FORMAL MODELING AND SIMULATION TO ANALYZE THE DYNAMICS OF MALWARE PROPAGATION IN NETWORKS USING CELL-DEVS

Baha Uddin Kazi, Gabriel Wainer

Dept. of Systems and Computer Engineering
Carleton University, Ottawa, ON, Canada
{bahauddinkazi, gwainer}@sce.carleton.ca

ABSTRACT

Modeling the propagation of malware can help us to understand this phenomenon, and to provide defense strategies for the system. We discuss the use of Cell-DEVS, a formal modeling approach to model malware propagation in different types of networks. We present different models to investigate the propagation of malware in different networks. We first study malware propagation in wireless sensor networks, and we introduce wireless channel access rules. Then, we study malware spread in wired networks. In this model, we introduce an attacker who can generate and update a worm dynamically, and a defender who can develop anti-malware into the network. Finally, we discuss the advantages of the use of formal modeling technique to model different kinds of malware propagation in networks.

Keywords: Malware, Computer networks, Wireless Sensor Networks, DEVS, Cell-DEVS, CD++.

1 INTRODUCTION

Networks and Internet brought many opportunities and challenges to users, enterprises and researchers. In particular, malware and their variants have been a persistent security threat in the networks and the internet. Malware causes large part of the internet temporarily inaccessible, financial loss and social disruptions. For instance, the Kaspersky security bulletin 2015 (KASPERSKY, 2015) mentioned that in 2015 one or more malware attacks blocked 58% of corporate computers. In 2001, the Code Red worm infected at least 359000 hosts in 24 hours and costs an estimated $2.6 billion in damage (Wang, Sheng, Yang, & Wanlei, 2014). According to the Symantec internet security thread report (Symantec, 2016), more than 430 million new unique pieces of malware were in 2015, which is 36% more than the year before. Therefore, industry and researchers have focused on modeling their propagation for investigating the optimized countermeasures.

In recent years, a variety of models and algorithms has been proposed for modeling the malware propagation mechanism. In order to control malware from propagating and to mitigate the impact of an epidemic, we need to have detailed understanding of malware spreading. In this study, we present two models available to study the dynamics of malware propagation using the Cell-DEVS formalism. One of them is a model for wireless sensor networks (WSNs) with medium access control (MAC) rules to observe the dynamics of malware propagation and energy efficiency. Another model is for malware propagation in wired computer networks, which is an extended Susceptible, Infected and Recovered (SIR) model (Rey & Martín, 2013). Moreover, we added attacker and defenders into the network. Attackers generate and update the malware in the network and spread them. The defender can generates anti-malware and annihilate the malware into the network.

The Cell-DEVS formalism is an extension of Discrete Event System (DEVS) specifications (Zeigler, Praehofer, & Kim, 2000) to define cellular models with explicit timing delay. Cell-DEVS improves the execution performance and provides simplicity and reusability of cellular models by using a discrete-

event approach (Wainer G. A., 2009). Furthermore, the use of formal modeling techniques enables automated model verification (Inostrosa-Psijas, Wainer, Gil-Costa, & Marin, 2014). For simulating the models, we use CD++ and distributed CD++ with RESTful Interoperability Simulation Environment (RISE) middleware (Al-Zoubi & Wainer, 2015). The CD++ toolkit is an open source platform for modeling and simulation that implements DEVS and the Cell-DEVS formalism.

The rest of the paper is organized as follows. In section 2, we discuss background and related work. In section 3, we present a model for malware propagation in wireless sensor networks. In this section, we simulate the model using CD++ toolkit and simulation results also presented. In section 4, we present another model to study the dynamics of malware spreading in computer networks. For simulating the model, we use distributed CD++ (DCD++) with the RISE middleware. The DCD++ environment (Wainer, Madhoun, & Al-Zoubi, 2008) is an extension of CD++ that enables distributed simulation of DEVS and Cell-DEVS models, and it has been integrated under the RISE middleware.

2 BACKGROUND

Communication networks are one of the most important inventions of humans, which makes it possible that people from all over the world can communicate with each other instantly. However, this environment also brings some security issues for users and their computers; malicious software (malware) is one of them. A malware is a program that self-propagates through the network exploiting its security flaws. It infects the host devices and uses the infected device to spread automatically or through human activities. Due to the complexity of the model under study, we use the Cell-DEVS formalism and the CD++ toolkit to model malware in computer networks and wireless sensor networks.

The Cell-DEVS is an extension of DEVS formalism (Zeigler, Praehofer, & Kim, 2000) that allows cellular modeling with explicit timing delay. A Cell-DEVS model is represented as a lattice of cells where each cell is a DEVS atomic model and cell space is a couple model. An atomic model represents a part of the system that describes the behavior of the system. A couple model is composed of several atomic models or coupled sub models. Each cell is connected to the local neighboring cells through ports. Figure 1 shows the basic idea of Cell-DEVS model.

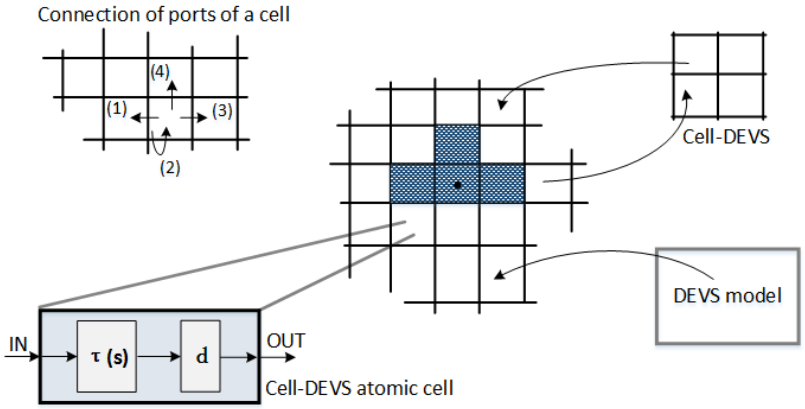

Figure 1: Cell-DEVS components.

A Cell-DEVS atomic model is defined formally as (Wainer G. A., 2009):

$$TDC = <X, Y, N, delay, d, \delta_{int}, \delta_{exit}, \tau, \lambda, D>$$

Where X is the set of external input events and Y is the set of external output events. N is a set of inputs, *delay* is the transport or inertial delay and d is the transport delay for the cell. δ_{int} is the internal transition

function and δ_{exit} is the external transition function. τ is the local computing function, λ is the output function, and *D* is the lifetime function of the state.

A Cell-DEVS coupled model is formally defined as:

$$GCC = <Xlist, Ylist, X, Y, n, \{t_1 \ldots t_n\}, N, C, B, Z>$$

Where Xlist is the input coupling list and Ylist is the output coupling list. X and Y is the set of external input and output events respectively. n is the dimension of the cell space and $\{t_1 \ldots t_n\}$ is the number of cells in each of the dimensions. N is the neighborhood set, C is the cell space, B is the border cells set and Z is the translation function.

CD++ and distributed CD++ (Wainer G. A., 2009) are open source toolkits that provide a development environment for implementing DEVS and Cell-DEVS theory. DEVS atomic models can be developed using state based approach programed in C++. Coupled models and Cell-DEVS models are defined using a built-in specification language based on the formal specifications of Cell-DEVS. The model specification includes the definition of the size and dimension of the lattice, borders and neighborhood. The local computing function of a cell is defined using a set of rules. The form of a rule is: POSTCONDITION ASSIGNMENTS DELAY {PRECONDITION} (Wainer G. A., 2009). The state of the cell will change to the designated POSTCONDITION when the PRECONDITION is satisfied. The values will be transmitted to other components through the ports after the DELAY. If the precondition is false, the next rule in the list is evaluated until a rule is satisfied or there are no more rules. The ASSIGNMENTS section can be used to modify the state variables of the model. The CD++ toolkit interprets this specification language and executes a simulation of the model.

In (Rey & Martín, 2013) authors proposed a model for malware propagation named SIR e-Epidemic based on cellular automata. This is a three states model: Susceptible, Infected and recovered. In (Karyotis & Symeon, 2014) the authors study the modeling of malware propagation for wireless distributed networks. They focus on the users that can dynamically join and leave the network because of the effects of malware. The authors in (Peng, Guojun, & Yu., 2013) present a model for worm propagation in smart phone using two-dimensional cellular automata based on epidemic theory. In (Ye, John, & Deborah, 2004), the authors propose a medium access control protocol for wireless sensor networks named S-MAC. S-MAC introduces techniques to reduce energy consumptions and support self-organization. The authors in (Yadav, Shirshu, & Malaviya, 2008) proposed an optimized medium access protocol focusing on energy efficiency for WSNs. In (Qela, Gabriel, & Hussein, 2009) authors introduce a model to study of large-scale wireless sensor networks using Cell-DEVS.

In this study, we have used the advantages of Cell-DEVS as we mentioned earlier to develop two advanced models for malware propagation in wireless sensor networks and computer networks. We proposed an advanced model to study the malware propagation in WSN by introducing the medium access control (MAC) rules. The details of the model are discussed in section 3. In addition, we also study the malware spreads in computer networks by extending the SIR model as stated before. In our model, we introduce five different states with anti-malware to make it closer to reality.

3 MODELING MALWARE PROPAGATION IN WIRELESS SENSOR NETWORKS

Wireless Sensor Networks (WSNs) are collections of nodes that can sense a variety of physical phenomena, partially process the raw data and communicate each other wirelessly (Stankovic, 2008). One of the main features of WSNs is its ability to collect information from the real world and communicating that information to more powerful devices that can process it and use accordingly. In recent years, WSNs have received tremendous attention in the research and industry that has potential applications in health care, transportation, energy management, industrial process monitoring, home automation, environmental monitoring, etc.

As the applications of WSNs increase gradually, sensor nodes become more vulnerable to security attacks like malware because of its broadcast nature, resource scarcity and unattended environment. A number of actions of a node such as computation activity in the MCU (Microcontroller Unit), data transmission, data reception consume the energy of a node. Among these, the most energy consuming activity is data transmission (Van Dam & Koen, 2003). Therefore, when the WSNs spread malware from one node to another node repeatedly, the energy of the nodes is exhausted and more and more nodes become dead.

In this section, we present a model of malware propagation in WSNs using Cell-DEVS. In (Song & Jiang, 2008), the authors analyzed the process of malware propagation in WSNs using cellular automata and our model is based on this work. We extend the work by introducing the medium access control (MAC) rules (Ye, John, & Deborah, 2004; Yadav, Shirshu, & Malaviya, 2008). The sensors are randomly deployed on a rectangular two-dimensional lattice composed of M×N cells. The entire coverage area of the network is considered as cell space and the coverage area of a particular node is considered as a cell. Each cell may or may not be occupied by sensor node. Each node is considered into one of the following four states (Song & Jiang, 2008):

- Susceptible (*S*): The nodes in this state have not been infected by malware yet but are vulnerable to become infected.
- Infected (*I*): In this state nodes have been infected by malware and may spread the malware to its neighbors.
- Recovered (*R*): The nodes in this state used to be infected by malware and recovered.
- Dead (*D*): In this state sensors have drained out of power that decreased during wireless communication.

Energy consumption of a node depends on a number of actions and states of the node. Initially all the nodes in the lattice are considered *susceptible,* with fixed power. In this state, the node consumes energy at the lowest rate. A node consumes energy at the highest rate in the *infected* state because it broadcasts the malware to other nodes. We considered the basic characteristics of the media access protocol to guarantee channel access fairness and minimize collision. The use of the MAC protocol also improves the battery power use in WSNs. So, broadcasting also depends on the availability of the channel. A node completely drained out of power is moved to the *dead* state. Cells in the grid without any sensor node are considered *dead*. Figure 2 shows the transition of states of a node based on the different probabilities. In the transition diagram P_1, P_2 and P_3 the probabilities to move into *infected, recovered* and *dead* state of a sensor node respectively. The details of the probability parameters will be discussed in later.

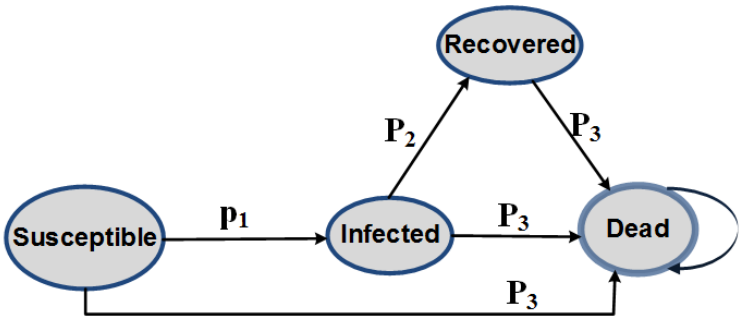

Figure 2: State transition of sensor nodes.

We built a Cell-DEVS model consisting of a 3D lattice (10*10*3 cells). We considered three planes: the first plane describes the different sensor nodes deployed and their transitions. The second plane represents the power status of the corresponding nodes in the first plane throughout the simulation. Finally, the third

plane shows the media access of the nodes. In this model, there are nine neighboring cells with the origin cell. The states of a node discussed earlier are denoted in Cell-DEVS as below:

- -1: The node or the cell is in *dead* or unoccupied within the coverage are of the network or within the cell space.
- 0: The node is in susceptible state within the coverage are of the network.
- 1: The node is in *infected* state.
- 2: The node is in *recovered* state.

The state of access plane is denoted by 30, 31 and 32. Where 30 represents the channel is free, 31 represents receiving the data and 32 represents it broadcasting data. Due to access rules, the node has to be waiting for channel to be free and to start broadcasting.

Each plane uses its own rules. Infected nodes in the sensor network try to diffuse the malware at each time step to their neighbors. All the nodes in the susceptible state become infected with probability P_1. From the infected state, a node can goes to recovered state by running a patch with a probability P_2. Sensor nodes transit to dead state with a rate of P_3 based on the restrained power of the sensors and the consumptions during communications among the nodes.

We use CD++ to simulate the model. To observe the malware propagation, three types of information (state transition, availability of channel and energy reduction) play a role in our model. The following are some sample rules we used.

```
rule : 32 {round(uniform(1,10))*100} { (0,0,-2)=1 and statecount(32) <1}
```

The above rule is for accessing the medium of a malware. If the node is infected that is the state value of the malware is 1 and the medium is free, start broadcasting.

```
rule : {(0,0,0)*1.0} 1000 {(0,0,-1)=0 or (0,0,-1)=2 }
```

The above rule shows the power consumption rate of *susceptible* and *recovered* nodes.

```
rule : 1 1000 { ((0,0,0)=0 and (0,0,2)=31)}
```

If the susceptible node received malware message it moves to the *infected* state that is state value change to 1.

We will discuss a few examples on the execution of this model showing how the malware propagates within the network and it affects the energy consumption of a sensor node. In our first scenario, we executed malware propagation for a 10×10×3 network. Initially all the sensors are considered with full power and the channels are considered free. We assume that one sensor is infected with malware and others are in the *susceptible* state. Figure 3 shows the initial state (time 0) of the network with three planes. It can be seen that in the malware plane, all the nodes are in susceptible state (value 0) except the node (0, 0, 0). We have initialized this node as in infected state (value 1), to observe the malware propagation behavior. We have set the initial battery power level of every node to its maximum value, i.e. 20. The access plane is initialized to show that the wireless media is free (state value 30)

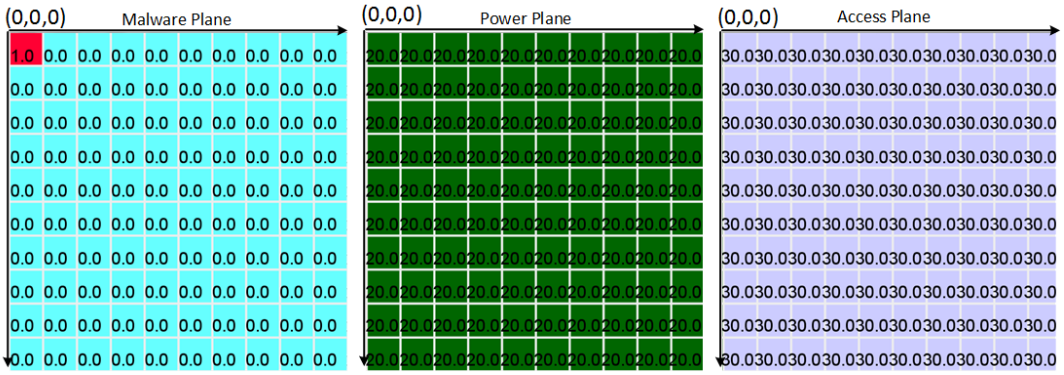

Figure 3: Initial values (00:00:00:000).

According to the rules, the infected node will start broadcasting after a random countdown. The output depicted in Figure 4 is observed in simulation time 600 and it can be noticed that node (0, 0, 0) started broadcasting from the value 32 appeared in access plane, which represent malware broadcast. In the next time step, node (0, 0, 0) has moved to the *recovered* state while its battery power has been reduced to 18 as shown in the Figure 4. In the meantime, three neighbors of the infected cell have received the malware message and changed to the *infection* spreading state as shown in figure 5. In the next time step, three infected neighboring nodes of (0, 0, 0) will broadcast, following media access rules, which are defined to minimize collisions by not allowing neighbors broadcast simultaneously.

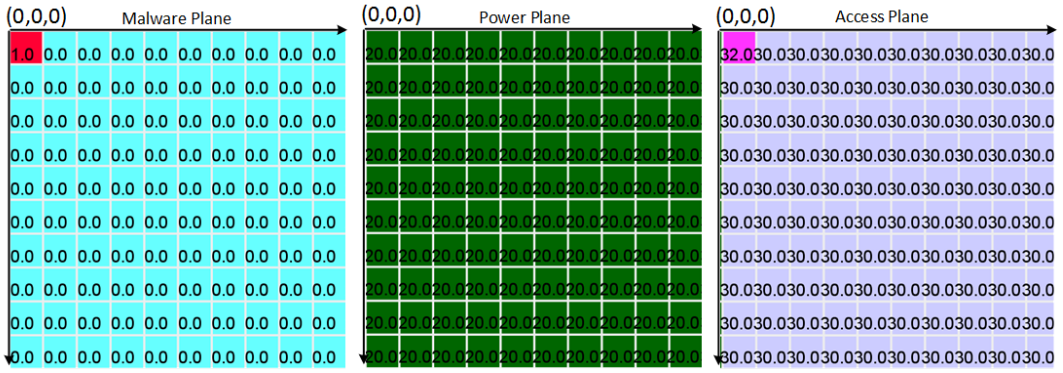

Figure 4: Start broadcasting.

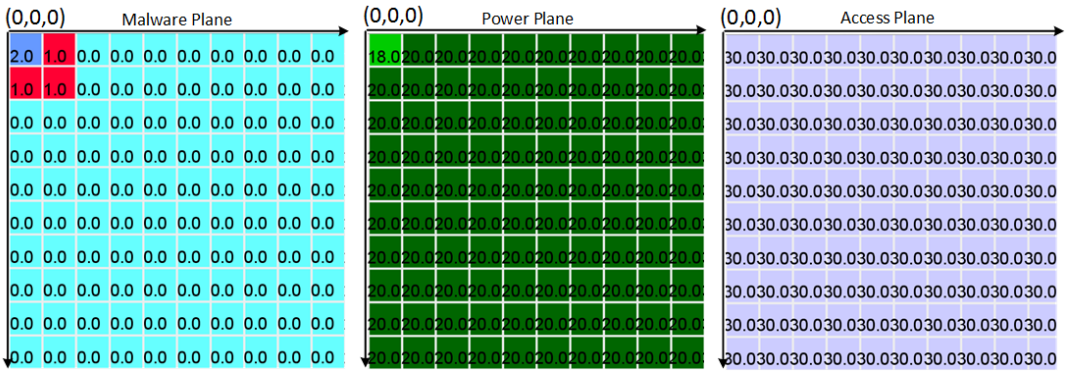

Figure 5: Malware propagated to neighbors.

An intermediary state taken at simulation time 00:02:08:000 is shown in Figure 6. In this figure, we can see the behavior of self-propagating malware.

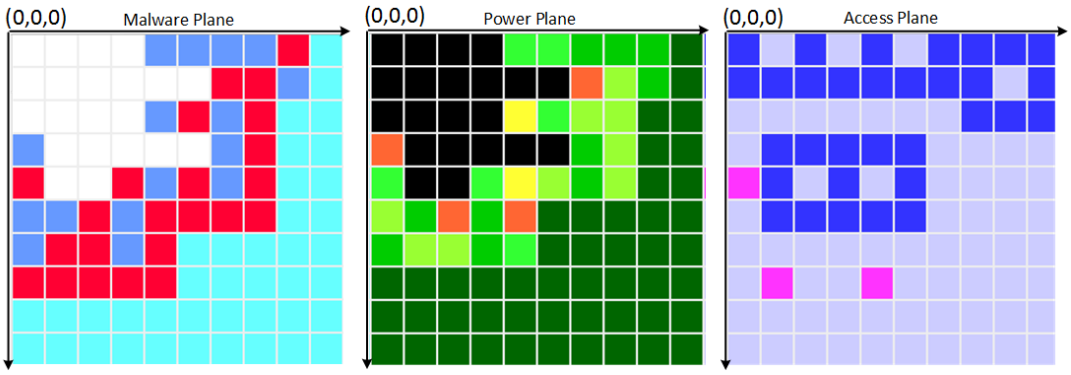

Figure 6: Intermediary output of the simulation (00:02:08:000).

From the access plane, we can see the broadcast of message initiation and broadcast receiving. From the power plane, we can see that malware-infected nodes consume more battery power for broadcasting and hence resulting dead nodes due to faster battery drain-out as shown in black cells. The malware plane also shows the propagation of malware and the dead cell. The dead cells as represented in white. This simulation process continue until the end of experimental time interval or all sensor nodes move to dead state due to complete battery drain-out.

4 MODELING MALWARE IN COMPUTER NETWORKS

In (Rey & Martín, 2013), the authors developed and proposed a model on malware spreading named SIR (susceptible, infected and recovered). We proposed an extended version of this model. In our model, we have five states and three different types of cell or computers: Civilian, Attacker and Defender. The attacker generates and updates the malware in the network at random time interval. The defender can generate anti-malware and wipe out the malware in the network. The civilian is an average computer that can be infected by malware and get immune by anti-malware. Moreover, each computer is considered into one of the following five states:

- Susceptible (S): Computers have not been infected yet but are potential to be infected in the network.
- Infected (I): Computers have been infected by malware in the network.
- Immune (Im): Computes in the network have been loaded with anti-malware.
- Isolated (Is): Computers have been isolated itself from the network.
- Eliminated (E): Malware wiped out from the computers and loaded with anti-malware.

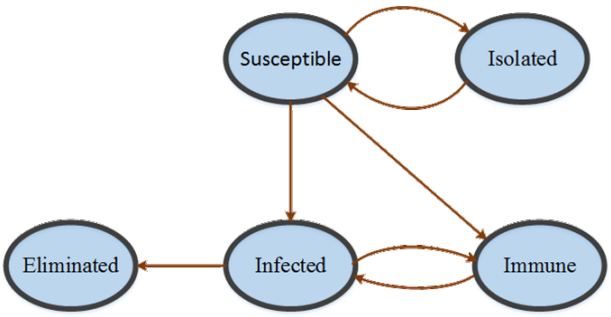

Figure 7: State transition diagram of a computer.

We built a Cell-DEVS model consisting of a 2D lattice (30*30 cells). To distinguish different types of cells as we discussed before a state variable "$identity" is used. $identity= {0,100,150, 200} where, an average cell has an identity value of 0 and moved to the isolate state from the network by changing its identity value to 150. The identity value 100 is for attacker and 200 for defender. The sates of a cell discussed earlier are denoted in Cell-DEVS as below:

- 0 or 1: The computer or cell is in the *susceptible* state within the network or within the cell space.
- -1: The cell is in *infected* state within the network or within the cell space.
- 2: The cell is in *immune* state within the network.
- 3: The cell is in *isolated* state within the network.
- 9: The cell is in *eliminated* state within the network or within the cell space.

The attacker generates and updates malware in random intervals. The aim of attacker is to spread the generated malware to other computers who have not antivirus software or just have an obsolete version of antivirus software that cannot check the malwares out and kill them. The following are some sample rules we used. Some sample rules for generating and updating malwares are 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 }
```

The above rules are for generating and updating versions of malware into the network. As we 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. This process of malware spread 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* state. In the second stage, a new version of 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.

```
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 above rule is for the spread of malware into the network. In this rule of spread, we take a factor of possibility into account. Cells are not infected unconditionally. Instead, it has a possibility to be infected. Here we use two macros "virusspreading" to find out the highest version number among the 8 neighbors of the cell and "vPossibility_Of_Spreading" to define the probability of malware spreading.

In this model and simulation of a network, we also put several defenders in the cells. Defenders monitor their neighbors; produce anti-malware software against the current version of the malware. To implement the function of producing and updating anti-malware software we use ports and state variables. First, a defender should output the current version number of anti-malware to its neighbor. That is required in the spread of anti-malware software. Thus, a new port called "~anvirusvers" is introduced to implement this feature. Also, a state variable "$anvirusvers" is used to record the current version number of anti-malware software. The state variable "$identity" is used again to indicate the defender cells. The following rule is used to produce and updating new version of antimalware software into the network.

```
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 defender encounters a new version of 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 anti-malware software can wipe out the malware. The rule for spreading of antimalware 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 in the above rule 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.

We run a number of simulations in different parameter settings. We have simulated the model in distributed CD++ using RESTfull Interoperability Simulation Environment (RISE) middleware (Al-Zoubi & Wainer, 2015). In this scene, the attacker in the network updates his malware frequently. This makes defender in the network has to update his anti-malware frequently and has little chance to wipe out the attacker finally.

In this scenario, the attacker in the network updates his malware frequently. 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. Figure 8(a) shows the defender detects there is a malware in his neighbors, and produces an anti-malware to fight against the malware. After 2100ms as shown in figure 8(b) 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. Figure 8(c) is the final scene of this simulation. The malware still controls most cells in the network. The defender 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 defender last for a long time and even there is no victory for defender.

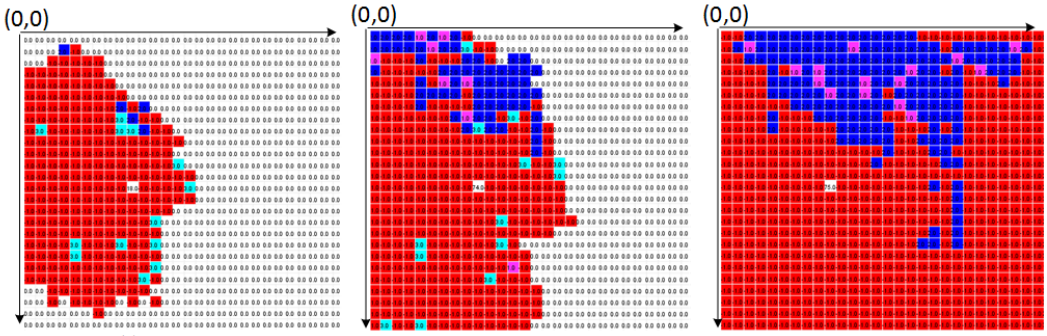

Figure 8: Simulation results for crazy attacker.

In this scenario, the attacker in the network updates his malware infrequently. This makes defender 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. At the 1500ms as shown in figure 9(b) from the beginning of simulation, the anti-malware successfully wipes out the attacker before it updates a new version of malware. In figure 8(c), at the 3300ms from the beginning of simulation, the anti-malware successfully secures or protects all the cells in the network from malware. In this case the malware is eliminated completely. This simulation shows the fact that if attacker updates its malware infrequently, the defender may have chance to wipe out the attacker and eliminate malware at all in the network.

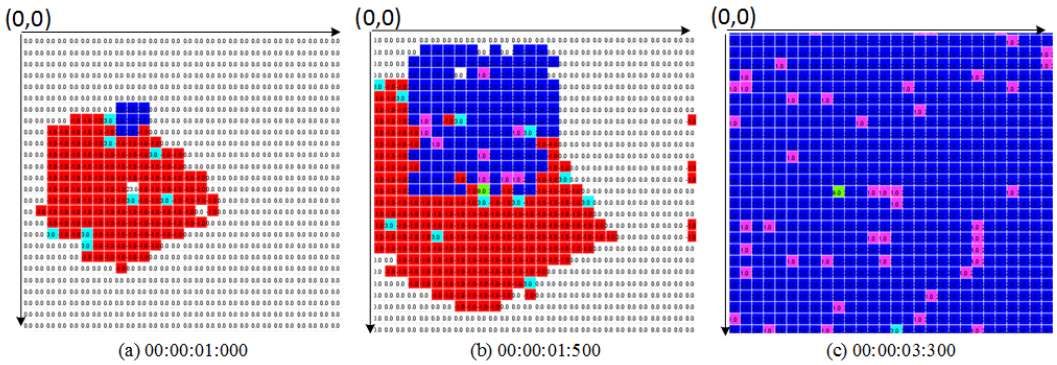

Figure 9: Simulation results for lazy attackers.

5 CONCLUSION AND FUTURE WORK

We provide a brief discussion of the use of the Cell-DEVS formalisms for modeling and simulation of malware in networks. In this paper, two models are presented to study the dynamics of malware propagation. In the first model, we used cell-DEVS to study the malware propagation in wireless sensor networks. We introduce the channel access rules to see how it influences the spreads of malware as well as the power depletion of a sensor node. Some simulations are performed using CD++ toolkit. We present simulation results graphically. In the second model, we study the malware propagation in wired network. In this model, we introduce attacker, who can dynamically generate and update malware to spread in the networks and defender who can produce anti-malware. Simulations are performed using distributed CD++ toolkit in RISE middleware. We also present simulation results graphically. These models showed how the Cell-DEVS formalisms could be used to model these kinds of problems in the networking area. We could reduce the complexity of the model by applying some level of simplification, and we implemented the models using CD++ and DCD++ toolkit. The modular and hierarchal nature of the Cell-DEVS formalism gives us the opportunity to reuse the model and improve it by adding new features easily according to the complexity of the system.

REFERENCES

Al-Zoubi, K., & Wainer, G. (2015). Distributed simulation of DEVS and Cell-DEVS models using the RISE middleware. *Simulation Modelling Practice and Theory, 55*, 27-45.

Inostrosa-Psijas, A., Wainer, G., Gil-Costa, V., & Marin, M. (2014). DEVS Modeling of Large Scale Web Search Engines. *Proceedings of the 2014 Winter Simulation Conference.* Savannah, GA.

Karyotis, V., & Symeon, P. (2014). Evaluation of Malware Spreading in Wireless Multihop Networks with Churn. *Evaluation of Malware Spreading in Wireless Multihop Networks with Churn." In International Conference on Ad Hoc Networks* (pp. 63-74). Springer International Publishing.

KASPERSKY. (2015). *Kaspersky Security Bulletin 2015.* Retrieved from https://securelist.com/files/2015/12/Kaspersky-Security-Bulletin-2015_FINAL_EN.pdf

Peng, S., Guojun, W., & Yu., S. (2013). Modeling the dynamics of worm propagation using two-dimensional cellular automata in smartphones. *Journal of Computer and System Sciences* , 586-595.

Qela, B., Gabriel, W., & Hussein, M. (2009). Simulation of large wireless sensor networks using Cell-DEVS. *In Proceedings of the 2009 Winter Simulation Conference (WSC)* (pp. 3189-3200). Austin, TX, USA: IEEE.

Rey, d., & Martín, Á. (2013). A SIR e-Epidemic model for computer worms based on cellular automata. *In Conference of the Spanish Association for Artificial Intelligence* (pp. 228-238). Springer Berlin Heidelberg.

Song, Y., & Jiang, G.-P. (2008). Modeling malware propagation in wireless sensor networks using cellular automata. *International Conference on Neural Networks and Signal Processing* (pp. 623-627). Zhenjiang, China: IEEE.

Stankovic, J. A. (2008). Wireless Sensor Networks. *IEEE Computer 41*, 92-95.

Symantec. (2016). *Internet Security Threat ReportInternet Report.* Retrieved from https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

Van Dam, T., & Koen, L. (2003). An adaptive energy-efficient MAC protocol for wireless sensor networks. *In Proceedings of the 1st international conference on Embedded networked sensor systems* (pp. 171-180). Los Angeles, California, USA: ACM.

Wainer, G. A. (2009). *Discrete Event Modeling and Simulation A Practitioner's approach.* Boca Raton, FL: CRC Press, Taylor & Francis Group.

Wainer, G. A., Madhoun, R., & Al-Zoubi, K. (2008). Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services. *Simulation Modelling Practice and Theory, 16*, 1266–1292.

Wainer, G. A., Madhoun, R., & Al-Zoubi, K. (2008). Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services. *Wainer, Gabriel A., Rami Madhoun, and Khaldoon Al-Zoubi. "Distributed simulation of DEVS and Cell-DEVSSimulation Modelling Practice and Theory*, 1266-1292.

Wang, Y., Sheng, W., Yang, X., & Wanlei, Z. (2014). Modeling the propagation of worms in networks: A survey. *IEEE Communications Surveys & Tutorials*, 942-960.

Yadav, R., Shirshu, V., & Malaviya, N. (2008). Optimized medium access control for wireless sensor network. *IJCSNS International Journal of Computer Science and Network Security*, 334-338.

Ye, W., John, H., & Deborah, E. (2004). Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on networking*, 493-506.

Zeigler, B., Praehofer, H., & Kim, T. (2000). *Theory of modeling and simulation.* San Diego, CA: Academic Press.

AUTHOR BIOGRAPHIES

BAHA UDDIN KAZI is a Ph.D. candidate in the department of Systems and Computer Engineering at Carleton University. He received his MASc in Computer Networks from Ryerson University, Canada. He is serving as a member of technical committee of several international conferences. His research interests focus on performance engineering, modeling and simulation, 5G cellular networks and network protocols. His email address is bahauddinkazi@sce.carleton.ca.

GABRIEL A. WAINER, FSCS, SMIEEE, is Professor and Associate Chair for Graduate Studies, Department of Systems and Computer Engineering (Carleton University). He held visiting positions at the University of Arizona; LSIS (CNRS), Université Paul Cézanne, University of Nice, INRIA Sophia-Antipolis, Université Bordeaux (France); UCM, UPC (Spain), and others. He is the author of three books and over 330 research articles; he helped organizing over 100 conferences, being one of the founders of the Symposium on Theory of Modeling and Simulation, SIMUTools and SimAUD. Prof. Wainer is Special Issues Editor of SIMULATION, member of the Editorial Board of IEEE/AIP CISE, Wireless Networks (Elsevier), and others. He received the IBM Eclipse Innovation Award, the First Bernard P. Zeigler DEVS Modeling and Simulation Award, the SCS Outstanding Professional Award (2011), the SCS Distinguished Professional Award (2013), the SCS Distinguished Service Award (2015) and various Best Paper awards. He is a Fellow of SCS. Email: gwainer@sce.carleton.ca