Modeling and Simulation of Routing Protocol for Mobile Ad Hoc Networks Using Colored Petri Nets

Chaoyue Xiong **Tadao Murata** **Jeffery Tsai**

Department of Computer Science
University of Illinois at Chicago,
Chicago, IL 60107 USA,
Email: {cxiong, murata, tsai}@cs.uic.edu

Abstract

In a *mobile ad hoc network* (MANET), mobile nodes directly send messages to each other via wireless transmission. A node can send a message to a destination node beyond its transmission range by using other nodes as relay points, and thus a node can function as a router. Typical applications of MANETs include defense systems such as battlefield survivability, and disaster recovery. The research on MANETs originates from part of the Advanced Research Projects Agency (ARPA) project in the 1970s. With the explosive growth of the Internet and mobile communication networks, challenging requirements have been introduced into MANETs and designing routing protocols has become more complex. For a successful application of MANETs, it is very important to ensure that a routing protocol is unambiguous, complete and functionally correct. One approach to ensuring correctness of an existing routing protocol is to create a formal model for the protocol, and analyze the model to determine if indeed the protocol provides the defined service correctly. Colored Petri Nets (CPNs) are a suitable modeling language for this purpose, as it can conveniently express non-determinism, concurrency and different levels of abstraction that are inherent in routing protocols. However, it is not easy to build a CPN model of a MANET because a node can move in and out of its transmission range and thus the MANET's topology (graph) dynamically changes. In this paper, we propose a *topology approximation* (TA) mechanism to address this problem of mobility and perform simulations of a typical routing protocol called *Ad Hoc On-Demand Distance Vector Routing* (AODV). Our simulation results show that our proposed TA mechanism can indeed mimic the dynamically changing graph (*mobility*) of a MANET.

Keywords: mobile ad hoc network (MANET), AODV routing protocol, colored Petri net

1 Introduction

Mobile ad hoc networking (Perkins 2000) is a technology for the cooperative engagement of a group of mobile nodes without requiring established infrastructure and centralized administration. In a *mobile ad hoc network* (MANET), mobile nodes directly send messages to each other via wireless transmission. A node can send a message to a destination node beyond its transmission range by using other nodes as relay points and thus a node can function as a router. This mode of communication is known as *wireless multihopping*. Compared with its hardwired counterpart, a MANET has additional constraints such as bandwidth-constrained, energy-constrained, and limited physical security. Because of these additional constraints and the dynamic topology, conventional routing protocols are not appropriate for MANETs. Many routing protocols such as *Ad Hoc On-Demand Distance Vector Routing* (AODV) (Perkins, Royer, and Das 2001), *Dynamic Source Routing* (DSR) (Johnson, and Maltz 1996), *Temporally-Ordered Routing Algorithm* (TORA) (Park, and Corson 1999), *Association Based Routing* (ABR) (Toh 1997) that are specific to mobile networking have been proposed. These protocols can be categorized as proactive, reactive, or both. Reactive protocols maintain routes only if needed, and thus have lower overhead. Proactive protocols determine routes independently of traffic pattern and have lower latency since routes are maintained at all times. Which protocol achieves a better trade-off depends on the traffic and mobility patterns of applications. Typical applications of MANETs include military affairs/events such as battlefield survivability, and disaster recovery communication (Corson 1998). The research on MANETs originates from part of the Advanced Research Projects Agency (ARPA) project in the 1970s. With the explosive growth of the Internet and mobile communication networks, challenging requirements have been added into MANETs and designing routing protocols has become more and more complex. Unexpected combinations of events can drive protocols into undesirable states, affect the protocol performance, and even lead protocols to errors. For a successful application of MANETs, it is vitally important to ensure that a routing protocol is unambiguous, complete and functionally correct. Bhargavan et al (Bhargavan, Gunter, Kim, Lee, Obradovic, Sokolsky, and Viswanathan (2002)) used an extended linear temporal logic as a formal method to verify AODV routing protocol. In this method, error traces are extracted from trace runs produced by a simulation package, and then are manually analyzed against the formula expressing the protocol's property in order to guess faults causing the errors.

As has been done in (Gordon 2001), one approach to ensuring correctness of an existing routing protocol is to create a formal model for the routing protocol, and analyze the model to determine if indeed the protocol provides the defined service correctly. By verifying the routing protocol using formal modeling, one can gain confidence in accuracy of the protocol and in using this technology, say, for military or defense systems. Colored Petri Nets (CPNs) (Jensen 1997) are a suitable modeling language for verification task, as they can conveniently express non-determinism, concurrency and different levels of abstraction that are inherent in routing protocols (Gordon 2001). CPNs have a graphical form that is based on an underlying mathematical definition. A major benefit of using CPNs is to obtain complete and unambiguous specifications in the design stage of a routing protocol, and verify if the routing protocol indeed provides the defined service correctly. CPNs provide several anal-

ysis methods, including simulation, state space analysis, sweep-line analysis and language analysis. In addition, it has a good computer tool support, Design/CPN (Jensen). Design/CPN is a suite of tools for editing, simulating and analyzing CPNs and it has a graphical editor that allows the user to create and layout the different net components. One of its nice features is that it uses pages to visually divide the model into components, enhancing its maintainability and readability without affecting the execution or analysis of the model. However, it is not easy to build a CPN model of a MANET because nodes (hosts) can move in and out of their transmission ranges and thus MANET's topology (graph) dynamically changes.

Currently, there are few mechanisms proposed to model a system's dynamic structure using CPN. Findlow and Billington (Findlow, and Billington (1990)) proposed a method to build a CPN model for a dynamic dining philosophers system whose structure changes in a predictable way. The method uses tokens in a place to describe neighboring pair of philosophers to record the system structure of philosophers' relevant position, and the system's dynamic structure is described by changes of tokens. If a new philosopher A joins the table and sits between philosophers B and C, the system's structure will definitely change in this way: Two tokens representing new neighboring information, (B, A) and (A, C), are added to the place and one old token, (B, C), is removed from the place. If a philosopher A, who sits between philosophers B and C, leaves the table, the structure will change in this way: Two corresponding tokens, (B, A) and (A, C), are removed from the place and one token, (B, C), is added to the place. Though the proposed method solves the problem of modeling dynamic structure of this system, it is difficult to capture structure changes of a MANET because the structure of a MANET changes in an *unpredictable way.* Nodes in a MANET move at will and their movements change neighboring relationship unpredictably. Thus in this paper, we propose a *topology approximation* (TA) mechanism to address the problem of mobility (dynamically changing topology) in MANETs and perform simulations of a typical routing protocol called AODV. The TA mechanism describes the aggregate behavior of nodes where their long-term average behaviors are of interest.

As for related work on simulation environments for wireless network systems, Bagrodia *et al.* at UCLA (Bagrodia) have developed a simulation package called GloMoSim. This package uses a C-based discrete-event simulation language and interactions among events are modeled by exchanging time-stamped messages. It is a powerful tool for simulating routing protocols for MANETs and comparing performances of different routing protocols. However, it is difficult to use GloMoSim to check *correctness* of routing protocols because GloMoSim is *not* a formal modeling tool.

The rest of this paper is organized as follows. Section 2 discusses the mobility problem of MANETs and the proposed topology approximation (TA) mechanism in detail. Section 3 presents our CPN models of a MANET using an AODV protocol and the TA mechanism. Simulation results are given in Section 4, and conclusions in Section 5.

2 Mobility Problem and Topology Approximation

2.1 Mobility Problem of MANET

We assume in this paper that a MANET is fully symmetric, i.e., every node has identical capabilities and responsibilities. A MANET can be represented by a graph G(V,E), where V is the set of nodes representing mobile hosts, and E is the set of edges representing links interconnecting mobile hosts. In a MANET, if node A lies within the transmission range of another node B, we say there is a link (edge) between them in the graph describing the MANET, where node A is called a neighbor of node B, and vice versa. For example, in Fig. 1, nodes A and B are neighbors since they are within the transmission ranges (shown by circles) of each other, but node C is not a neighbor of node A since node C is not within the transmission range of node A.

Every node is allowed to move at will in a MANET, and thus a link between any two nodes may disappear and reappear in an unpredictable manner. For example, in Fig. 1, when node A moves out of the transmission range of node B, the link between nodes A and B breaks as shown in the graph of Fig. 2. When node B moves back within the transmission range of node A, the link between them reappears as shown in the graph of Fig. 3. In an extreme case, the graph of a MANET remains the same even if all nodes have moved: e.g., the MANET graphs in Figs. 1 and 4 are the same, although all three nodes have moved from their original positions. Therefore, it is reasonable to disregard the exact locations or movements of nodes when we build a CPN model for a MANET. In other words, it is possible to model a MANET without information on its exact graph structure, as is done in our CPN model of a MANET in this paper.

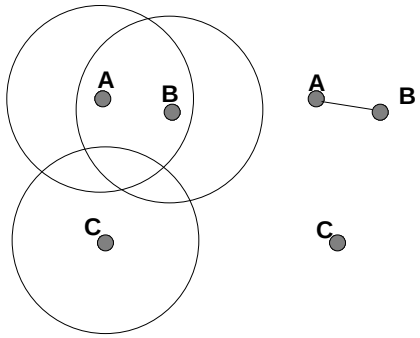

Figure 1: Nodes A and B are neighbors

Nevertheless, to verify the performance of a routing protocol, a CPN model still should simulate the mobility of a MANET, while it is difficult to build a dynamic structure in CPN modeling. In addition, to find a route from a source node to a destination, the source node in the CPN model should have its neighbors' identifications to send broadcast messages. (Note that it is not necessary for nodes in a real MANET to know their neighbors.) We propose a topology approximation (TA) mechanism to address these problems in building our CPN models for a routing protocol. The TA mechanism can simulate the mobility of a MANET without any loss of useful information that represents the semantics and performance of routing protocols used in MANETs. Even if neighbors' identifications are recorded at every node for a reactive protocol, this information on neighbors is time-varying and may be inaccurate after a certain time because nodes can move. The selection of a route across a large network or set of networks is typically performed using only partial or approximate information (Guerin, and Orda 1997, Lorenz, and Orda 1998). In the TA mechanism proposed in this paper, we use a receiving function for selecting sender's neighbors which receive messages sent by a sender node, in order to mimic the dynamic structure of a MANET.

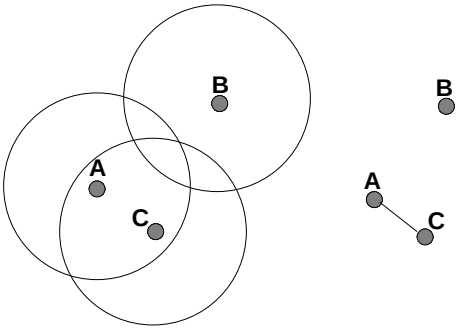

Figure 2: Nodes A moves out of the transmission range of B and moves into that of C

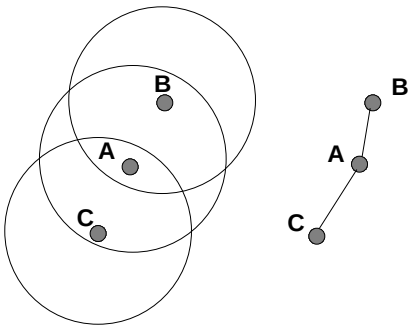

Figure 3: Node B moves back the transmission range of A

2.2 Topology Approximation

The TA mechanism uses a receiving function to describe the dynamic structure of a MANET. We use the general assumption in MANET that every node has the same distance of transmission range. Thus, if a MANET is composed of a fixed number of nodes, the average number of neighbors of a node will depend on the area it covers: If all nodes are dispersed in a small area, the average number of neighbors will be larger than that when nodes are dispersed in a wider area. Also, as shown by the simulation given in (Perkins, Royer, and Das 2001), if every node has the same responsibility and can move within a fixed area at will, the average number of neighbors for a node is nearly a constant if there are a fixed number of nodes within that area. Thus, the average number of neighbors can well represent the approximate topology information of a MANET. Based on this observation, the TA mechanism uses a receiving function to decide which nodes receive which messages. In the graph describing a MANET, the average number of neighbors can be described by the average degree of nodes in the graph. Thus by using the average degree of the graph representing a MANET, our CPN model can provide enough information for a routing protocol to find routes.

Assume every node in a MANET has the same number of neighbors which is equal to the average degree of the MANET graph. Since every node has the same capacity and responsibility, every node in a MANET has the same chance of receiving or forwarding broadcast messages. Broadcast messages are sent or forwarded by a node through radio waves. In a real situation, neighboring nodes will directly receive a broadcast message sent by this node. In CPN modeling, we create a place called *Store* to hold all the messages in transmission. After the place *Store* receives messages, a receiving function directs messages to the corresponding neighboring nodes. Clearly, a node's maximum number of neighbors is equal to $(n-1)$ where n is the number of nodes in a MANET, and all the node's neighbors will receive the broadcast message sent by the node. Thus a node should maximally send $(n-1)$ copies of a broadcast message to the place Store. For convenience in CPN modeling, we always choose $(n-1)$ as the number of copies of broadcast messages sent or forwarded by a node. But only x out of these $(n-1)$ copies of the broadcast message will be actually received by other nodes where x is the number of neighboring nodes of this node, and $(n-1-x)$ copies will be thrown away by the CPN model. The function of how many copies of broadcast messages will be received by other nodes is achieved by the receiving function in using probability bar (PB). PB is the probability of a node that will receive a broadcast message. Let d be the average degree of the MANET graph. Then on average, d nodes will receive a broadcast message among all $(n-1)(n-1)$ broadcast messages ($(n-1)$ copies of messages received by $(n-1)$ nodes). Thus we have:

$$PB = \frac{d}{(n-1)(n-1)} \quad (1)$$

Nodes in a MANET also receive unicast messages such as *route reply (RREP) message*, which are different from broadcast messages such as *route request (RREQ) message*. Thus the receiving function needs to check if a message is a unicast or broadcast message. The complete pseudo-code for the receiving function is shown in Fig. 5.

```
If ( the message is of type RREQ and is not sent by the node)
{
 Generate a random integer, K,  between 0 and 100;
   if (K<100PB)                /* PB is computed by Equation (1)*/
  the node will receive the message;
 else
  the node will not receive the message;
}
else if (the message is of type RREP and is for the node)
 the node will receive the message;
else
 the node will not receive the message;
```

Figure 5: Pseudo-code for the receiving function

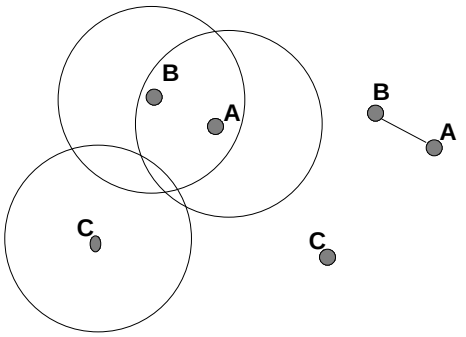

Figure 4: The topology stays the same as that in Fig. 1, although all three nodes have moved

3 CPN Model of a MANET with AODV Routing Protocol

3.1 CPN Model of a MANET

We choose a typical reactive routing protocol called AODV to demonstrate the effectiveness of the TA mechanism. The hierarchy page (Jensen) of our CPN model is depicted in Fig. 6, and shows the overall organization of the modules comprising the CPN

model of our MANET. Page Prime#1 is the top level of our CPN model for a MANET, which consists of five nodes in our example. The second level (page Node#2) is our node template for implementing AODV routing protocol. By instantiating this node template, we can create a CPN model for a MANET composed of nodes as many as allowed in the computer using Design/CPN tool. The third level has three pages (page RREQInit#3, page RREQProcess#4 and page RREPProcess#5), each of which is named by an AODV state.

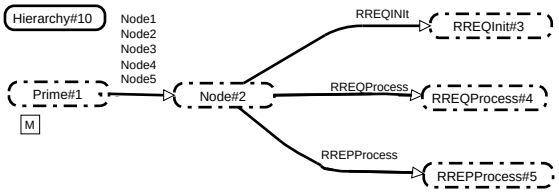

Figure 6: The hierarchy page

The function of the node template is similar to that of *class* in an object-oriented programming language. If we want to create a node instance in a MANET, we can simply instantiate the node template. Like an *object* in an object-oriented programming language, every instantiated node has its own local variables and provides interfaces to the outside world. In our CPN modeling, an instantiated node is represented by a substitution transition, and sockets provided by Design/CPN are its interfaces to the outside world. If more nodes are to be added to this MANET model, we simply add substitution transitions to the model. Fig. 7 shows a node instance in a MANET, which is a part of the CPN model in the top level.

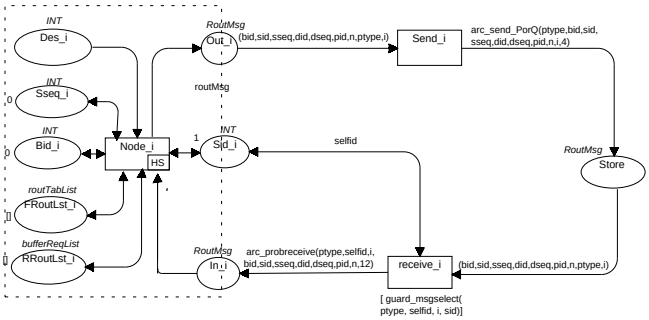

Figure 7: Node instance of a MANET at the top level showing nodes' local variables, and connections with the place *Store*

As is shown in Fig. 7, Places *Dest-i, Sseq-i, Bid-i, FroutLst-i,* and *RroutLst-i* are the local variables of *Node-i.* The place *Dest-i* describes the destination node ID(s) to which packets are sent. A routing protocol can get destination ID(s) from its upper level protocol. The place *Sseq-i* represents the sequence number of the node. The place *Bid-i* is the broadcast ID of the node. To easily analyze simulation results, we separate forward paths from reverse paths using different places to represent them. The place *FroutLst-i* represents forward paths information contained in RREP messages received by the node. The place *RroutLst-i* represents the reverse paths information contained in RREQ messages received by the node. Places *Sid-i, In-i,* and *Out-i* are interfaces to the outside world. The place *Sid-i* is the node's ID, which is equal to *i* for node *i* in our CPN model. The place *Out-i* holds all the messages that the node wants to send or forward, and the place *In-i* contains all the incoming messages that the node receives from the place *Store.*

The arc inscription from the transition *Receive* to the place *In-i, arc_probReceive(ptype, selfid, i, bid, sid, sseq, did, dseq, pid, n, 12),* is the receiving function, which has the same meaning as described in the pseudo-code shown in Fig. 5. It directs all RREP messages routed for this node and selectively directs RREQ messages based on PB ($PB = 12$ in this example) to the receiving buffer, which is represented by the place *In-i.* For details on the implementation of *arc_probReceive(),* see lines 119 - 125 in the appendix. The function *arc_send_PorQ (ptype, bid, sid, sseq, did, dseq, pid, n, i, MAXNeighbor)* is the arc inscription from the transition *Send-i* to the place *Store.* It is used to control the number of copies of a message sent by the node. If the message is of RREP type, one copy of this message will be sent. If it is of RREQ type, $(n-1)$ copies of this message will be sent ($n = 5$ in this example). For details on implementation of *arc_send_PorQ()*, see lines 119 - 125 in the appendix. The function *guard_Msgselect(ptype, selfid, i, sid)* (line 115-118 in the appendix) is the guard of the transition *Receive-i.* It returns *true* if there is an RREP message routed for this node in the place *Store* or there is an RREQ message sent by other nodes in the place *Store.*

3.2 CPN Model of AODV Routing Protocol

As mentioned in the above, the second level is the node template implementing AODV protocol. AODV is a typical reactive routing protocol. It creates routes in an on-demand fashion and builds routes using a route request/route reply query cycle. When a source node needs to send data packets to a destination node, it first checks its route table to see if it has an unexpired (i.e., existing and valid) route to the destination. If it does, it sends data packets using this route immediately. Otherwise, it broadcasts a *RREQ message.* This RREQ message contains a sequence of numbers: source ID, broadcast ID, sequence number of source, sequence number of destination, previous ID, hop count and destination ID. The purpose of a destination sequence number is to prevent a loop in route discovery process. Each node maintains its own sequence number which will be increased by one whenever any link between the node and its neighbor breaks. The information of link breakage can be obtained from a lower level of the routing protocol, which is beyond the scope of this paper. The hop count determines the current distance from the node to the source node initiating the *RREQ message.* The initial RREQ has this field of the hop count which is set to zero and is increased by one at every subsequent node . A node, upon receiving this broadcast message, checks if it has received this RREQ message before. If yes, the node simply discards the message. If no, the node processes this message as follows: If the node is the destination or has a valid route to the destination, it unicasts the RREP message to the source along the path from which the RREQ message has come. Otherwise, it forwards this RREQ message to all of its neighbors. Based on RREQ and RREP messages, nodes capture the route information. When an intermediate node receives a RREP message, it sets up a forward path entry to the destination in its route table. When an intermediate node receives a fresh RREQ message, the node sets up a reverse route entry to the source node in its route table. Except for their origins, there is no distinction in functionalities for forward and reverse path entries. AODV protocol has four states:

- *RoutCheck:* Check the route table to see if the

node has an existing and valid path to the destination.

- *RREQInit:* Initiate RREQ message when necessary.
- *RREQProcess:* Process the incoming RREQ message and output proper results.
- *RREPProcess:* Process the incoming RREP message and output proper results.

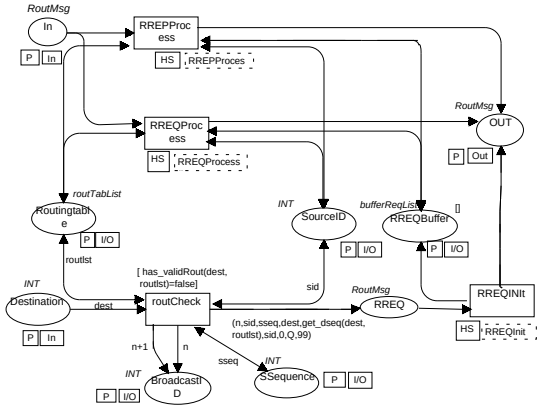

Figure 8: CPN model for a node template (page *Node#1*)

Fig. 8 shows the template of a CPN node for implementing AODV routing protocol. It is represented by the substitution transition, *Node-i*, in Fig. 7. All the incoming messages to the node are directed to the place *In*, and all the outgoing messages from the node are directed to the place *Out*. The node template has three substitution transitions named *RREQInit*, *RREQProcess* and *RREPProcess* corresponding to three of AODV states. If a node wants to send a packet, it first enters *RoutCheck* state. If there is no route in its route table, the node enters *RREQInit* state to initiate route discovery mechanism and broadcasts a RREQ message. If a node receives a RREQ/RREP message, it enters *RREQProcess/RREPProcess* state. We have modeled the AODV routing protocol with the following features (services):

- *Node generating a broadcast message:* If a node wants to send a packet to a destination node (we can get this information from the upper layer of routing protocol), a token representing the destination ID is put into the place *Destination* in Fig. 8, thus initiating the working process of the model. Immediately, the node enters *RoutCheck* state and checks its route table, which is represented by the place *RoutingTable*, to see if it has a valid route to the destination by the guard *has_validRout(dest, routlst)* of the transition *routCheck* in Fig. 8. For detailed information on the implementation of *has_validRout(dest, routlst)*, see lines 49-52 in the appendix. If *has_validRout(dest, routlst)* returns true, which means that the node has a valid route to the destination, the node immediately sends the packet and nothing needs to be done for our routing protocol, and if *has_validRout(dest, routlst)* returns false, the node generates a *RREQ message* in the place *RREQ* and goes into *RREQInit* state, which is represented by the substitution transition *RREQInit* in Fig. 8. For detailed information on the implementation of subpage *RREQInit*, the

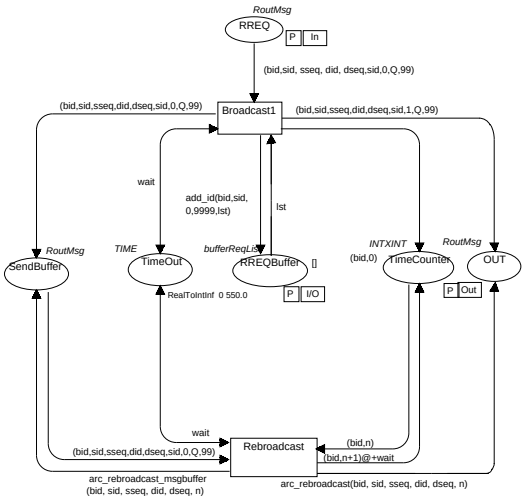

Figure 9: *RREQInit* subpage

reader is referred to the CPN model shown in Fig. 9, which is explained below:

The main functions of subpage *RREQInit* is to direct the previously generated *RREQ* message to the place *Out* and rebroadcast the RREQ message if necessary. The node directs the RREQ message by the transition *Broadcast1* in Fig.9. At the same time, the node puts the RREQ message into the retransmission buffer, which is represented by the place *SendBuffer*, and sets a timer (represented by the place *TimeOut)* for this broadcast message. If the node doesn't get any feedback from other nodes after the timer is off, it rebroadcasts the RREQ message by the transition *Rebroadcast* in Fig. 9. The maximum number of rebroadcasts for a specific RREQ message is set to three times. For more detailed information about rebroadcast, see the function, *arc_rebroadcast()* (lines 107-109 in the appendix) and *arc_rebroadcast_msgbuffer()*(lines 110-112 in the appendix).

- *Node unicasting a RREP message to its previous node:* If a node receives a RREP message, it enters *RREPProcess* state, which is represented by the substitution transition *RREPProcess* in Fig. 8. Fig 10 shows detailed information on the implementation of subpage *RREPProcess*, which is explained below.

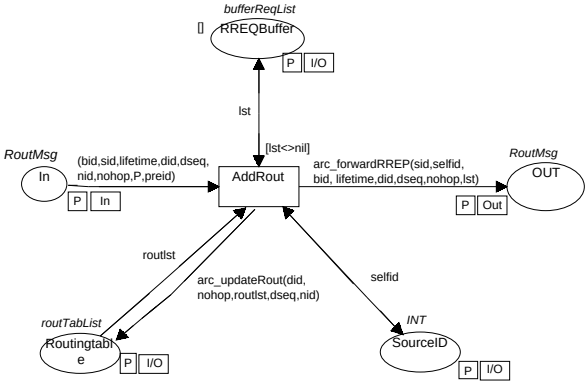

Figure 10: *RREPProcess* subpage

The main function of subpage *RREPProcess* is to update the route table and forward the received RREP message if necessary. To

achieve these functionalities, two functions, *arc_updateRout(did, nohop, routlst, dseq, nid)* and *arc_forwardRREP(sid, selfid, bid, lifetime, did, dseq, nohop, lst)* play important roles on this subpage. The function *arc_updateRout()* is the arc inscription from the transition *AddRout* to the place *RoutingTable*. It first checks the ID of the node to see if it is the source recipient for this RREP message. If yes, it simply adds this discovered route contained in the RREP message to its route table; and if no, it updates its own route table if this incoming new route message has less hops than the old one. For more detailed information on the implementation of this function, see lines 84-91 in the appendix. The function *arc_forwardRREP()* is the arc inscription from the transition *AddRout* to the place *Out*. This function first checks if the node is the source recipient for this RREP message. If yes, it does nothing, if no, it finds the ID of the next node which should receive this RREP message and directs this RREP message to the place Out. For detailed information on the implementation of the function, see lines 92-95 in the appendix.

- *Node broadcasting an incoming RREQ message to its neighboring nodes:* If a node receives a RREQ message, it enters *RREQProcess* state, which is represented by the substitution transition *RREQProcess* in Fig. 8. Fig. 11 shows detailed information on the implementation of subpage *RREQProcess*, which is explained below.

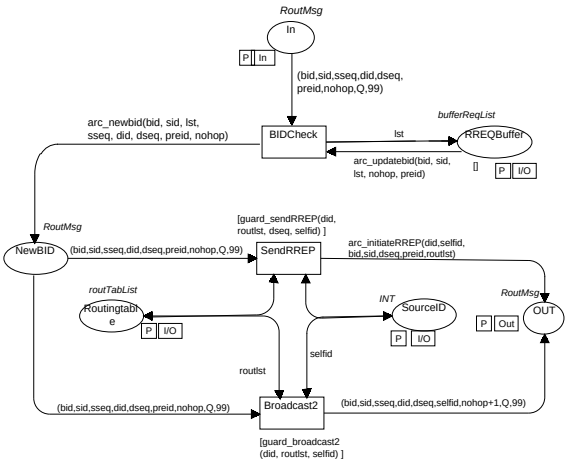

Figure 11: *RREQProcess* subpage

The main function of subpage *RREQProcess* is to initiate the corresponding *RREP* message if possible or forward the RREQ message if necessary. Upon receiving a RREQ message (represented by a token in the place *InRREQ)*, the transition *BIDCheck* is enabled, thus entering the *RREQProcess* state for the routing protocol. If the received RREQ message is fresh (received for the first time), it is directed to the place *NewBID*. Otherwise, it is simply discarded. This functionality is achieved by the arc inscription *arc_newbid(bid, sid, lst, sseq, did, dseq, pried, nohop)*. For more detailed information on the implementation of this function, see lines 104-106 in the appendix. After the place *NewBID* gets the token for a fresh RREQ message, either the transition *SendRREP* or the transition *Broadcast2* is enabled. If the node is the destination or has a valid route to the destination, guard *guard_sendRREP()* returns true. Otherwise, the *guard guard_broadcast2()* returns true. For detailed information on the implementation of these two functions, see lines 78-83 in the appendix. If it's *SendRREP's* turn, a RREP message is initiated by the arc inscription *arc_initiateRREP(did, selfid, bid, sid, dseq, pried, routlst)* and directed to the place *Out*. Otherwise, transition *Broadcast2* increases the number of hops by one and directs the received RREQ message to the place *Out*.

3.3 Global Declarations

The global declarations comprise the color sets and variables for the AODV protocol. The color sets show data structures storing different routing information such as route entry, RREQ and RREP messages.

The color set *RoutMsg* (line 14 in Fig. 12) defines the routing message. It contains the following fields:

- *BID:* Broadcast ID.
- *SID:* Source node that initiates RREQ message. *(BID, SID)* uniquely specifies a RREQ message.
- *SSEQ:* Most recent recorded sequence number for the source.
- *DID:* Destination node.
- *DSEQ:* Most recent recorded sequence number for destination *DID*.
- *PID:* The immediately previous node from which the node receives the message.
- *INT:* Distance from the sender that initiates the message. The sender is *DID* for RREP message and *SID* for RREQ message.
- *TYPE:* Type of the message. Q means RREQ, and P means RREP.
- *DUMMY:* Only useful for RREP message: Next node to which the message is sent. It is set to 99 if the message is of type RREQ.

The color set *BufferREQ* (line 15 in Fig. 12) defines buffered identification of RREQ message that a node has processed. The color set *RoutTable* (line 20 in Fig. 12) defines the structure of a route entry in route table. It contains the following fields:

- *DID:*Destination of the route.
- *DSEQ:* Most recent recorded sequence number for destination *DID*.
- *INT:* Distance from the destination *DID*, measured in the number of hops that need to be traversed to reach *DID*.
- *NID:* Next node on the route to *DID*.
- *LIFETIME:* Remaining time before the route expires.

Since the focus of this paper is our TA mechanism, we don't implement fields *DSEQ* and *LIFETIME*. But for consistence and convenience in future work, we keep these two fields and set them to be 0.

4 Simulation Experiment

For our illustrative purpose, we use a MANET example consisting of 5 nodes. Thus, when a node sends a broadcast message, it will send 4 copies of the message to the place called *Store*. In our simulation model, we chose two as the average degree of the MANET graph,

```
1  color INT = int timed;
2  color LIFETIME, DSEQ, SSEQ, DID, SID, BID, NID,
       PID, DEST, DUMMY = INT;
3  color TYPE = with P | Q;

14 color RoutMsg = product
BID*SID*SSEQ*DID*DSEQ*PID*INT*TYPE*DUMMY;
15 color BufferREQ = product BID*SID*INT*PID;
16 color bufferReqList = list BufferREQ;
17 var lst:bufferReqList;
18 color INTXINT = product INT*INT;
19 color INTXINTXINT = product INT*INT*INT;
20 color RoutTable = product DID*DSEQ*INT*NID*LIFETIME;
21 color routTabList = list RoutTable;

23 var routlst:routTabList;
24 var wait:TIME;
25  var bid,sid,sseq,did,dseq,selfid,hopNo, preid:INT;
26 var bbid,ssid,ddid,ddseq: INT;
27 var lifetime,nohop,dest,n,nexthop:INT;
28 var ptype:TYPE;
29 var nid,pid:NID;
30 var msg, routMsg:RoutMsg;
31 var i:INT;
```

Figure 12: Color sets and variables

i.e., the average number of neighbors of a node is two. Thus, the probability bar PBcan be found from

$$PB = \frac{d}{(n-1)^2} = \frac{2}{(5-1)^2} = \frac{2}{16} = 0.125$$

or

$$100PB = 100 \times 0.125 = 12.5$$

We choose $100PB$ to be 12, an integer, for convenience using Design/CPN.

The main function of a routing protocol is to discover routes. Thus, we are only interested in routes found at the end of each simulation by the CPN model. For detailed explanation on the structure of a route entry in a route table, please refer to Section 3.3. Here we give an example to explain the relation between a route and its corresponding graph. For example, (4, 0, 4, 2, 0) is a route entry in node 1's route table. Then $DID = 4$, $DSEQ = 0$, $INT = 4$, $NID = 2$, and $LIFETIME = 0$, which mean that node 1 has a route with four hops to node 4, and node 2 is the next node on this route, as shown in Fig. 13.

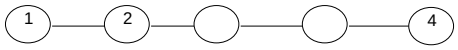

Figure 13: Example of route entry

To check if the TA mechanism can really mimic the mobility (dynamically changing graph) of a MANET, we use the same initial conditions for the following simulation experiments: *There is no existing route kept at any node, and node 1 wants to send packets to node 2.* Thus, in order to find a valid route to node 2, node 1 has to initiate the route discovery mechanism.

Node id	Forward path	Reverse path
1	{(2 0 1 2 0)}	{}
2	{}	{(1 0 1 1 0)}
3	{}	{(1 0 2 4 0)}
4	{}	{(1 0 1 1 0)}
5	{}	{(1 0 1 1 0)}

Table 1: Route tables resulting from simulation #1

From these initial conditions, we performed three simulations. Route tables resulting from simulation #1 are given in Table 1, where row i is corresponding to the route table of node i . In this table, row1 shows route table of node 1, and (2 0 1 2 0) indicates a path from node 1 to node 2 with one hop, row2- (1 0 1 1 0) a path from node 2 to node 1 with one hop, row3- (1 0 2 4 0) a path from node 3 to node 1 with two hops and node 4 is an immediate next node of node 3, row4- and row5-(1 0 1 1 0) two paths from nodes 4 and 5 to node 1 with one hop. Thus the MANET graph found from simulation #1 is the one shown in Fig. 14.

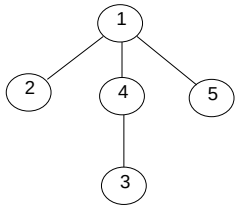

Figure 14: The MANET graph showing the routes found by simulation#1

Node id	Forward path	Reverse path
1	{(2 0 1 2 0)}	{}
2	{}	{1 0 1 1 0}
3	{}	{}
4	{}	{}
5	{}	{(1 0 1 1 0)}

Table 2: Route tables resulting from simulation #2

Node id	Initial route table	Destination node id
1	{(2 0 2 3 0)}	{}
2	{}	{(1 0 2 3 0)}
3	{(2 0 1 2 0)}	{(1 0 1 1 0)}
4	{}	{}
5	{}	{(1 0 2 3 0)}

Table 3: Route tables resulting from simulation #3

The results of simulations #2 and #3 are shown in Tables 2 and 3, in Figs 15 and 16, respectively. In every simulation above, the CPN model finds expected routes (those from node 1 to node 2) as specified in the initial conditions and the formal model of AODV shown in Figs. 8, through 11. This simulation experiment confirms that the AODV protocol provides correctly its features (services) defined in Section 3.2. And the simulation results also show that our proposed TA mechanism can indeed mimic the dynamically changing graph (mobility) of a MANET without knowing its actual topology (graph), since the above three simulations result in different MANET graphs from the same initial condition.

5 Conclusions and Future Work

To the best of our knowledge, this paper presents for the first time a CPN model and simulation of a MANET. Routing protocols used for MANETs are not novel in themselves and there have been few formal methods presently available for designing and testing protocols for MANET's due to their dynamically changing graph structures. This paper has proposed a method called topology approximation (TA) mechanism to address this problem of mobility, and then used this mechanism to build a CPN model of a MANET with an AODV routing protocol. The simulation results show that the TA mechanism can sim-

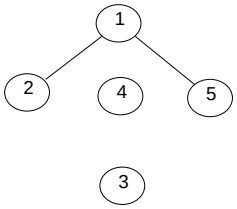

Figure 15: The MANET graph showing the routes found by simulation#2

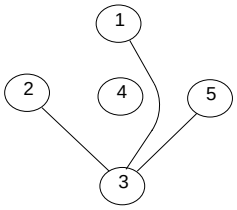

Figure 16: The MANET graph showing the routes found by simulation#3

ulate the mobility (dynamically changing graphs) of a MANET without knowing its actual topology. The simulation results also show that we can gain great insight into MANET routing protocol design by using CPN models and Design/CPN as a simulation tool.

Latency or delay is a very important consideration in deciding routes in MANETs and can be easily described by Fuzzy-Timing High-Level Petri Nets (FTHNs) (Zhou, Murata, and DeFanti 2000, Murata, Suzuki, and Shatz 1999). Most of existing routing protocols select routes by choosing a route with the least hop counts. As a future work, we plan to use FTHNs to improve the present CPN model, and then design a new routing protocol which can select a route with the least fuzzy latency. We also plan to investigate how formal analysis methods such as state space analysis can be performed on CPN models of a MANET.

Acknowledgment: This work was supported by the National Science Foundation under research grant: CCR-9988326.

References

Bagrodia, R. (1999), GloMoSim Online. Parallel Computing Laboratory, Department of Computer Science, University of California at Los Angeles. [online]. Available: http://pcl.cs.ucla.edu/projects/glomosim/.

Bhargavan, K., Gunter, C.A., Kim, M., Lee, I., Obradovic, D., Sokolsky, O. and Viswanathan, M. (2002), Verisim: Formal Analysis of network simulations, *IEEE Transactions on Software enginerring*, **28**(2) 129-145

Corson, M.S. (1998), Thoughts on the future of mobile ad hoc networking, *in*U.S. Army Research Office Strategy Planning Workshop.

Findlow, G. & Billington, J. (1990), High-Level nets for dynamic dining philosophers systems, *in* M.Z. Kwiatkowska, M.W. Shields, & R.M. Thomas, eds, Semantics for Concurrency, Proc. International BCS-FACS Workshop, 23-25 July 1990', University of Leicester, UK, pp. 185-203.

Gordon, S.D. (2001), Verification of the WAP Transaction Layer using coloured Petri nets. Ph.D., University of South Australia, Adelaide, South Australia.

Guerin, R. and Orda, A. (1997), QoS-based Routing in networks with inaccurate information, *in* Theory and algorithms. *Proc. IEEE INFOCOM'97.* pp. 75-83.

Jensen, K.: Design/CPN Online. Dept. Computer Science, Univ. Aarhus, Denmark. [online]. Available: http://www.daimi.au.dk/designCPN/.

Jensen, K. (1997), *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use*, Monographs in Theoretical Computer Science, Springer-Verlag.

Johnson, D.B. and Maltz, D.A. (1996), *Dynamic Source Routing in Ad Hoc Wireless Networks, Mobile Computing*, Kluwer Academic Publishers.

Lorenz, D.H. and Orda, A. (1998), QoS routing in networks with uncertain parameters', *Journal of IEEE/ACM Transactions on Networking* **6**(6) 768-778.

McDonald, B. and Znati, T. (1999), A Path Availability Model for Wireless Ad-Hoc Networks, *in* Proc. of IEEE WCNC 99', New Orleans, LA, USA.

Murata, T., Suzuki, T. and Shatz, S. (1999), Fuzzy-Timing High-Level Petri Nets (FTHNs) for Time-Critical Systems', *Fuzziness in Petri Nets***22** 88-114.

Park, V. and Corson, S. (1999), The Temporally-Ordered Routing Protocol (TORA) Specification. Internet Draft.

Perkins, C.E., Royer, E.M. and Das, S. (2001), Ad Hoc On Demand Distance Vector (AODV) Routing. IETF Internet draft, draft-ietf-manet-aodv-09.txt.

Perkins, C.E. and Royer, E.M. (1999), Ad hoc On-Demand Distance Vector Routing, *in* Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications', New Orleans, LA, USA, pp. 90-100.

Perkins, C.E. (2000), *Ad Hoc Networking*, Reading, MA, Addison-Wesley.

Toh, C.-K. (1997), Associativity-based routing for ad-hoc mobile networks', *Journal of Wireless Personal Commun.*, **4** 103-139.

Zhou, Y., Murata, T. and Defanti, T. (2000), Modeling and Performance Analysis Using Extended Fuzzy-Timing Petri Nets for Networked Virtual Environments', *Journal of IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, **30**(5) 737-756

Appendix

```
39 fun has_id(bid,sid,[])=false
40   | has_id(bid,sid,(b,s,h,p)::rest)=
41   if (bid=b andalso sid=s) then true
42   else has_id(bid,sid,rest);

43 fun get_Pid (bid,sid,(b,s,h,p)::rest)=
44   if (bid=b andalso sid=s) then p
45   else get_Pid(bid,sid,rest);

46 fun add_id(bid,sid,hopno,pid,[])= [(bid,sid,hopno,pid)]
47   | add_id(bid,sid,hopno,pid,lst) =
48     (bid,sid,hopno,pid)::lst;

49 fun has_validRout(did,[]) = false
```

```
50    | has_validRout(did,(ddid,dseq,i,nid,lifetime)::rest) =
51      if (did=ddid andalso lifetime<100) then true
52      else has_validRout(did,rest);

53 fun has_betterRout(did,i,[])= 0
54    | has_betterRout(did,i,(ddid,dseq,ii,nid,lifetime)::rest)=
55      if (did=ddid andalso (i<ii orelse lifetime>100)) then 1
56      else
57        if (did=ddid andalso i≥ii andalso lifetime<100) then 2
58        else has_betterRout(did,i,rest);

59 fun add_rout(did,dseq,i,nid,lifetime,lst) =
60    (did,dseq,i,nid,0)::lst;

61 fun remove_rout(did,[]) = []
62    | remove_rout(did, (ddid,ddseq,ii,nnid,llifetime)::rest) =
63        if(did=ddid) then rest
64        else (ddid,ddseq,ii,nnid,llifetime)::remove_rout(did,rest);

65 fun remove_add_rout(did,dseq,i,nid,lifetime,lst) = (
66    add_rout(did,dseq,i,nid,lifetime,remove_rout(did,lst))
67    );

68 fun get_lifetime(did,(ddid,dseq,ii,nid,lifetime)::rest) =
69    if (did=ddid) then lifetime
70    else get_lifetime(did,rest);

71  fun get_hop(did,(ddid,dseq,ii,nid,lifetime)::rest)=
72    if (did=ddid) then ii
73    else get_hop(did,rest);

74 fun get_dseq(did,[]) = 0
75    | get_dseq(did,(ddid,dseq,ii,nid,lifetime)::rest) =
76        if (did=ddid) then dseq
77        else get_dseq(did,rest);

78 fun guard_sendRREP(did,routlst,dseq,selfid) =
79    if ((dseq≤get_dseq(did,routlst) andalso
has_validRout(did,routlst))
80    orelse (did=selfid) ) then true
81    else false;

82 fun guard_broadcast2(did,routlst,selfid)=
83    (not (has_validRout(did,routlst))) andalso (did≠=selfid);

84 fun arc_updateRout(did,nohop,routlst,dseq,nid)=
85    case has_betterRout(did,nohop,routlst) of
86    1⇒
87        remove_add_rout(did,dseq,nohop,nid,0,routlst)
88    | 0⇒
89        add_rout(did,dseq,nohop,nid,0,routlst)
90    | 2⇒
91        routlst;

92 fun arc_forwardRREP(sid,selfid,bid,lifetime,did,dseq,nohop,lst)
=
93    if (sid≠selfid) then 1`(bid,sid,lifetime,did,
94        dseq,selfid,nohop+1,P,get_Pid(bid,sid,lst))
95    else empty;

96 fun arc_initiateRREP(did,selfid,bid,sid,dseq,preid,routlst)=
97    if (did=selfid) then
98        1`(bid,sid,0,did,dseq,selfid,1,P,preid)
99    else
100       1`(bid,sid,0,did,dseq,selfid,1+get_hop(did,routlst),P,preid);

101 fun arc_updatebid(bid,sid,lst,nohop,preid)=
102    if has_id(bid,sid,lst) then lst
103    else add_id(bid,sid,nohop,preid,lst);

104 fun arc_newbid(bid,sid,lst,sseq,did,dseq,preid,nohop)=
105    if has_id(bid,sid,lst) then empty
106    else 1`(bid,sid,sseq,did,dseq,preid,nohop,Q,99);

107 fun arc_rebroadcast(bid,sid,sseq,did,dseq,n)=
108    if (n<3) then 1`(bid,sid,sseq,did,dseq,sid,1,Q,99)
109    else empty;

110 fun arc_rebroadcast_msgbuffer(bid,sid,sseq,did,dseq,n)=
111    if (n<3) then 1`(bid,sid,sseq,did,dseq,sid,0,Q,99)
112    else empty;

113(* function on the prime page*)
114
115   fun guard_Msgselect(ptype,selfid,i,sid)=
116    (ptype=P andalso selfid=i)
117    orelse
118    (ptype=Q andalso sid≠selfid);

119 fun arc_probreceive(ptype,selfid,i,bid,sid,sseq,did,dseq,pid,n,PB)
=
120    if (ptype=P andalso selfid=i) then
121        1`(bid,sid,sseq,did,dseq,pid,n,ptype,i)
122    else if (ptype=Q andalso
123        ( sid≠selfid andalso CPN'randint(0,100)<PB)
124        then 1`(bid,sid,sseq,did,dseq,pid,n,ptype,i)
125        else empty;

126  fun arc_send_PorQ(ptype,bid,sid,sseq,did,dseq,pid,n,i,MAXNeighbor)
=
127    if ( ptype=Q)
128        then MAXNeighbor`(bid,sid,sseq,did,dseq,pid,n,ptype,i)
129    else 1`(bid,sid,sseq,did,dseq,pid,n,ptype,i);
```