Effectiveness's of MAC Techniques and Security patches on Restricting Malware Propagation in Wireless Sensor Networks

Term paper prepared by
H. Amarasinghe (6152735), University of Ottawa
Sysc 5104: Methodologies for Discrete Event Modeling and Simulation (fall 2012), Carleton University

***Abstract*—Wireless sensor nodes are highly resource constrained in terms of memory, processing, battery power and communication. Due to these limitations, sophisticated security features are not cost effective in most wireless sensor network (WSN) applications. Hence WSNs are often targeted by viruses and malware, interrupting or completely terminating their desired functionalities. In this work, we investigate malware propagation patterns in WSNs and examine the abilities of different Media Access Control (MAC) rules and techniques to limit malware propagation. Effectiveness of security patching is also analyzed by implementing self-propagating patches. We have modeled and simulated wireless communication in WSN using parallel Cell-DEVS and investigated malware propagation by implementing basic characteristics of two commonly used WSN MAC protocols.**

Keywords— Parallel Cell-DEVS, Simulation, Sensor Networks, Malware Propagation, MAC protocols;

I. INTRODUCTION

Wireless Sensor Networks (WSNs) are regarded as one of the dominant technology trends of this decade that has potential usage in monitoring applications in number of fields including defense, health care, environment science, and chemical industries. Malware and virus attacks on wireless computer networks and wireless telecommunication networks have been widely investigated and hold a strong literature. However, WSNs differ from traditional computer networks and wireless telecommunication networks in various aspects: First, wireless sensor networks are highly distributed system and consist of a great number of distributed nodes (sensor nodes) with the ability to monitor its surroundings.

Second, sensor nodes are limited in power, computational capacities, and memory. Finally, self-organization is a fundamental feature of wireless sensor networks, allowing normal operation with minimal human interaction. Security mechanisms employed on traditional wireless computer networks or wireless telecommunication networks could not be applied directly to wireless sensor networks. Several research works has recently given attention to study malware propagation in WSN and proposed number of techniques to minimize effects and limit propagation. Some of these techniques involve application of signal processing technique to model space-time propagation dynamics of topologically-aware malware in a sensor network with uniformly distributed nodes [1]. Some of these authors make use of epidemic theoretic model for evaluating malware propagation and broadcast protocols in wireless sensor networks. On the other hand, substantial amount of work available in the literature, that uses Cellular Automata to simulate their proposed approaches. Considering the inherent characteristics of WSN, existing models for epidemiology cannot be directly applied to study malware propagation of wireless sensor network [2].

Cellular automata (CA) models complex natural systems, containing large numbers of simple identical components with local interactions. It defines an infinite regular n-dimensional lattice in which each cell can take a finite value. There is a substantial literature [3, 4] on the mathematical model based on cellular automata which simulates behaviors of natural systems. These models based on cellular automata focus on the local characteristics of single component that influence the global behavior of the system.

Cell-DEVS can be identified as an advanced version of CA which uses DEVS formalism to enhance capabilities of conventional CA. It allows interactions with external entities and components, explicit timing delays, when computing individual cell states. Hence events that trigger local cell state computation can be fired by entities that are external to the cellular neighborhood. Also more resent versions have allowed convenient behavioral modeling using multiple states and multiple input and ports that define cells characteristics.

We have focused on the inherent characteristics of WSN and the dynamic process of malware propagation in WSN by modeling it employing Cell-DEVS modeling technique. We have validated the proposed malware propagation models using CD++ [5-7], a simulation engine, which was designed specifically to simulate Cell-DEVS models. Simulation study revealed that MAC mechanisms of wireless sensor networks has the ability greatly slow down the speed of malware propagation and reduces the risk of large-scale malware prevalence in WSN. Moreover, we show that use of self-propagating security patches can significantly reduce adverse effects of malware attacks. The developed WSN model can be used to describe accurately the dynamic behavior of malware propagation on WSN, and can be used for developing robust and efficient defense system on WSN.

In this work we talk about three WSN models. First, we have developed our initial model to simulate in general CD++. Second, we have re-created the model to use multiple ports

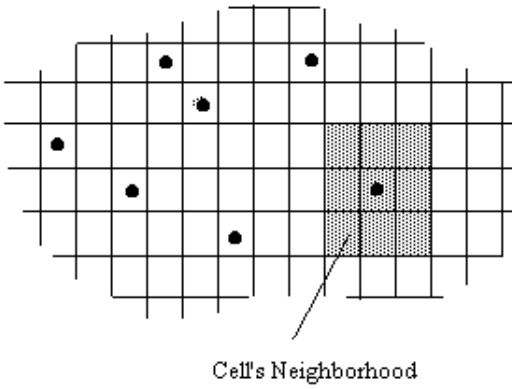


Figure 2 Neighborhood of a cell

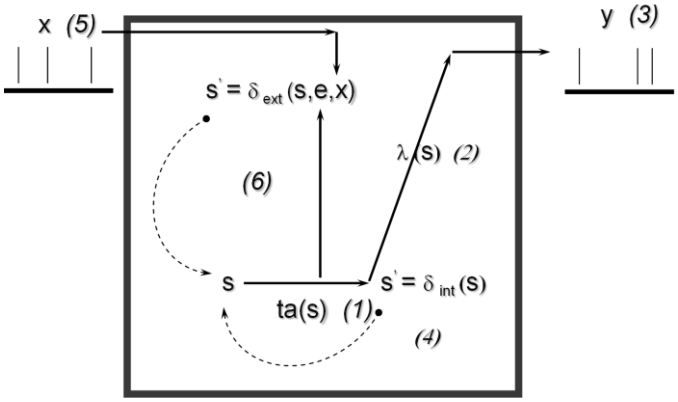


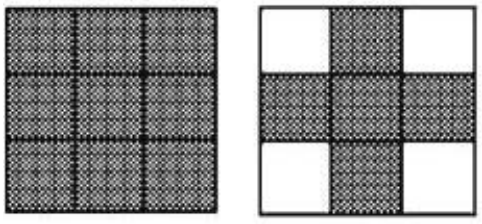
Figure 3 DEVS Atomic Model

and simulate in parallel CD++. Finally, we have re-engineered the simulation model to resemble more realistic WSN, and compare effectiveness of different MAC protocols and self-propagating patching, in malware spread controlling. In order to avoid confusion, we name these three models as initial version, converted version and extended version respectively in following sections.

The rest of this term-paper is organized as follows. Section 2 provides detailed background information, introducing Cell-DEVS approach towards modeling cellular space. Section 3 describes conceptual modeling and Cell-DEVS formalism of initial version and extended version. We do not talk about converted version in this section since its modeling approach is similar to initial model. In section 4, simulation methodology of the initial version is illustrated with brief description of obtained simulation results. The extended version is simulated using multi-port parallel CD++ and this simulation procedure is discussed in section 5. In section 6, performance of initial version is compared with converted version. Moreover, effectiveness of different MAC protocols and self-propagating patching is analyzed using simulation results obtained from extended version. Brief conclusion is given in section 7, discussing potential future research directions.

II. BACKGROUND

This section focuses on providing brief introduction to closely related and employed technologies such as cellular automata, DEVS, Cell-DEVS and several MAC techniques specifically designed for WSN.



(a) ***(b)***

Figure 1 Neighborhood types; (a) Moor's neighborhood (b) Von-Neumann's neighborhood

A. Cellular Automata (CA)

Cellular Automata is a discrete dynamical system formed by a finite number of identical objects called cells which are arranged uniformly in a two-dimensional cellular space. Each cell is endowed with a state (from a finite state set S), that changes at every step of time accordingly to a local transition rule.

In this sense, the state of a particular cell at time *t* depends on the states of a set of cells, called its neighborhood, at the previous time step *t-1*. More precisely, a CA is defined by;

$$CA = < S, n, C, \eta, N, T, \tau >$$

Where *C* is the cell's set of state variables and ***S*** is the finite alphabet to represent each cell's state. η represents the dimensional space while ***N*** defines set of neighboring cells or in other words, cells neighborhood. **T** global transition function and τ is the local computing function of each cell in the cell space.

As shown in Figure 1, all the cells in the cell space have the same neighborhood and state of each cell is computed by using their neighborhood cell states. There are two basic neighborhoods. Von Neumann's neighborhood has the cells to up, down, left and right while Moor's neighborhood has all eight cells around a cell in 2-dimensional cell space as shown in Figure 2.

B. Discrete Event system Specification (DEVS)

DEVS is a system that allows modeling real system experimental frames in to a module hierarchy [6]. As shown in Figure 3, a real system modeled using DEVS can be represented as a set of atomic or coupled sub-models. The atomic DEVS model is defined as:

$$M = < X, S, Y, d_{int}, d_{ext}, l, ta >$$

where *X* is the input events set, *S* is the state set, *Y* is the output events set, dint is the internal transition function, d_{ext} is the external transition function, *l* is the output function, and *ta* is the time advance function.

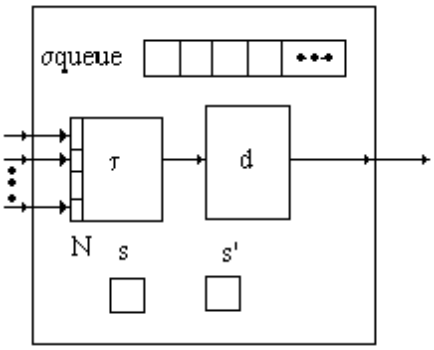


Figure 4 A Cell-DEVS atomic model with transport delay

The atomic model can be considered as the base element in which we define dynamics of any system, while the coupled structural model consists of one or more atomic and/or coupled models. Coupled models are defined as a set of basic components (atomic or coupled).The coupled model can be defined as:

$$CM = < X, Y, D, \{Md \mid d \in D\}, EIC, EOC, IC, select >$$

where X is the input events set, Y is the output events set, D is the set of component names, Md is a DEVS basic model(i.e., atomic or coupled), EIC is the set of external input couplings, EOC is the set of external output couplings, IC is the set of internal coupling, and select is the tie-breaker function. The coupled model explains how to convert the outputs of a model into inputs for the other models, and how to handle inputs/ outputs to and from external models.

C. Cell-DEVS

Cell-DEVS [8] has extended the DEVS formalism, allowing us to implement cellular models with timing delays. Once the behavior of a cell is defined, a coupled Cell-DEVS can be created by interconnecting a number of cells with their neighbors. Each cell is defined as a DEVS atomic model, as shown in Figure 4.

Each cell uses N inputs to compute its next state. These inputs, which are received through the model's interface, activate a local computing function *τ*. A delay *d* can be associated with each cell. A coupled Cell-DEVS model is the resulting array of cells (atomic models) with given dimensions, borders, and zones.

D. WSN MAC protocols

In order to evaluate and compare malware propagation patterns in WSN, we have simulated basic characteristics of following MAC protocols.

Sensor S-MAC [9] a contention based MAC protocol is modification of IEEE 802.11 protocol specially designed for WSN. In this MAC protocol sensor node periodically goes to the fixed listen/sleep cycle. A time frame in S-MAC is divided into two parts: one for a listening session and the other for a sleeping session. Only for a listen period, sensor nodes are able to communicate with other nodes and send some control packets such as SYNC, RTS (Request to Send), CTS (Clear to Send) and ACK (Acknowledgement). By a SYNC packet exchange all neighboring nodes can synchronize together and using RTS/CTS exchange the two nodes can communicate with each other. A lot of energy is still wasted in this protocol during listen period as the sensor will be awake even if there is no reception/transmission.

In the Optimized MAC protocol [10], the sensors duty cycle is changed based on the network load. If the traffic is more than the duty cycle will be more and for low traffic the duty cycle will be less. The network load is identified based on the number of messages in the queue pending at a particular sensor. The control packet overhead is minimized by reducing the number and size of the control packets as compared to those used in the S-MAC protocol. This protocol may be suited for applications in which apart from energy efficiency there is need for low latency.

III. MODELING MALWARE IN WSN

We have performed simulations on two different versions of CD++. Initially few basic characteristics of WSN is modeled and simulated using general CD++ version. After that, model was further enhanced and simulated in more powerful, Parallel CD++ [11] version. Parallel CD++ allows using multiple state variables that define sensor node state, to be defined and handled in the same plane. Moreover it allows defining multiple input and output ports for cells. Use of this extended version of CD++ has significantly reduced the complexity of the implementation. Hence we were able to introduce additional functionalities and fine-tune the model to represent more detailed realistic malware propagation scenarios in WSN.

In this section, we provide detailed information of our WSN modeling approach. Conceptual and Cell-DEVS modeling for the general CD++ is described in first two subsection followed by enhanced modeling for parallel CD++.

A. Conceptual Modeling of the initial design

We have modeled the wireless sensor network as a 2 dimensional lattice, with each cell may or may not occupied by a stationary wireless sensor node as shown in Figure 5.

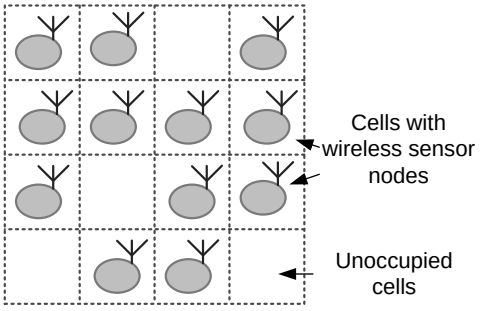


Figure 5 WSN lattice

Each node can have one of the states given in Table 1.

Node States	Value
Dead or unoccupied	-1
Susceptible	0
Infected & spreading node	1
Infected & dormant node	2

Table 1 Sensor node states

Changes of states and assignment of these state values can be represented by state transformation diagram as shown in Figure 6.

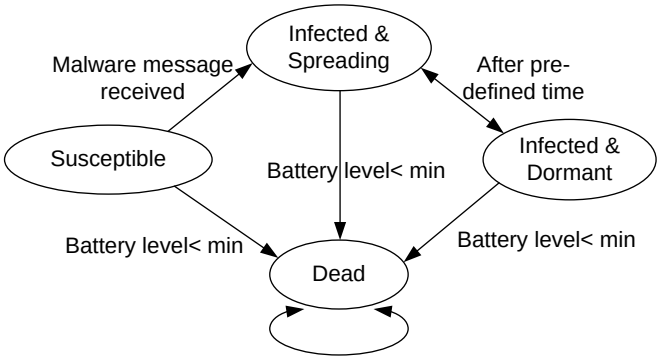


Figure 8 States transformation diagram for malware plane (initial model)

Initially all the nodes in the lattice are in the ***susceptible*** state with fixed battery power. If no malware is received, it assumes sensor nodes perform their normal designed functions using relatively smaller battery power. Hence un-infected node will take long time to go to dead state. However if node receive a malware message, it moves to ***infected & spreading*** state and start broadcasting malware to its neighbors. Broadcasting consumes significant amount of power, draining its battery faster and result in moving to ***dead*** state quickly.

Wireless media access protocols are used in wireless sensor networks to transfer data minimizing collisions and improve battery power usage. In the initial model, we have considered basic characteristics of media access protocol to guarantee channel access fairness and minimize collisions. Table 2 shows media access states of each node used to model wireless data transfer with minimal collisions. Transformation of states and assignment of these state values can be represented by state transformation diagram in Figure 7.

Due to media access (MAC layer) rules, node has to wait for channel to be free, to start broadcasting. Basic characteristic of wireless channel access is modeled by

MAC layer node state	value
Free	30
Receiving	31
Broadcasting	32

Table 2 States in media access plane

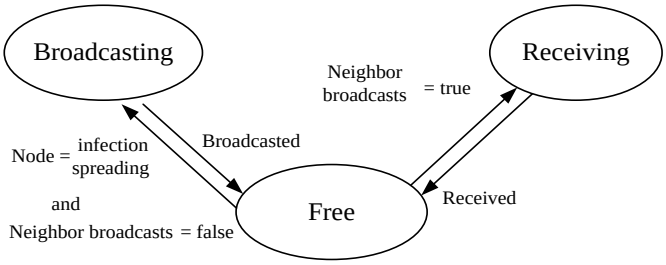


Figure 6 States transformation diagram for access plane (initial model)

introducing ***infected & dormant*** state. Infected & spreading node wait for random delay before next malware broadcast by moving to infected & dormant state after each broadcast.

Finally sensor node battery power is modeled by starting with fixed sized battery power of value 20 and reduced to minimum of 10 in different rates for each node state.

B. Cell-DEVS modeling of the initial design

The general version of CD++ does not facilitate assigning multiple state variables or multiple ports in a single plane. Hence we have modeled the system in Cell-DEVS using three planes to represent three state variables of a sensor node that can exist at a particular time instance. These three state variables are defined, and handled by malware plane, power plane and access plane as shown in Figure 8.

Malware plane is used to handle malware related states variable values of wireless sensor nodes. As we discussed earlier, a cell in the malware plane can be; dead/unoccupied, susceptible, spreading or dormant. Power plane is used to keep power state values of the sensor node. This power state can be any real number between 10 and 20. Initial power value 20 represents battery levels of sensor node is full and value 10 represents completely depleted battery power in the sensor node. Access defines MAC related states and has state variable values; channel free, receiving and broadcasting.

Table 3 provides formal Cell-DEVS specifications for each plane and detailed descriptions of each element are given below.

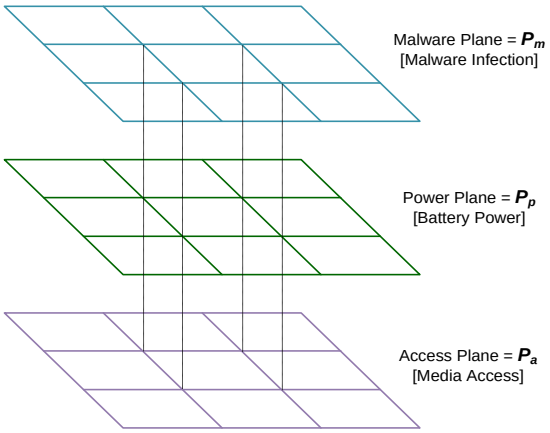


Figure 7 Three planes used to assign three state variables in general CD++

Plane Name	Function	Specification
Malware Plane (P_m)	Model malware infection	<I_m, X_m, Y_m, $Xlist_m$, $Ylist_m$, η_m, N_m, {r_m, c_m}, C_m, B_m, Z_m, $Select_m$>
Power Plane (P_p)	Model battery usage	<I_p, X_p, Y_p, $Xlist_p$, $Ylist_p$, η_p, N_p, {r_p, c_p}, C_p, B_p, Z_p, $Select_p$>
Access Plane (P_a)	Model media access	<I_a, X_a, Y_a, $Xlist_a$, $Ylist_a$, η_a, N_a, {r_a, c_a}, C_a, B_a, Z_a, $Select_a$>

Table 3 Formal Cell-DEVS specifications for three planes

Malware Plane (P_m) Specifications

GCC_{bm} = <I_m, X_m, Y_m, $Xlist_m$, $Ylist_m$, η_m, N_m, {r_m, c_m}, C_m, B_m, Z_m, $Select_m$>

$Xlist_m$= { 0 };

$Ylist_m$= { 0 }.

η_m= 6x6x3;

Z_m: [malware-propagation-rules]

rule : -1 1000 { (0,0,1)<11 }
rule : 1 1000 { ((0,0,0)=0 and (0,0,2)=31)}
rule : 2 1000 { (0,0,2)=32}
rule : 1 5000{ (0,0,0)=2}
rule : {(0,0,0)} 1000 {t}

N_m:

neighbors : sensor(-1,-1,0) sensor(-1,0,0) sensor(-1,1,0)
neighbors : sensor(0,-1,0) sensor(0,0,0) sensor(0,1,0)
neighbors : sensor(1,-1,0) sensor(1,0,0) sensor(1,1,0)
neighbors : sensor(0,0,1)
neighbors : sensor(0,0,2)

B_m= nowrapped;

S_m:

-1 : Dead node or Unoccupied by a sensor
0 : Susceptible node
1 : Infected & spreading
2 : Infected & dormant

d = inertial delay (default 1000)

Power Plane (P_p) Specifications

GCC_{bp}= <I_p, X_p, Y_p, $Xlist_p$, $Ylist_p$, η_p, N_p, {r_p, c_p}, C_p, B_p, Z_p, $Select_p$>

$Xlist_p$= { 0 };

$Ylist_p$= { 0 }.

η_p= 6x6x3;

Z_p: [power-usage-rules]

rule : 10 1000 { (0,0,-1)=-1}
rule : 10 1000 { (0,0,0)<=11 }
rule : {(0,0,0)*.9} 1000 {(0,0,-1)=1}
rule : {(0,0,0)*1.0} 1000 {(0,0,-1)=0 or (0,0,-1)=2 }
rule : {(0,0,0)} 1000 {t}

N_p:

neighbors : sensor(0,0,-1)
neighbors : sensor(0,0,0)

B_p= nowrapped;

S_p: Battery power $\{\beta \in \mathbb{R}^+ | 10 \leq \beta \leq 20\}$

d = inertial delay (default 1000)

Access Plane (P_a) Specifications

GCC_{bp}= <I_a, X_a, Y_a, $Xlist_a$, $Ylist_a$, η_a, N_a, {r_a, c_a}, C_a, B_a, Z_a, $Select_a$>

$Xlist_a$= { 0 };

$Ylist_a$= { 0 }.

η_a= 6x6x3;

Z_a: [media-access-rules]

rule : 31 1000 { (0,0,0)=30 and statecount(32)>0}
rule : 30 1000 { (0,0,0)=31 and statecount(32)<1}
rule : 32 {round(uniform(1,10))*100} { (0,0,-2)=1 and statecount(32)<1}
rule : 30 1000 { (0,0,0)=32}
rule : {(0,0,0)} 1000 {t}

N_a:

neighbors : sensor(-1,-1,0) sensor(-1,0,0) sensor(-1,1,0)
neighbors : sensor(0,-1,0) sensor(0,0,0) sensor(0,1,0)
neighbors : sensor(1,-1,0) sensor(1,0,0) sensor(1,1,0)
neighbors : sensor(0,0,-1)
neighbors : sensor(0,0,-2)

B_a= nowrapped;

S_a:

30 :channel free
31 :receiving
32 :broadcasting

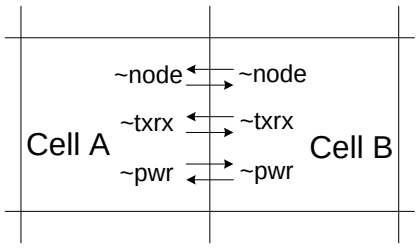


Figure 9 Multiple input and output ports between cells

d = inertial delay (default 1000)

C. Extensions for the initial design

Initial model was further enhanced by defining more states and transition rules to resemble more realistic simulation environment, exploiting additional functionalities introduced by parallel CD++. For the first phase, we have converted model designed for general CD++ directly to parallel CD++, in order to analyze and compare simulation performance. We have preserved most of the model characteristics and simulation rules are generated using multiple input and output

port	value	Represented state
node	-1	Dead or unoccupied node
	0	Susceptible sleeping node
	1	Susceptible active node
	2	Infection spreading node
	3	Infected but dormant node
	4	Patched active node
	5	Patched sleeping node
pwd	25	Battery power vary between maximum value 25 and minimum value 0
	:	
	0	
txrx	0	No channel access
	1	Normal ad-hoc messages
	2	Malware broadcast
	3	Patch messages

Table 4 States represented by each port value

ports. We discuss these implementation details and compare performance of these two models using general and parallel CD++ versions in following sections.

After converting the initial multiple plane design to multiple ports design, we further enhanced it by adding new functionalities. These new modifications can be highlighted as;

- Cells were modified to compute their states by receiving neighbor inputs from three input ports
- Probabilistic approach was followed to model shared wireless media access
- Characteristics of S-MAC [9] protocol and Optimized MAC protocol [10] is modeled introducing fixed and variable sleep states.
- Effectiveness security patching was analyzed by introducing self-propagating patching.

Modification to access multiple input ports has reduced complexity of the initial multiple plane model. We have used three ports, namely; node, txrx and pwd to interface node state changes, data transmission and channel access state changes and power changes respectively. Figure 9 illustrates this multi-port communication and state values handled by these ports are shown in Table 4.

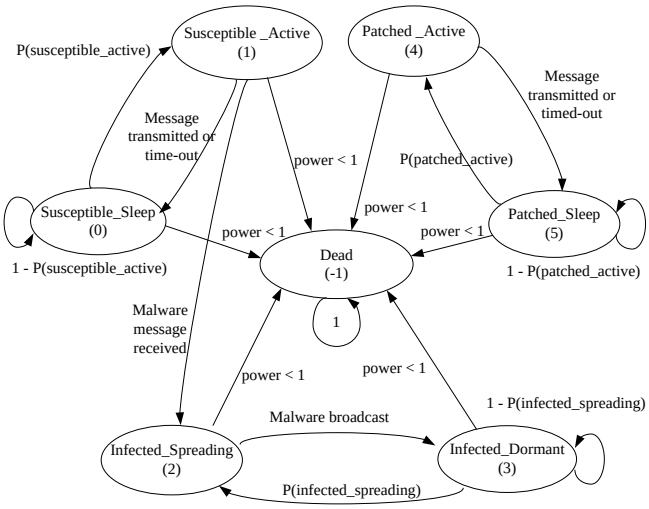


Figure 10 State changes represented by port: node

Each port name represents input and output ports with the same name. Changes of the state of a cell is delivered to output port named *node* and cells neighbors are informed about this state change through their input port named *node*. Relationships between states represented by port *node* are

State/Event	Power consumption per 100ms
Dead	0
Susceptible sleeping	0.1
Susceptible active	0.2
Infection spreading	0.5
Infected but dormant	0.2
Patched active	0.2
Patched sleeping	0.1
Broadcast (event)	1.0
Ad-hoc message (event)	0.25

Table 5 power consumption at each state and after channel access events

presented in the state diagram in Figure 10.

We modeled battery power consumption at each node state and after broadcast and normal ad-hoc message transfer event using the power usage scheme shown in Table 5.

Basic characteristics of Optimized MAC protocol and S-MAC protocols were used to model media access in WSN. Optimized mac protocol increase duty cycle based on total load in the sensor network. Since we have followed a probabilistic approach, this characteristic was modeled by maintaining inversely proportional relationship between ad-hoc message generations and sleeping probabilities. On the other hand, S-MAC protocol uses fixed duty cycles regardless of the load on the sensor network. Hence we have modeled this S-MAC characteristic by keeping fixed sleeping probability and observing malware propagation patterns for different operation message generation probabilities.

Beside these characteristics, simple set of rules were defined to provide shared access to wireless media, minimizing collisions. Implementation of collision avoidance algorithm such as CSMA is out of the focus of this work. Hence we simply make nodes wait if it detect channel is used by its neighbors and only transfer data if none of the neighbors use the channel (i.e. channel is free). However this simple approach does not guarantee collision free data transfer. Therefore if more than one neighbors of a particular cell access the channel at the same time, node assumes a possible collision and simply ignores the message.

IV. Simulation of the Initial Model

In this section, we discuss simulation approach followed to analyze basic malware propagation patterns illustrating rules used in the initial model. As we discussed before, for the simulation of the initial model, we have implemented three planes, each has 20X20 lattice containing wireless sensor nodes. We have used inertial delay with default delay time 1000ms. Boarders were not-wrapped and no special rule was defined for boarders since boarder cell behavior is similar to internal cells. Brief descriptions of the CD++ rules are as follows;

[media-access-rules]

rule : 31 1000 { (0,0,0)=30 and statecount(32)>0}

If any of the neighbors are broadcasting=32, set state to receive=31

rule : 30 1000{ (0,0,0)=31 and statecount(32)<1}

If the the current state is receive and none of the neighbors are broadcasting, set state to free=30

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

If the node is infected=1 and media is free=30, start broadcasting=32

rule : 30 1000{ (0,0,0)=32}

After broadcasting, set media free=30

rule : {(0,0,0)} 1000{t}

Default rule

[power-usage-rules]

rule : 10 1000 { (0,0,-1)=-1}

Set power value of all non-occupied cells to 10

rule : 10 1000 { (0,0,0)<=11 }

If the power value is less than 11, make it 10

rule : {(0,0,0).9} 1000 {(0,0,-1)=1}*

Power consumption rate of infected cells

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

Power consumption rate of susceptible and recovered nodes

rule : 10 1000 {t}

Default rule

[malware-propagation-rules]

rule : -1 1000 { (0,0,1)<11 }

If battery power is less than 11, node will die

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

If susceptible node received malware message, go infected_spread=1

rule : 2 1000 { (0,0,2)=32}

After doing a broadcast, go infected_dormant=2

rule : 1 5000{ (0,0,0)=2}

After staying in infected_dormant=2, go back to infected_spread=1

rule : {(0,0,0)} 1000 {t}

Default rule

V. Simulation of the Extended Model

In this section, we provide implementation details and insight to the simulation rules defined for the extended model simulated in parallel CD++ lopez version [11]. For the simulation of the extended model, we have used three ports in single plane as we discussed in section III. A two dimensional 20X20 lattice containing wireless sensor nodes was used with transport delay and default delay time 100ms. Similar to the initial model, boarders were not-wrapped and no special rule was defined for boarders since boarder cell behavior is similar to internal cells. An insight towards the implemented rules in the .ma file is given below.

Initializing port values in the parallel CD++ can be done in several ways. In this work we have employed run-time port initialization approach. .val file initialize all the variables in a particular cell by a given single value. In our model, we were required to put malware and patch seeds in the cell space and

three ports (i.e. ~node, ~pwr and ~txrx) to be initialized by different values. Hence, we have used .val file to pre-initialize with values 99, 100 and 101 respectively for susceptible, infected and patched nodes and have defined following rules for actual initialization.

1. *rule : { ~node := 1; ~pwr := 20; ~txrx := 0; } 100 { (0,0)~node = 99 and ((0,0)~pwr = 99 and round(uniform(1,10)) > 5) }*
2. *rule : { ~node := 0; ~pwr := 20; ~txrx := 0; } 100 { (0,0)~node = 99 and (0,0)~pwr = 99}*
3. *rule : { ~node := 2;~pwr := 20; ~txrx := 2; } 100 { (0,0)~node = 100 and (0,0)~pwr = 100 }*
4. *rule : { ~node := 4; ~pwr := 20; ~txrx := 3; } 100 { (0,0)~node = 101 and (0,0)~pwr = 101 }*

Rule 1 and 2 initialize susceptible sensor nodes active in 0.5 probabilities, keeping rest sleeping. Rule 3 initializes infected nodes as infected-spreading (2) and rule 4 initialize patched node as patched-active (4). Battery powers of all nodes are initialized as 20 power units. Although these four rules are discussed before others, to improve execution time, we have placed them at the end of the .ma file, just before the default (always true) rule assuming sequential execution of rules. Thus we can make sure these initialization rules do not affect the performance of the simulation significantly.

Rules that define dead (-1) state appears at the top of the rule sequence.

5. *rule : { } 100 { (0,0)~node = -1 }*
6. *rule : { ~node := -1; ~pwr := 0; ~txrx := 0; } 100 { (0,0)~pwr <= 1 }*

Rule 5 keeps all the dead cells as it is and rule 6 sends any node with battery power less than or equal to 1, to dead state after making its remaining battery power equal to zero and making channel access equal to none (0). Except the dead state, all other states and events such broadcasting and ad-hoc channel access for normal operational messages consume battery power. These battery power consumptions corresponding to each state and event are tabulated in Table 5.

7. *rule : { ~node := 1; ~pwr := (0,0)~pwr - 0.1; } 100 { (0,0)~node = 0 and round(uniform(1,10)) > 6 }*
8. *rule : { ~pwr := (0,0)~pwr - 0.1; } 100 { (0,0)~node = 0 }*

Rule 7 and 8 changes state of susceptible-sleep (0) nodes to active with a pre-defined probability (in this example, probability = 0.4).

9. *rule : { ~node := 4; ~pwr := (0,0)~pwr - 0.1; } 100 { (0,0)~node = 5 and round(uniform(1,10)) <= (statecount(2, ~node)*3 + 4) }*
10. *rule : { ~pwr := (0,0)~pwr - 0.1; } 100 { (0,0)~node = 5 }*

If node is in patched-sleep (5) state, rules 9 and 10 change node states to patched-active (4) with the probability given by a linear function of number of infected neighbors. This makes sure that if any of the patched nodes neighbors are infected-spreading, patched node move active with higher probability, increasing the speed of patching susceptible and infected nodes.

11. *rule : { ~node := 4; ~pwr := (0,0)~pwr - 0.2; } 100 { statecount(3, ~txrx) > 0 and ((0,0)~node = 1 or (0,0)~node = 2) }*

If the node is susceptible-active or infected-spreading and any of its neighbors are transmitting a patch message, rule 11 will change the node state to patched-active.

12. *rule : { ~node := 2; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 and (statecount(2, ~txrx) = 1 and statecount(1, ~txrx) = 0) }*

Rule 12 defines spreading of malware to neighbors of an infected node. If a node is active and only one neighbor is broadcasting malware, while being the only node accessing the wireless channel at that time, malware message is successfully received and node will move to infected-spreading state.

This rule can be considered as one of the important improvement over the initial version of malware propagation model since it assumes if more than one neighbor is broadcasting, it results collision and no legible message is received.

13. *rule : { ~txrx := 0; ~node := 3; ~pwr := (0,0)~pwr - 1; } 100 { (0,0)~node = 2 and (0,0)~txrx = 2 }*

Rule 13 make sure that if an infected-spreading node is broadcasted malware messages to its neighbors, its remaining power has to be reduced by 1 power unit and moved to infected-dormant (3) state, changing channel access port value to none. This simulates higher power consumptions for message broadcasting in WSN.

14. *rule : { ~txrx := 2; ~pwr := (0,0)~pwr - 0.5; } 100 { (0,0)~node = 2 and statecount(1, ~txrx) = 0}*

Rule 14 defines how an infected-spreading sensor node makes the malware broadcast decision. These broadcasts are bounded by channel access rules. Hence broadcast cannot be initiated if any other node is accessing the channel for ad-hoc message transmission. More strict media access rule can be defined by taking other broadcasts also in to account by modifying the above rule as follows;

rule : { ~txrx := 2; ~pwr := (0,0)~pwr - 0.5; } 100 { (0,0)~node = 2 and (statecount(1, ~txrx) = 0 and statecount(2, ~txrx) = 0 }

However we have noticed during simulations that if such media access condition was implemented, malware propagation speed is significantly reduced and simulation time is considerably increased. We discuss these media access rules in detail in performance evaluation section.

15. *rule : { ~pwr := (0,0)~pwr - 0.5; } 100 { (0,0)~node = 2 }*

Rule 15 address the case where if an infected spreading node could not access the channel because its used by another node. Infected-spreading node simply wait for the next round, staying in the infected-spreading state, until the channel is free or move to dead state due to low battery power

16. *rule : { ~node := 2; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 3 and round(uniform(1,10)) > (statecount(2, ~node)*2 + 2) }*

17. *rule : { ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 3 }*

If the sensor node is infected-dormant (3), rule 16 and 17 will allow it to move back to infected-spreading state with the probability decreased by a linear function of number of infected-spreading nodes in the neighbourhood. These rules model a smart malware that does not waste battery power due to unnecessary movements to infected-spreading state.

18. *rule : { ~txrx := 0; ~node := 5; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 4 and ((0,0)~txrx = 3 or (0,0)~txrx = 1) }*

Rule 18 determine that if the node is patched-active (4) and patch message or normal ad-hoc message has been transmitted, node will move to patched-sleep (5) state.

19. *rule : { ~txrx := 3; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 4 and round(uniform(1,10)) <= (statecount(2, ~node)*4 + 2) }*

According to the rule 19, if the node is patched-active (4), patch message will be transmitted with a probability increased by a linear function based on number of infected-spreading neighbors in the neighborhood. Hence if number of infected-spreading nodes is high in the neighborhood, there is a higher probability to transmit patch messages to its neighbors.

We can call this kind of approach as an intelligent self-propagating malware patching since patching overhead is minimized while maintaining effectiveness of patching by employing a function to change patching probability. Moreover, we have noticed in the simulations output that if patch is properly seeded close to infected nodes, damage from the malware can be greatly reduced.

20. *rule : { ~txrx := 0; ~node := 0; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 and round(uniform(1,10)) > 6 }*

21. *rule : { ~txrx := 0; ~node := 5; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 4 and round(uniform(1,10)) <= (statecount(2, ~node)*6 + 4) }*

If the node is susceptible-active or patched-active, there is a probability of going back to susceptible-sleep or patched-sleep respectively, without accessing the channel at all. Rules 20 and 21 models realistic cases such as, message to be transmitted is timed out and events that make sensor nodes active but does not require data transmission.

22. *rule : { ~txrx := 0; ~node := 0; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 and (0,0)~txrx = 1 }*

Rule 22 defines if the node is susceptible-active and has already transmitted ad-hoc messages in the previous time step, state will be changed to susceptible-sleep after setting channel access to none.

23. *rule : { ~txrx := 1; ~pwr := (0,0)~pwr - 0.2; } 100 { ((0,0)~node = 1 or (0,0)~node = 4) and (statecount(1, ~txrx) = 0 and statecount(2, ~txrx) = 0) }*

Rule 23 defines MAC rule for susceptible-active and patched-active nodes. If the node is susceptible-active or patched-active and none of the neighbors are using the channel, it will be accessed by the node for ad-hoc messages.

24. *rule : { ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 or (0,0)~node = 4 }*

According to rule 24, if the node is susceptible-active or patched-active and if any of the neighbors are access the channel, node has to wait for the channel to be free.

25. *rule rule : { ~node := 50; ~pwr := 50; ~txrx := 50; } 100 { t }*

Finally, we have modified the default rule, making it possible to verify the accuracy of other rules.

VI. Performance Evaluation

In this section, we analyze the outputs of both initial and extended model simulations and compare performance of initial and converted versions. Effectiveness of MAC techniques and patching in controlling malware propagation is evaluated using the extended version. We have compared two MAC protocols which are uniquely designed for sensor networks. Optimized MAC protocol aims to optimize sensor battery usage by changing duty cycle based on sensor load. But S-MAC protocol keeps fixed duty cycle. We compare these two MAC protocols in terms of how they react to a self-propagating virus such as malware. Moreover in the latter part of this section, we introduce a self-propagating, malware parching which dynamically change its patch message emission speed, linearly with the number of infected nodes.

A. Output analysis of initial model

In this sub section we discuss simulation outputs at observed at different simulation times.

Malware-Plane						Power-Plane						Access-Plane					
1.0	0.0	0.0	0.0	0.0	0.0	20.0	20.0	20.0	20.0	20.0	20.0	30.0	30.0	30.0	30.0	30.0	30.0
0.0	0.0	0.0	0.0	0.0	0.0	20.0	20.0	20.0	20.0	20.0	20.0	30.0	30.0	30.0	30.0	30.0	30.0
0.0	0.0	0.0	0.0	0.0	0.0	20.0	20.0	20.0	20.0	20.0	20.0	30.0	30.0	30.0	30.0	30.0	30.0
0.0	0.0	0.0	0.0	0.0	0.0	20.0	20.0	20.0	20.0	20.0	20.0	30.0	30.0	30.0	30.0	30.0	30.0
0.0	0.0	0.0	0.0	0.0	0.0	20.0	20.0	20.0	20.0	20.0	20.0	30.0	30.0	30.0	30.0	30.0	30.0
0.0	0.0	0.0	0.0	0.0	0.0	20.0	20.0	20.0	20.0	20.0	20.0	30.0	30.0	30.0	30.0	30.0	30.0

Figure 11 Initial values

Cells in three planes of the initial model are initialized using a *.val* file. Output *.log* file is visualized using Drawlog tool and CD++ Modeler tool in the simulator. In this section, we have analyzed CD++ Modeler outputs of few important steps of the simulation execution. Figure 11 shows CD++ Modeler output at simulation time 0 ms. Hence it shows just after assigning initial values to cells in malware plane, power plane and access plane.

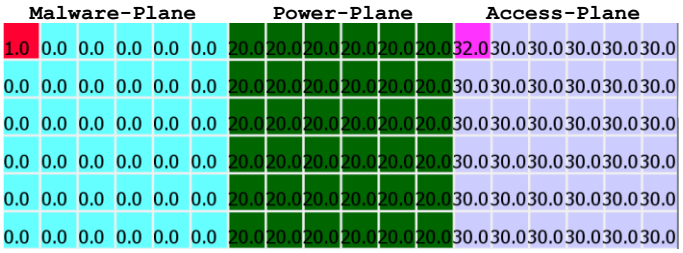


Figure 12 Starting broadcast

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 an infection spreading node (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) and none of the nodes are broadcasting (state value 31) or receiving (state value 32).

According to rules, infected node will start broadcasting after random countdown time. Output depicted in Figure 12 is observed in simulation time, 600ms 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 infected & dormant state while its battery power has been reduced to 18 as shown in the Figure 13. In the meantime, three neighbors of the infected cell has received the malware message (not shown in the figure) and moved to infection spreading state.

In the next time step, three infected neighboring nodes of (0,0,0) will state broadcasting following media access rules, which are defined to minimize collisions by not allowing neighbors broadcast simultaneously. However, we have noticed that occasionally, neighboring cell broadcast simultaneously and in this initial version, we did not attempt to address this issue.

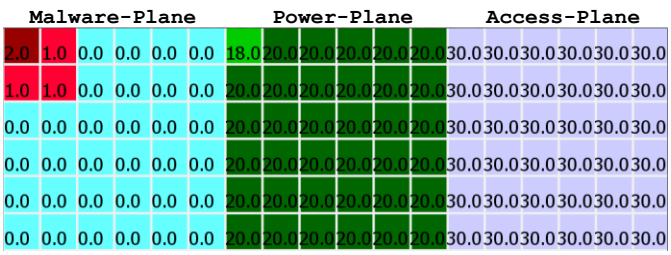


Figure 13 Malware propagation to neighbors

An intermediary state taken at simulation time 14000ms is shown in Figure 14. In this figure we can see behavior of self-propagating malware. From the access plane shows broadcast (value 32) message initiation and broadcast receiving (value 31). This gives a good example of minimal simultaneous multiple broadcasts in same neighborhood. 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..

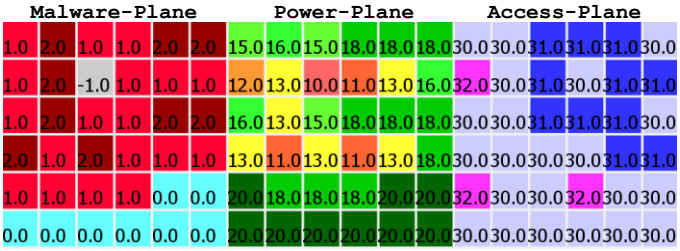


Figure 14 Intermediary output of the simulation

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.

B. Performance comparison with converted model

Simulation performance compression of initial version simulated in general CD++ with directly converted version in parallel CD++ is discussed in this sub-section. As we have discussed before, same rules used for initial version has been used in the converted version with slight modifications to make use of multiple input/output ports, instead of multiple planes.

We have simulated both versions in parallel CD++ using RESTful Interoperability Simulation Environment (RISE) [12]. Both simulations were executed by changing size of lattice dimensions from 20X20 to 5X5 in four steps. Execution times of each simulation run for both models were tabulated in tables 6 and 7.

simulation run	lattice size			
	5x5	10x10	15x15	20x20
1st run	0.708	4.716	10.624	25.426
2nd run	0.778	4.668	10.595	25.404
3rd run	0.796	4.699	10.605	25.468
4th run	0.771	4.694	10.601	25.422
5th run	0.706	4.713	10.634	25.362
average	0.7518	4.698	10.6118	25.4164

Table 6 Execution times of initial model

simulation run	lattice size			
	5x5	10x10	15x15	20x20
1st run	0.068	1.204	3.727	7.749
2nd run	0.066	1.198	3.802	7.773
3rd run	0.072	1.118	3.759	7.693
4th run	0.064	1.144	3.784	7.725
5th run	0.068	1.16	3.765	7.762
average	0.0676	1.1648	3.7674	7.7404

Table 7 Execution times of converted model

Average execution times with respect to lattice dimension sizes were plotted and shown in Figure 15.

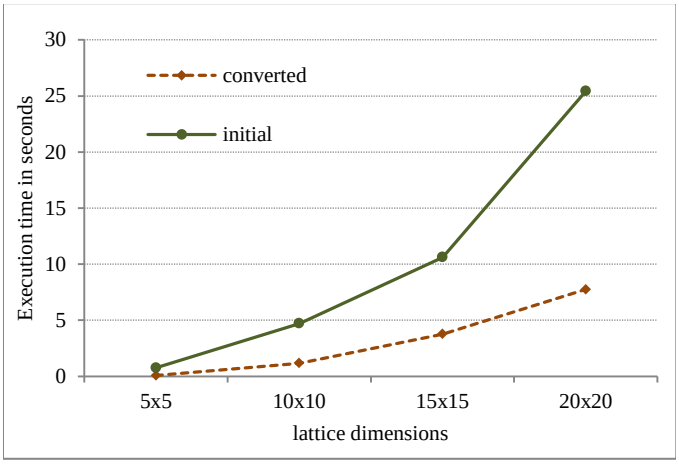


Figure 15 Average execution times vs. lattice dimensions

According to plotted execution times shown in Figure 15, we can see that the performance of the converted model is better than the initial version in terms of execution times. We can assume that, this performance improvement is due to reduced simulation complexity in the converted model, which uses multiple ports instead of multiple planes.

C. Effects of MAC protocols on malware propagation

Extended model introduced in Section V was also simulated using the RISE version of parallel CD++ using a 25X25 2-dimensional lattice. Initially we have observed malware propagation patterns with different media access rules.

As we have discussed previously, Optimized MAC protocol increases duty cycle based on total load in the sensor network. This characteristic was modeled by maintaining inversely proportional relationship between ad-hoc message generations and sleeping probabilities. S-MAC protocol on the other hand, maintains fixed sleep/active cycles regardless of the load. In order to compare these protocols, we have obtained number of infected sensor nodes after 2000ms simulation time, for different combinations of ad-hoc message generation and sleep probabilities. We have made some modifications to few existing rules discussed in Section V, as follows;

%ORIGINAL RULE 20

rule : { ~txrx := 0; ~node := 0; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 and round(uniform(1,10)) > 6 }

%MODIFIED RULE 20

*rule : { ~txrx := 0; ~node := 0; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 and round(uniform(1,10)) <= **$[S_p]$** }*

Original rule 20 was introduced to model message time-outs and active nodes that move to sleep state without transmitting any message. We have used this moving-to-sleep rule to change sleeping probability in different simulation runs. Hence as shown on the modified rule 20, S_p which stand for sleeping probability was changed in each simulation run to obtain results shown in table 8.

%ORIGINAL RULE 22

rule : { ~txrx := 0; ~node := 0; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 and (0,0)~txrx = 1 }

%MODIFIED RULE 22

rule : { ~txrx := 0; ~pwr := (0,0)~pwr - 0.2; } 100 { (0,0)~node = 1 and (0,0)~txrx = 1 }

Original rule 22 was used to send nodes from active state to sleep after an ad-hoc message transmission. We have modified this rule by removing its post-condition, move-to-sleep, to make sure that moving to sleep is only controlled by modified rule 20. Hence from modified rule 22, we only expect to change channel access state back to no-channel-access, after transmitting a message.

%ORIGINAL RULE 23

rule : { ~txrx := 1; ~pwr := (0,0)~pwr - 0.2; } 100 { ((0,0)~node = 1 or (0,0)~node = 4) and (statecount(1, ~txrx) = 0 and statecount(2, ~txrx) = 0) }

%MODIFIED RULE 23

*rule : { ~txrx := 1; ~pwr := (0,0)~pwr - 0.2; } 100 { ((0,0)~node = 1 or (0,0)~node = 4) and round(uniform(1,10)) > **$[L_p]$** }*

Original media access control rule of susceptible and patched nodes was modified by adding a probability condition and removing channel lock constraints by neighbors from pre-condition. In the modified rule 24, load probability can be changed by changing L_p.

Figure 16 shows CD++ Modeler outputs at 2000ms, obtained by keeping S_p constant at 0.6 and changing from L_p 0.2 to 0.8 in 0.2 steps.

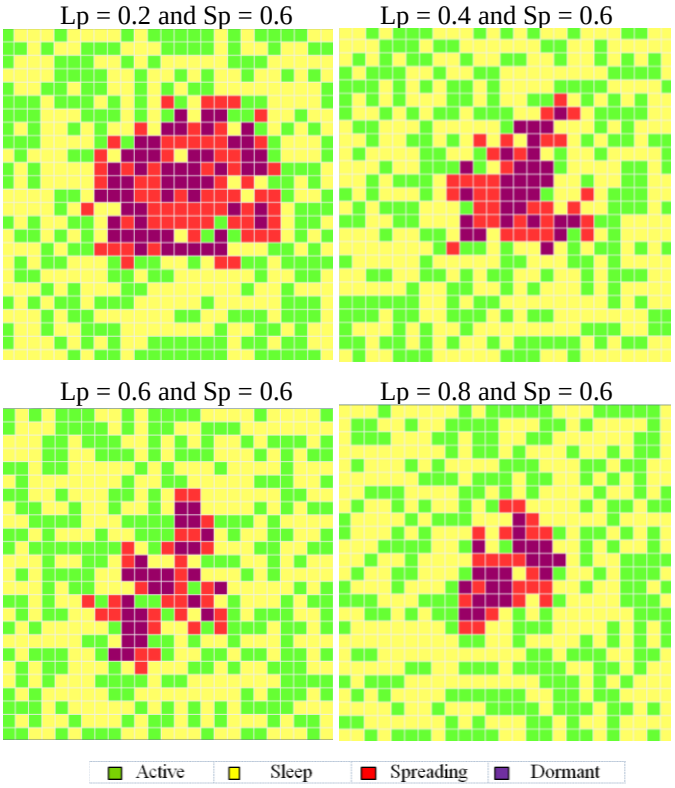


Figure 16 Malware spreading patterns for constant sleep and increasing load probabilities

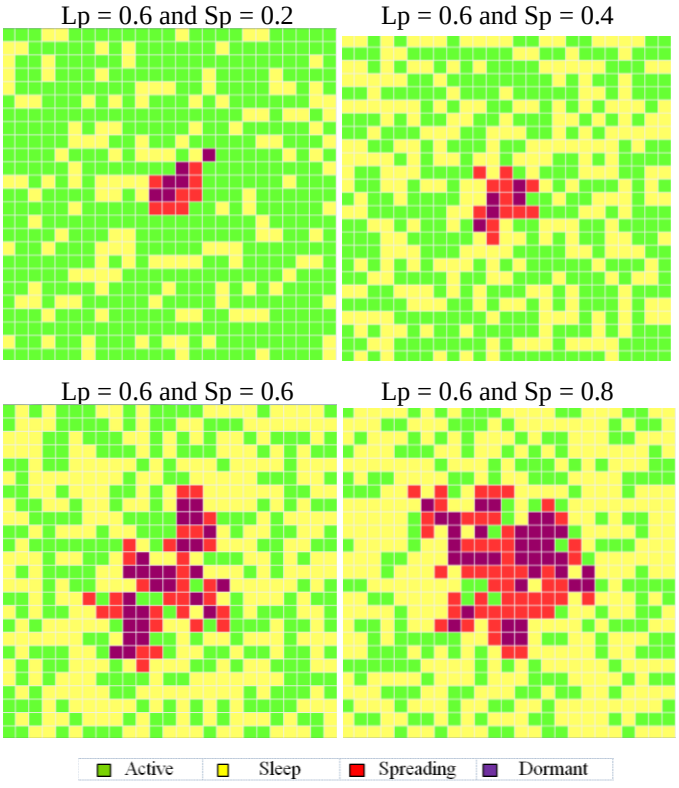


Figure 17 Malware spreading patterns for constant node load and different sleep probabilities

According to the outputs shown in Figure 16, It can be seen that when node load in terms of messages to be transmitted is increased, malware spreading speed is decreased. This is because different node loads make WSN channel access highly competitive and broadcast collisions prevent faster spreading.

On the other hand, if we keep node load L_p constant and sleeping probability S_p increased, increased malware propagation speeds can be observed as shown in the CD++ modeller outputs in Figure 17.

By changing workloads L_p and sleep probabilities S_p, in sensor nodes, number of infected nodes in the network after 2000ms simulation time is obtained and tabulated in Table 8.

Message Generation Probability [Lp]	Sleeping Probability [Sp]			
	0.2	0.4	0.6	0.8
0.2	80	77	123	128
0.4	20	31	64	92
0.6	14	17	57	90
0.8	8	15	43	78
1	7	10	16	56

Table 8 Number of infected nodes after 2000ms, for different Sp and Lp values

For this work, each simulation was executed only once and hence we recognize that the error percentage can be relatively high. We expect to perform more sophisticated simulations in our future work to obtain more accurate results.

We have plotted number of infected nodes at 2000ms by changing L_p and S_p. Figure 18 shows the number of infected nodes plotted against S_p.

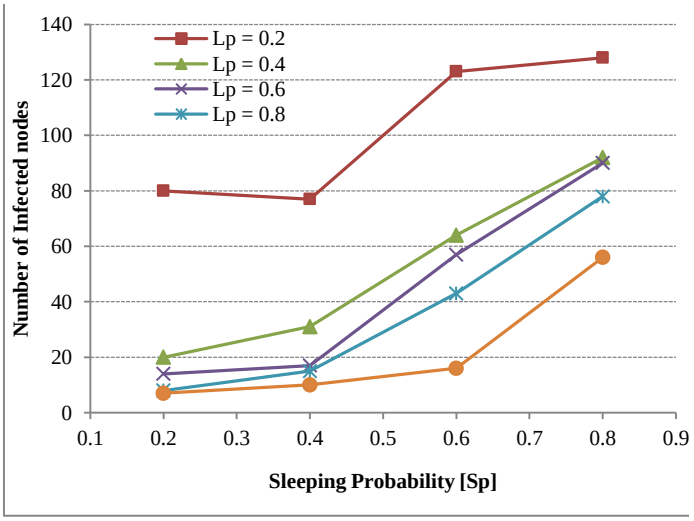


Figure 18 Number of infected nodes at 2000ms vs. sleeping probability of sensor nodes

From the graph in Figure 18, we can see that when sleeping probabilities are increased, numbers of infected nodes are increased.

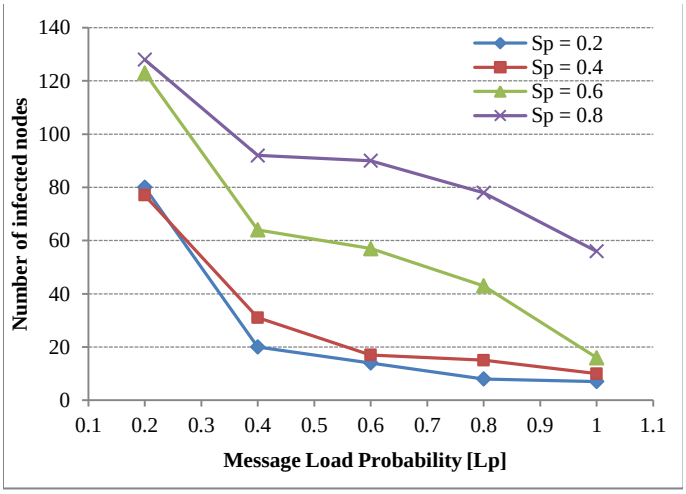


Figure 19 Number of infected nodes at 2000ms vs. sensor node communication load

Figure 19, shows infected node variation with respect to sensor node load probabilities L_p from this graph, we can observe that, when the communication load over sensor nodes due to normal operational messages are increased, malware propagation speeds are decreased.

Based on these two observations, we can compare effectiveness of considered Optimized-MAC and S-MAC protocols, in controlling malware propagation speeds. We have previously discussed that Optimized-MAC protocol changes the duty cycle based on sensor node load and S-MAC protocol keeps fixed duty cycles irrespective if the load. Hence we can build an argument that, if we change load probabilities keeping sleep probabilities constant, it resembles operation of S-MAC protocol. Furthermore, if we reduce sleep probability inversely proportional to load probability, it resembles operation of Optimized-MAC protocol. Extraction of data from Table 8 based on this argument is highlighted by two dotted rectangles. We have arranged these extracted data in Table 9.

Message Generation Probability Lp	Optimized MAC	S-MAC
0.2	123	123
0.4	64	92
0.6	17	90
0.8	8	78

Table 9 infected nodes at 2000ms for different MAC protocols

We have plotted number of infected nodes for each MAC protocol as shown in Figure 20.

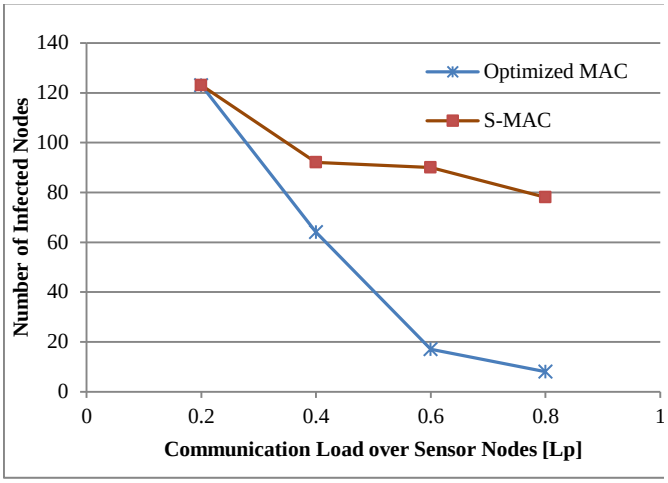


Figure 20 Malware propagation behaviors in Optimized-MAC and S-MAC protocols

From the graph shown in Figure 20, we can observe that Optimized-MAC protocol is more effective than the S-MAC protocol in controlling malware propagation speeds over WSN.

D. Effectiveness of Patching

In this sub-section, we discuss the effectiveness of patching in minimizing the harmful effects of malware in WSN. We have simulated four scenarios, where single patching seed placed far from the malware seeds, two patching seeds placed far from malware, single patching seed placed close the malware seeds and multiple patching seeds surrounding malware seeds. For all simulations, sleeping probability is maintained at 0.4 and non-probabilistic channel access rules which were defined in section V are used. We have placed five malware seeds in Von-Neumann neighborhood instead of one seed at the starting of the simulation, to model the realistic scenario where malware patching is usually done reactively after noticing a malware attack on the WSN.

First we have carried out a simulation by keeping a single patching seed 10 nodes away from the central malware seed. CD++ Modeler outputs observed at different simulation times are shown in Figure 21.

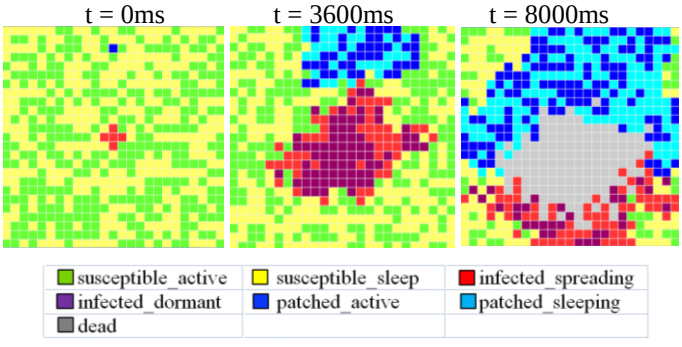


Figure 21 Single patching seed placed far from malware

In the second scenario, we have placed two patching seeds on the opposite sides of the malware, each with 10 nodes away from the central infected node. CD++ Modeler output corresponding to this scenario is shown in Figure 22.

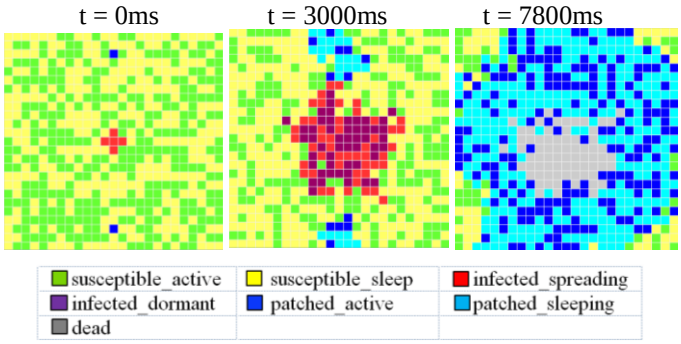


Figure 22 Two patching seeds placed far from malware

In the third scenario, we have observed the effectiveness of patching by placing a single patching seed 5 nodes away from the central infected node. Observed CD++ Modeler outputs are illustrated in Figure 23.

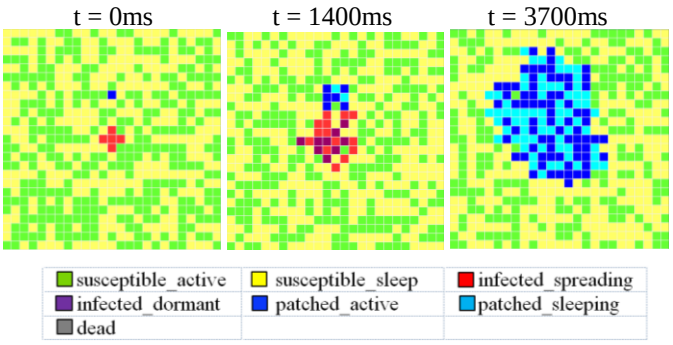


Figure 23 Single patching seed placed closed to malware

In the fourth scenario, we have observed the reaction of the malware to patching by placing two patching seeds 5 nodes away from central infected node on opposite sides. Observed CD++ Modeler outputs are shown in Figure 23.

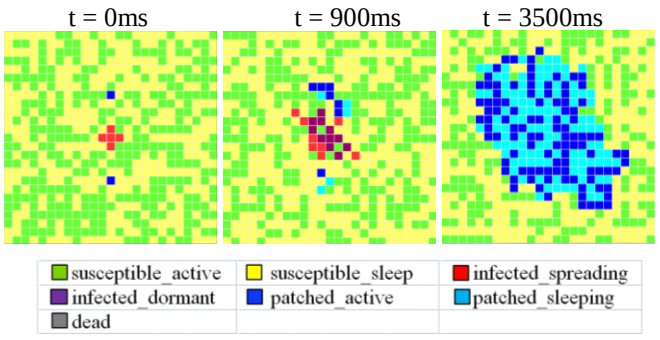


Figure 24 Two patching seeds placed close to the malware

From Figure 21-24, each figure shows three simulation outputs taken at the simulation time *t*, written on top of each output. Outputs in the left side of each figure shows initial seed placement and middle output shows the instances where the patched-active nodes first see infected-spreading nodes in their neighborhood. Output to the right in Figures 22-24 shows the simulation step where all the malware was completely removed by self-propagating patching. However in Figure 21, it shows an intermediary step that can result in topological fragmentation due to dead sensor nodes block patches from propagating to other parts of the topology. Patching message generation rate of patched-active nodes is increased linearly with the number of infected-spreading nodes in their neighborhood. Dynamic patching message generation helps WSN in two ways. First, it will not impose significant network overhead on the uninfected nodes. Secondly it will attack malware faster minimizing its harmful effects.

Outputs shown in figure 21 shows the patch and malware behavior when infected and patching seeds placed far away from each other. In such a scenario, time taken by patches to reach the infected nodes is considerably high and thus malware gets enough time to spread over the WSN and do higher damage. Damage done by malware can be seen from the higher number of dead nodes created by draining battery from sensor nodes. Moreover by killing topologically critical sensor nodes, malware can block patching in certain parts of the network topology.

By placing multiple patching seeds in different parts of the WSN topology as shown in Figure 22, fragmentation of topology can be minimized. However, still the malware gets enough time to do a significant damage to WSN by draining battery power completely from several sensor nodes. In figures 23 and 24, we see that that when patching seeds are placed close to the malware seeds, effectiveness of the patching is maximized and malware can be quickly removed from WSN without allowing it to do a significant damage to sensor nodes.

VII. CONCLUSION AND FUTURE WORK

In this term paper, have followed a simulation based approach to study malware propagation patterns, in highly resource constrained WSN environments. WSN are highly vulnerable to viruses, worms and malicious programs such as self-propagating malware. Due to limitations of processing, memory and battery power, powerful security features are not cost effective for most WSNs. Hence media access controlling is widely regarded as an efficient and cost effective method of restricting malware propagation in WSNs. In this work, we have investigated the ability of two widely used MAC protocols; Optimized-MAC and S-MAC in restraining malware propagation.

Furthermore, Patching can be considered as a reactive approach towards controlling malware propagation in WSN. Efficient placement of self-propagating patching seeds can significantly mitigate adverse effects of malware attacks. However, if patches are not delivered to sensor nodes in time or not placed properly, malware can still damage sensor nodes draining battery power or compromising its functions. Hence

use of proper MAC protocols along with efficient patching is highly important to ensure secure operation in WSN.

Due to limitations of time and resources, we have limited our study only to two most basic MAC protocols used in WSN. As for our future work, we recognize the importance of carrying out a more detailed study in this area focusing on other popular WSN MAC technologies. Moreover, we are planning to integrate topological details in our future simulations and develop more powerful patching algorithms comparing effectiveness's of different distributions instead of limiting our scope to simple linear functions.

REFERENCES

[1] S. A. Khayam and H. Radha, "Using signal processing techniques to model worm propagation over wireless sensor networks," Signal Processing Magazine 23, 164–169 (2006)

[2] Y. Song and G.P. Jiang, "Modeling malware propagation in wireless sensor networks using cellular automata," in IEEE international conference on Neural Networks and Signal Processing, Zhenjing, China, June 2008.

[3] I. G. Georgoudas, G.C. Sirakoulis and I. Andreadis, "Modelling earthquake activity features using cellular automata," in Mathematical and Computer Modelling, vol. 46, pp. 124–137, 2007.

[4] L.H.Encinas, S. H. White, A. M. D. Rey and R. G. Sanchez, "Modelling forest fire spread using hexagonal cellular automata," in Applied mathematical modelling vol. 31, pp. 1213–1227 2007.

[5] G. Wainer, "CD++: a toolkit to define discrete-event models," In Software, Practice and Experience. Wiley. vol. 32, no.3, pp. 1261-130, 2002.

[6] G. A. Wainer and N. Giambiasi, "Timed Cell-DEVS: modelling and simulation of cell spaces," Springer-Verlag, 2001.

[7] D. A. Rodriguez and G. A. Wainer, "CD++ User's Guide," Departamento de Computación, Universidad de Buenos Aires, Buenos Aires, Argentina, 1999

[8] G. A. Wainer. "Discrete-Event Modeling and Simulation: a Practitioner approach," Taylor and Francis, 2009.

[9] W. Ye, J. Heidemann, D. Estrin, "Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks," IEEE/ACM Transactions on Networking, vol. 12, issue. 3, pp.:493 - 506, June 2004.

[10] R. Yadav, S. Varma and N. Malaviya, "Optimized Medium Access Control for Wireless Sensor Network," in IJCSNS International Journal of Computer Science and Network Security, vol. 8, no. 2, pp. 334 -338, February 2008.

[11] A. López and G. Wainer, "Improved Cell-DEVS model definition in CD++," P.M.A. Sloot, B. Chopard, and A.G. Hoekstra (Eds.): ACRI 2004, LNCS 3305. Springer-Verlag. 2004.

[12] S. Wang and G. A. Wainer, "RISE User's Guild Manual (Integrated version)," Dept. of Systems and Computer Engineering Carleton University, Ottawa, Canada