MODELING ROUTING IN WIRELESS AD-HOC NETWORKS USING CELL-DEVS

Umar Farooq **Bengu Balya** **Gabriel Wainer**

Dept. of Systems and Computer Engineering
Carleton University. 1125 Colonel By Dr. Ottawa, ON. K1S 5B6. Canada
{ufarooq, bengu, gwainer}@sce.carleton.ca

Keywords: DEVS, Cell-DEVS, ad-hoc networks, AODV.

Abstract
Ad-hoc networks are self-organizing systems conformed by wireless cooperating nodes that use the concept of neighboring nodes to build networks with variable topology. Analyzing these networks is a complex task due to the irregular nature of their behavior. Cell-DEVS is an extension to Cellular Automata in which each cell in the system is considered a DEVS model. The approach permits defining models with asynchronous behavior, and to execute them with high efficiency. Here we show how these techniques can be used to model ad-hoc networks, easing model definition, and developing new experiments and to define new routing techniques.

1. INTRODUCTION

In recent years, we have witnessed a large number of research efforts in the area of ad-hoc networks. Ad-Hoc networks are characterized by several wireless nodes (mobile or static) communicating with each other without an infrastructure. Each node can act as a router, forwarding data from one neighboring node to another. These networks are decentralized, with temporary interconnections, and arbitrary behavior of the nodes. Due to the complex characteristics, modeling and simulation is a natural choice to analyze the behavior of the components, and to study the execution results of new algorithms [1].

One of the challenging aspects in the study of ad-hoc networks is the routing algorithm for data transmission between the nodes. Several algorithms were developed, and some of them consider power a cost metric rather than physical distance or shortest path in order to determine best route path. Any routing algorithm has to take into account the highly dynamic nature of the mobile nodes in the network and unreliable and bandwidth-limited wireless links. Different modeling and simulation tools have been developed [2][3], but in recent years, different authors [4][5] decided to represent this problem using Cellular Automata (CA) [6]. CA are discrete-time discrete models described as cells organized as n-dimensional infinite lattices, and they evolve executing a local transition function, which uses the present value for the cell and a finite set of neighbors.

The use of a discrete time base constrain the precision of these model; consequently, executing complex CA usually requires large amounts of CPU time, primarily due to its synchronous nature. Cell-DEVS [7] solved these problems using the DEVS (Discrete Events systems Specification) formalism [8] to define a cell space where each cell is defined as a DEVS model. Cell-DEVS permits building discrete-event cell spaces, and it improves their definition by making the timing specification more expressive. DEVS is used to formally specify discrete events systems using a modular description. This strategy allows the reuse of tested models, improving the safety of the simulations and allowing to reduce the development time. As it is a discrete event formalism, it uses a continuous time base, which allows accurate timing representation, and reduces CPU time requirements.

The CD++ tool [9] was built to implement DEVS and Cell-DEVS theory. Our goal was to build models of ad-hoc networks using the improved advantages of Cell-DEVS. Defining new models using this method is relatively simple compared with other traditional methods. The use of Cell-DEVS also enables integration with other existing models, permitting to define multi-formalism applications. An advantage of this approach is that in ad-hoc models are often composed of different subcomponents (for instance, the routing algorithm and the city topology) interacting together. We can also make use of existing infrastructure, including parallel simulators and distributed environments, and a variety of visual tools.

We used a variant of Lee's Algorithm [10] to find out the shortest path between two communicating nodes on a network plane, implementing it easily and efficiently using CD++. We will show the definition of the Ad-hoc on Demand Distance Vector (AODV) [11] routing algorithm in CD++, and different extensions proposed. We will also show how to model the mobility behavior of mobile nodes. A basic routing algorithm is implemented to determine the minimum distance of mobile nodes from the gateway. This algorithm is useful for networks where the main traffic is through the gateway. Finally, a network coverage model is implemented, to determine the cells with potential to reach to the gateway. Every cell, which has potential to reach to the gateway, is considered to be within the coverage area.

2. BACKGROUND

Ad-Hoc networks are characterized by dynamic topology changes, severe power constraints and unpredictable wire-

less environment [1]. There is no infrastructure, and routers are mobile themselves, depending on battery power. Nodes are connected with wireless links, which are prone to environment factors (fading, shadowing and noise). Therefore, link failures or congestion are not considered problems, but the normal behavior of the network.

Several ad-hoc routing algorithms are proposed in recent years, such as Destination-Sequenced Distance Vector (DSDV) [12], Dynamic Source Routing (DSR) [13], AODV [11] and Temporally Ordered Routing Algorithm (TORA) [14]. DSDV is a table driven routing algorithm as traditional routing protocols. On the other hand, DSR, TORA and AODV are on demand algorithms. We used AODV, which is one of the first ad-hoc routing algorithms chosen by IETF as an experimental RFC standard. While having low processing and memory overhead, AODV offers quick adaptation to dynamic link conditions.

Cell-DEVS [7] was defined as a combination of DEVS [8] and CA with timing delays. This approach allows describing cell spaces as discrete events models, where each cell is seen as a DEVS atomic model that can be delayed using several constructions. In Cell-DEVS, each cell is defined as a DEVS model, and a procedure to couple cells is depicted (as showed in Figure 1). The delay functions allow to define, complex behavior for each cell, improving the definition for each of the submodels: transport delays have anticipatory semantics (every output event is transmitted after a delay), and inertial delays have preemptive semantics (a scheduled event will not necessary be executed).

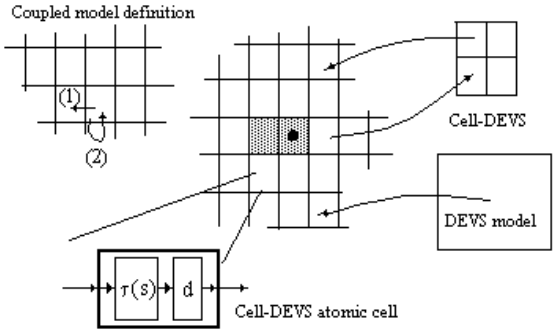


Figure 1. Informal definition of Cell-DEVS.

A DEVS model [8] is seen as composed by behavioral (atomic) submodels than can be combined into structural (coupled) models. A DEVS atomic model can be described as $M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$. Input external events are considered to be received in input ports (***X***). When an event arrives, the model executes the external transition function $\boldsymbol{\delta_{ext}}$ to produce a state change. Each state has an associated duration **D**. When this lifetime is consumed, the internal transition function $\boldsymbol{\delta_{int}}$ is activated to produce internal state changes. The internal state **S** can be used to provide model outputs, which are sent through the output ports (***Y***). They are sent by the output function $\boldsymbol{\lambda}$, which executes before the internal transition.

An atomic model can be combined with other DEVS models to build a structural model. These models are called coupled, and they are defined as: $CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\} >$. Each coupled model consists of a set of **D** basic models $\mathbf{M_i}$ connected through input/output ports. The list of influencees $\mathbf{I_i}$ of a given model is used to determine the models to which outputs must be sent. These sets are used to build the translation function $\mathbf{Z_{ij}}$, which is in charge of translating outputs of a model into inputs for the others. An index of influencees is created for each model (I_i). For every *j* in the index, outputs of model M_i are connected to inputs in model M_j.

CD++ [9] is a modeling tool that was defined using the DEVS and Cell-DEVS specifications. CD++ makes use of the independence between modeling and simulation provided by DEVS, and different simulation engines have been defined for the platform: a stand-alone version, a Real-Time simulator, and a Parallel simulator. DEVS Atomic models can be programmed and incorporated onto a class hierarchy programmed in C++. Coupled models can be defined using a built-in specification language. Cell-DEVS models are built using a specification language provided to describe them. The model specification includes the definition of the size and dimension of the cell space, the shape of the neighborhood and borders. The cell's local computing function is defined using a set of rules with the form: POST-CONDITION DELAY {PRECONDITION}. These indicate that when the *PRECONDITION* is satisfied, the state of the cell will change to the designated *POSTCONDITION*, whose value will be transmitted to other components after consuming the *DELAY*.

We have used CD++ to model the core functionality of the AODV Protocol using Cell-DEVS. In the following sections, we will describe how to analyze the execution behavior of these applications.

3. MODELING AODV

The AODV protocol assumes bi-directional links, and creates routes on demand. Whenever a node needs to communicate with another one, it broadcasts a Route Request message (RREQ) to its neighbors. They re-broadcast the message and set up a reverse path pointing towards the source. When the intended destination receives a RREQ message, it replies by sending a Route Reply (RREP) that travels along the reverse path set up by RREQ.

Our model considered a network plane in which nodes are spread at random. The network plane does not make any assumptions about the physical location of the nodes in the area; thus, each cell may have a different size in terms of the physical area represented. Data movement between two cells represent one hop, as routing takes into account the shortest hop count instead of the actual physical. Each node

can communicate to the nodes to the N, S, E and W. However, if each neighbor is not a node, we have a *dead cell* through which communication cannot take place. These cells represent physical obstacles (such as a high rise building) or simply the absence of a communication link. Two nodes with a dead cell in between cannot communicate directly. If we assume the cost of the communication links between any two nodes that can directly communicate is the same, modeling AODV using Cell-DEVS involves finding the shortest path between two nodes.

We used of a variant of the classical Lee's Algorithm [10]. Figure 2 shows a simple example of a network plane. Here, S represents a sender node and D, a destination node, while black cells represent dead cells. In order to find a route from S to D, the node S broadcasts a RREQ message to all its neighbors (called the *wave* nodes). The wave nodes re-broadcast the message to their neighbors, and set up a reverse path to the sender. These nodes further re-broadcast this message and set up a reverse path to the nodes from which they received the message. This process continues until the message reaches the destination node D.

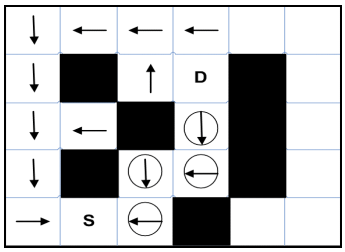


Figure 2. AODV Routing in Cell-DEVS.

Since there is more than one path, the destination may receive multiple RREQ messages for the same sender. However, the route through which the destination node receives the RREQ message first is the shortest path between the sender and the destination. The destination thus ignores all RREQ messages for the same sender except the first one, and it replies sending a RREP message using the reverse path. All the wave nodes on this route become *path* nodes (represented with circles containing arrows in the figure). All other wave nodes move to a *clear* state (not shown in the figure). This model can be formally defined using Cell-DEVS specifications as follows:

CD=< *X*, *Y*, S, N, d, δ_{int}, δ_{ext}, τ, λ, D >, X=Φ, Y=Φ, S= **dead**, **init**, **initD** (initial state of the destination), **DR** (Destination Ready; state after receiving a send request), **InitS** (Initial State of the Sender), **WaveU** (Wave Up), **WaveD** (Down), **WaveR** (Right), **WaveL** (Left), **PathU** (Path Up), **PathD** (Down), **PathR** (Right), **PathL** (Left), **Clear** and **Found** (destination found; final state of Sender).

N={(-1,0), (0,-1), (0,0), (0,1), (1,0)}, d=100ms

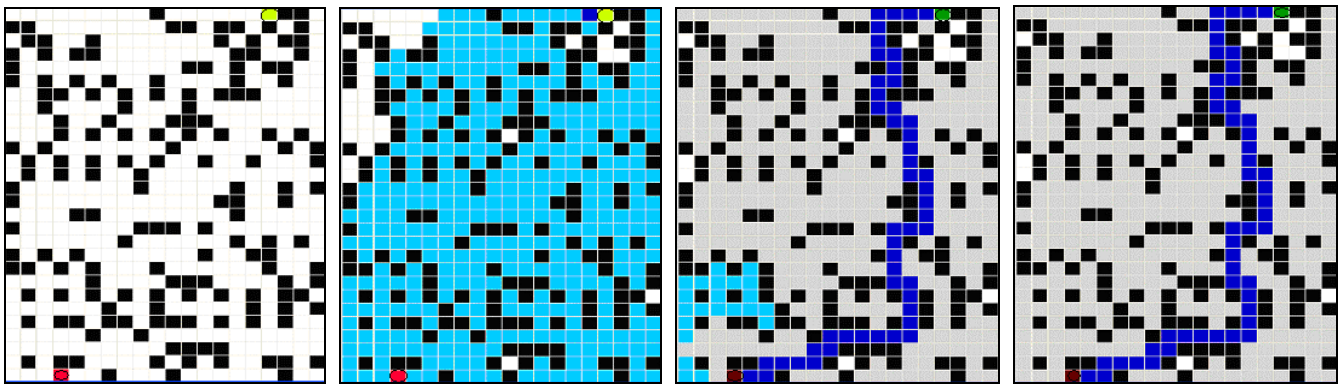
τ: $N^{n} \rightarrow S$ are the rules defined according to the algorithm discussed, and Θ, δ_{int}, δ_{ext}, λ, D are defined according to the definitions of Cell-DEVS atomic models. Using this specification as a base, the model was implemented in CD++, as presented in Figure 3.

```
[path]
type : cell  width : 20  height : 28   delay : transport
neighbors : (-1,0) (0,-1) (0,0) (0,1) (1,0)
localtransition : path-rule   border : nowrapped

[path-rule]
rule: DR 100 { (0,0) = InitD and stateCount(PathU) > 0}
rule: DR 100 { (0,0) = InitD and stateCount(PathD) > 0}
rule: DR 100 { (0,0) = InitD and stateCount(PathR) > 0}
rule: DR 100 { (0,0) = InitD and stateCount(PathL) > 0}
rule:WaveU 100{(0,0)=Init and(-1,0)>DR and (-1,0)< PathU}
rule: WaveD 100 {(0,0)=Init and (1,0)>DR and (1,0)<PathU}
rule: WaveR 100 {(0,0)=Init and(0,1)>DR and (0,1)<PathU}
rule: WaveL 100 {(0,0)=Init and(0,-1)>DR and(0,-1)<PathU}
rule: PathU 100 {(0,0)=WaveU and stateCount(InitD) = 1}
rule: PathD 100 {(0,0)=WaveD and stateCount(InitD) = 1}
rule: PathR 100 {(0,0)=WaveR and stateCount(InitD) = 1}
rule: PathL 100 {(0,0)= WaveL and stateCount(InitD) = 1}
rule: PathU 100 { (0,0) = WaveU and (0,-1)= PathR}
rule: PathU 100 { (0,0) = WaveU and (0,1) = PathL}
rule: PathU 100 { (0,0) = WaveU and (1,0) = PathU}
rule: PathD 100 { (0,0) = WaveD and (0,-1)= PathR}
rule: PathD 100 { (0,0) = WaveD and (0,1) = PathL}
rule: PathD 100 { (0,0) = WaveD and (-1,0)= PathD}
rule: PathR 100 { (0,0) = WaveR and (0,-1)= PathR}
rule: PathR 100 { (0,0) = WaveR and (-1,0)= PathD}
rule: PathR 100 { (0,0) = WaveR and (1,0) = PathU}
rule: PathL 100 { (0,0) = WaveL and (0,1) = PathL}
rule: PathL 100 { (0,0) = WaveL and (-1,0)= PathD}
rule: PathL 100 { (0,0) = WaveL and (1,0) = PathU}
rule: clear 100 { (0,0) = Init and stateCount(clear) > 0}
rule: clear 100 { (0,0)>InitS and (0,0)<PathU and
          stateCount(clear) > 0}
rule: clear 100 {(0,0)>InitS and (0,0)<PathU and
          stateCount(DR) > 0}
rule: clear 100 { (0,0) > InitS and (0,0) < PathU and
          stateCount(found) > 0}
rule: clear 100 { (0,0) > InitS and (0,0) < PathU and
        (-1,0) >WaveL and (-1,0)<clear and (-1,0)!=PathD}
rule: clear 100 { (0,0) > InitS and (0,0) < PathU and
        (1,0) > WaveL and (1,0) < clear and (1,0)!=PathU}
rule: clear 100 { (0,0) > InitS and (0,0) < PathU and
        (0,-1) >WaveL and (0,-1)<clear and (0,-1)!=PathR}
rule: clear 100 { (0,0) > InitS and (0,0) < PathU and
        (0,1) > WaveL and (0,1) < clear and (0,1)!=PathL}
rule: found 100 {(0,0)=InitS and stateCount(PathU)>0}
rule: found 100 {(0,0)=InitS and stateCount(PathD)>0}
rule: found 100 {(0,0)=InitS and stateCount(PathR)>0}
rule: found 100 {(0,0)=InitS and stateCount(PathL)>0}
rule: {(0,0)} 100 {t}
```

Figure 3. Implementing AODV Routing in CD++.

A number of tests were conducted on the model (detailed results can be found in [15]). shows screen shots taken from execution on CD++ (dead cells are in black cells that have not received any message are white). We used 20 x 28 cells. Initially, the sender (shown in gray in the lower-left part of the figure) broadcasts a RREQ message to the destination (shown in the top-right part of the figure). This is defined by rules *Wave{U, D, L, R}* in Figure 3. After 50 steps of execution, we see the light gray nodes representing those nodes that have received a RREQ message while establishing a reverse path to the sender. The dark gray node (close to the source) carries a RREP (in accordance with the rules *Path{D, R, L, U}*).



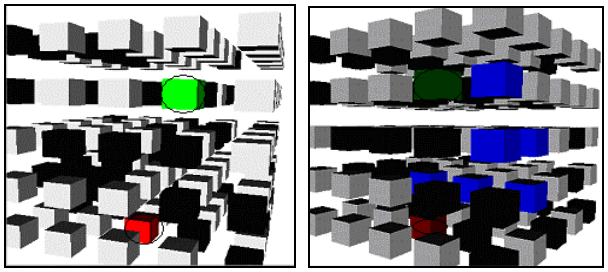
(a) **(b)** **(c)** **(d)**

Figure 4. AODV simulation (a) Initial distribution (b) State after 50 steps (c) after 100 steps (d) Final state after 106 steps.

After 100 steps, a successful route has been established. Clear are represented in light gray (rule *clear* in Figure 3). The final state after 106 steps successfully established the shortest route between the sender and the receiver and all wave nodes end up in clear state.

4. ADVANCED AODV MODELS

We extended the above model to **3 dimensions**, as a way to represent inter-network routing. This is useful if, for example, if one of the planes represents a wireless ad-hoc network and the other a wired network, it would make sense to transmit the data in the ad-hoc plane to the wired plane through the nearest gateway (because the cost in a wired network is generally less than in a wireless one). Implementation details of this model can be found in [15]. Figure 5 shows some of the execution results.



(a) (b)

Figure 5. 3D AODV simulation (a) Initial (b) Final state.

We also extended the model in section 3.1 to provide multicasting in AODV. The construction of multicast trees on cellular planes is complex, and Hochberger [4] has shown that as the number of receiver and/or sender nodes on a plane increases, the number of states for each cell goes beyond practical limits. Another to take into account during the construction of multicast trees is its optimality (i.e., the tree should duplicate data as less as possible). Consider for example the distribution of nodes in Figure 6: S represents a sender that wants to multicast data to R1 and R2. The shortest path between S and R1 is shown in light gray. For R2, there are two shortest paths. As the data is being multicast, it would make more sense to send data through Path 1 (data would be duplicated only for the last 4 hops from S to R2). On the other hand, if the data is sent through Path 2, it is duplicated right in the beginning.

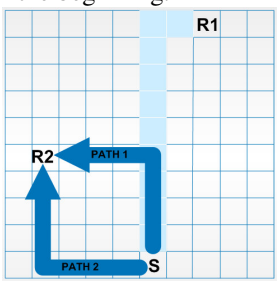


Figure 6. Optimal Multicast Trees Concept.

We proposed the following algorithm:

1. Establish the shortest route using Section 3.
2. All the path nodes become *tree* nodes, and a tree is formed between the sender and the receiver. Note that the sender and a receiver belong to the multicast group but the tree nodes are not a part of the group. During this step, all clear state nodes are re-initialized to their initial value.
3. A new node wanting to join a multicast group broadcasts a RREQ message to join the group.
4. A path is established between the new node and the nearest tree node, following the algorithm in step 1.
5. Since step 4 may generate more than one path between the tree and the new node, all non-optimal paths are purged.
6. We have now the optimal path from the new node to the tree. This path becomes the tree during this step.
7. For each subsequent node that wants to join a multicast group, steps 3 to 6 are repeated.

The formal specifications of this model are similar to the one presented in Section 3, with the addition of three states: **Tree** (nodes belonging to the multicast tree), **MS** (final state of the sender after building the tree), and **MR** (final state of the receiver after the construction of the tree).

```
 [path]
type : cell   width : 20    height : 28 delay : transport
neighbors : (-1,0)   (0,-1)  (0,0)  (0,1) (1,0)
border : nowrapped    localtransition : path-rule

[path-rule]
% Remaining DR, Wave, Clear and Path rules: see Figure 3
rule: PathU 100 {(0,0)=WaveU and (stateCount(initD)=1 or
 stateCount(tree)=1) and (stateCount(PathU)+stateCount
  (PathD)+stateCount(PathR)+stateCount(PathL)= 0)}
rule: PathD 100 {(0,0)=WaveD and (stateCount(initD)=1 or
 stateCount(tree)=1) and (stateCount(PathU)+ stateCount
   (PathD)+stateCount(PathR)+stateCount(PathL)= 0)}
rule: PathR 100 {(0,0)=WaveR and (stateCount(initD)=1 or
  stateCount(tree)=1) and (stateCount(PathU)+stateCount
    (PathD)+ stateCount(PathR)+ stateCount(PathL) = 0)}
rule: PathL 100 {(0,0)=WaveL and (stateCount(initD)= 1 or
 stateCount(tree)=1) and (stateCount(PathU)+stateCount
    (PathD)+ stateCount(PathR)+ stateCount(PathL)= 0)}
...
rule: clear 100 { (0,0) = PathU and (-1,0) = clear }
rule: clear 100 { (0,0) = PathD and (1,0) = clear }
rule: clear 100 { (0,0) = PathR and (0,1) = clear }
rule: clear 100 { (0,0) = PathL and (0,-1) = clear }
rule: found 100 {(0,0) = initS and stateCount(PathU) > 0}
rule: found 100 {(0,0) = initS and stateCount(PathD) > 0}
rule: found 100 {(0,0) = initS and stateCount(PathR) > 0}
rule: found 100 {(0,0) = initS and stateCount(PathL) > 0}
rule: init 100 {(0,0)=clear and (stateCount(initS)+
 stateCount(WaveU)+ stateCount(WaveD) + stateCount(WaveR)
     + stateCount(WaveL) = 0) }
rule: tree 100 {(0,0)>WaveL and (0,0)<clear and
          stateCount(found) > 0 }
rule: tree 100 {(0,0)> WaveL and (0,0)<clear and
   stateCount(tree)>0 and ((0,1)=PathL or (0,-1)=PathR
     or (-1,0) = PathD or (1,0) = PathU)}
rule: tree 100 {(0,0)>WaveL and (0,0)<clear and
         (stateCount(DR) + stateCount(tree) > 1)}
rule: MS 100 { (0,0) = found }
rule: MR 100 { (0,0) = DR and stateCount(tree) > 0 }
```

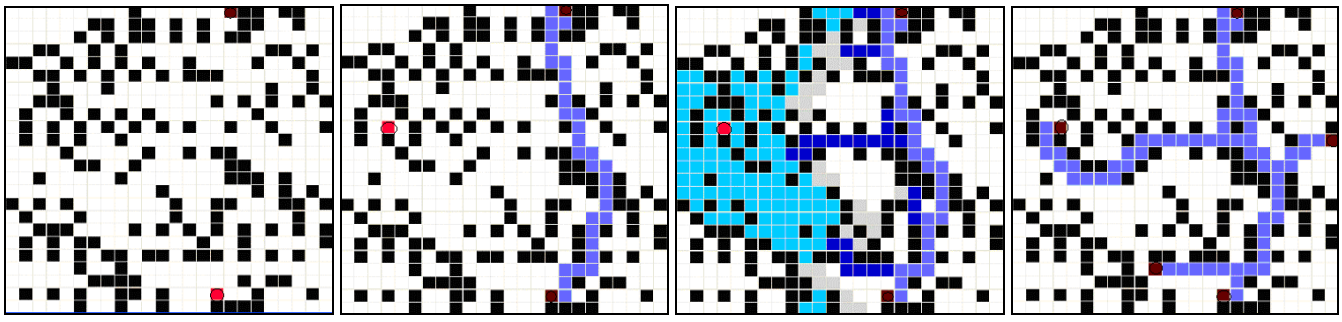
Figure 7. Implementing Multicast AODV Routing in CD++

CD++ implementation is showed in Figure 7. Figure 8 shows screen shots taken from execution of a sample model at different intervals during the test, which consists of a 25x25 space. Initially there are only two nodes(one at the top and the other at the bottom). Subsequent nodes join the group after successful completion of tree construction.

After 80 steps of execution, the path between the two nodes has been established and that path is becoming a tree according to rule *tree* in Figure 7. All clear state nodes have been re-initialized. A new node (near the left center of the figure) joins the multicast group. In Figure 8.c) the new node has broadcasted a RREQ message. As there are multiple tree nodes, multiple (non-optimal) paths are being formed. However, the algorithm logic is detecting and purging all such non-optimal paths (rule *clear* in Figure 7). The new node finds the shortest optimal route, which becomes a tree (rule *tree*). All non-optimal paths are purged and all the clear state nodes re-initialized. The final state of the model after 217 steps of execution is shown in Figure 8 (d): 3 new nodes have been successfully added to the multicast tree.

The example shows how the model successfully built an optimal multicast tree. Many non-optimal paths are generated during the addition of every new node, but the model successfully detects and builds the shortest path and purges the non-optimal paths.

We have also defined models for routing among multiple pairs of senders and receivers. The variant of Lee's Algorithm used in Section 3, fails for multiple pairs of senders and receivers: it generates deadlocks and may prevent the generation of routing path between pairs of nodes that can communicate [4]. To overcome this problem, we have exploited the inherent parallelism in Cell-DEVS and have found a simple solution to the problem: each pair of sender/receiver is allocated a plane in a 3D Cell-DEVS. The total number of planes thus depends on the total number of pairs of receivers and senders to be routed in parallel.



(a) **(b)** **(c)** **(d)**

Figure 8. AODV Multicast Modeling (a) Initial (b) After 93 steps (c) After 125 steps (d) Final state after 217 Steps.

In each plane, we run the variant of Lee's Algorithm discussed in Section 3 (with a total of 15 states for each cell). The scheme permits to route multiple pairs of senders and receivers without having to define more states. As each pair is routed separately in each plane, routing messages for each pair do not interfere with each other. By avoiding this interference, we can successfully prevent the generation of deadlocks. Moreover, our approach exploits the inherent parallelism in the Cell-DEVS model as multiple pairs are routed simultaneously. we present here a simple test as a

sample, in order to facilitate the understanding of the execution results obtained. Figure The test consists of a 5x5 cells space having 2 planes and thus routes two pairs of senders and receivers, one in each plane. Since the two planes represent the same network, we can assume the distribution of the nodes on the two planes to be the same. However, to show that our model works even if we assume different distribution of nodes in each plane, the example chosen has different distribution of nodes in plane 1 and plane 2 as shown in Figure 9 (a) and Figure 9 (b) respectively. The state of the model after 10 steps of execution in plane 1 and plane 2 is shown in Figure 9 (c) and Figure 9 (d) respectively. In plane 1, a successful route has been established between the sender and the receiver while in plane 2 RREQ message has not yet reached the destination. The final state of the model in plane 1 and plane 2 after 25 steps of execution is shown in Figure 9 (e) and Figure 9 (f) respectively. The figures show that the model has successfully established the shortest path between the sender and the receiver in each of the planes. The model thus is capable of routing among multiple pairs of senders and receivers simultaneously without having to define more states. Although we have shown an example in which two pairs are routed simultaneously, an arbitrary number of pairs can be routed simultaneously by defining each pair in a separate plane.

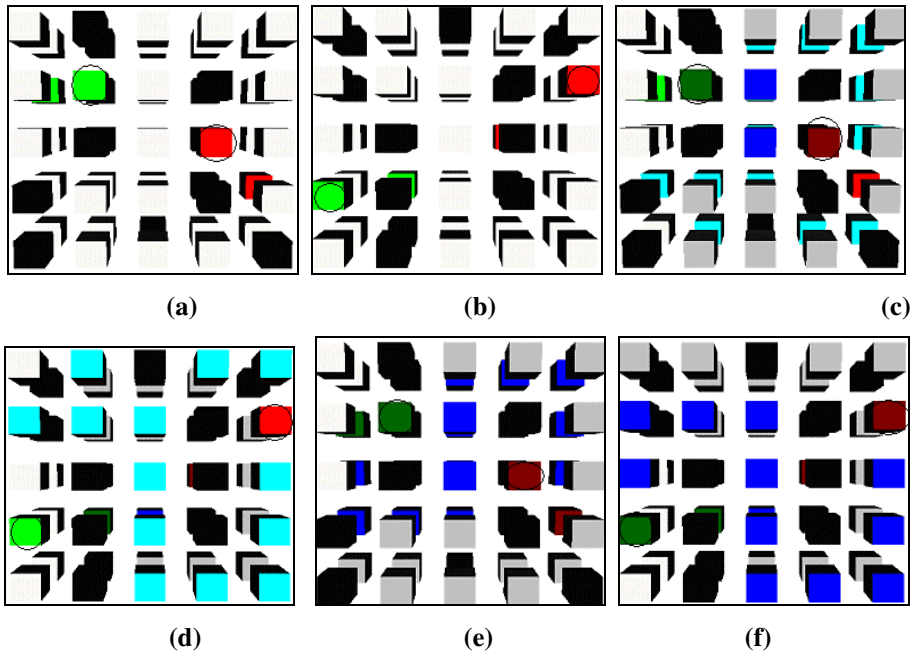


Figure 9. AODV Routing among Multiple Pairs of Senders and Receivers (a) Initial Distribution of the nodes in Plane 1 (b) In Plane 2 (c, d) State after 10 steps (e, f) Final state after 25 Steps

4. MODELING MOBILE NODES.

We implemented mobility behavior of ad-hoc nodes, allowing the mobile nodes to move in diagonal directions and to bounce back when they reach to the edges of the plane. Collision avoidance is implemented by checking the cell that is in the moving direction of the mobile node. If that cell is not empty, then reverse the direction of the mobile node. If it is empty, then check if there is any other mobile node approaching to the same cell from other directions. If there is any, then reverse the direction of the mobile node. Otherwise, assign the direction value of the mobile node to the new cell.

There are both static and mobile nodes in the model, and one or more gateways. The coupled model as presented in Figure 10 have 20x20 cells, and the surrounding 25 cells will form the neighborhood. The *mobilenode* model implements all the desired behavior: mobility, routing and coverage; however, in Figure 10 we only show partial mobility and coverage related rules (please refer to [15] for further details).

Numerical values are used to represent the model's state variables as follows: S= 0 (Empty), 1 (Moves to SE), 2 (Moves to NE), 3 (Moves to SW), 4 (Moves to NW), 5 (Static Node), 6 (Gateway), 10 (1 hop), 20 (2 hops), 30 (3 hops), 40 (4 hops), 50 (5 hops), 60 (cannot reach the gateway), 7 (within coverage).

```
[mobilenode]
type : cell      width: 20   length : 20   height : 3
delay : transportborder : nowrapped
neighbors : (-2,-2,0) (-2,-1,0) (-2,0,0) (-2,1,0)
(-2,2,0) (-1,-2,0) (-1,-1,0) (-1,0,0)(-1,1,0)(-1,2,0)
...
(0,1,-1)  (0,2,-1)  (1,-2,-1)  (1,-1,-1)  (1,0,-1)  (1,1,-1)
(1,2,-1) (2,-2,-1) (2,-1,-1) (2,0,-1) (2,1,-1) (2,2,-1)
zone : cornerUL-rule { (0,0) }
zone : cornerUR-rule { (0,19) }
zone : cornerDL-rule { (19,0) }
zone : cornerDR-rule { (19,19) }
zone : top-rule { (0,1)..(0,18) }
zone : bottom-rule { (19,1)..(19,18) }
zone : left-rule { (1,0)..(18,0) }
zone : right-rule { (1,19)..(18,19) }
[mobility_CA-rule]
rule: 1 1000 {(0,0)=4 and ((-1,-1)!=0 or (-2,-2)=1 or
        (-2,0)=3 or (0,-2)=2)}
rule: 4 1000 {(0,0)=1 and ((1,1) != 0 or (0,2)=3 or
        (2,2)=4 or (2,0)=2 )}
rule: 3 1000 {(0,0)=2 and ((-1,1) != 0 or (-2,0)=1 or
        (-2,2)=3 or (0,2)=4)}
rule: 2 1000 {(0,0)=3 and ((1,-1) != 0 or (2,-2)=2 or
        (2,0)=4 or (0,-2)=1)}
[coverage-rule]
rule: 7 1000 { statecount(6) > 0 }
rule: 7 1000 { statecount(10) > 0 }
rule: 7 1000 { statecount(20) > 0 }
rule: 7 1000 { statecount(30) > 0 }
rule: 7 1000 { statecount(40) > 0 }
rule: 0 1000 {statecount(40)=0 and statecount(30)=0 and
statecount(20)=0 and statecount(10)=0 and statecount(6)=
0 }
```

Figure 10. Implementing Mobility Model in CD++.

Nine different collision scenarios are created. 4 of them are between static and mobile nodes: 3 of them are between two mobile nodes and 2 of them are between a mobile node and a gateway. All mobile nodes change their directions at the next time unit, in order to avoid collision.

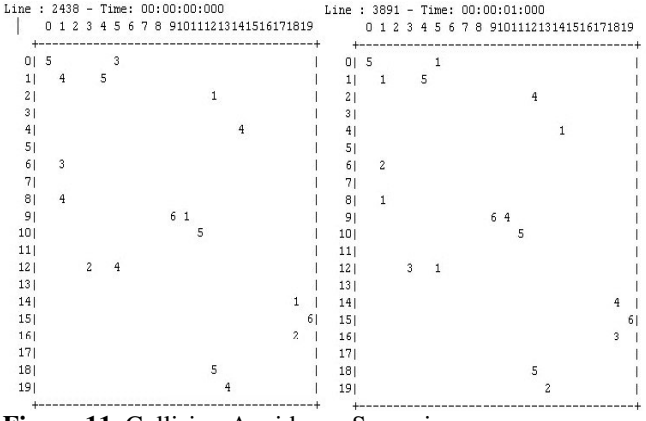


Figure 11. Collision Avoidance Scenarios.

We incorporated a hop-count sub model in which every node determines the next neighbor that can reach to the gateway with smallest number of hops. We created a topology for routing (hop count) plane as follows: there are be 2 gateways, 22 static nodes and 4 mobile nodes, each with different initial directions, as shown in Figure 12. When preparing the initial layout of nodes, loops are created between nodes and gateways, in order to observe if the particular node can choose the path with lowest number of hop counts.

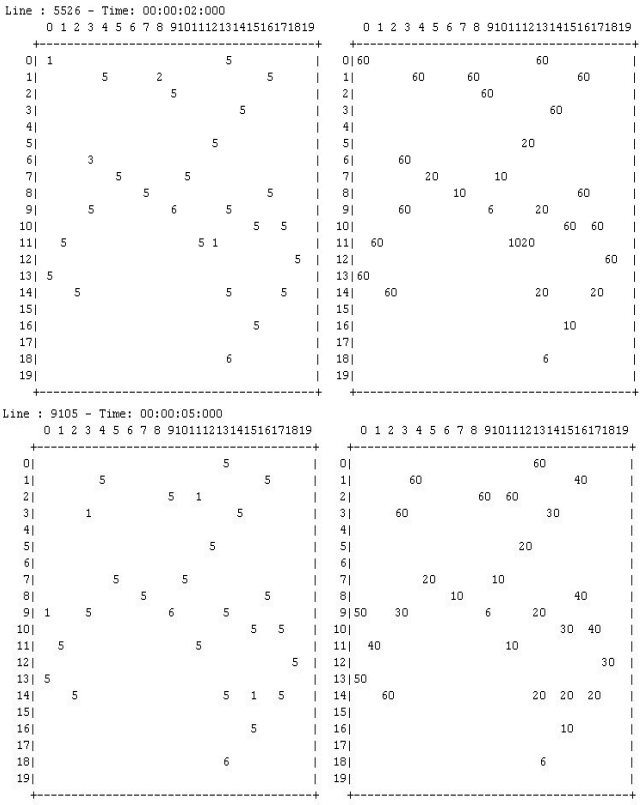


Figure 12. Mobility (left) and hop Count (right).

In Figure 12, the node located at the rightmost end of hop count plane has paths to both gateways. This node chooses the one with lowest hop count and sets its value as 30. Figure 12 shows both the mobility (to the left) and the hop count values (to the right). First, all nodes within the neighborhood of gateways set their hop count value to 10. Then, all nodes within the neighborhood with value 10 set their value to 20, etc. At the fifth iteration, all nodes 5 hops away from the gateway set their values to 50.

A node in the neighborhood of another node that can reach to the gateway, can also reach it. All cells occupied by the nodes that can reach the gateway, and all the cells in the neighborhood of these nodes will be in the coverage area. This information can be quite useful for network engineers to decide on whether or not locate new gateways in those areas. Figure 13 shows the coverage values for the hop count plane values presented in Figure 12. As it can be seen, there are two areas totally out of coverage. As service demand increases in these areas, network engineers should install more gateways in these regions.

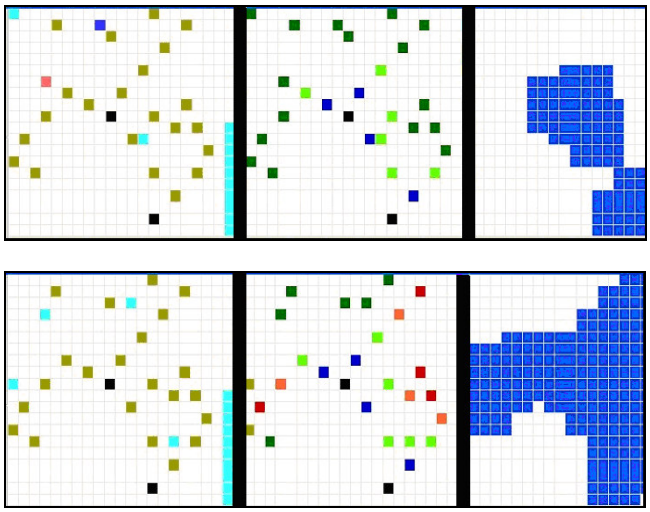


Figure 13. Mobility, hop count and coverage.

5. CONCLUSION

We presented modeling mobility, network coverage and routing in wireless ad-hoc networks using Cell-DEVS. Our research shows that routing protocols, such as AODV can be successfully mapped onto Cell-DEVS.

We extended the AODV routing algorithm to represent inter-network routing and multicasting. We not only successfully modeled AODV multicast for ad-hoc networks but also ensure the optimality in multicast tree construction (i.e., the multicast trees are constructed in such a way that ensure least duplication of data). We also modeled routing using AODV among multiple pairs of senders and receivers. Hochberger [4] showed that multiple pairs of senders/ receivers on a plane, might generate deadlocks. We have found a very simple solution to the problem by exploiting the inherent parallelism in Cell-DEVS. We introduced a new algorithm to overcome this problem, and by introducing the notion of trees model AODV multicast with a small number of states for each cell. Moreover, in our implementation, the number of states for each cell is independent of the number of nodes in the tree

We also built models of mobile nodes. Our model works correctly and transients occur when mobile nodes move in or out of a neighborhood. This is the characteristic of wireless mobile ad-hoc networks and happens in real life. As nodes can be either static or mobile, collision avoidance techniques were implemented

As a future work, the models can be easily improved by adding extra features, such as terrain or wireless media modeling. Mobility model can be further improved by including mobility in all directions. In addition, collision avoidance may further be defined by taking into account, giving right of way or waiting. Addition of such phenomena is straightforward in Cell-DEVS.

REFERENCES

[1] T.S. Rappaport. “Wireless Communications: Principles and Practice”, Second Ed., *Prentice Hall*, 2002.

[2] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. "GloMoSim: A Scalable Network Simulation Environment". *UCLA Computer Science Department Technical Report 990027*, May 1999.

[3] Kevin Fall, Kannan Varadhan, Eds. "The ns Manual (formerly ns Notes and Documentation)". http://www.isi.edu/nsnam/ns/. Accessed February 2004.

[4] C. Hochberger; R. Hoffmann. “Solving routing problems with cellular automata”, in *Proceedings of the Second Conference on Cellular Automata for Research and Industry*, Milan, Italy. 1996.

[5] R. Subrata and A.Y. Zomaya. "Evolving Cellular Automata for Location Management in Mobile Computing Networks". *IEEE Trans. on Parallel and Distributed Systems* 14:1 (Jan 2003), 13- 26.

[6] S. Wolfram "A new kind of science". *Wolfram Media, Inc*. 2002.

[7] G. Wainer, N. Giambiasi. "Timed Cell-DEVS: modelling and simulation of cell spaces ". In *Discrete Event Modeling & Simulation: Enabling Future Technologies*. Springer-Verlag. 2001.

[8] B. Zeigler, T. Kim, H. Praehofer. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". *Academic Press*. 2000.

[9] G. Wainer "CD++: a toolkit to define discrete-event models". G. Wainer. *Software, Practice and Experience*. Wiley. Vol. 32, No.3. pp. 1261-1306. November 2002.

[10] C. Y. Lee, “An algorithm for path connections and its applications”, in *IRE Transaction on Electronic Computers*, Sep. 1961. pp. 345-365.

[11] C. Perkins, E. Belding-Royer, S. Das, “Ad-hoc On Demand Distance Vector (AODV) Routing”, *IETF Network Working Group, RFC 3561*, July 2003.

[12] C. E. Perkins, P. Bhagwat, “Highly Dynamic DSDV for Mobile Computers”, *Communications Architectures, Protocols and Applications Conference Proceeding*, Pages 234-244, August 1994.

[13] D. Johnson, D. Maltz, Y. Hu, “The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)”, *IETF MANET Working Group Internet Draft*, April 2003.

[14] J. Raju and J. J. Garcia-Luna-Aceves. “A New Approach to On-demand Loop-Free Multipath Routing”. In *Proceedings of the Int'l Conf. on Computer Communications and Networks* (IC3N), pp. 522-527, 1999.

[15] U. Farooq, B. Balya, G. Wainer. " Routing in Mobile Ad-hoc Networks using Cell-DEVS". *Technical Report. Dept. of Systems and Computer Engineering. Carleton University*. 2003.