Simulations of Malware and Anti-malware in a Network using CD++ (lopez service)

Wubin Ouyang(100934089)
Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON. K1S-5B6 Canada
Wubin.ouyang@carleton.ca

***Abstract*:** Nowadays, services of computer network have experienced a dramatic increase. However, the availability of these brand-new services also increases the vulnerability to malwares. In fact, malwares have put a serious threat on networks and computers in it. Malware is a general topic which includes several types of programs such as Worm and Trojan. Besides, hackers have played a more important role in the spread of malwares. In this paper, two models of malwares and their spread will be simulated. To characterize the propagation dynamics of malwares, we propose two modelling schemes using a two-dimensional cellular automata in a new version CD++, which supports multiple ports and multiple state variables in a cell. The first model mainly focuses on a relatively static model of malware, while the second model introduces three characters to make simulations more dynamic and approximate to the real world. The effectiveness and rationality of the proposed models have been validated through a series of simulations.

Keywords—Simulation; Malware; Anti-malware; Worm; Network security; Cellular automata; Parallel CD++;

I. INTRODUCTION

Nowadays, the use of personal computers and the computer network makes it possible that people from all over the world can communicate and share their information in seconds. However, this increasingly developed field also raises a significant worry in terms of security issues. Various malwares have been produced and spread on the Internet, which threatens the security of Internet and makes losses on Internet users. There is not a satisfactory definition of a computer virus because this notion has been overloaded with many definitions over the years [1]. However, typically a computer virus is a hidden and malicious program that infects a computer by copying itself to other programs or files.

Among kinds of malwares, worm is a serious kind in recent years. Worms are self-replicating computer viruses, which can propagate through computer networks without any human intervention. They have been rampant in the Internet for more than two decades. The dramatic developments of Internet and its services as well as the features of worm virus make hackers more appalled to write worms and use them to make illegal profit. Thus worms have threatened the network for years and are still challenging to Internet users and network administrators.

Mathematical epidemiology has existed for over a hundred years. Epidemic modeling is used to imitate the spreading of infectious diseases for a given population, such as H1N1, SARS, and influenza [2]. Infected individuals spread the virus to healthy individuals that they contact with. Because worms are very similar to biological viruses in their self-replicating and spread behaviors, epidemiological models has been used to model the propagation of Internet worms over the past decades. The study of computer worms in general, and Internet worms in particular, is a very popular topic of research.

In this paper, we use cellular automata as an efficient to characterize malware propagation. In fact, cellular automata can model the computation capability characterizing physical, biological or environmental complex phenomena, such as growth processes, reaction-diffusion systems, epidemic models, and the spread of forest fire. In our simulations, every cell in a two-dimensional plane is regarded as a computer in the network. Two different models will be proposed. The first model is a simplified modelling, which mainly focuses on the propagation of a worm that is brought to a LAN. The worm cannot get updated and it can be eliminated by every cell in the network initially. In this simple simulation, we just want to demonstrate how a worm can be modeled in cellular automata properly and how we can analysis the outcome of this simulation. By modifying and rewriting the model in a new version CD++ (lopez), we then implement the second modelling. The second modelling is complicated, which assigns different characters to each cell. The worm is now generated by an attacker in the network instead of being brought to. Moreover, worms can be updated by the attacker at random time interval. Another new character is introduced as policeman in the simulation. This simulation can be called a simulated network battle in the network, which may be more approximate to the real world and network in some cases.

The remainder of this paper is structured as follows: In Section 2, we provide an overview of related work. In Section 3, we will present the initial modelling of worm. In Section 4, we will have a discussion on how to improve the modelling in new version CD++ (lopze). We present the new simulation in Section

5, and analysis the outcome in Section 6.Finally, we draw a conclusion in section 7.

II. RELATED WORK

Since malwares have raised attentions largely in the world, many works have been done to simulate malwares and their different types of propagations in different network. A number of studies have demonstrated the threat of malware in the network.

Among these related work, most epidemic models have focused almost entirely on the technology of the differential equations and the Markov chain. Although most previous work can provide some valuable insight into the characteristics and dynamics of worm propagation, the models based on differential equations fail to capture the local characteristics of spreading processes, nor do they include interaction behaviors among individuals. Furthermore, the models based on the Markov chain are difficult to describe the spatial–temporal process of worm propagation.

In recent years, cellular automata have been used as an alternative to model epidemics as well as worm propagation in networks. Several papers have been published to demonstrate that cellular automata is a powerful tool to model this sort of issues. Some papers [1] [3] think that a cell representing computer in a network has three states, S (susceptible), which means computers are those that have not been infected by the computer virus; E (exposed), Exposed computers have been infected by the virus but it is non-activated; or I (Infected), infected computers are those that are infectious, and the virus is activated and it is able to propagate to another computer. In a modelling of worm propagation in smartphones [2], five states of cell are proposed. Susceptible state: nodes have not been infected by any worm in the network but are prone to infection. Exposed state: nodes have been infected by the worm but have not spread the worm to the susceptible smartphone while transmitting data or controlling the messages sent to the phones for the time being. Infectious state: nodes have been infected by worms in the network and they may infect some nodes in state S. Diagnosed state: nodes have been diagnosed to be infected by some kind of specific worm. Another paper "Simulation of Worm Viruses Spread in Network Based on Cellular Automata" [4] lists five states of a cell may belong to as "Susceptible", "Questionable", "Destroyed", "Infected" and "Immune".

Though there may be different states in papers, some common states can be concluded as "susceptible", "immune", "and infected". In our simplified simulation (first modelling), we propose a four-states model to show the spread of worm, which is composed by S (susceptible), Q (questionable), I (Infected), R (Recovered). Further details are provided later. And in our complicated simulation (second modelling), each cell can have five different states as "unprotected", which means the cell has a potential to be infected later; "Immune", in which the cell has been loaded with anti-malware, "Isolated", which means the cell has isolated itself from the network and "Eliminated" representing the attacker has been wiped out.

Through simulations demonstrated later, it can be proved that this type of states is reasonable and similar to the real world.

III. STATIC WORM MODELLING

In this section, we propose a static worm modelling in Cell-DEVS. The worm modelled here is assumed being brought to the LAN, and cannot get updated. Every cell in this network has the ability to check out it has been infected and can wipe out the malware by itself. We call it static worm because the worm cannot be updated in this simulation. Later, in the second modeling, we will propose a model in which worm or malware is generated in the network and gets updated constantly, which can be regarded as a dynamic modelling of malware and its propagation.

In this simulation, we introduce several states and parameters to make it more realistic. However, parameters may depend on the functionality of a worm and the level of awareness of users. Even the pattern of communication in this LAN may play an important role in the process of spread. Thus, in this simulation, we do not expect to achieve a precise process of spread. Instead, we provide a framework that can adjust to different type of worms and LANs by setting different parameters. And in the following demonstration, several examples will be explained to show this idea.

A. Cellular definitions

The following is the formal definition for the CELL-DEVS model.

CD = < X, Y, I, S, θ, N, d, δint, δext, τ, λ, D >
X = Ø
Y = Ø
S = {-1, 0, 1, 2}
N = { (-1, 0), (0, -1), (0, 1), (1, 0), (0,0), (-1, -1), (-1, 1), (1, -1), (1, 1),(-2, -2), (-2, -1), (-2, 0), (-2, 1), (-2, 2), (-1, -2), (-1, 2), (0, -2), (0, 2), (1, -2), (1, 2), (2, -2), (2, -1), (2, 0), (2, 1), (2, 2) }
d = 100 ms
δint: based on the rules explained later.

Fig. 1. Cell-DEVS definition for the static worm modelling

B. States and transitions

To balance the accuracy and complexity, we will use a four-state model to show the spread of worm, which is composed by S (susceptible), Q (questionable), I (Infected), R (Recovered).

The four states and their descriptions are shown below:

Values	States	Descriptions
0	S(Susceptible)	Not infected and immunized, may be infected in the future, or turn into questionable.
-1	R(Recovered)	Recovered from an infected state Computers will not be infected again.
1	Q(Questionable)	computer is questioned
2	I(Infected)	computer is infected

Table 1- States of cell.

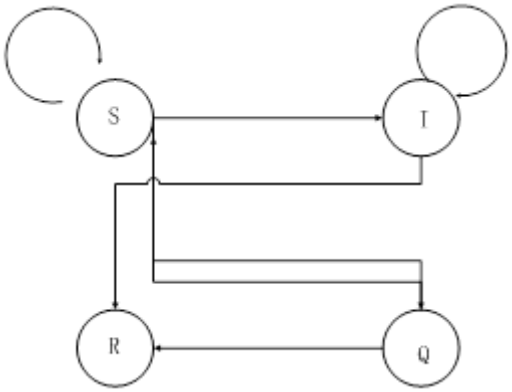


Fig. 2. States transitions of cells.

C. *Two kinds of neighbor*

To distinguish influences among different neighbors, we introduce two kinds of neighbor. The first sort is called “adjacent category” which is next to the central cell. Another one called “remote category” which is a little farther. In my model, the “adjacent category” includes cells of { (-1, 0), (0, -1), (0, 1), (1, 0), (0,0), (-1, -1), (-1, 1), (1, -1), (1, 1) }, while the “remote category” includes { (-2, -2), (-2, -1), (-2, 0), (-2, 1), (-2, 2), (-1, -2), (-1, 2), (0, -2), (0, 2), (1, -2), (1, 2), (2, -2), (2, -1), (2, 0), (2, 1), (2, 2) }.

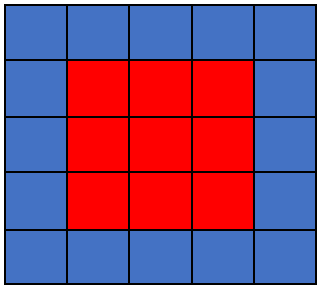


Fig. 3. Two kinds of neighbors.

In this model, we consider a LAN containing 2500 computers. These computers do not have access to outer network (e.g Internet). The worm is initially rooted in one or several computers, assuming that it is produced deliberately or brought in incautiously. Computers in this LAN have 24 contacts at most, which can be used to spread the worm. 16 of them are in “remote category”, while others are in “adjacent category”. Cells in “adjacent category” have more influences than ones in “remote category”. In this model, this feature is implemented by two macros “Inner_Factor” and “Outer_Factor”.

D. *Simulations*

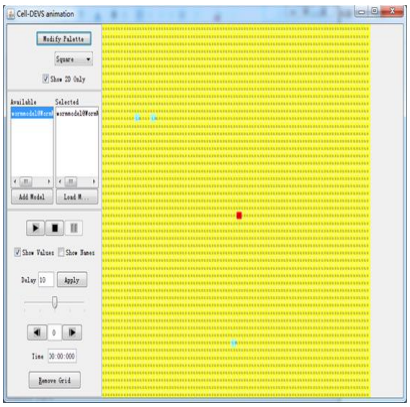


Fig. 4. Initial scene of simulation

In this graph, red cell is in the state of infected, while yellow one is in state of susceptible and green one is in the state of recovered. Later, a blue cell indicates that the cell is in the state of questionable. Initially, we put one infected cell and three recovered cells which will not get infected in the whole process of simulation.

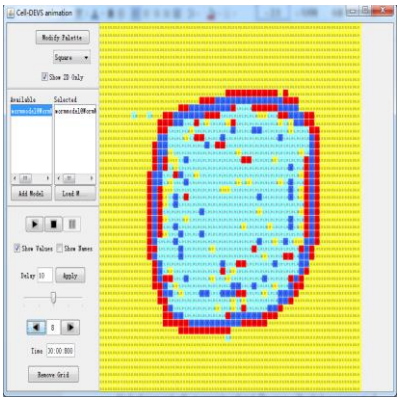


Fig. 5. Intermediate scene

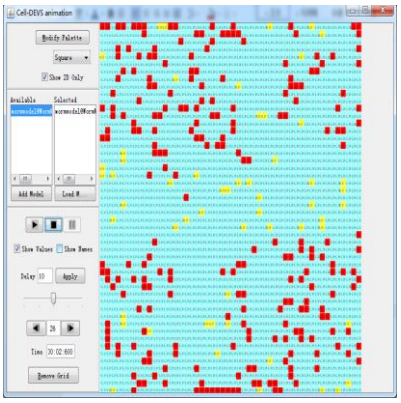


Fig. 6. Final scene

In the final scene, red cells are still in the state of infected. But because of the large number of recovered cells, it cannot spread the worm any more. The infected cells are then isolated to patches. The yellow cells are in the state of susceptible. Similarly, due to the block of recovered cells, it cannot be

infected. The green cells which account most of the 2500 cells are recovered, and this decisively contributes to the end of spread.

IV. IMPROVEMENTS

A. New situation

In the previous section, we have witnessed a static worm and its propagation in a network. The worm in that simulation is assumed been brought to the network. That means, it has no likelihood to get updated. And finally, as we expected, the propagation only experienced a short time and almost every cell in the network is secured at the end of the simulation.

However, in a real world, situations may be more complicated. First, a malware cannot always be static as we assumed in last section. While hackers and worm writers pay more attentions to the network attack, malwares may be updated frequently to avoid being killed by anti-malwares. Moreover, in recent years, an increasing number of malwares have features of both Trojan horses and Backdoor, which makes hackers or worm writers easier to update their worms.

Therefore, in the second modelling, we update our model by making worms have ability to be updated. Correspondingly, cells now are not identical absolutely. We assign cells with three roles, "attacker", "civilian", "policeman" to make this model more like to a real world. Attacker is supposed to generate the malware and updates it at a random time interval; Civilian is average cell, which can be infected by malware or get immune by anti-malware. Policeman is supposed to respond to malware by producing anti-malware to fight against it. The anti-malware can also spread in the network cell by cell to wipe out the malware. The final scene may be in two situations. The first situation is that the malware and attacker are wiped out from this network, while in the second situation, malware is still rampant in the network.

B. New requirements

Based on the discussion, the modelling is getting more complicated, compared with the first simulation. Inevitably, the new model demands a more complex structure and definitions to implement.

First, in the previous simulation, every cell can be seen as identical to each other. But in this new model, we assign cells with different roles. That means a cell should keep its identity in the entire simulation. Second, malwares are not static any more, which involves the information of version in this modelling. Finally, policeman can also update its anti-malware, which also requires extra structure to implement.

In the standalone version CD++ which we used for the first modelling, it is hard to implement these new features. An alternative method is to transform the two-dimensional model to three-dimensional model, and use new layers to keep extra information and implement the adding functionality. To be honest, we have tried this method and found it is a limited alternative which is hard to design and modify. Fortunately, a new version CD++ that can satisfy these requirements fairly.

C. New version CD++ (lopez)

The Cell-DEVS formalism [5] extended the DEVS formalism allowing the simulation of discrete-event cellular models. CD++ [5] is a DEVS/Cell-DEVS modeling and simulation toolkit. It is a standalone version running on a local PC, which simulation results can be seen in 2D scenes using CD++. RISE (RESTful Interoperability Simulation Environment) [6] is a simulation middleware to support RESTful-CD++ web services for the remote simulation, which aims to support interoperability and mash-ups of distributed simulations regardless of the model formalisms, model languages or simulation engines [7].

In the second modelling and its simulation, we use a new version called "lopez". This new version CD++ supports multiple ports and multiple state variables, which is powerful to simulate a complicated modelling. The details of implementation is going to be provided in the later sections.

V. MODELLING DEFINITIONS AND TESTS

In this section, we will model the complicated and advanced simulation by each functionality. And tests attaching to them will be implemented and analyzed to make sure it works properly.

A. Attacker generates malware and update it

In this section, the process of generating and updating malwares will be described. And the scene in a new version CD++ will be shown as well.

The role who generates malwares and updating them is attacker. The aim of attacker is to make malwares and spread them to other computers who have not anti-virus software or just have an obsolete version of anti-virus software that cannot check the malwares out and kill them. The attacker can then control these infected computers to do what he wants.

The time interval of updating malwares is not fixed. In this case, we introduce a timer which randomly increases its value every 100ms. When the value of timer exceeds a fixed number, then a new malware will be generated and the timer is set to be 0 and counts again.

Besides the timer, we need to introduce another variable to keep and update the version number of the current malware. When a new malware is generated, the version number is added by two. A port will output the new version number of the variable.

With powerful multiple input and output ports and multiple state variables of the new version of CD++, we can properly implement the process of generating malwares. Based on the description of attacker and its behaviors, we can introduce several ports and state variables to implement it. The attacker is a cell in a 2-D plate. To distinguish it from other cells, we introduce a state variable "$identity" to represent it identity of being an attacker. And it also requires a port to output the current version of malware to inform other cells. We then introduce a port called "~virusvers" to implement this feature. Besides, it also need a state variable to record the current version of malware. Thus a state variable called "$virusvers" is introduced

to realize it. The cell of attacker also requires a state variable as a timer to implement a random time interval between issuing malwares and the state variable "$timer" is to be used in this case.

The rule for generating and updating malwares is shown as below:

```
rule : { ~virusvers := $virusvers ; } { $timer := $timer +
uniform(4, 6) ; } 100 { $identity = 100 and $timer < 60 }
rule : { ~virusvers := $virusvers ;} { $timer := 0 ;
$virusvers := $virusvers + 2 ;} 100 { $identity = 100 and
$timer > 60 }
```

As you can see, the attacker's identity is marked as 100 and it updates the malware among every 1000ms to 1500ms. Every update will make the version number increase 2 to a new value.

There may be a doubt that why not we use a random delay time in rules instead of an introduction of state variable timer. This is because in the entirely completed simulation we design that the attacker will be eliminated finally by anti-virus software. If we use a random delay time here, the attacker may "revive" even it has been eliminated. This phenomenon is due to the mechanism of DEVS. Therefore, we use a fixed delay and implement a random function by using a timer.

The graph below shows the simulation scene of generating and updating malwares.

Fig. 7. Scene of cells at 00:00:00:00

This graph shows that at the very beginning, there is no malware among these cells. But an attacker has been put among these cells. We will witness its generating and updating malwares in following graphs.

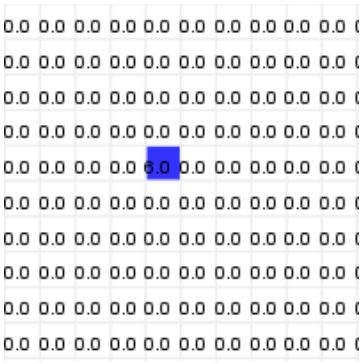


Fig. 8. Scene of cells at 00:00:01:50

This graph shows that at the 1500ms from the beginning of simulation, the attacker begins to generate malwares. The current version number is 6 and marked blue.

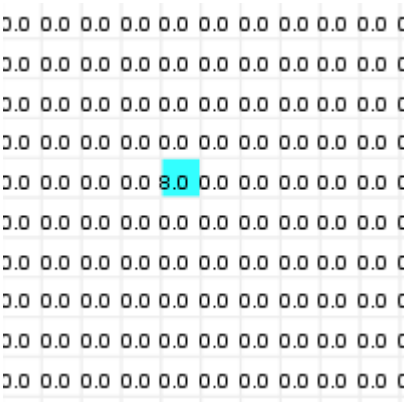


Fig. 9. Scene of cells at 00:00:02:90

This graph shows that at the 2900ms from the beginning of simulation, the attacker updates the malware. The current version number is 8 and marked green.

Fig. 10. Scene of cells at 00:00:04:10

This graph shows that at the 4100ms from the beginning of simulation, the attacker updates the malware. The current version number is 10 and marked yellow.

From the graphs shown above, we witness the process of generating and updating the malware. The version of malware is increasing during a random time interval. Notice that the value of 6 lasts 1400ms, while the value of 8 lasts 1200ms.

In this section, the function of generating and updating malware by attacker is proved correct. In next section, the process of malware spread will be demonstrated.

B. Spread of malware

In last section, we introduced the process of generating and updating malware by the attacker in a network. The goal of attacker is to spread it as widely as possible. Then it can intrude and even control these infected computer.

This process can be divided to two stages. In the first stage, an unprotected cell is infected by the malware. Then its state changes from "unprotected" to "infected". In the second stage, a new version malware with higher value of "$virusvers" will

replace the current version of malware in the cell. The state of cell is still “infected”, but the cell has been infected again by an advanced malware with higher version value.

Here we assume that every cell can be affected by at most 8 cells directly in the simulation. That means a cell has 8 neighbors. An infection or malware update in a cell will notice its 8 neighbor cells. Neighbors then checks if the version of malware is higher than their current versions. If so, then neighbors are also infected or update their version of malware to make it as same as the cell which informs them.

These features in spread of a malware require several ports and state variables. First, we need the state variable “$identity” to declare cells as average computers in network, which means that these computers are not attacker or police and they can be infected. Besides, these cells also need a port to inform other cells their new version of malware. Therefore a port called “~virusvers” is introduced here as well as a state variable “$virusvers” which is used to record the current version number of the cell. Finally, a port called “~state” is introduced to output the state of cell to its neighbors.

Rules for the spread of malware are shown as below:

```
rule : { ~virusvers := #Macro(virusspreading) ; ~state := -
1 ; } { $virusvers := #Macro(virusspreading) ; } 100
{ #Macro(virusspreading) > $virusvers and $identity = 0 }
```

Here we use a macro “virusspreading” to find out the highest version number among a cell’s 8 neighbors. The macro is defined as below:

```
max( max( max((0,-1)~virusvers,(-
1,0)~virusvers),max((1,1)~virusvers,(0,1)~virusvers)),max(
 max((1,0)~virusvers,(1,-1)~virusvers),max((-
1,1)~virusvers,(-1,-1)~virusvers)))
```

A simulation of spread of malware is implemented. And the scenes are shown in the following graphs.

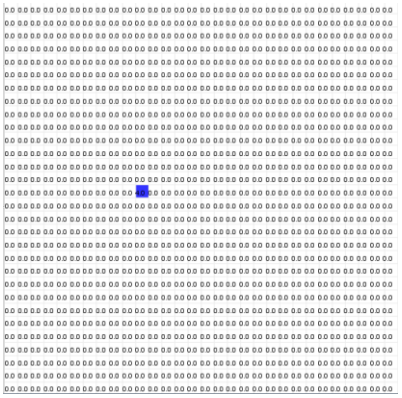


Fig. 11. scene of cells at 00:00:00:15

This figure shows that at the beginning, the attacker generates a malware. But it has not spread yet. The blue cell represents the attacker and white cells represent unprotected computers without infection.

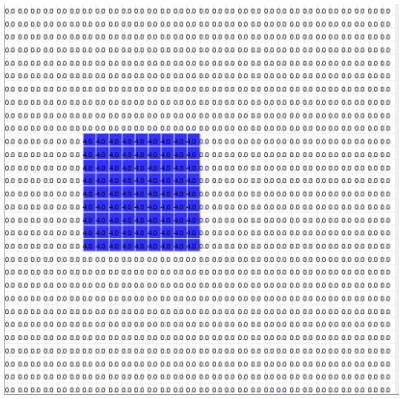


Fig. 12. scene of cells at 00:00:00:50

This scene shows that at the 500ms from the beginning of simulation, the malware has spread to cells near the attacker. The infected cells are marked blue indicating the current virus version is 4.

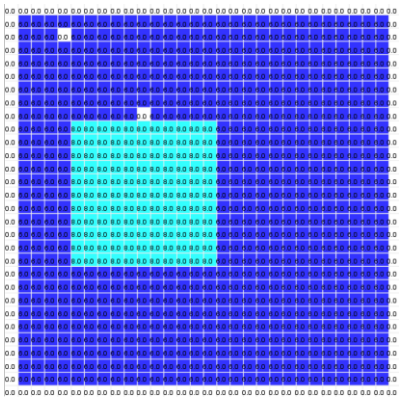


Fig. 13. scene of cells at 00:00:03:40

This scene shows that at the 3400ms from the beginning of simulation, almost every cell in network has been infected. Moreover, that attacker also has updated its malware and spread it. The infected cells with a new version of malware are marked green whereas the cells with an old version are marked as blue.

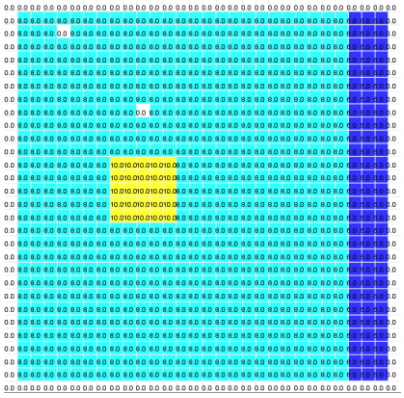


Fig. 14. scene of cells at 00:00:04:40

This scene shows that at the 4400ms from the beginning of simulation, there are three versions of malware in the network. The nearer the cells from the attacker, the newer version of

malware is harbored. It indicates that the new version of malware can replace the old one and then spread again.

These four graphs depict the scenes of spread of malware. However, in these cases, unprotected cells are infected unconditionally. The realm of infected cells shown in these graph are always square, which reflects this feature. It may not be realistic if we want to make the simulation approximate to a real case.

In an advanced rule of spread, we take a factor of possibility into account. Cells are not infected unconditionally. Instead, it has a possibility to be infected. The advanced rule is:

```
rule : { ~virusvers := #Macro(virusspreading) ; ~state := -
1 ; } { $virusvers := #Macro(virusspreading) ; } 100
{ #Macro(virusspreading) > $virusvers and
#Macro(virusspreading) > $anvirusvers and $identity = 0
and random <  ( #Macro(vPossibility_Of_Spreading)) }
```

The scenes of running this advanced rule are shown below:

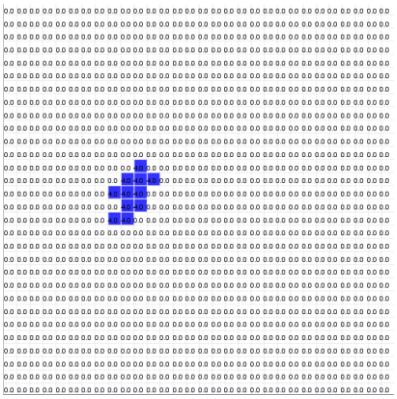


Fig. 15. scene of cells at 00:00:00:30

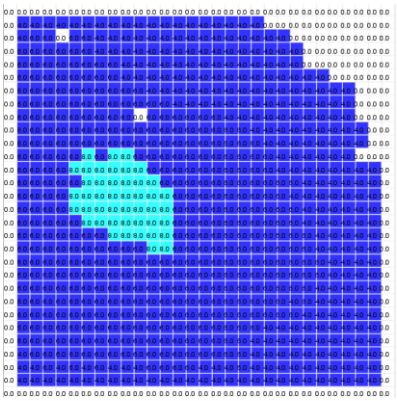


Fig. 16. scene of cells at 00:00:03:20

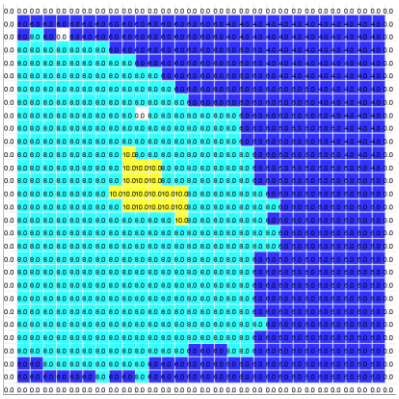


Fig. 17. scene of cells at 00:00:04:40

These graphs show that the realm of infected cells shown in these graph are not square any more, which indicates that a cell has a possibility to be infected by its neighbors instead of being infected unconditionally. It is more similar to the real network.

In the later section, we will also assign different weigh to a cell's eight neighbors. The possibility of a cell to be infected is also depending on the distance from the infected neighbor. This will add more reality to the simulation. We will give the details in the later sections.

C. The producing of anti-malware software

In this section, we will talk about the process of producing of anti-malware software and simulate it in the new version CD++.

In the simulation of a network, we also put several policemen in the cells. The policemen cells have a technique that exempts them of being infected by malware. That means the policemen are always isolated from the cells of being infected.

But the police forces are not ignoring what is happening in the network. Actually, they will monitor their eight neighbors and check out if they are infected. An infected neighbor will stimulate the police cells to produce an anti-malware software against the current version of malware. The anti-malware software also has a number to indicate its current version. Moreover, the version number of an anti-malware software is always higher than the version of malware it can eliminate. For example, a malware with a version number of 8 may incent the police forces to produce an anti-malware with a version number of 9, and then it may be eliminated by this anti-malware.

It is should be admitted that the action of policemen is always later than the counterpart of attacker. That means a policeman in the network cannot produce an anti-malware software that can eliminate the malware that is going to be generated by the attacker. Thus, the police forces should respond to a new version of malware rapidly and positively. In this simulation, the police forces generate the anti-malware software unconditionally when they encounter a new version of malware.

Another situation may also happen. What if a policeman encounter a malware which has a lower version number compared with the newest anti-malware software's. Here, the

policeman will ignore this situation because an anti-malware software has been deployed to eliminate this malware in the network.

Based on the discussion we made above, some ports and state variables are demanded to implement the function of producing and updating anti-malware software. First, a policeman should output the current version number of anti-malware to its neighbor. That is required in the spread of anti-malware software, whose process will be talked about in the later section. Thus a new port called "~anvirusvers" is introduced to implement this feature. Also, a state variable "$anvirusvers" is used to record the current version number of anti-malware software. Besides, the police forces also need an identity marker to exempt them from being infected. The state variable "$identity" is used again to indicate the police cells. By the way, a port called "~state" is added to inform the neighbors of policeman that here is a policeman.

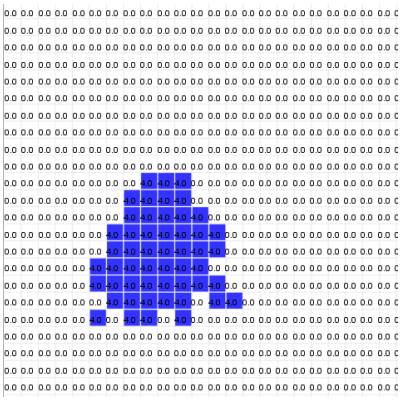
The rule of producing and updating new version of anti-malware software is shown as below:

The rule of producing and updating new version of anti-malware software is shown as below:
rule : { ~anvirusvers := #Macro(virusspreading) + 1 ; ~virusvers := 0 ; ~state := 2 ; } { $virusvers := 0 ; $anvirusvers := #Macro(virusspreading) + 1 ; } 100 { (#Macro(virusspreading) > $anvirusvers or #Macro(anvirusspreading) > $anvirusvers) and $identity = 200 }

In this rule, every time the policeman encounters a new version malware, it updates the anti-malware software. The new version number of anti-malware software is the version number of malware plus 1. This ensures that the malware can be wiped out by the anti-malware software.

Several scenes below demonstrate the process of producing anti-malware and updating it.

The following graphs show the process of two policemen in the network producing anti-malware and updating it according to a new version of malware. This time, we need combine graphs of spread of malware and graphs of policeman producing anti-malware together to get a story about what happens.



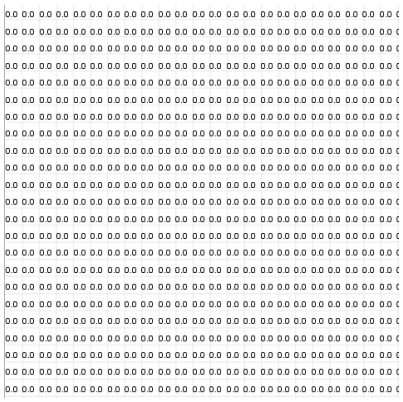


Fig. 18. Versions of malware and anti-malware at 00:00:00:60

This figure depicts the scene at 600ms from the beginning of simulation. In the upper graph, it can be seen that the malware has begun to spread. Several cells have been infected. But in the lower graph, a fact can be drawn that the policeman has not responded to the malware and its spread yet.

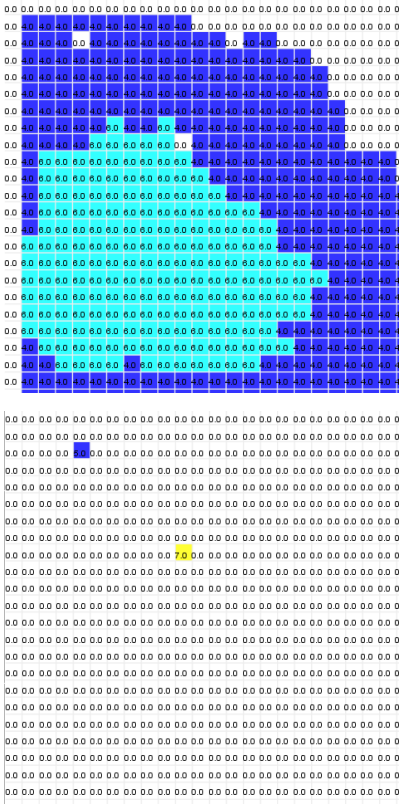


Fig. 19. Versions of malware and anti-malware at 00:00:02:40

This figure depicts the scene at 2400ms from the beginning of simulation. In the upper graph, it can be seen that the malware has updated its version number to 6. And in the lower graph, a fact that the policeman marked yellow has also detected this malware and updated its version of anti-malware to 7 to fight against the malware.

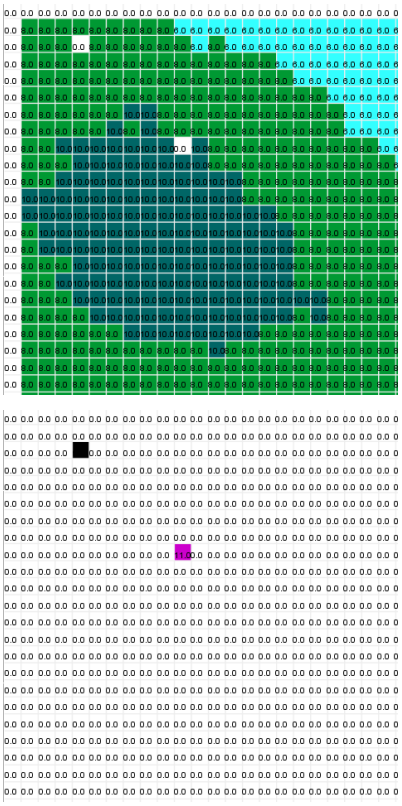


Fig. 20. versions of malware and anti-malware at 00:00:04:90

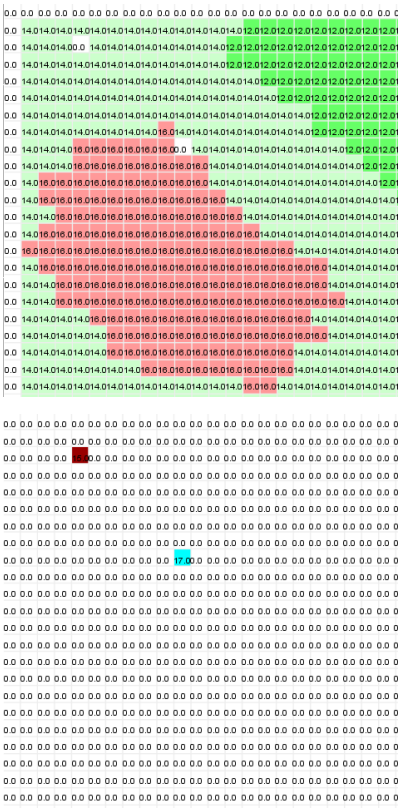


Fig. 21. versions of malware and anti-malware at 00:00:09:20

These graph shows the process of policemen producing and generating the version of anti-malware software to fight against the malware in the network. The functionality has been proved correct through a bunch of simulation in the new version CD++.

Only generating and updating the anti-malware cannot secure the network and eventually eliminate the attacker. We will talk about the spread of anti-malware software in the next section.f malware and anti-malware at 00:00:09:20

D. Spread of anti-malware in the network.

In this section, the process of spread of anti-malware in the network will be described.

Last section has disclosed the mechanism of producing and updating anti-malware software by a policeman in the network. As we said early, only producing and updating anti-malware cannot secure the network and eliminate the attacker finally. A policeman needs to spread its anti-malware software to its neighbors and eventually the whole networks to fight against the malware made by attacker.

Typical anti-malware software cannot spread in the network automatically. Therefore, compared with the spread of malware, the spread of anti-malware software is more passive, which leads to a poor efficiency to fight against the malware in the network.

Take this feature into account, we decide to design a pattern of anti-malware which can also spread from one computer to others automatically. This process is similar to the process of spread of malware. In other words, anti-malware now is regarded as a “malware” as well in terms of spread. The difference between anti-malware and malware is that the anti-malware will not do harm to the computer, and it will wipe out the malware in the computer if it has a higher version number.

As depicted in the previous section, the producing and updating anti-malware is triggered by intrusion happened in a neighbor cell of the policeman. After generating a corresponding anti-malware software, from the cell of policeman, this anti-malware begins to spread neighbor by neighbor. In this course, three situations may happen. The first situation is that the cell has a malware in it, but the version of malware is lower than the advancing anti-malware. Then the malware will be wiped out and the anti-malware is loaded. In the second situation, the cell has a malware with a higher version number compared with the advancing anti-malware. In this case, the anti-malware is unable to eliminate the malware in this cell because it is obsolete. In the final situation, the cell is already secured by having a previous version of anti-malware. The new anti-malware with a higher version number will replace the obsolete version of anti-malware, otherwise, the cell keeps its current anti-malware. This situation can be regarded as an update of anti-malware in cells.

Based on the discussion above, we can find that the spread of anti-malware is also triggered by the spread of malware. We need take the process of spread of malware into simulation of anti-malware as well.

Before we get into the simulation, we should figure out the required ports and state variables to implement the spread of anti-malware. Here, a cell needs a port “$anvirusvers” to output

the current version number of the anti-malware software the cell owns. Besides, a state variable “$anvirusvers” is demanded to record the current version number of anti-malware. A port called “state” is used to output value of 2 when the cell is loaded with the new anti-malware. Finally, a state variable “$indentity” is used to mark the cell as a “civilian” which can be infected by both malware and anti-malware. The rule of spread of anti-malware is as below:

```
rule:{~anvirusvers:=#Macro(anvirusspreading);~virusvers:
=0;~state:=2;}
{$anvirusvers:=#Macro(anvirusspreading);$virusvers:=0;}
100 {#Macro(anvirusspreading)>$anvirusvers and
#Macro(anvirusspreading)>$virusvers and  $identity =0}
```

The macro “anvirusspreading” is introduced to find the newest version of anti-malware in a cell’s 8 neighbors. The anti-malware will get updated when the version number of anti-malware in the cell’s neighbors are higher than the cell’s current version number of anti-malware, or the cell is infected and the coming anti-malware has a higher version number than the malware, which means the anti-malware is able to defeat the malware.

After being loaded the new version of anti-malware, the cell will output the current version number of its anti-malware as well as a value of “2” to indicate that it has been immune without any malware in it.

The graphs below show the simulation of spread of anti-malware. Notice that in spreads of both malware and anti-malware, the cell will output “-1” or “2” to indicate that it has been infected or loaded with anti-malware. Thus, in this simulation outcome, we just look at the value of port “~state” and ignore the concrete version numbers.

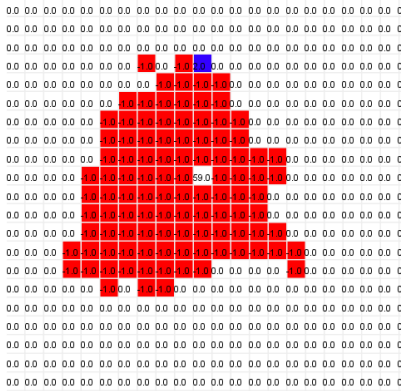


Fig. 22. states of cells at 00:00:00:80

This graph depicts the spread of malware and the producing of anti-malware at 800ms from the beginning of this simulation. The cells marked red are infected and the blue one is the policeman in the network who has produced an anti-malware and is ready to spread it.

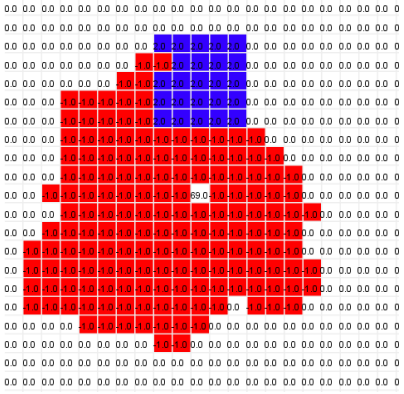


Fig. 23. states of cells at 00:00:01:00

This graph depicts the spread of malware and the spread of anti-malware at 1000ms from the beginning of this simulation. The anti-malware has occupied some cells which were unprotected or infected before. However, the malware is still spreading in other directions.

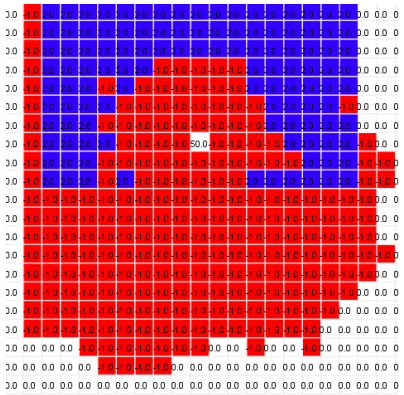


Fig. 24. Figure states of cells at 00:00:01:60

This graph depicts the battle between malware and anti-malware at 1600ms from the beginning of this simulation. The new version of malware is retaking cells which were occupied by previous anti-malware. The policeman may capture the new version of malware and be producing a new version of anti-malware.

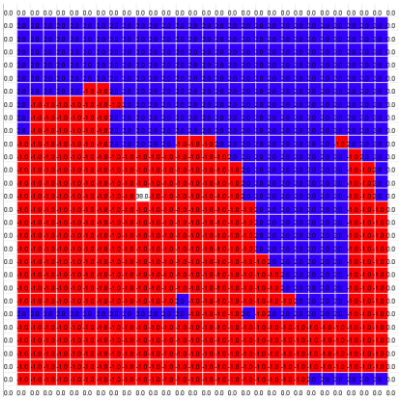


Fig. 25. Figure states of cells at 00:00:06:70

This graph depicts the battle between malware and anti-malware at 6700ms from the beginning of this simulation. The battle is still ongoing. Both malware and anti-malware take half of the whole cells in the network. The battle may last and finish at the moment the attacker is being wiped out, or has no ending because of the frequent update of malware made by attacker.

In this section, the spread of anti-malware has been proved correct in this simulation. At this time, most important functionalities in this simulation have been introduced and verified.

E. Improvements in spread of malware and anti-malware.

In this section, we will make some modifications and improvements to the rule of spread of malware and anti-malware.

In reality, effects that neighbors put on a cell may not be identical. For example, a computer may frequently interact with some computers, but only has little interaction with others. Thus, the possibilities of being infected through different neighbors are different as well.

In this simulation, we divide the 8 neighbors of a cell to two different group. The first group has a fewer weigh while the second one has more weigh when interacting with the central cell. The neighbors (1,-1), (-1,1), (1,1), (-1,-1) are in the "lighter" group, which have weak effects on the cell (0,0). Other neighbors (1,0), (-1,0), (0,-1), (0,1) are in the "heavier" group, which have strong effects on the cell (0,0). This setting represents the reality we discussed above. We assign one weight to the "lighter" neighbors, and assign three weights to the "heavier" neighbors. This assignments are set in the macros "vFactor_Outer" and "vFactor_Inner".

Another improvement involves states of a cell. In the discussion above, average cell ($identity=0) only has three states, "unprotected", "infected" and "immune". However, some computers' owners may have a high vigilance. When they find some neighbors have been infected, they may isolate their computer from the network for a while to prevent potential infection. In this course, the cell changes its identity from civilian ($identity = 0) to "hermit" ($identity = 150). And it cannot be exploited to spread the malware and anti-malware. We assign an output at port "~state" with the value of 3 to indicate a cell is isolating itself from the network. The rule is shown as below:

By this rule, when a cell finds that more than one of its neighbors have been infected, it will have a possibility (0.6) to isolate itself from the network by changing its identity value to 150. Besides, "hermit" can also return to the state of "unprotected". This reflects the possible scene that a computer owner reconnects his computer to the network after considering the network may be safe even he had isolated it before. The rule is designed as below:

```
rule : { ~state := 3 ; } { $identity := 150 ; } 100
{ ( #Macro(vFactor_Outer)/2 +
#Macro(vFactor_Inner) ) > 2 and (0, 0)~state != -1
and (0, 0)~state != 2 and random < 0.6}
```

By this rule, when a cell finds that more than one of its neighbors have been infected, it will have a possibility (0.6) to isolate itself from the network by changing its identity value to 150. Besides, "hermit" can also return to the state of "unprotected". This reflects the possible scene that a computer owner reconnects his computer to the network after considering the network may be safe even he had isolated it before. The rule is designed as below:

```
rule : { ~state := 1 ; ~anvirusvers := 0 ; ~virusvers :=
0 ; } { $identity := 0 ; $anvirusvers := 0 ; } 100 { (0,
0)~state = 3 and random < 0.2 }
```

This scene happens in a low possibility (0.2). And if it happens, the cell changes its value of "$identity" from 150 to 0, indicating that now it is unprotected and may be infected in the future.

In the improvements and modifications we did above, possibilities are introduced to make this simulation more realistic and more flexible. If we want to change the model slightly, we may only need to adjust the possibility in rules, which is easy and direct.

F. Cell-DEVS model

By now, we have completed the modelling and simulations of every functionality. And we will do a series of simulations with different parameters in next section. Before simulations, we summarize the modelling by listing its parameters and states in DEVS form.

States(~state)	values	descriptions
unprotected	0 or 1	Cell can be infected
Immune	2	Cell has been loaded with anti-malware
Infected	-1	Cell has been infected.
Isolated	3	Cell has isolated itself from the network
Eliminated	9	Mark the attacker who has been wiped out

Table 2- States of cell.

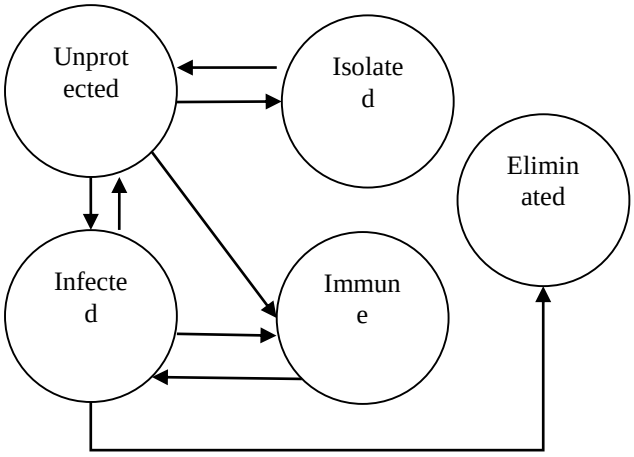


Fig. 26. State transitions

Ports	values	descriptions
~virusvers	2,4,6,8…(even)	Output the current version of malware
~anvirusvers	3,5,7,9…(odd)	Output the current version of anti-malware
~state	0,1,2,-1,9	Describe the state of cells

Table 3- Multiple ports of a cell.

Variables	values	descriptions
$virusvers	2,4,6,8…(even)	Keep the current version of malware
$anvirusvers	3,5,7,9…(odd)	Keep the current version of anti-malware
$identity	0, 100, 200,150	Represents the identity of cell
$timer	Less than 60	Generate random time intervals

Table 4- Multiple state variables of a cell.

VI. SIMULATIONS

We then do a series of simulations in different parameter settings. These simulations have different scenes which are interesting and meaningful, and they comply with different occasions in the real world.

A. Crazy attacker

In this scene, the attacker in the network updates his malware frequently. This makes policeman in the network has to update his anti-malware frequently as well and has little chance to wipe out the attacker finally.

We can make the timer faster to satisfy this setting. The timer adds a larger number every 100ms and it makes the value of timer easier to exceed the threshold 60, which triggers an update of malware.

The scenes of simulation are shown as below:

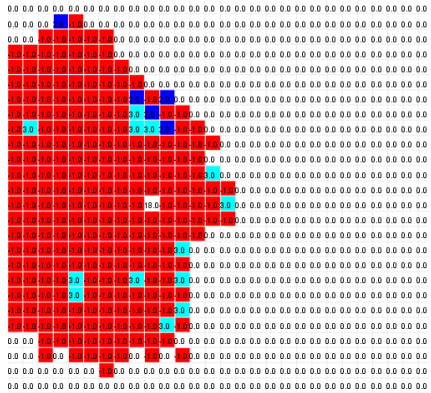


Fig. 27. state scene of cells at 00:00:01:70

At this time, the policeman detects there is a malware in his neighbors, and produces an anti-malware to fight against the malware.

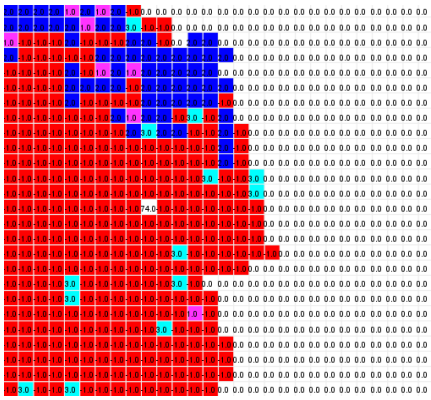


Fig. 28. state scene of cells at 00:00:02:10

After 2100ms from the beginning of simulation, the malware spreads fast and the anti-malware can only secure a small part of computers in the upper-left corner of this scene.

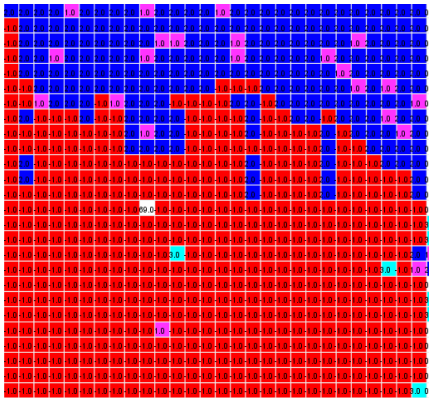


Fig. 29. state scene of cells at 00:00:04:00

After 4000ms from the beginning of simulation, every cell in the network has been involved to this battle. But the malware still controls most of the cells.

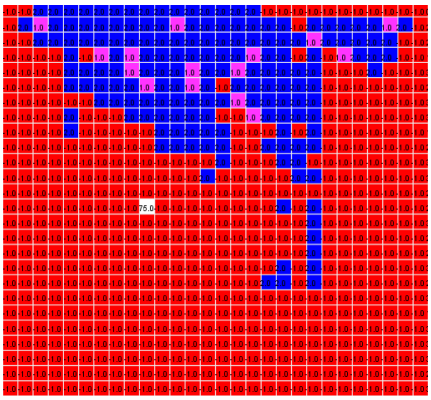


Fig. 30. state scene of cells at 00:00:18:00

This is the final scene of this simulation. The malware still controls most cells in the network. The policeman has no chance to eliminate the attacker because of the frequent update of malware.

This simulation demonstrates that a crazy attacker who frequently updates its malware may make the battle between attacker and policeman last for a long time and even there is no victory for policeman.

B. Lazy attacker

In this scene, the attacker in the network updates his malware infrequently. This makes policeman and his anti-malware wipe out the attacker eventually before it updates its malware.

We can make the timer slower to satisfy this setting. The timer adds a little number every 100ms.

The scenes of simulation are shown as below:

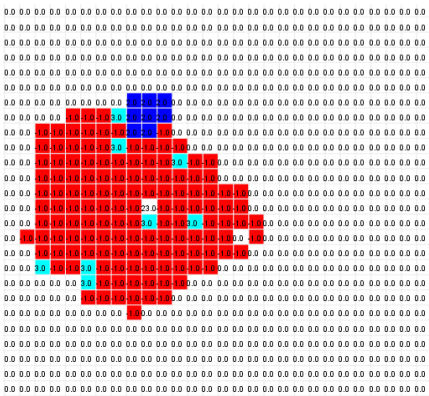


Fig. 31. state scene of cells at 00:00:01:00

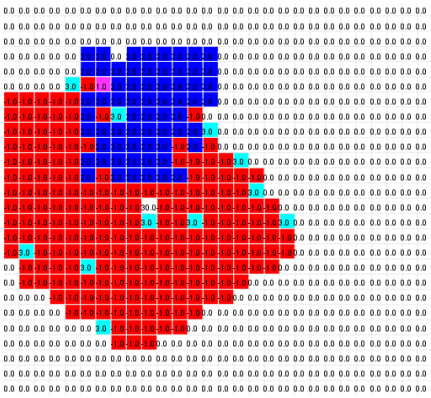


Fig. 32. state scene of cells at 00:00:01:30

At the 1300ms from the beginning of simulation, the anti-malware has strongly spread in the network, compared with the previous case of “crazy attacker”.

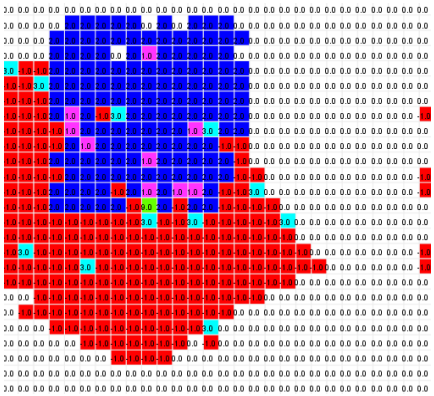


Fig. 33. state scene of cells at 00:00:01:50

At the 1500ms from the beginning of simulation, the anti-malware successfully wipes out the attacker before it updates a new version of malware.

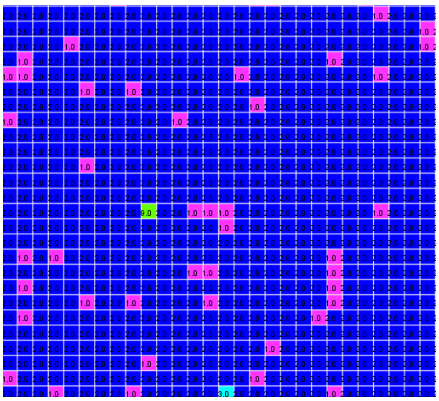


Fig. 34. state scene of cells at 00:00:03:30

At the 3300ms from the beginning of simulation, the anti-malware successfully secures or protects all the cells in the network from malware. It is an amazing moment that the malware is eliminated completely.

This simulation shows the fact that if attacker updates its malware infrequently, the policeman may have chance to wipe

out the attacker and eliminate malware at all in the network, though the mechanism of producing anti-malware is passive.

C. *Strong vigilance*

In the third simulation, we assume that every owner in the network has a strong vigilance that will make the spread of malware more difficult and the spread of anti-malware easier.

As introduced before, we only need to adjust the possibilities to implement this setting of simulation. We lower the possibility of spread of malware whereas we increase the possibility of anti-malware. And all other settings are identical to the test case “crazy attacker”. The scenes of simulation are shown below:

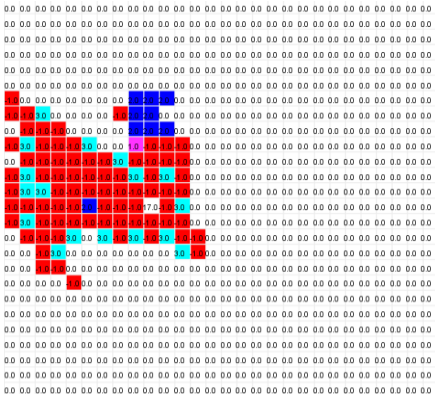


Fig. 35. state scene of cells at 00:00:01:90

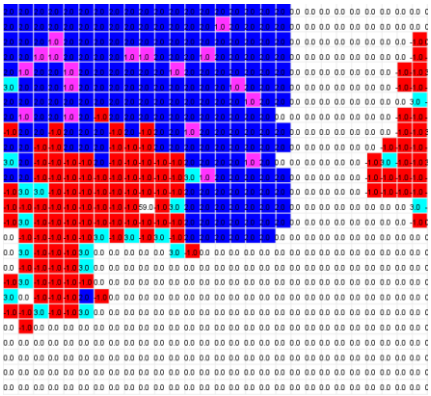


Fig. 36. state scene of cells at 00:00:02:70

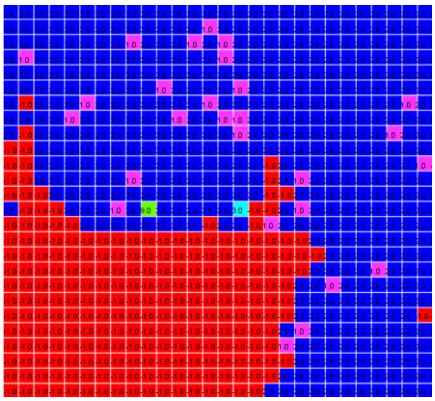


Fig. 37. state scene of cells at 00:00:13:00

At 1300ms after the beginning of simulation, the attacker is wiped out. It is truly a dramatic moment. In the first simulation of “crazy attacker”, there is no ending of battle, which means the police has no chance to wipe out the attacker. But in this simulation, the police succeeds to achieve this goal. This is due to a stronger vigilance among the cells, which prevents the spread of malware and encourages the spread of malware to some extents.

D. *More policemen*

The final simulation focus on the how the number of policemen in the network matters. 9 policemen are settled in the network with random positions. Other settings are identical with the previous case “crazy attacker”.

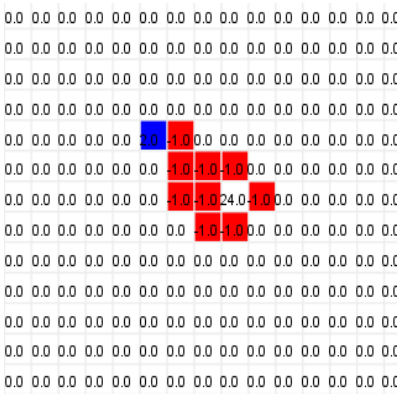


Fig. 38. state scene of cells at 00:00:00:60

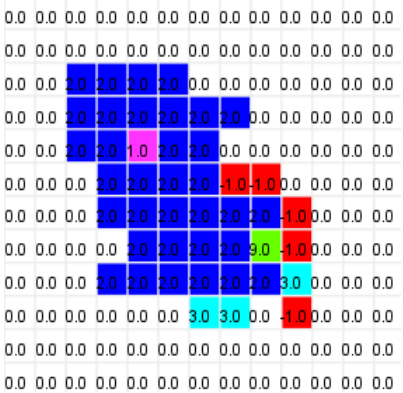


Fig. 39. state scene of cells at 00:00:00:90

At 1300ms after the beginning of simulation, the attacker is wiped out, and the malware is hardly spread in the network.

This scene shows that more policemen in the network may contribute to a faster response to the malware. It makes police forces more able to wipe out the attacker.

The four simulations with respect to different parameters disclose the fact that a slight change in the parameters of simulation may lead to an absolutely different scene.

VII. CONCLUSION

In this paper, we propose two types of malware propagation in networks by using standalone CD++ and new version CD++ (lopez) [6].With the powerful new version CD++, we simulate

a virtual battle between hackers or worm writers and network security forces. And several analysis have been made and drawn to conclusions in this paper.

In the first modelling and simulation, we present a static worm and its spread in a network. This simulation takes the level of interaction between cells and possibilities into account. The outcome complies with our expectation, in which the worm will be isolated or eliminated finally.

In the second modelling and simulation, we present a dynamic and complicated malware and its propagation in a network. There are also other features being introduced, such as identity of cell, multiple ports and state variables.

These modelling and simulations can be proved rational and meaningful. Thus, besides, this model can be a framework to simulate epidemics and other types of malware. People can adjust or modify some parameters in this modelling or do some improvements to it.

REFERENCES

[1] A. Mart´ın del Rey, "A Computer Virus Spread Model Based On Cellular Automata of Graphs,"Lecture Notes in Computer Science, vol. 5518, pp. 503-506, 2009

[2] S. Peng, G. Wang, and S. Yu, "Modeling the dynamics of worm propagation using two-dimensional cellular automata in smartphones", ;presented at J. Comput. Syst. Sci., 2013, pp.586-595.

[3] A. Mart´ın del Rey, "A SIR e-Epidemic model for computer worms based on cellular automata,"Lecture Notes in Artificial Intelligence, vol. 8109, pp. 228-238, 2013.

[4] HUANG Guang-qiu1, LIU Xiu-ping. "Simulation of Worm Viruses Spread in Network Based on Cellular ", Computer Engineering,vol.35, pp. 168-169, October 2009

[5] Wainer , G.. 2009. Discrete-event Modeling and Simulation: a Practitioner's Approach. CRC/Taylor & Francis.

[6] Al-Zoubi, K., and G. Wainer. 2009. Using REST Web Services Architecture for Distributed Simulation. In: Proceedings of Principles of Advanced and Distributed Simulation (PADS 2009). pp. 114-121. Lake Placid, New York, USA.

[7] Sixuan Wang, Gabriel A. Wainer, V Subashini Rajus, R Woodbury,"Occupancy Analysis using Building Information Modeling and Cell-DEVS Simulation", Proceedings of 2013 SCS/ACM/IEEE Symposium on Theory of Modeling and Simulation, TMS/DEVS'13, page 26, San Diego, CA - April 201.