Validation of Routing Protocol for Mobile Ad Hoc Networks using Colored Petri Nets

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology
in
Computer Science and Engineering

By

Piyush Prasad (10606045)
Baltej Singh (10606014)
Asish Kumar Sahoo (10606048)

Department of Computer Science and Engineering
National Institute of Technology
Rourkela

2009

Validation of Routing Protocol for Mobile Ad Hoc Networks using Colored Petri Nets

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology
in
Computer Science and Engineering

By

Piyush Prasad (10606045)
Baltej Singh (10606014)
Asish Kumar Sahoo (10606048)

Under the Guidance Of
Prof. S. K. Rath

Department of Computer Science and Engineering
National Institute of Technology
Rourkela
2009

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the thesis entitled, "**Validation of Routing Protocol for Mobile Ad Hoc Networks using Colored Petri Nets**" submitted by Sri Piyush Prasad, Sri Baltej Singh and Sri Asish Kumar sahoo in partial fulfillments for the requirements for the award of Bachelor of Technology Degree in Computer Science and Engineering at National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Date:

Prof. S.K. Rath
Dept .of Computer Science and Engineering
National Institute of Technology
Rourkela – 769008

Acknowledgment

We express our sincere gratitude to Prof. S.K. Rath, Department of Computer Science and Engineering, National Institute of Technology, Rourkela, for his valuable guidance and timely suggestions during the entire duration of our project work, without which this work would not have been possible.

We would also like to convey our deep regards to our respected HOD. Prof. B .Majhi ,Prof. S. Chinara and all other faculty members and staff of Department of Computer Science and Engineering, NIT Rourkela, who have bestowed their great effort and guidance at appropriate times without which it would have been very difficult on our part to finish this project work.

Piyush Prasad
Roll no: 10606045

Baltej Singh
Roll no:10606014

Asish Kumar Sahoo
Roll no: 10606048

Date:-

Contents

Chapter 1 - Introduction to Petrinet

1.1 Predicate/Transition Nets
1.2 Firing of a transition
1.3 Transition rule

Chapter 2 - Colored Petrinet

2.1Introduction
2.2The CPN modeling language

Chapter 3 - Sliding Window Implementation

Chapter 4 - Mobile Ad-hoc Network

4.1MANETs have several salient characteristics
4.2Mobility Problem
4.3Topology Approximation
4.4CPN Model of AODV

Chapter 5 – Dynamic Source Routing

5.1Introduction
5.2Assumptions
5.3DSR Protocol Description
5.4Basic DSR Route Discovery

Chapter 6- Comparison Study & Conclusion

Chapter 7- Bibliography

Appendix

ABSTRACT

In a Mobile Ad Hoc Network (MANET), mobile nodes directly send messages to each other via other nodes in a wireless environment. 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. 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 needed 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 dynamically changes. So a topology approximation (TA) mechanism has been proposed to address this problem of mobility and perform simulations of routing protocol called Ad Hoc On demand Distance Vector Routing (AODV) and Distance Source Routing(DSR) and to perform comparison based on the simulation results.

Overview

Chapter 1

This chapter describes the details of the petrinet. Here discussion about the types of pertinets has been done, along with the ways to fire a transition and what the various transition rules are. Further petrinets have been categorized according to firing conditions i.e. Original Petri nets and Weighted Petri nets.

Chapter 2

This chapter describes Colored Petrinet . Here discussion about how to use Colored petrinet tool has been done, along with what CPN Modeling Language is and how to fire transitions using this language. All these concepts have been described by using a model of CPN and also we have described all the syntax of the language.

Chapter 3

This chapter discusses about the implementation of Sliding Window using Colored Petrinet tool. It also describes a self designed model for Sliding Window and explains the working of the model.

Chapter 4

This chapter discusses the Mobile Ad-hoc network. Here some of the Applications of MANET along with several salient characteristics of MANET have been discussed. Along with various issues like mobility problem, complexity in a Mobile Ad-hoc network have been explained. In order to solve this mobility problem we have followed methodology called *Topology Approximation (TA)* Algorithm [1]. Then the CPN model for AODV(ad-hoc on demand distance vector) routing protocol [1] is discussed in order to do a comparison study between AODV and DSR(Dynamic source routing) protocol.

Chapter 5

This chapter discusses the Dynamic Source Routing protocol (DSR).This chapter discusses a self designed model for DSR using CPN tool, further discussions are focused on some of the characteristics of DSR. The basic Route Discovery mechanism and Route Maintenance mechanisms have been explained. The detailed

working steps of the self designed DSR model have been discussed with snapshots from the running model .

Chapter 6

This chapter covers the performance comparison between DSR and AODV routing protocol. Here the comparison is made using graphs between the no. of simulation and the time taken for network discovery . It is then followed by the conclusion related to the comparative study between DSR and AODV.

The appendix consists of the code for the implementation of DSR.

List of Figures

Fig. No.	Title of Fig.	Page No.
1	Basic Petri net	12
1.1(a) and 1.1(b)	Petri net before a transition fires	14
1.2(a) and 1.2(b)	Petri net after a transition fires	15
2	Net structure of a CPN	21
3	Sliding Window	24
3.1	Snapshot of Sliding window	25
4(a),4(b),4(c)	Mobility Problem	31-32-33
4.1	Hierarchy Page	36
4.2	Mobility Problem	37
4.3	CPN Model for a node Template	40
4.4	RREQInit Subpage	41
4.5	RREPProcess Subpage	43
4.6	RREQProcess	44
5	top level of DSR	56
5.1	Node Instance of DSR	58
5.2	Node Mechanism of DSR	60
5.3	Route request initialization	61
5.4	Route Request Processing	62
5.5	Route reply processing	63
5.6	Node Instance for DSR	65
5.7	Node mechanism for DSR	66
5.8	RREP process for DSR	67
5.9	RREQ Init for DSR	68
5.10	RREQ process for DSR	69

CHAPTER 1

Introduction to Petrinet

1. Introduction to Petrinets

Petri nets were developed in the early 1960s by C.A. Petri in his Ph.D. dissertation C.A. Petri. Kommunikation mit Automaten. PhD thesis, Institut for instrumentelle Mathematik, Bonn, 1962 .
Petri Nets are graphical and mathematical modeling tool for describing and studying information processing systems that are characterized as being concurrent, synchronous, distributed and non-deterministic [7].

1.1 Predicate/Transition Nets (Pr/T-nets):-

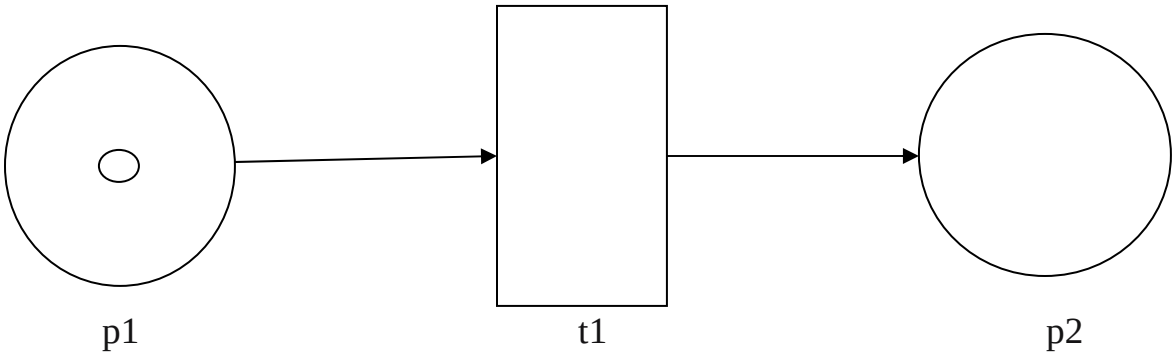
A Petri net consists of places, transitions, and directed arcs. Arcs are either from a place to a transition or from a transition to a place, never between places or between transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition.

Places may contain any non-negative number of tokens. A distribution of tokens over the places of a net is called a marking.

A Petri Net is a 5 tuple

PN = (P,T,F,W,M0), where P = {p1, p2, … pm} is a finite set of places T = {t1, t2, …, tn} is a finite set of transitions F Í (P X T) È (T X P) is a set of arcs W: F® {1,2,3,…} is a weighting function M0 :P® {0,1,2,…} is the initial marking // defines Number of tokens per place [7].

Example:-



(Fig 1: showing a basic Petri net as in [7])

Description of the above figure:
Places- p1, p2
Transitions-t1
M(p1)=1 , i.e.- p1 has one token
M(p2)=0 , i.e.- p2 has no token

1.2 Firing of a transition:-

- A transition of a Petri net may fire whenever there is a token at the end of all input arcs; when it fires, it consumes these tokens, and places tokens at the end of all output arcs .
- A firing is atomic, i.e., a single non-interruptible step. Execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire [7]. If a transition is enabled, it may fire, but it doesn't have to.
- Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurrent behavior of distributed systems [7].
- In order to simulate the dynamic behavior of a system, a state or marking in a Petri nets is changed according to the following transition (firing) rule.

1.3 Transition rule:-

I. A transition t is said to be enabled if each input place p of t is marked with at least w (p,t) tokens , where w(p,t) is the weight of the arc from p to t as in [2].

II. An enabled transition may or may not fire (depending on whether or not the event actually takes place) [2].

III. A firing of an enabled transition t removes w(p,t) tokens from each input place p of t and adds w(t,p) tokens to each output place p of t, where w(t,p) is the weight of the arc from t to p [2].

Basing upon these firing conditions Petri nets can be of 2 types:

1.Original Petri nets

2. Weighted Petri nets

1.3.1 Original Petri Nets:

Only 1 token can be removed/added from a place when a transition fires(i.e., the weight of the arc is always 1) as shown in the figure 1.1(a) and 1.1(b)

Example:

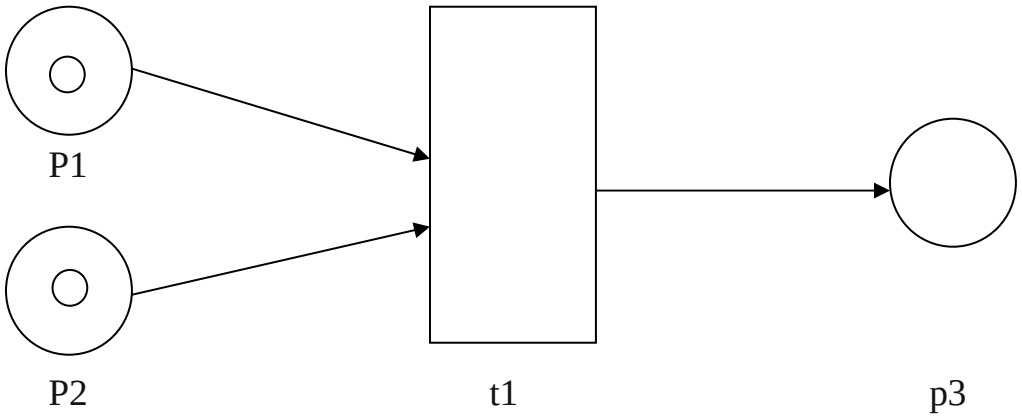


Fig-1.1(a): Petri net before t1 fires

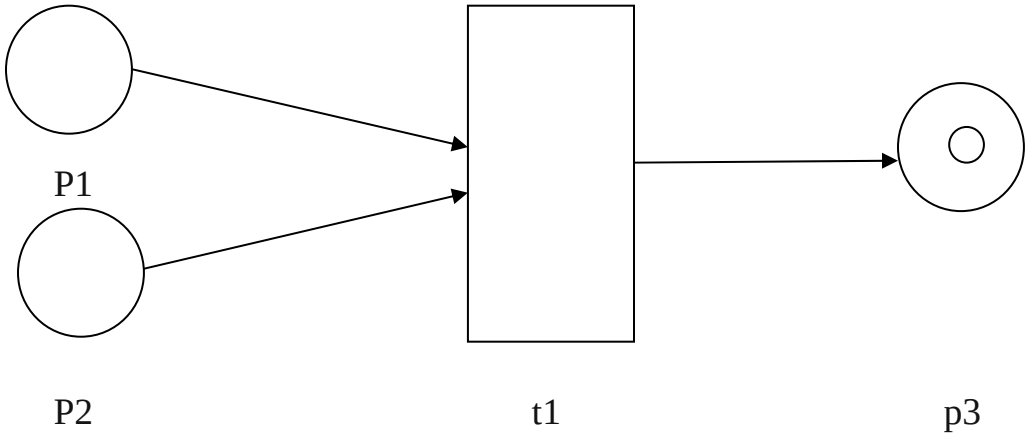


Fig-1.1(b): Petri net after t1 fires

1.3.2 Weighted Petri Nets:

- Generalized the original Petri net to allow *multiple tokens* to be Added/removed when a transition fires as shown in the fig. 1.2

- The edges are labeled with the weight (i.e., number of tokens).
- If there is no label, then the default value is 1.

Example:

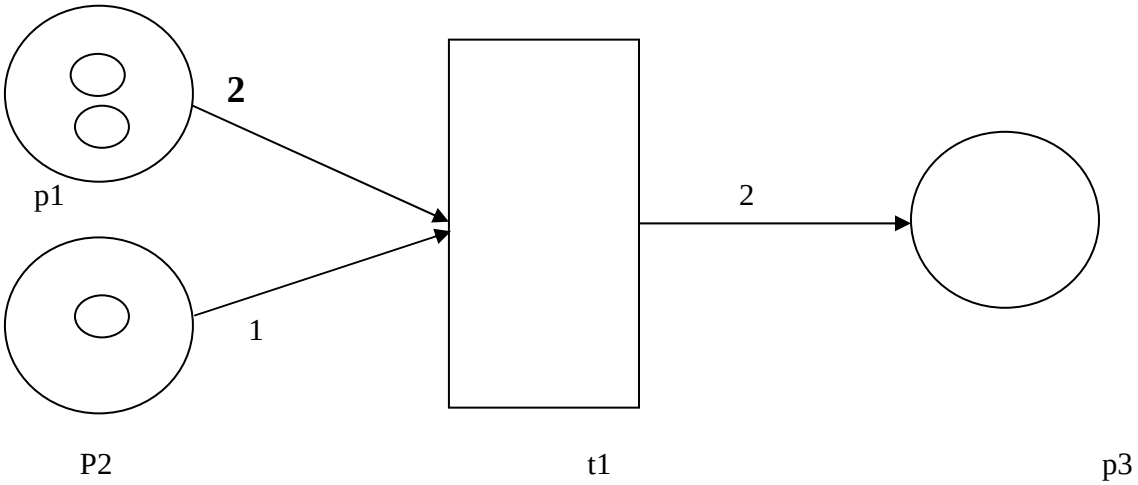


Fig-1.2(a): Petri net before t1 fires

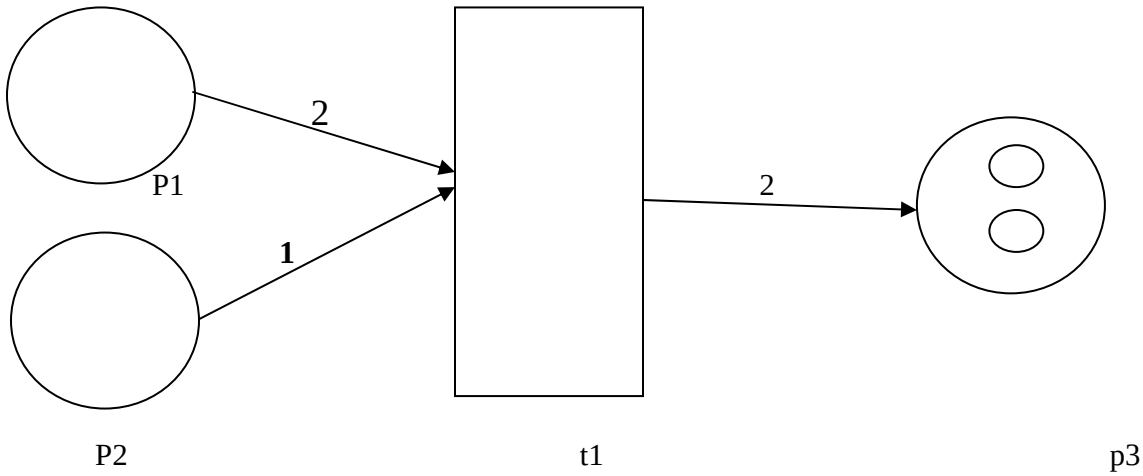


Fig-1.2(b): Petri net after t1 fires

CHAPTER 2

Colored Petrinet

2.1 Introduction

Colored Petri Nets (CPNs) is a language for the modeling and validation of systems in which concurrency, communication, and synchronization play a major role. Colored Petri Nets is a discrete-event modeling language combining Petri nets with the functional programming language Standard ML. Petri nets provide the foundation of the graphical notation and the basic primitives for modeling concurrency, communication, and synchronization. Standard ML provides the primitives for the definition of data types, describing data manipulation, and for creating compact and parameterisable models. A CPN model of a system is an executable model representing the states of the system and the events (transitions) that can cause the system to change state. The CPN language makes it possible to organize a model as a set of modules, and it includes a time concept for representing the time taken to execute events in the modeled system. CPN Tools is an industrial-strength computer tool for constructing and analyzing CPN models. Using CPN Tools, it is possible to investigate the behavior of the modeled system using simulation, to verify properties by means of state space methods and model checking, and to conduct simulation- based performance analysis. User interaction with CPN Tools is based on direct manipulation of the graphical representation of the CPN model using interaction techniques, such as tool palettes and marking menus. A license for CPN Tools can be obtained free of charge, also for commercial use [2].

Colored Petri Nets (CP-nets or CPNs) is a graphical language for constructing models of concurrent systems and analyzing their properties. CP-nets is a discrete-event modeling language combining Petri nets and the functional programming language CPN ML which is based on Standard ML. The CPN modeling language is a general purpose modeling language, i.e., it is not focused on modeling a specific class of systems, but aimed towards a very broad class of systems that can be characterized as concurrent systems. Typical application domains of CP-nets are communication protocols, data networks, distributed algorithms, and embedded systems. CP-nets are, however, also applicable more generally for modeling systems where concurrency and communication are key characteristics. Examples of these are business process and workflow modeling, manufacturing systems, and agent systems. A CPN model of a system describes the states of the system and the events (transitions) that can cause the system to change state as described in [2].
By making simulations of the CPN model, it is possible to investigate different scenarios and explore the behaviors of the system. Very often, the goal of simulation is to debug and investigate the system design. CP-nets can be simulated interactively or automatically. An interactive simulation is similar to single-step debugging. It provides a way to “walk through” a CPN model, investigating different scenarios in

detail and checking whether the model works as expected. During an interactive simulation, the modeler is in charge and determines the next step by selecting between the enabled events in the current state. It is possible to observe the effects of the individual steps directly on the graphical representation of the CPN model. Automatic simulation is similar to program execution. The purpose is to simulate the model as fast as possible and it is typically used for testing and performance analysis. For testing purposes, the modeler typically sets up appropriate breakpoints and stop criteria. For performance analysis the model is instrumented with data collectors to collect data concerning the performance of the system [2].

Time plays a significant role in a wide range of concurrent systems. The correct functioning of some systems crucially depends on the time taken by certain activities, and different design decisions may have a significant impact on the performance of a system. CP-nets include a time concept that makes it possible to capture the time taken to execute activities in the system. The time concept also means that CP-nets can be applied for simulation-based performance analysis, investigating performance measures such as delays, throughput, and queue lengths in the system, and for modeling and validation of real-time systems.

CPN models can be structured into a set of modules to handle large specifications. The modules interact with each other through a set of well-defined interfaces, in a similar way as in programming languages. The module concept of CP-nets is based on a hierarchical structuring mechanism, allowing a module to have sub modules and allowing a set of modules to be composed to form a new module.

2.2 The CPN modeling language

In this section, we introduce the CPN modeling language by means of a small running example modeling a communication protocol. We use a simple protocol since it is easy to explain and understand, and because it involves concurrency, non-determinism, communication, and synchronization which are key characteristics of concurrent systems. The protocol itself is unsophisticated, but yet complex enough to illustrate the constructs of the CPN modeling language. No prior knowledge of protocols is required as given in [2].

The simple protocol consists of a sender transferring a number of data packets to a receiver. Communication takes place on an unreliable network, i.e., packets may be lost and overtaking is possible. The protocol uses sequence numbers, acknowledgements, and retransmissions to ensure that the data packets are delivered exactly once and in the correct order at the receiving end. The protocol uses a stop-and-wait strategy, i.e., the same data packet is transmitted until a corresponding acknowledgement is received. The data packets consist of a sequence number and the data (payload) to be transmitted. An acknowledgement consists of a sequence number specifying the number of the data packet expected next by the receiver as given in [2].

Net structure, declarations, and inscriptions:
A CPN model is usually created as a graphical drawing, and Fig.2 shows the basic CPN model of the protocol. The left part models the sender, the middle part models the network, and the right part models the receiver. The CPN model contains eight places (drawn as ellipses or circles), five transitions (drawn as rectangular boxes), a number of directed arcs connecting places and transitions, and finally some textual inscriptions next to the places, transitions, and arcs. The inscriptions are written in the CPN ML programming language which is an extension of the Standard ML language. Places and transitions are called nodes. Together with the directed arcs they constitute the net structure. An arc always connects a place to a transition or a transition to a place. It is illegal to have an arc between two nodes of the same kind, i.e., between two transitions or two places.

The state of the modeled system is represented by the places. Each place can be marked with one or more tokens, and each token has a data value attached to it. This data value is called the token color. It is the number of tokens and the token colors on the individual places which together represent the state of the system. This is called a marking of the CPN model, while the tokens on a specific place constitute the marking of that place. By convention, we write the names of the places inside the ellipses. The names have no formal meaning— but they have huge practical importance for the readability of a CPN model (just like the use of mnemonic names in traditional programming). The state of the sender is modeled by the two places PacketsToSend and Next Send. The state of the receiver is modeled by the two places Data Received and NextRec, and the state of the network is modeled by the places A, B, C, and D. Next to each place, there is an inscription which determines the set of token colours (data values) that the tokens on the place are allowed to have. The set of possible token colours is specified by means of a type (as known from programming languages), and it is called the color set of the place. By convention the color set is written below the place. The places NextSend, NextRec, C, and D have the color set NO. In CPN Tools, color sets are defined using the CPN ML keyword colset, and the color set NO is defined to be equal to the integer [2].

```
type int:
colset NO = int;
```

This means that tokens residing on the four places NextSend, NextRec, C, and D will have an integer as their token color. The color set NO is used to model the sequence numbers in the protocol. The place Data Received has the color set DATA defined to be the set of all text strings string. The color set DATA is used to model the payload of data packets. The remaining three places have the color set NOxDATA which is defined to be the product of the types NO and DATA. This type contains all two-tuples (pairs) where the first element is an integer and the second element is a text string. Tuples are written using parentheses (and) around a comma separated list. The color set NOxDATA is used to model the data packets which contain a sequence number and some data. The color sets are defined as:

```
colset DATA = string;
colset NOxDATA = product NO*DATA;
```

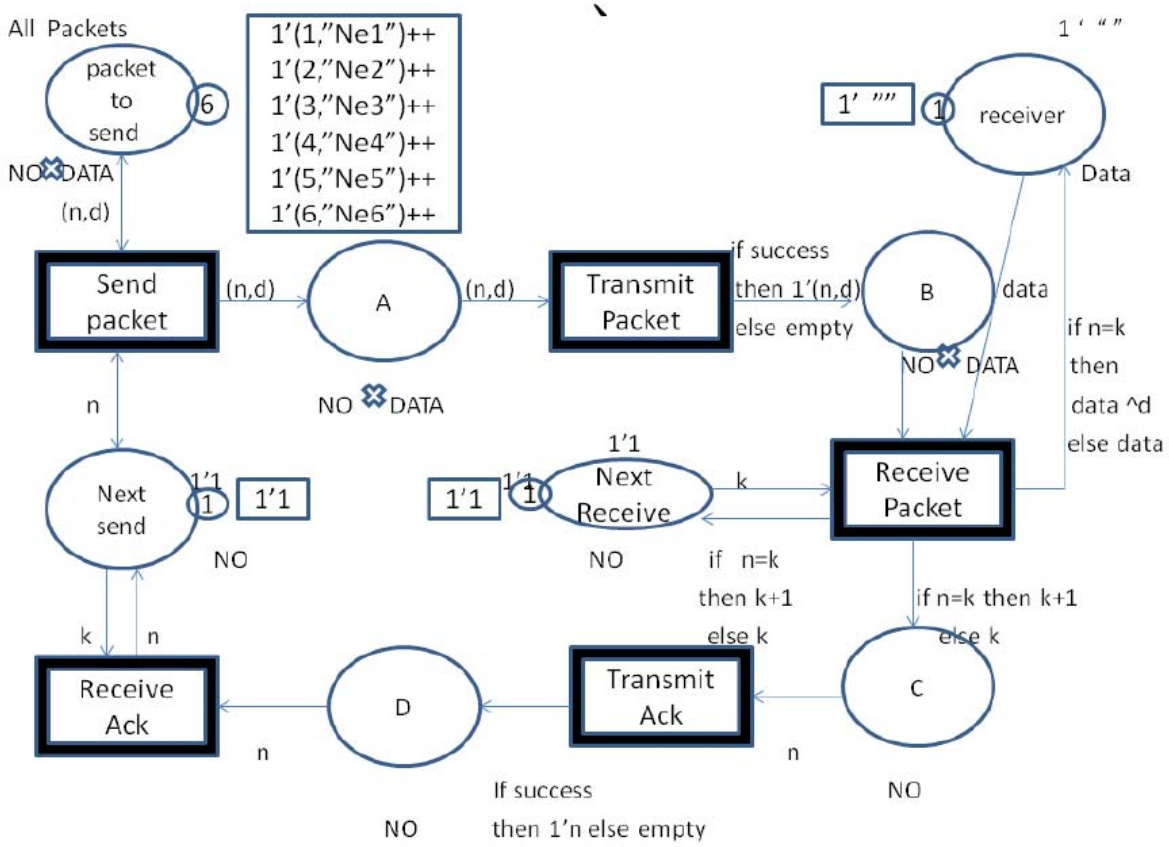


fig 2

Next to each place, we find another inscription which determines the initial marking of the place. The initial marking inscription of a place is by convention written above the place. For example, the inscription at the upper right side of the place NextSend specifies that the initial marking of this place consists of one token with the color (value) 1. This indicates that we want data packet number 1 to be the first data packet to be sent. Analogously, the place NextRec has an initial marking consisting of a single token with the color 1. This indicates that the receiver is initially expecting the data packet with sequence number 1. The place DataReceived has an initial marking which consists of one token with color ""(which is the empty text string). This indicates that the receiver has initially received no data as given in [2].

The inscription AllPackets at the upper left side of place PacketsToSend is a constant defined as:

val AllPackets = 1‘(1,"Ne1") ++ 1‘(2,"Ne2") ++ 1‘(3,"Ne3 ") ++ 1‘(4,"Ne4") ++ 1‘(5,"Ne5 ") ++ 1‘(6,"Ne6");

which specifies that the initial marking of this place consists of six tokens with the data values:

(1,"Ne1"),(2,"Ne2"),(3,"Ne3 "),(4,"Ne4"), (5,"Ne5 "),(6,"Ne6").

The ++ and ‘are operators that allow for the construction of a multi-set consisting of token colours. A multi-set is similar to a set, except that values can appear more than once. The infix operator ‘ takes a nonnegative integer as left argument specifying the number of appearances of the element provided as the right argument. The ++ takes two multi-sets as arguments and returns their union (sum). The initial marking of PacketsToSend consists of six tokens representing the data packets which we want to transmit. The absence of an inscription specifying the initial marking means that the place initially contains no tokens. This is the case for the places A, B, C, and D.

CHAPTER 3

Sliding Window Implementation

Implementation of sliding Window using colored Petri net

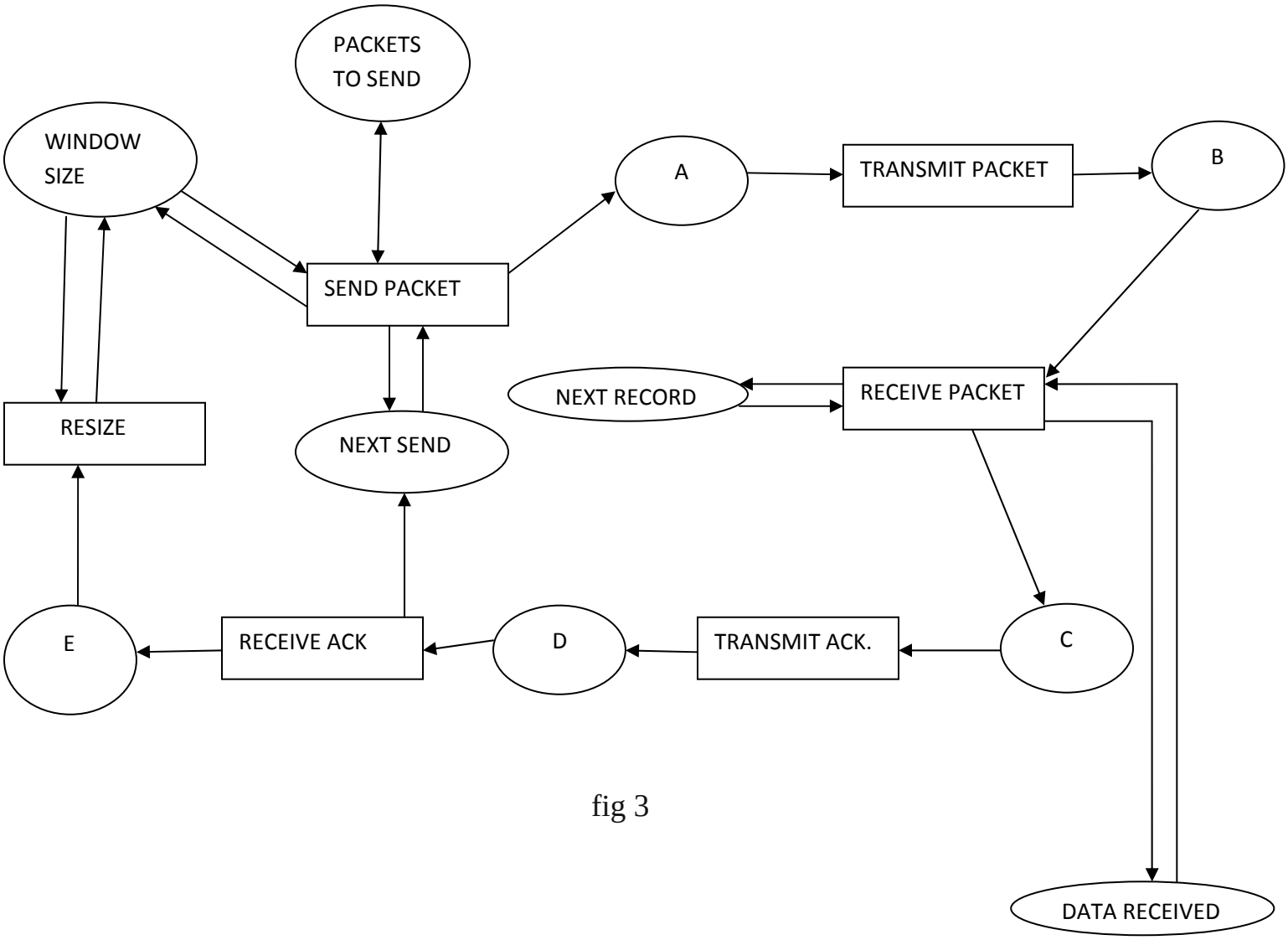


fig 3

- Sliding Window protocol was implemented using colored Petri net.

Snapshot during working condition:-

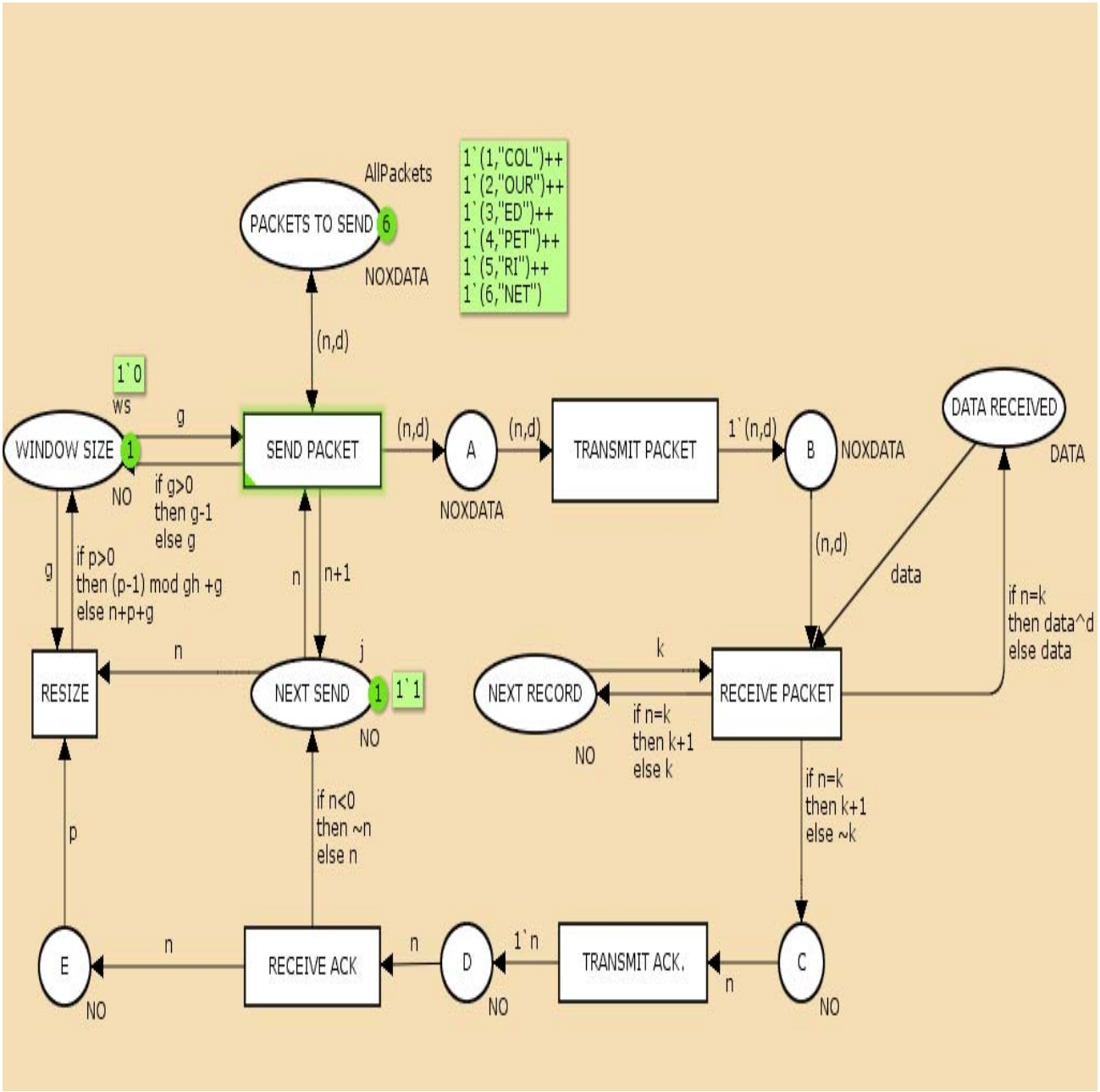


fig 3.1

In sliding window we have used the go-back-n mechanism. In this mechanism when a packet is lost, the receiver-end sends a NAK (not acknowledged). Once the sender receives the NAK it shifts its window back to the appropriate packet and retransmits all the packets from that packet onwards.

So in our implementation we have sender side and receiver side and by using Boolean we made it probabilistic regarding if a packet will be sent or not.

In the sender side there is a separate place to store “window size”, also we have a “next send” which stores the value of the next packet to be sent to the network. We have a transition resize which upon receiving the Acknowledgment from receiver resizes its window accordingly.

In the receiver side the working is very simple upon receiving the packet it appends the data into a single location and sends back the acknowledgment to the sender or else if it doesn’t receive the correct packet it sends the NAK packet to the sender so that the sender sends the correct packet required at the receiver end.

Here we have used 3 Color sets basically:-

1. Colset NO:INT;
2. Colset DATA:STRING;
3. Colset NOXDATA:INT*STRING;

Apart from that we have defined various local variables of the type of these color sets.

CHAPTER 4

Mobile Ad-hoc Network

MOBILE AD-HOC NETWORK

In a Mobile Ad Hoc Network (MANET), which implies a wireless environment wherein mobile nodes directly send messages to each other via other nodes. A node can function as a router and can thus send a message to a destination node beyond its transmission range by using other nodes as relay points. Challenging requirements have been introduced into MANETs and designing with the explosive growth of the Internet and mobile communication networks, thus making routing protocols become more complex. 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 needed the protocol provides the defined service correctly, thus providing for a successful application of MANETS.

Application of MANET span through various type of networks as given in [6] which are namely:

1. Enterprise network
2. Home Network
3. Tactical Network
4. Sensor Network
5. Pervasive Networks
6. Wearable Computing
7. Automated vehicle Network

Mobile ad hoc networking is a technology which works without requiring an already established infrastructure and centralized administration and provides for the cooperative engagement of a group of mobile nodes. In a mobile ad hoc network (MANET), each node can function as router and thus mobile nodes directly send messages to each other via wireless to a destination node beyond its transmission range by using other nodes as relay points. This mode of communication is known as multihopping. There are additional constraints compared with its hardwired counterpart, constraints such as bandwidth-constraint,enery-constrained and limited physical security .Because of these additional constraints and the dynamic topology, conventional routing protocols are not appropriate for MANETs thus new protocols have been introduced which can be broadly categorized as reactive and proactive or

both e.g. routing protocols such as Ad Hoc On-Demand Distance vector Routing(AODV) ,Dynamic Source Routing(DSR) , Temporarily-Ordered Routing Algorithm (TORA), Association based Routing (ABR) that are specific to mobile net working have been proposed . Which protocol achieves a better trade-off depends on the traffic and mobility patterns of an application. Reactive protocols maintain routes only if needed, and thus have lower overhead. For proactive protocols the latency time is lower since herein the routes are determined independently of the traffic pattern thus the routes are maintained at all times. The research on MANETs originates from part of the Advanced Research 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 have been added into MANET and the designing routing protocols has become more and more complex. A routing protocol should be unambiguous, complete and functionally correct in order to have a successful application of MANET. These properties ensure that the protocol is robust to the unexpected combinations of events that can lead the protocol to undesirable states and might even affect its performance or produce errors.

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 needed 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.

4.1 MANETs have several salient characteristics:

1) Dynamic topologies: Given the fact that for dynamic nature of the real time scenario the nodes are allowed to move arbitrarily which accounts for the topology changes taking place randomly and at unpredictable times and might also include links which can be unidirectional and bidirectional depending upon the nature of the nodes and the mode of communication between them.

2) Bandwidth- constraint which is variable among the various links. Comparing the hardwired against the wireless communication , the wireless always have been with lower capacity which can be further accounted for the effects of multiple access , noise and other interference conditions , fading etc.

One effect of the relatively low to moderate link capacities is that congestion is typically the norm rather than the exception, i.e. aggregate application demand will likely approach or exceed network capacity frequently. As the mobile network is often simply an extension of the fixed network infrastructure, mobile ad hoc users will demand similar services. These demands will continue to increase as multimedia computing and collaborative networking applications rise.

3) Energy –constraint operation: For the nodes present in the MANET which rely on batteries or other exhaustible means optimization of these resources is an important design criteria.

4)Limited physical security: The decentralized nature of the MANETs provides for the additional robustness against the single point failure of the centralized approaches present. But the wireless networks are more prone to physical security than the fixed-cable nets which further increases the chances of eavesdropping, spoofing, and even denial-of-service attacks. To prevent these attacks the existing link security techniques are often applied within wireless networks to reduce security threats.

4.2 Mobility Problem:

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.

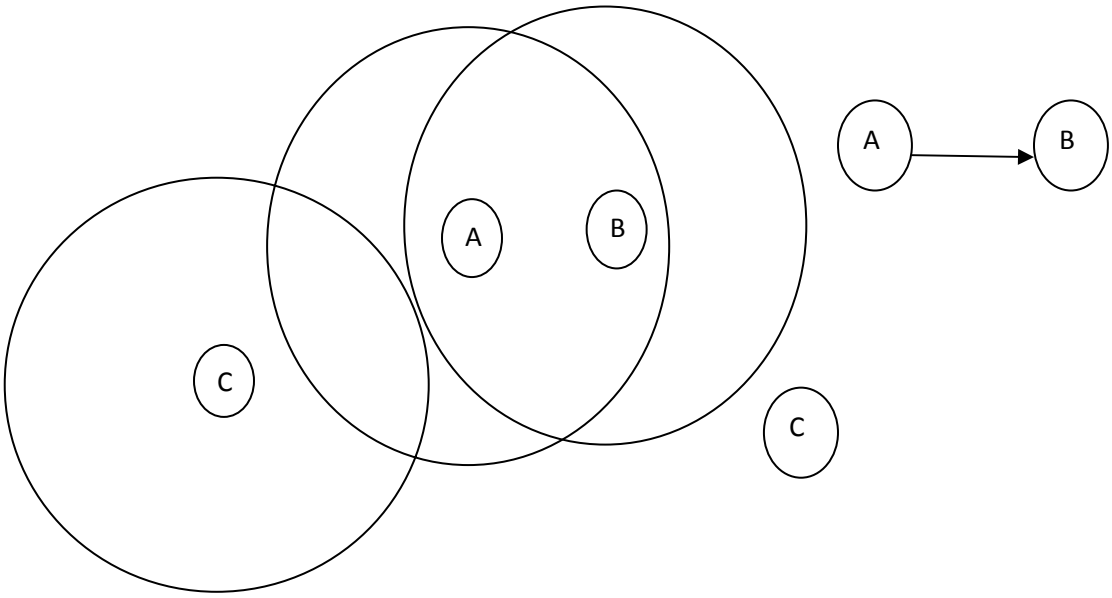


Fig.4(a)- Nodes A and B are neighbors

For example, in Fig.4(a), 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 nodes may disappear and reappear in an unpredictable manner. For example in Fig.4(a) when node A moves out of the transmission range of node B, the link between nodes A and B breaks as shown in the Fig.4(b). When node B moves back within the transmission range of node A, the link between them reappears.

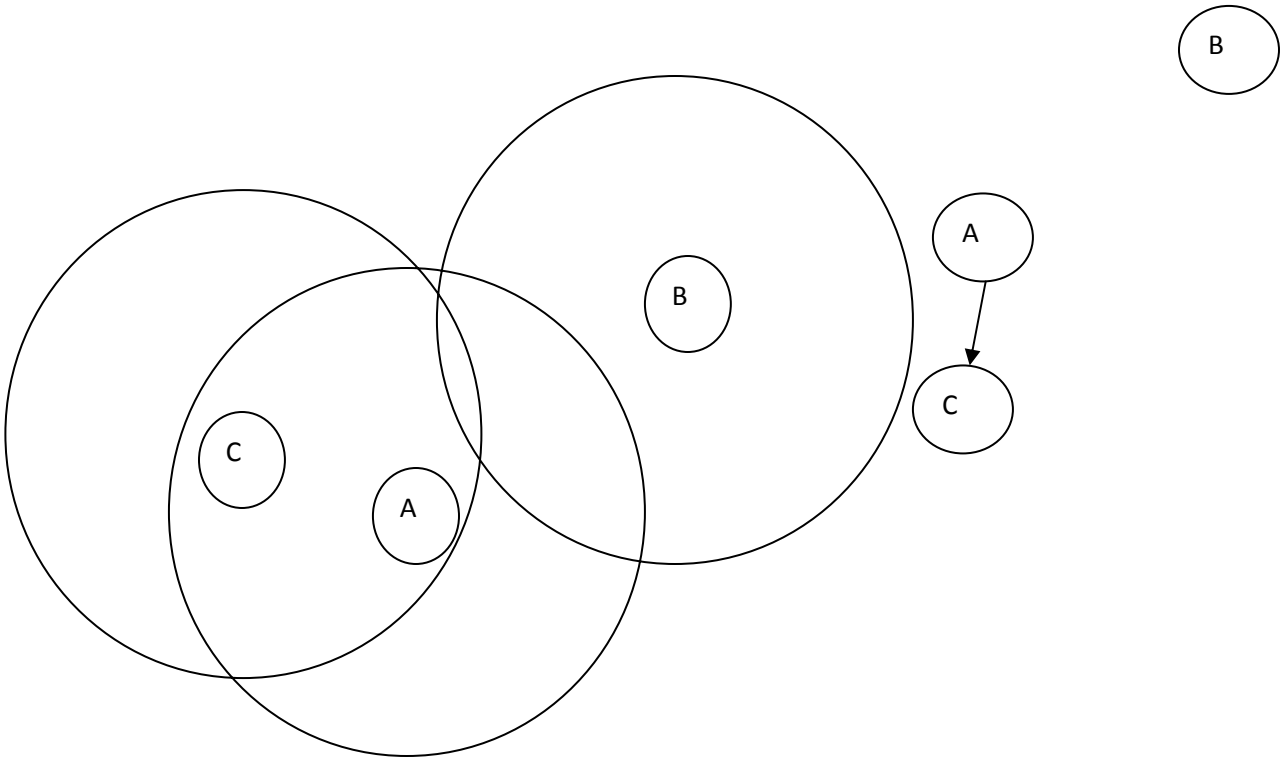


Fig.4(b)- Node A moves out of the transmission range of node B moves into C

In an extreme case the graph of a MANET remains the same even if all nodes have moved; e.g., the MANET graphs in Fig. 4 (a) and 4(c) 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 nods when we build a CPN model for a MANET.

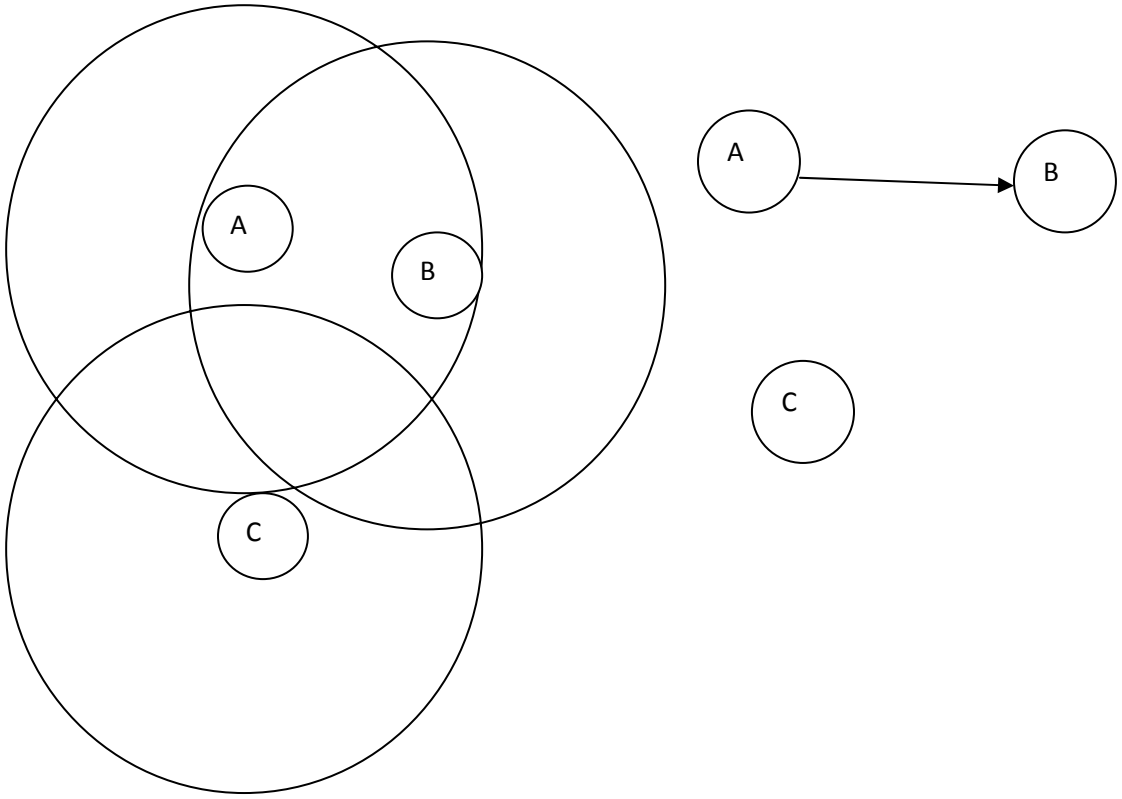


Fig.4(c): The topology remains the same as that in Fig.4(a) although all three nodes have moved .

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 neighbor's identifications to send messages. It is difficult to capture structure changes in 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 relationships unpredictably. Therefore, it is not easy to build a CPN model of a MANET as nodes can move in and out of their transmission ranges and thus MANET's topology(graph) dynamically changes. To address this problem we have discussed topology approximation that has already implemented algorithm in section 4.3 and the proposed algorithm in section 4.4.

4.3 Topology Approximation:

To address the mobility problem, a topology approximation (TA) algorithm has been already proposed in building CPN models for a routing protocol. The TA mechanisms 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 or set of networks is typically performed using only partial or approximate information. In the TA mechanism proposed, uses a receiving function for selecting senders neighbors which receive messages sent by a sender node, in order to mimic the dynamic structure of a MANET as given in [1].

Topology Approximation Algorithm:

The TA mechanism uses a receiving function to describe the dynamic structure of a MANET. It uses the general assumption in MANET that every node has the same distance of transmission range. Thus, if a MANET is composed of a fixed no of nodes, the average no 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 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 constant, if there are a fixed number of nodes within that area. Thus average number of neighbors can well represent the approximate topology in formation of a MANET. Based on this observation the TA mechanism uses a receiving function to decide which node receives which message.
Assume every node in a MANET has the same number of neighbors which is equal to average degree of 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 message. 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, it creates a place called Store to hold the entire message in transmission. After the place Store receives messages, a receiving function directs messages to the corresponding neighboring node. 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 neighbor will receive 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 the broadcast message sent or forwarded by a node. But only x out of these (n-1) copies of the broadcast message will 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 message 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. Thus it results:

PB =d/[(n-1)(n-1)]

Nodes in a MANET also receive unicast message such as route reply(RREP) message, which are different from broadcast message such as route request (RREQ) message. Thus the receiving function needs to check if a message is a unicast or broadcast message as given in [1].

If(the message is of type RREQ and is not sent by the node)

{

Generate a random integer K, between a and 100;

If(100PB>K)

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;

The following AODV model is referred from the thesis paper [1] and is being referred in its original structure for the comparative study between AODV and DSR.

4.4 CPN Model of AODV:

Here we have demonstrated the effectiveness of the topology approximation by considering a reactive routing protocol called AODV. The hierarchy pages are presented here. It explains the overall organization of the comprising modules comprising the CPN model of our MANET.

Here Prime#1 is the top level of our Model for a MANET. And the second level is the node template for implementing AODV routing protocol.

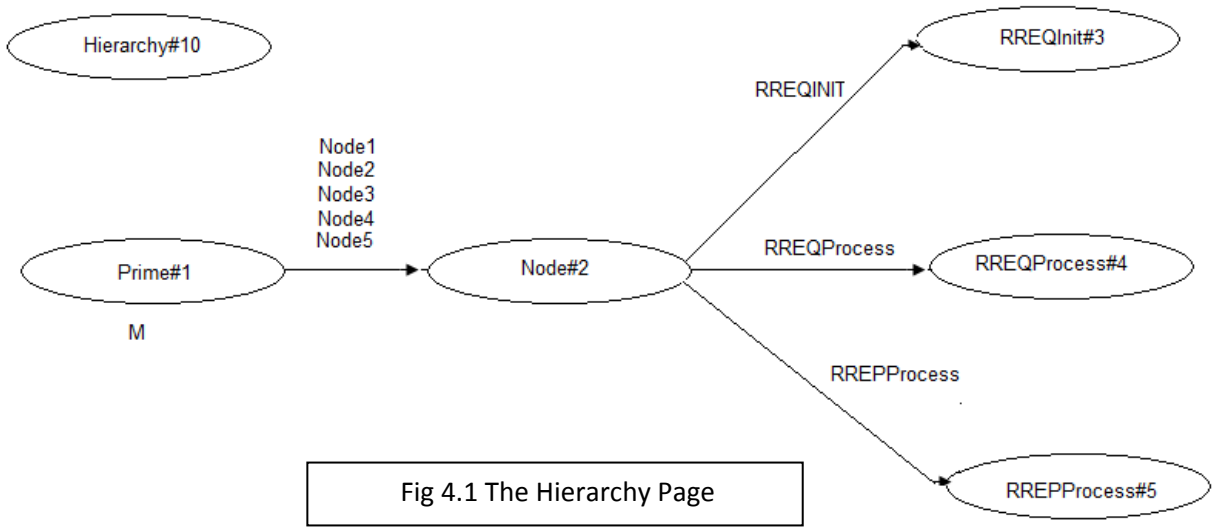


Fig 4.1 The Hierarchy Page

Here we can create model for any manet composed of nodes as many as allowed by instantiating this node template using Design/CPN tool .The third level has three pages (page RREQInit#3,page RREQProcess#4 and RREPProcess#5) , each of which is named by an AODV state as given in [1].

The function of the node template is just 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 do it by instantiating the node template. Like an object in an object –oriented programming language, every instantiated node has its own local variables and provides interface to the outside world. In our CPN modeling, an instantiated node has its own local variables and provide 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 can simply add substitution transition to the model. Fig 4.2. Shows a node instance in a MANET, which is a part of the CPN model in the top level as given in [1].

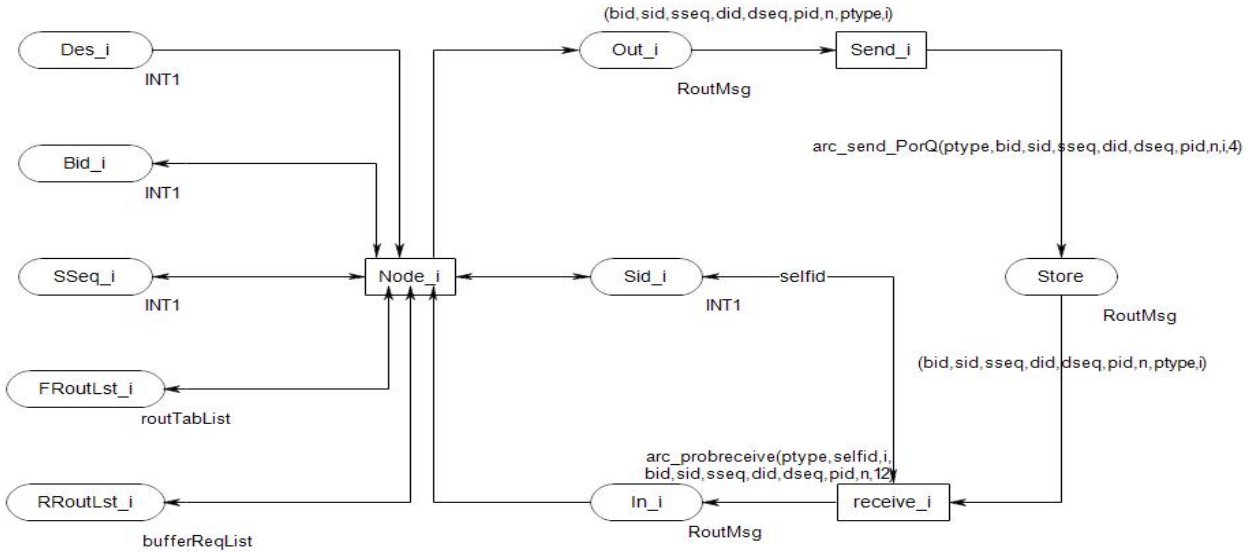


Fig 4.2: Node Instance

As shown in fig 4.2, 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 which provides interface for the node communication .The place SSeq-i represents the sequence number of the node. The place Bid-i is the broadcast ID of the node. We separate forward paths from reverse paths using different places to represent them and which ultimately helps to easily analyze and understand the

simulation results. 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 is responsible for holding all the messages that the node wants to send or forward, and the place In-I is responsible for containing all the incoming messages that the node receives from the place store. 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 as given in [1].

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 .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. It directs all RREP messages routed for this node and selectively directs RREQ messages based on PB to the receiving buffer, which is represented by the place In-i. The function guard_Msgselect (ptype,selfid,I,sid) 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 as given in [1].

4.4.1 CPN Model of AODV Routing 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 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 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 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 messages to all of its neighbors. Based on RREQ and RREP messages, nodes capture the route information. When an intermediate node receives a RREQ message, the node sets up a reverse route entry to the source node in its route table. There is no distinction in functionalities for forward and reverse path entries except their origins. AODV protocol has four states as given in [1].

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

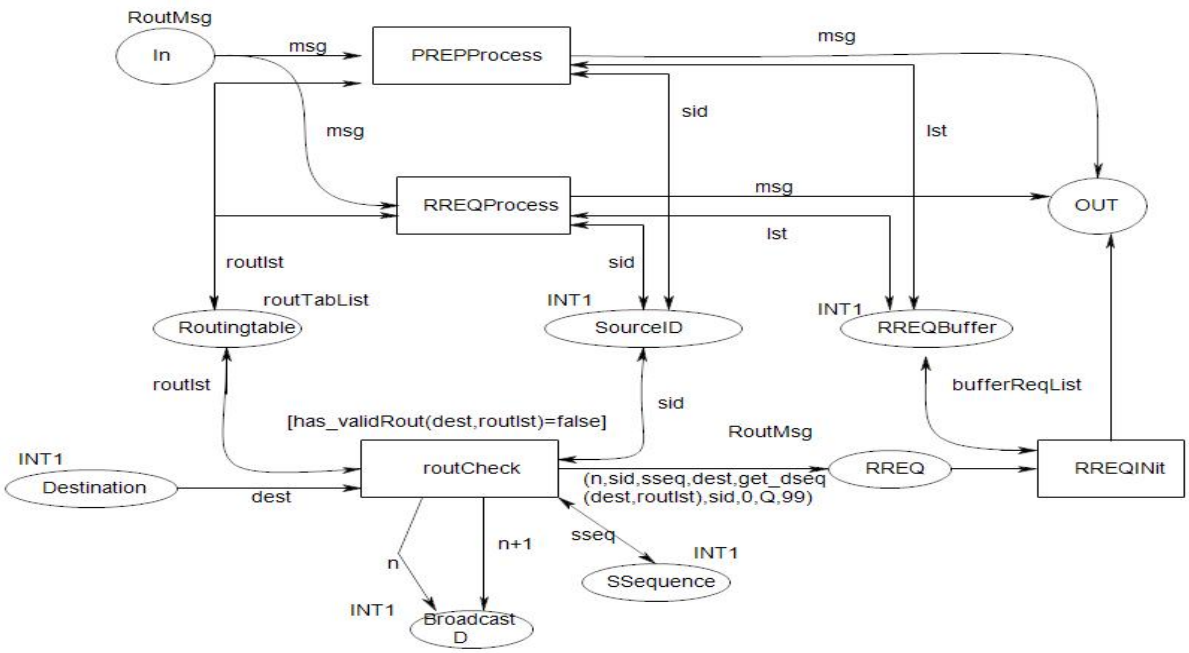


Fig 4.3: CPN Model for a node Template

The above figure shows the template of a CPN node for implementing AODV routing protocol. It is represented by the substitution transition Node-i, in fig 3. 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 consists of 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 and if a node receives a RREQ/RREP message, it enters RREQProcess/RREPProcess state. The various modules are as follows according to [1]:

4.4.2 Node generating a broadcast message:

If a node wants to send a packet to a destination node, a token representing the destination ID is put into the place Destination in fig 4.3, thus initiating the working process of the model. After that, 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 4.3.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 4. 3 and 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.

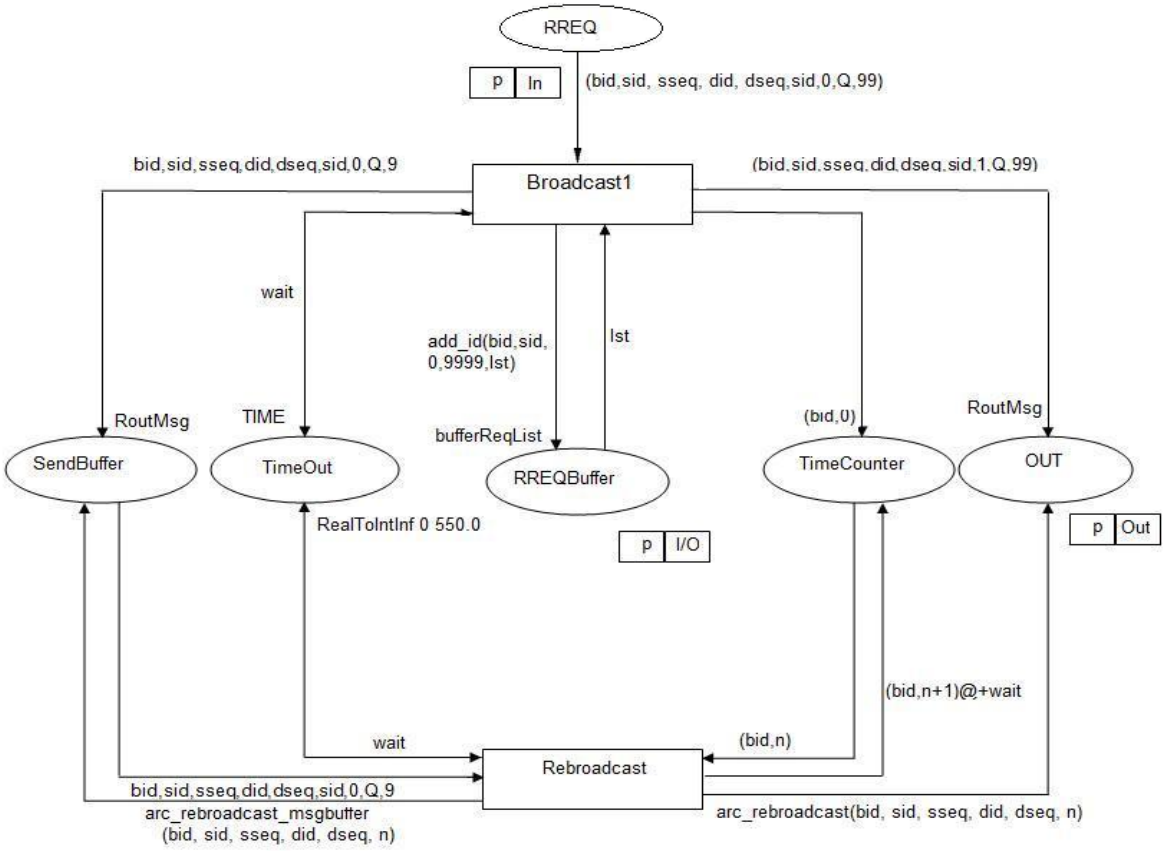


Fig 4.4 : RREQInit Subpage

The main functionality 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 4.4. At the same time, the node puts the RREQ message into the retransmission buffer, which is represented by the place SendBuffer, sets a timer for the broadcast message. If the node does not get any feedback from other nodes after the timer is off, it rebroadcasts the RREQ message by the transition Rebroadcast in fig 4.4 as was in [1].

4.4.3 Node Unicasting a RREP message to its previous node:

When a node receives a RREP message ,it enter into the RREPProcess state, which is represented by the substitution transition RREPprocess in fig 4.5.Here fig 4.5 shows the detailed implementation of the RREPProcess.

The main functionality of this page is to update the route table and forward the received RREP message if necessary. So we have used two function named arc_updateRout(did,nohop,routlst,bid,lifetime,did,dseq,nohop,lst) play important roles on this subpage. The function arc_forwardRREP() is the arc inscription from the transition ADDRoute 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. 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 recipient for the 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 as given in [1].

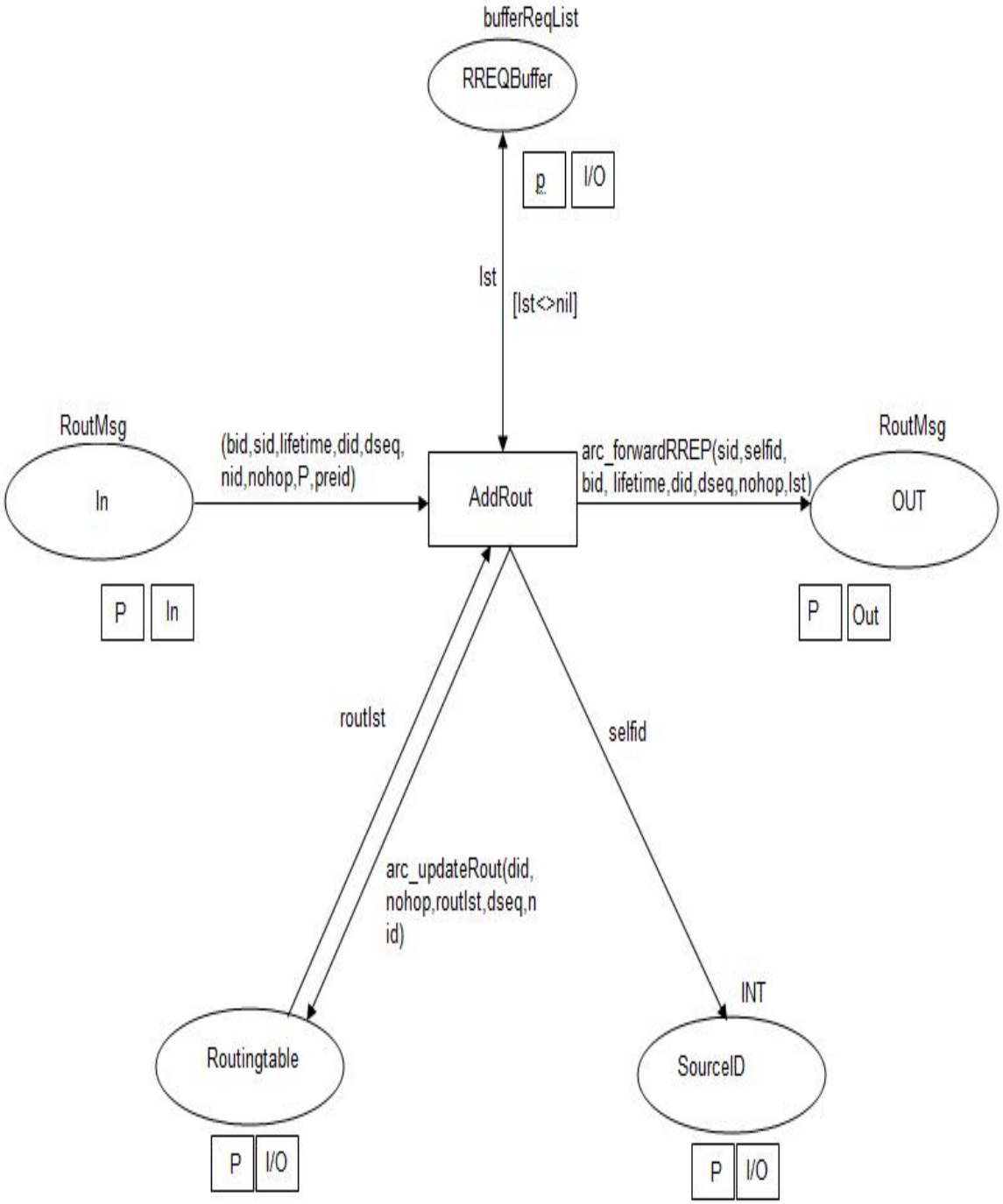


Fig 4.5: RREPProcess Subpage

4.4.4 Node broadcasting an incoming RREQ message to its neighboring nodes:

If a node receives a RREQ message, then it enters RREQProcess state, which is represented by the substitution transition RREQProcess in fig4.3. Fig 4.6 shows the detailed information regarding REQProcess.

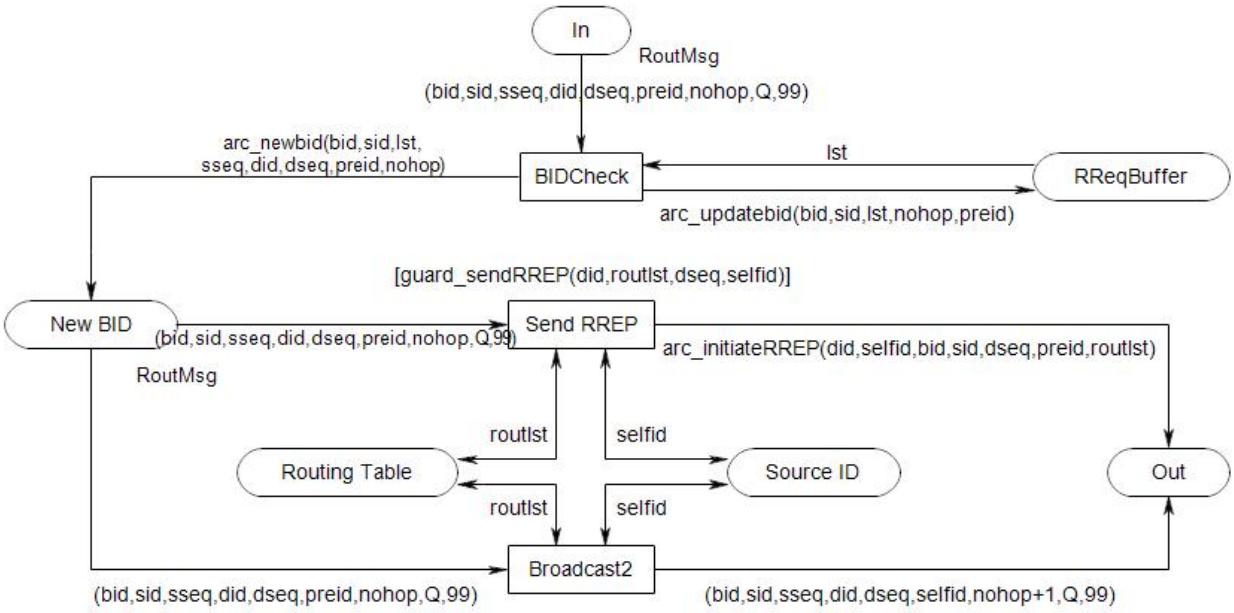


Fig 4. 6: RREQProcess

The actual 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, the transition Bidcheck is enabled, thus entering the RREQProcess state for the routing protocol. If the received RREQ message is fresh, it is then simply discarded .This functionality is achieved by the arc inscription arc_newbid(bid, sid,lst,sseq,did,dseq,preid,nohop). 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_sendRREP() returns true. Otherwise ,the guard guard_broadcast2() returns true. If its SendRREP's turn, a RREP message is initiated by the arc inscription arc_initiateRREP(did,selfid,bid,sid,dseq,preid,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 as given in [1].

4.4.5 Global Declarations

The global declarations comprise the color sets and variables for the AODV. The color sets show data structures storing different routing information such as route entry,RREQ and RREP messages as given in [1].

The color set RoutMsg 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.

DID : Destination node.

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.

DSEQ : Most recent recorded sequence number for destination DID

PID : The immediately previous node from which the node receives the message.The color set BufferREQ defines buffered identification of RREQ message that a node has processed. The color set RoutTable defines the structure of a rout entry in route table. It contains the following fields :

DID : Destination of the route.

NID : Next node on the route to DID.

DSEQ : Lost 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.

LIFETIME : Remaining time before the route expires.

CHAPTER 5

Dynamic Source Routing Protocol

5.1 INTRODUCTION:

The *Dynamic Source Routing* protocol (DSR) [Johnson 1994, Johnson 1996a, Broch 1999a] is an efficient and simple protocol that has been designed to be used in multi-hop wireless ad-hoc networks of mobile nodes. DSR can be said to be a self-organizing and self-configuring protocol which does not require an existing network infrastructure . There are two mechanisms composing this protocol namely *RouteDiscovery* and *Route Maintenance*, which together lead to the discovery of other nodes present in the network and also to help in maintaining of the routes to the various destinations present in the ad-hoc network. DSR provides for the avoidance to perform the periodic or up-to-date routing information between the intermediate nodes present within the network and also provides a loop-free environment . It also allows nodes forwarding or overhearing packets to cache the routing information in them for their own future use. Various aspects of the DSR protocol operate entirely on an on-demand basis, thereby allowing the overhead involved to reduce automatically to only a level that would be required to track the changes in routes currently in use as given in [4].

DSR protocol allows for the discovery of the source routes i.e. route to destination from the source node for any destination in the ad-hoc network. For each of the data packet being sent across the network, each packet is required to carry the source route i.e. the ordered list of the nodes through which it must pass in order to reach the destination. This further helps to have a loop free scenario in the network and avoids the need to perform periodic routing of packets in order to keep various nodes updated about the changes taking in the network topology. This inclusion of source route helps other nodes which can then cache the routes available to the various destination nodes while overhearing or forwarding these packets across the network and can then use routes for the same use of sending of packets in future. DSR keeps track of the changes taking place in the topology .i.e. changing of the no. of intermediate steps etc on an on-demand met to by the Route Maintenance mechanism and thus keeps track of the overall changing topology which is involved in the current communication routes.

5.2 Assumptions :

- An assumption regarding the willingness of the nodes present within the ad hoc network to communicate or rather participate, states that all nodes agree to participate fully for forwarding the packets for other nodes present in the network.
- Nodes present in the adhoc network can move in or out of the network or can even move continuously taking into assumption that the speed with which these nodes move is moderate with respect to the packet transmission latency and the range of wireless transmission for a particular network. It can be said about the DSR , that it supports rapid rates of arbitrary mobility of nodes but assumes that the nodes do not move continuously so as to become a reason for the flooding of the each of the individual packets the option left to be used as a routing protocol.
- For the nodes it is assumed that they may be able to enable *promiscuous* receive mode on their network interface hardware , which can further cause the hardware to deliver the received packets without any filtering based on the link layer destination address to the network driver software .
- For the communication taking place between nodes, the mode of communication may be *bi-directional* or may be *uni-directional* allowing only one of the nodes to send the packets to the other node and restricting the communication in the reverse direction. DSR can discover and forward packets for types of communication mode discussed above.
- It is assumed that each node has a *single* IP address by which it is to be known in an ad-hoc network. It is possible that a single node may have different physical network interfaces , it is required in DSR that the node should select one of these IP's made available due to different network interfaces . This further allows each node to be recognized as a single entity independent of the network interface it might be using to communicate with other nodes.

5.3 DSR Protocol Description :

The DSR protocol is composed of two mechanisms that work together to allow the discovery and maintenance of source routes in the ad hoc network as given in [4]:

- *Route Discovery* Is the mechanism a source node **S** in order to send a packet to the destination node **D** would obtain a route to it , the route being obtained is known as the SourceRoute. This mechanism is used only when a node **S** does not have the route to the node **D.**
- *Route Maintenance* Is the mechanism by which a node **S** detects if the network topology has changed by using the source route to the destination node **D** such that it can then no longer send packets to **D** via using the same links present in the route. After the detection of the broken link in the source route the node S attempts to use other available routes to **D** and if that is not available then S can also initiate RouteDiscovery to the same destination node **D**. This mechanism is used only when **S** actually send packets to **D**.

In DSR both the mechanisms involved operate completely on-demand basis. There is no periodic update packets i.e. any advertisement, link status sensing, or neighbor detection packets used to keep routes available with a node updated within the network. DSR neither does not rely on the lower layer protocols of the network to perform the same act. For this reason i.e. on demand behavior and lack of periodic activity the no. of overhead packets caused due to the protocol (DSR) reduces to *zero* , wherein all the routes which are required for the current communication are known beforehand. For the movement of the nodes within the network the routing packet overhead automatically would reach to a level so as to keep up with the updated routes which are currently in use.

For the a single RouteDiscovery being made in DSR , it is possible that the node might learn and cache multiple routes to the same destination . This caching of multiple routes helps a node to send a packet to the destination node via an alternative route if the one it has been using would fail. This further helps in avoiding the overhead that would have been involved to perform a new RouteDiscovery each time a route in use breaks. Another case in networks is that , in wireless networks it might be possible that a link between two nodes might not work well in both directions , which might be due to several reasons e.g. sources of interference . DSR allows uni-directional links to be used whenever necessary thus improving the overall performance and the network connectivity in the system.

5.4 Basic DSR Route Discovery :

RouteRequest :

For when a node **S** originates a new packet to be sent to a destination node **D** , it places in the header of the packet the source route which gives the sequence of hops that must be followed by the packet in order to reach **D**. In most cases the node S can find a route to **D** from the available *RouteCache* previously learned. But in the case no valid route to the node **D** is found the node **S** initiates the RouteDiscovery mechanism to dynamically find a route to **D**. In such a scenario , node **S** is called the initiator and the destination node **D** the target of the RouteDiscovery . Each of the ROUTEREQUEST message identifies the initiator node and the target node , along with these entities also contains a unique request id , which is determined by the initiator node **S**. The ROUTEREQUEST also contains a record which lists to store the address of all the intermediate nodes through which this particular packet has been forwarded to. Initially this route record is initialized to an empty list by the initiator for the RouteDiscovery .

For whenever a node receives a ROUTEREQUEST , if that is the target node D, then D sends a ROUTEREPLY message to the initiator of the ROUTEREQUEST, when the initiator S receives the ROUTEREPLY , it caches the route made available in the header of the ROUTEREPLY packet to be used for sending subsequent packets to this destination. Otherwise if this node is not the target node D and has recently seen another ROUTEREQUEST message from this node S bearing the same request id or if it finds its own address is already listed in the header of ROUTEREQUEST message which carries the list of the nodes that have been visited by the packet , then the packet would be discarded . Otherwise , this node appends its own address to the route record available in the ROUTEREQUEST message header and would further propagate it by a local broadcast with the same request id as given in [4].

RouteReply:

For returning ROUTE REPLY messages to the initiator S of the Route Discovery , the destination node D checks its own Route Cache for a route back to the initiator, which if found would be used to send the packet containing ROUTEREPLY. Otherwise , the destination may perform Route Discovery mechanism for the target node being the initiator node. Also the node returning the reply can simply reverse the sequence of hops in the route record of the ROUTEDISCOVERY just received in order to use as the source route on the packet carrying the ROUTE REPLY itself as given in [4].

Send Buffer :

For whenever the RouteDiscovery take place the sending nodes would save a copy of the original message or the packet in the local buffer called *SEND BUFFER*. All the packets present in the buffer are timestamped for the time it was placed in the buffer and gets discarded after staying in the buffer for some timeout period. FIFO or other strategy might be used in order to prevent the send buffer from overflowing. For a packet which is present in the buffer the node must occasionally initiate a new RouteDiscovery for the packet. But the node must make sure that the rate at which the new RouteDiscoveries are being made for the same address are kept under control since it is possible that the destination might not be available at that point of time.

For if a large no. of RouteDiscoveries would be made for a single packet that would lead to a large no. of unproductive ROUTEREQUEST packets . And thus the node must not initiate more no. of RouteDiscovery until the passage of a minimum allowable interval between the latest RouteDiscovery for the required destination has been reached as given in [4].

Basic DSR Route Maintenance :

For RouteMaintenance it is required for each node transmitting the packet that when originating or forwarding a packet using a source route it is responsible for confirming that the packet has been received by the next hop along the source route. The retransmission of the packet is done until its maximum no. of attempts is reached or its required confirmation is received. A RouteError message is received by the original sender of the packet when the packet is retransmitted by some hop the maximum number of times and no receipt confirmation is received. The source node upon receiving the error message identifies the link over which the packet could not be forwarded and thus removes the same link from its cache. For the retransmission to be done the source node searches another source route to the destination node in its cache table , which if not found would lead to another RouteDiscovery for the destination as given in [4].

5.5 OUR WORK ON MODELING AND SIMULATION OF DSR USING COLORED PETRI-NET

Here we have modeled the DSR algorithm based on the skeleton of the AODV model that was previously implemented using CPN.

In this algorithm we have implemented both of the mechanisms which constitute the protocol namely:-

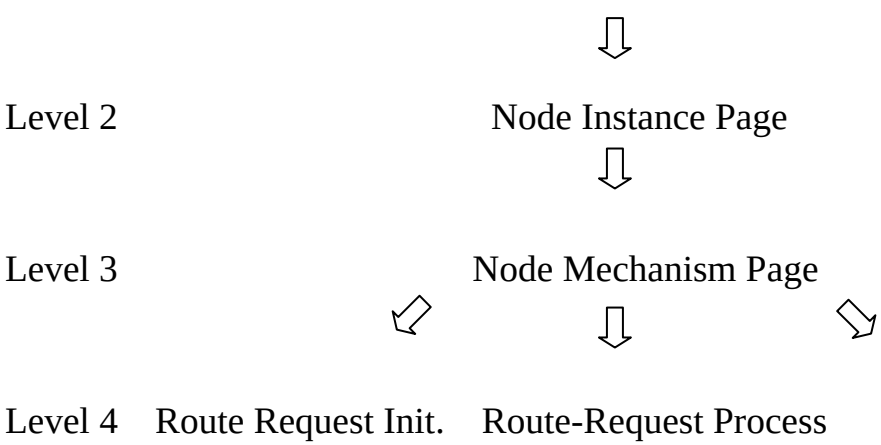
1. Route Discovery
2. Route Maintenance

In this model we have 5 pages which are:-

1. Node Instance Page
2. Node Mechanism Page
3. Route Request Initialization Page
4. Route request Processing Page
5. Route Reply Processing Page

Other Than These 5 pages we have one top level hierarchy Page.

The complete model is divided into 4 level of abstraction



Top Level Page:-

In this page we can declare the instances of various nodes in the network and connect them to a common shared place “STORE” from where they communicate with each other.

To add new node to the network we instantiate a new node in this page and connect it to the “STORE” .In this page every node has its own instance of the other 3 levels of abstraction. So here we can see the number of nodes that are currently available in the network.

Each of the messages that have to be sent or received is stored in a common place “STORE”. Now if a node wants to send a message then it sends its packet or broadcast a definite number of packets and store it in the common place from where based on the CPNTool simulation any node can receive the packet randomly by taking a packet from the store if it satisfies the criteria for the receiver. Similarly if a node wants to send a reply message back a node then it provides a single copy of the packet to the store that can be sent via the path that was determined earlier either by the route cache or by the request packet. We have considered bi-directional communication within the network.

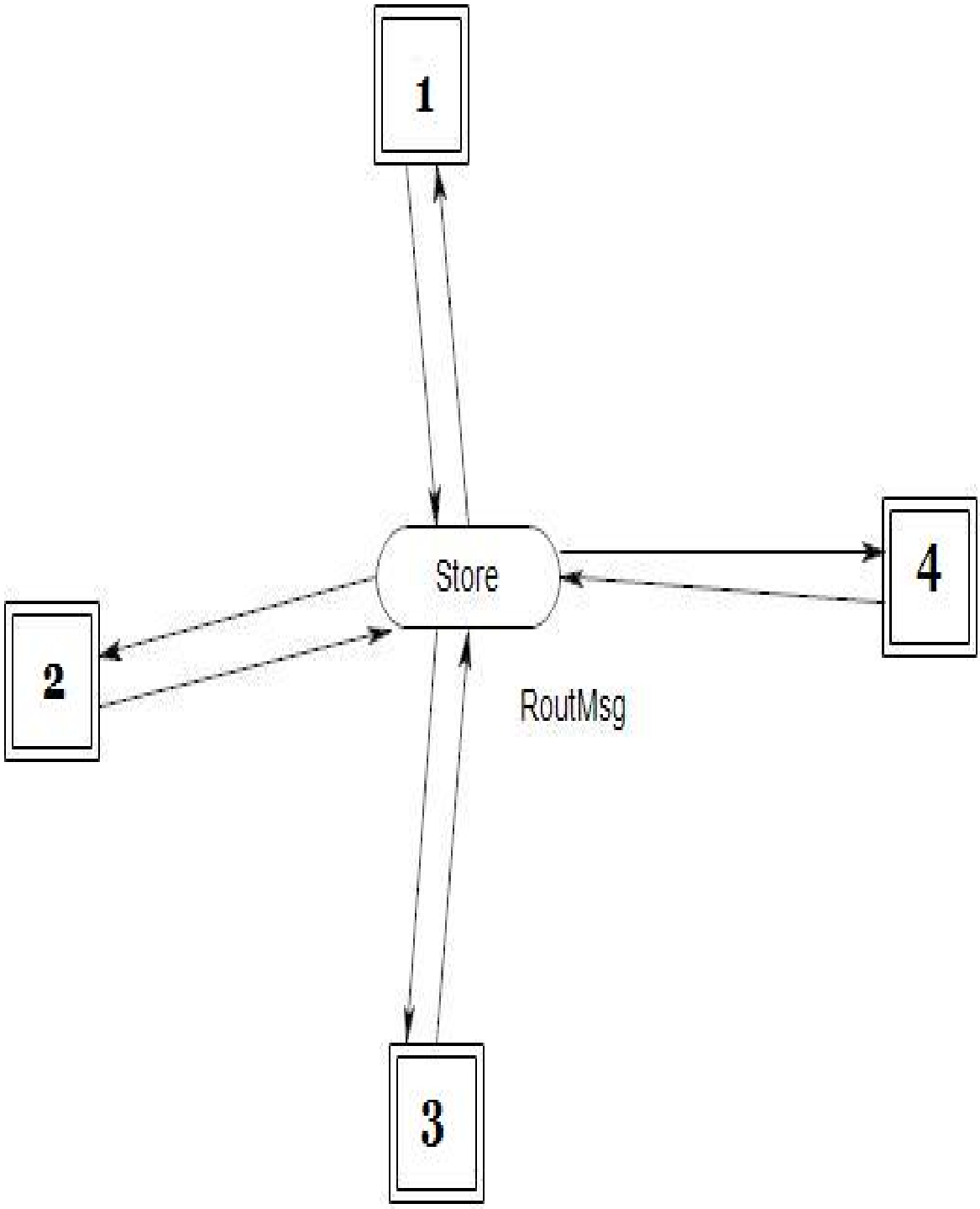


Fig 5

Node Instance Page:-

For each and every node there is one node instance page where there are places that stores the values associated with each node which are namely route cache table, self id, destination id etc.

At this level of abstraction we can actually see the packets that are sent and received by a node at any point of the time. Here we can set parameters for a node to communicate with other nodes. In this page there are two places which are “IN” and “OUT” that acts as a buffer for the packets that are being received and sent. Here also we can see the packets that are present currently in the global “STORE” that is shared by all the nodes present in the network. “Route Cache” is a place to store the routing table information about the network present with the node. Every node has a unique identification number which is used to distinguish a node from others which is stored at a place called “Self id”. For every node the destination node’s identification number is stored at “Destination Id”.

Here we have 2 transitions which are “SEND” and “RECEIVE” respectively. The “RECEIVE” transition has a guard associated with it which ensures that the node is receiving the right packet that is meant for it. The “SEND” transition simply provides the store with multiple copies of the packet if the message has to be broadcasted otherwise in case of direct reply it provides a single copy of the packet meant just for the valid node.

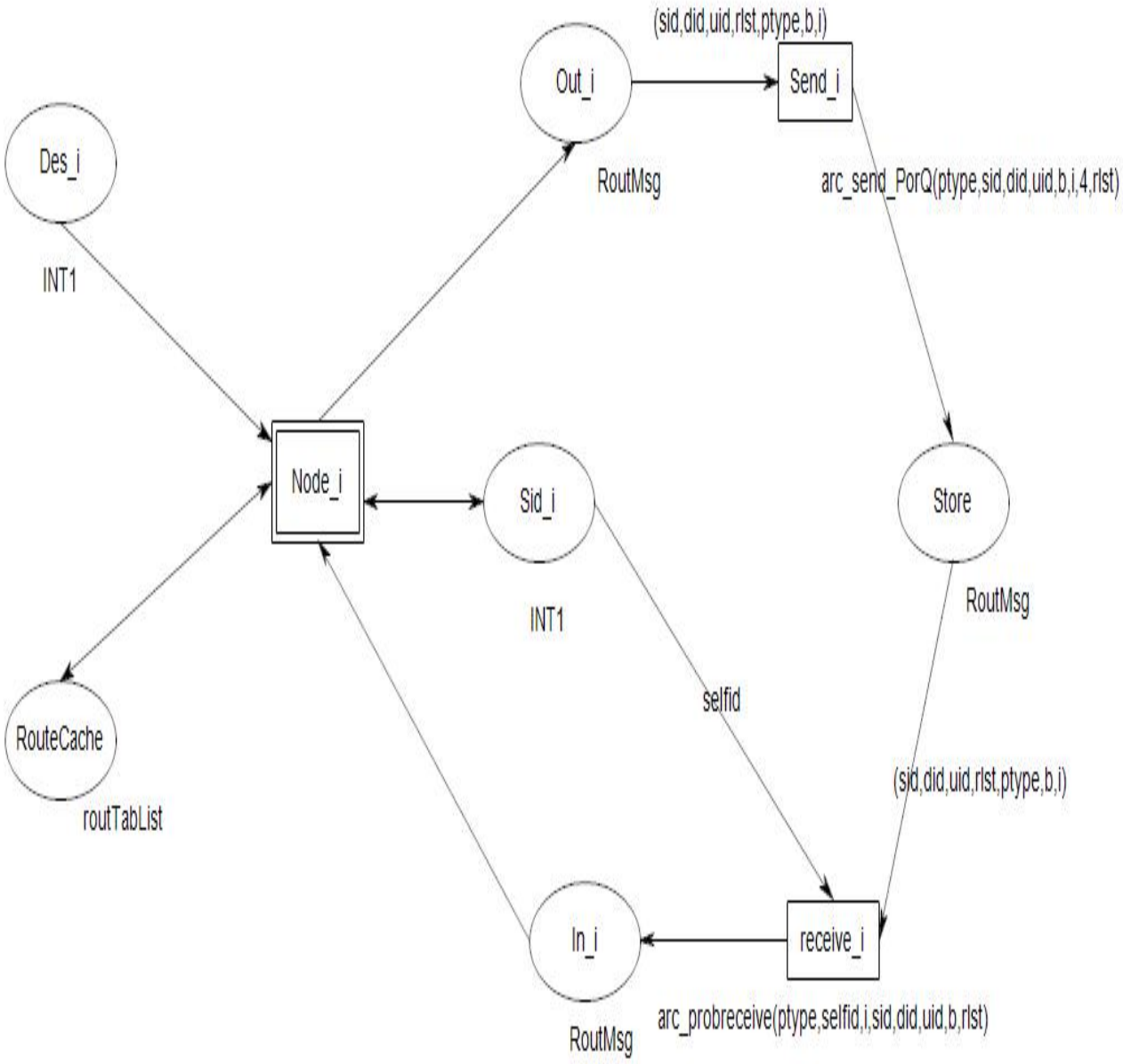

(sid,did,uid,rlst,ptype,b,i)
Out_i
Send_i
Des_i
RoutMsg
arc_send_PorQ(ptype,sid,did,uid,b,i,4,rlst)
INT1
Node_i
Sid_i
Store
INT1
RoutMsg
selfid
RouteCache
(sid,did,uid,rlst,ptype,b,i)
routTabList
In_i
receive_i
RoutMsg
arc_probreceive(ptype,selfid,i,sid,did,uid,b,rlst)

Fig 5.1

Node Mechanism Page:-

This page performs all the functionalities associated with a node that are defined in DSR.

Every node that wants to send packet first checks its own routing table to see if there is already a path available in it which is done by the transition “ROUTE CHECK”. If there is no definite route available in the table then it initialize the route discovery mechanism by the transition “RREQ INIT” which initialize the route request mechanism and broadcasts a RREQ packet in the network asking for a route to the destination node. Once a node receives a RREQ packet it can act on it by the functionality provided by the “RREQ Process” transition. If the destination node gets the broadcasted message then it sends back a RREP packet to the source node by following the path that was followed by the RREQ packet. Once a node receives a RREP packet then it can act on it by using the transition “RREP process”. Also there is a unique id associated with each packet that is sent over at the network so that the same packet is not received again and again by the same node thus avoiding the loops.

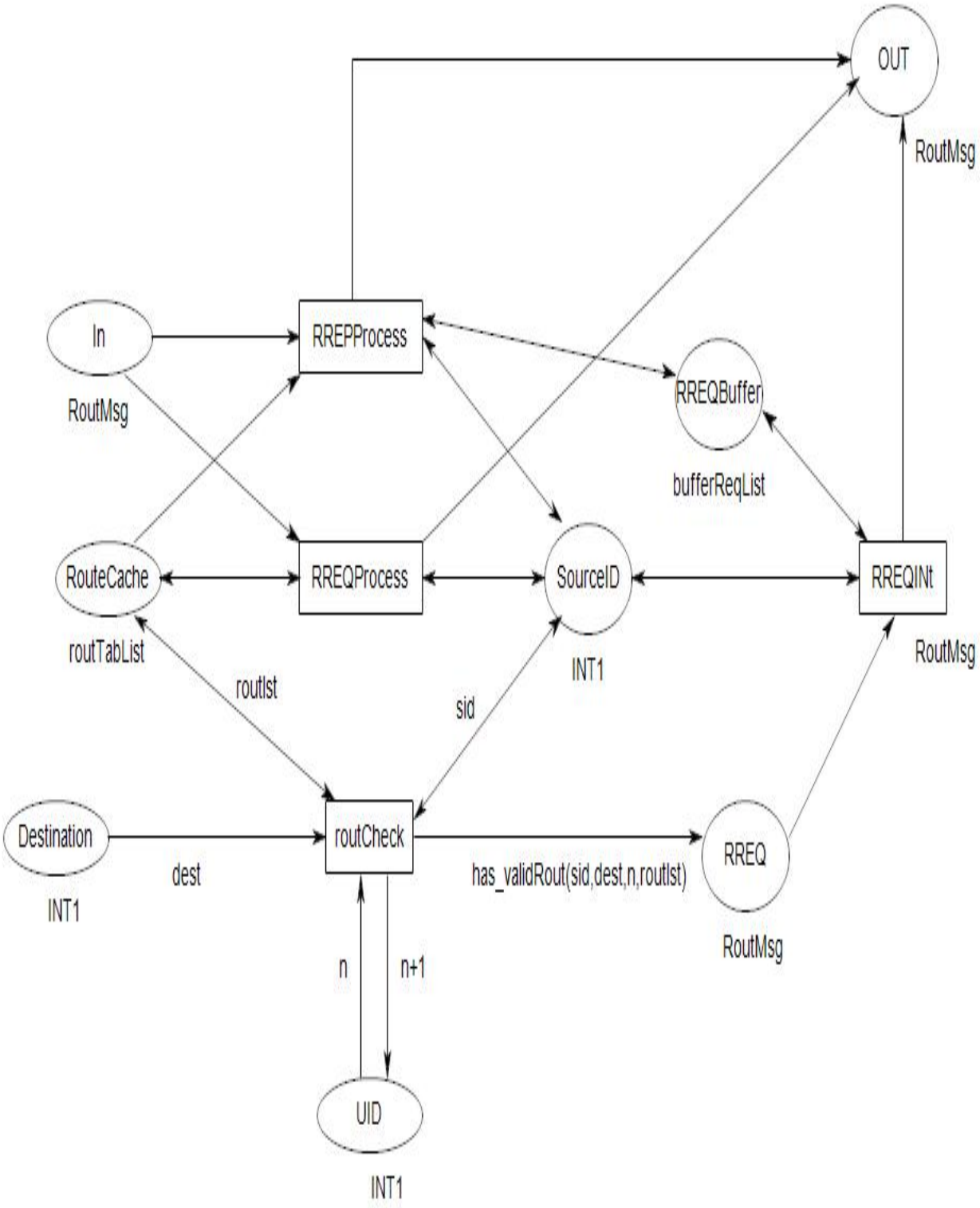

OUT
RoutMsg
In
RREPProcess
RREQBuffer
RoutMsg
bufferReqList
RouteCache
RREQProcess
SourceID
RREQINt
routTabList
routlst
sid
INT1
RoutMsg
Destination
routCheck
RREQ
dest
has_validRout(sid,dest,n,routlst)
INT1
RoutMsg
n
n+1
UID
INT1

Fig 5.2

Route Request Initialization Page:-

As mentioned earlier once the route check returns no valid route in the table it sends a copy of packet to be broadcasted in the “RREQBuffer”. In this page that packet is taken from the RREQ Buffer and is broadcasted in the network and the required number of copies gets stored in the “STORE”. While broadcasting nodes also add its own Identification Number in the route list present with the packet. Also one separate copy of the packet is stored in the “SEND BUFFER” to be rebroadcasted if the timeout occurs which is defined at the place “TIMEOUT”. Also the copy can be rebroadcasted only a limited number of times that is predefined and is stored at “COUNTER”. So if timer expires and the counter value is valid then the same packet is again rebroadcasted into the network.

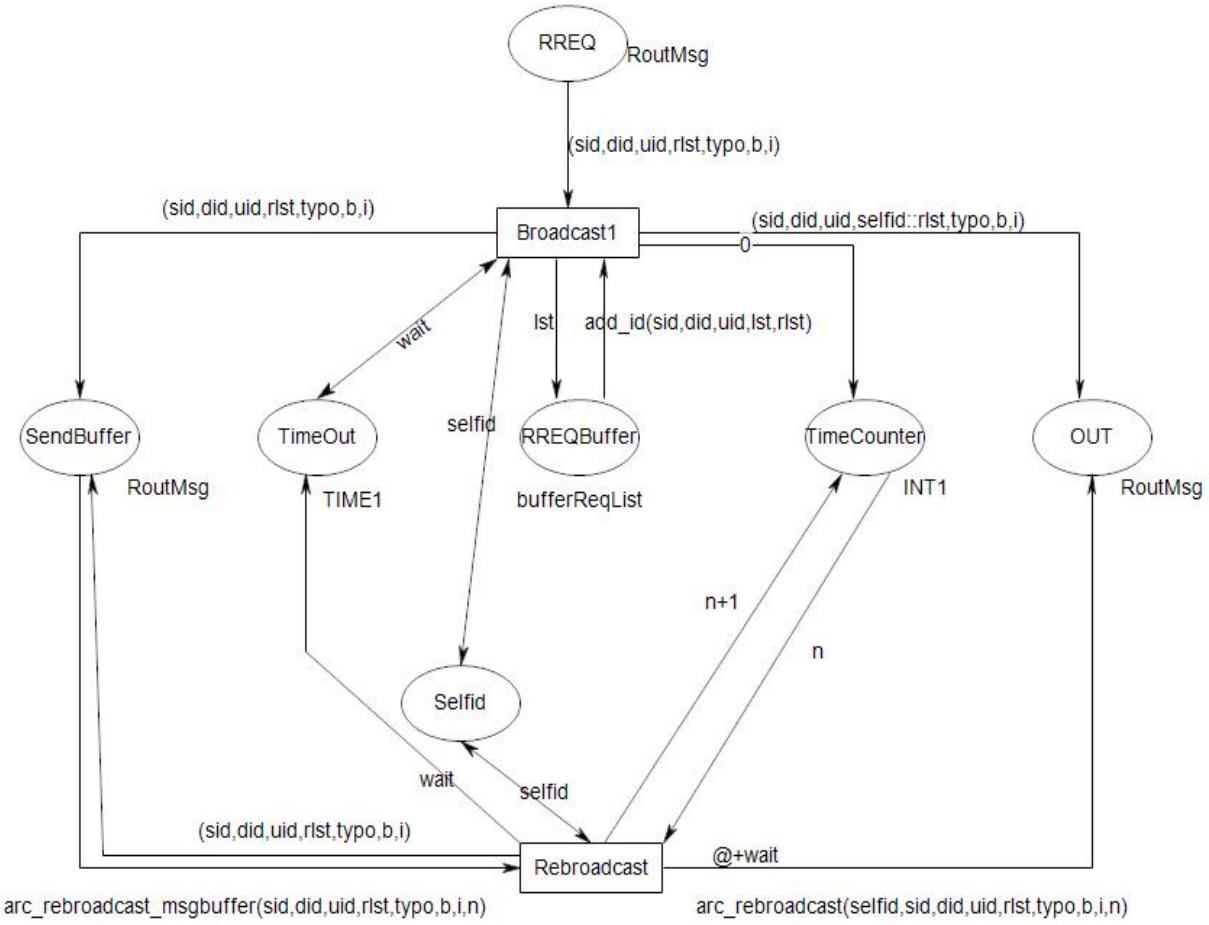


Fig 5.3

Route Request Processing Page:-

Now if a node receives a RREQ packet then it checks, if its own identification number is already in the route list present with the packet, check is performed by the transition “BIDCheck”. If it is not listed then it means that it is coming to this node for the first time so the route table present with the node is checked whether this node is destination or if it is having a route to destination. If this is the destination then a Route Reply packet is sent to the source or if it is having the route to the destination then it appends the route in the route contained with the packet and sends back Route Reply packet to the source. If it is none of the above two then it simple updates the packet’s route list and broadcast it on the network.

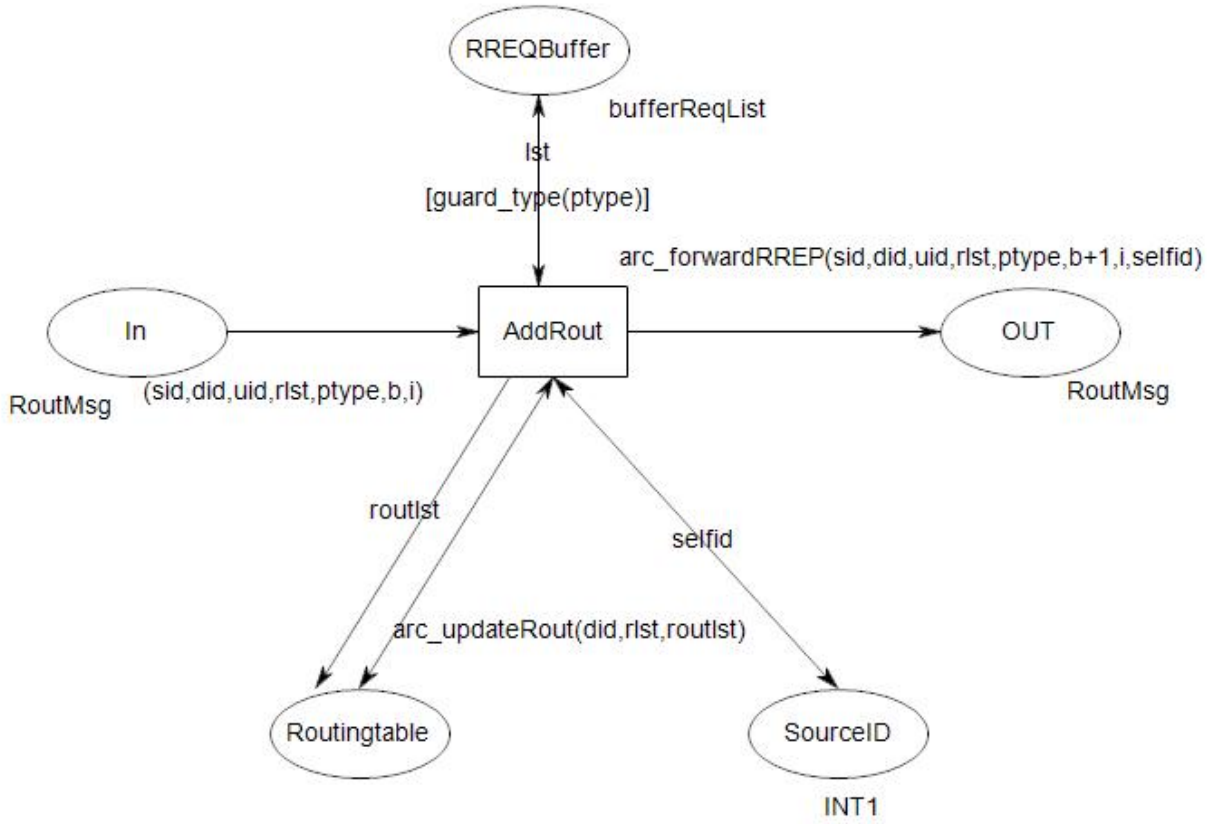


Fig 5.4

Route Reply Processing Page:-

Once a packet reaches its destination or a node that has a valid route to the destination, a route reply packet is sent back to the source retracing the path followed by the packet that is stored in the form of route along with the packet. So if a node receives a route reply then it can be the source or it can be the intermediate node. If it is the intermediate node then it updates its route table with the information contained in the packet and sends the packet to the next node in the path to the source. If it is source then it simple updates its route table and now when it has a valid route to the destination it can send its message to the destination.

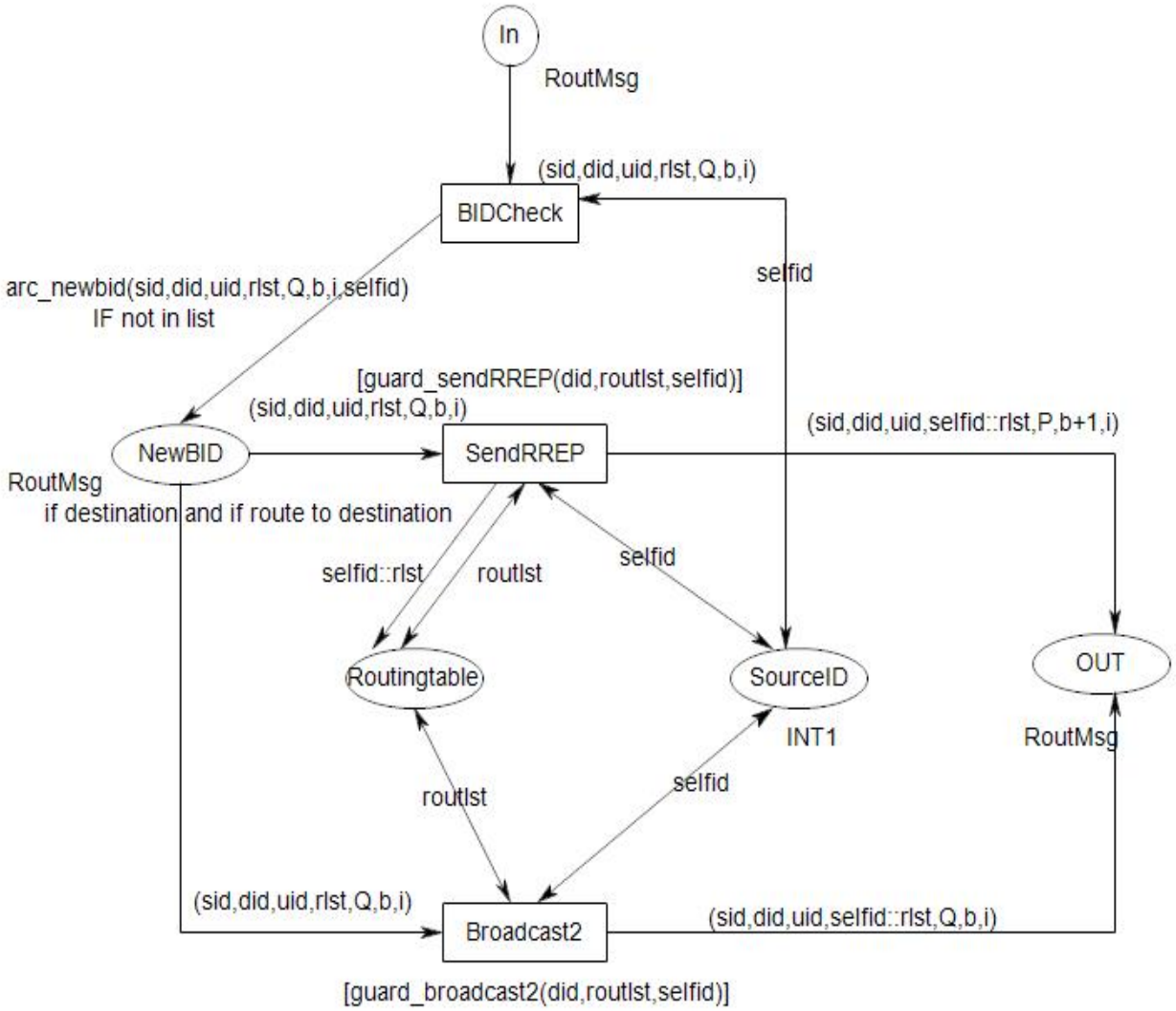


Fig 5.5

Various Colorsets and Variables Used :

```
colset INT1 = int;
colset SID = int;
colset DID = int;
colset UID=int;
colset PID= int;
colset TYPE = with P|Q;
colset TIME1= int timed;
colset lstt=list INT1;
var selfid:INT1;
colset BufferREQ= product SID*DID*UID*lstt;
colset bufferReqList = list BufferREQ;
colset RoutMsg=product SID*DID*UID*lstt*TYPE*PID*INT1;
var lst : bufferReqList;
var rlst:lstt;
var typo:TYPE;
colset RoutTable = lstt ;
colset routTabList = RoutTable;
var routlst : routTabList;
var wait : TIME1;
var sid,did,uid: INT1;
var dest,n: INT1;
var ptype : TYPE;
var b,s,h,p:INT1;
var msg,routMsg :RoutMsg;
var i :INT1;
```

Snapshots of DSR protocol while running in Colored Petrinet:

Node Instance

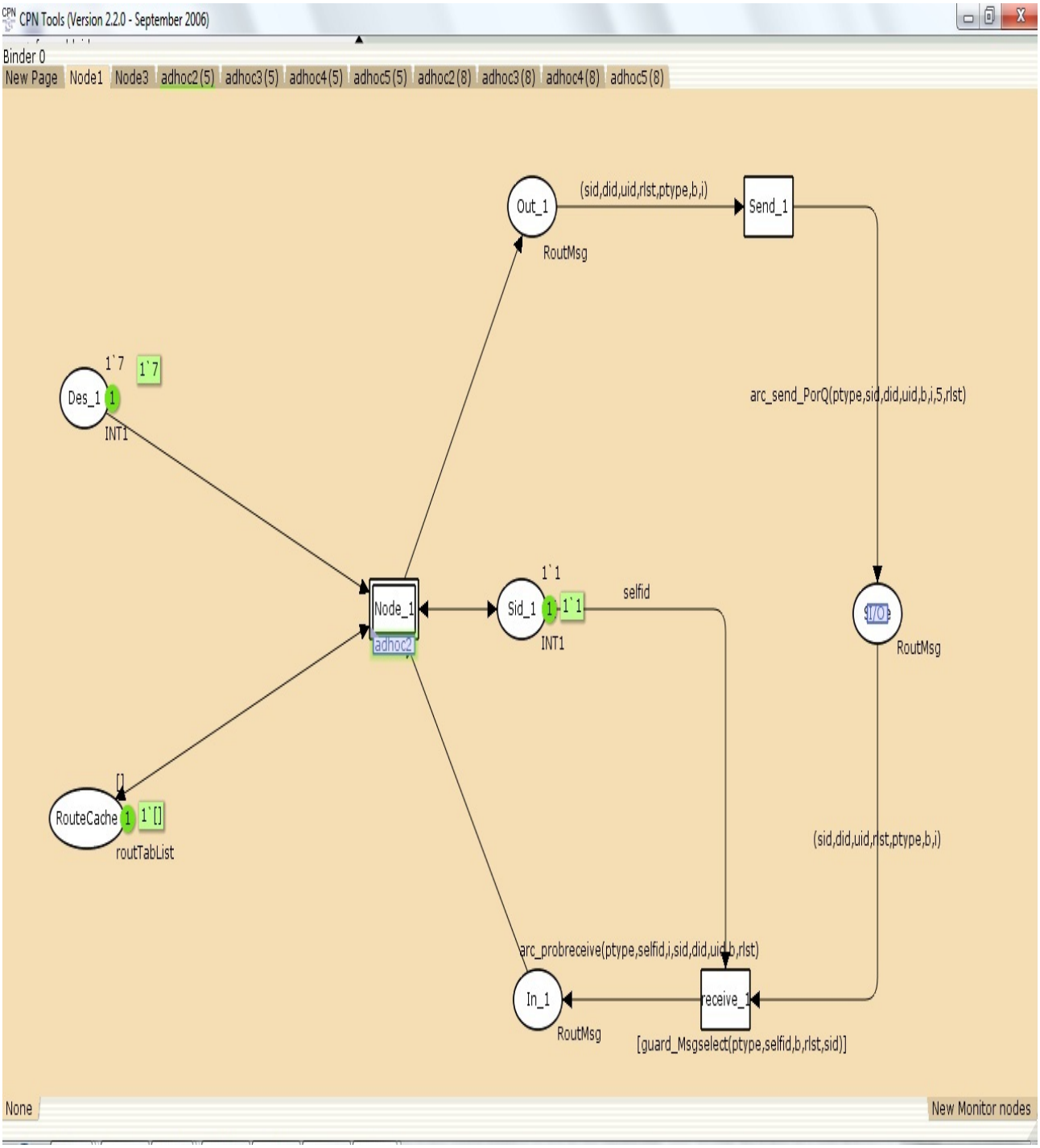


Fig 5.6

Node Mechanism

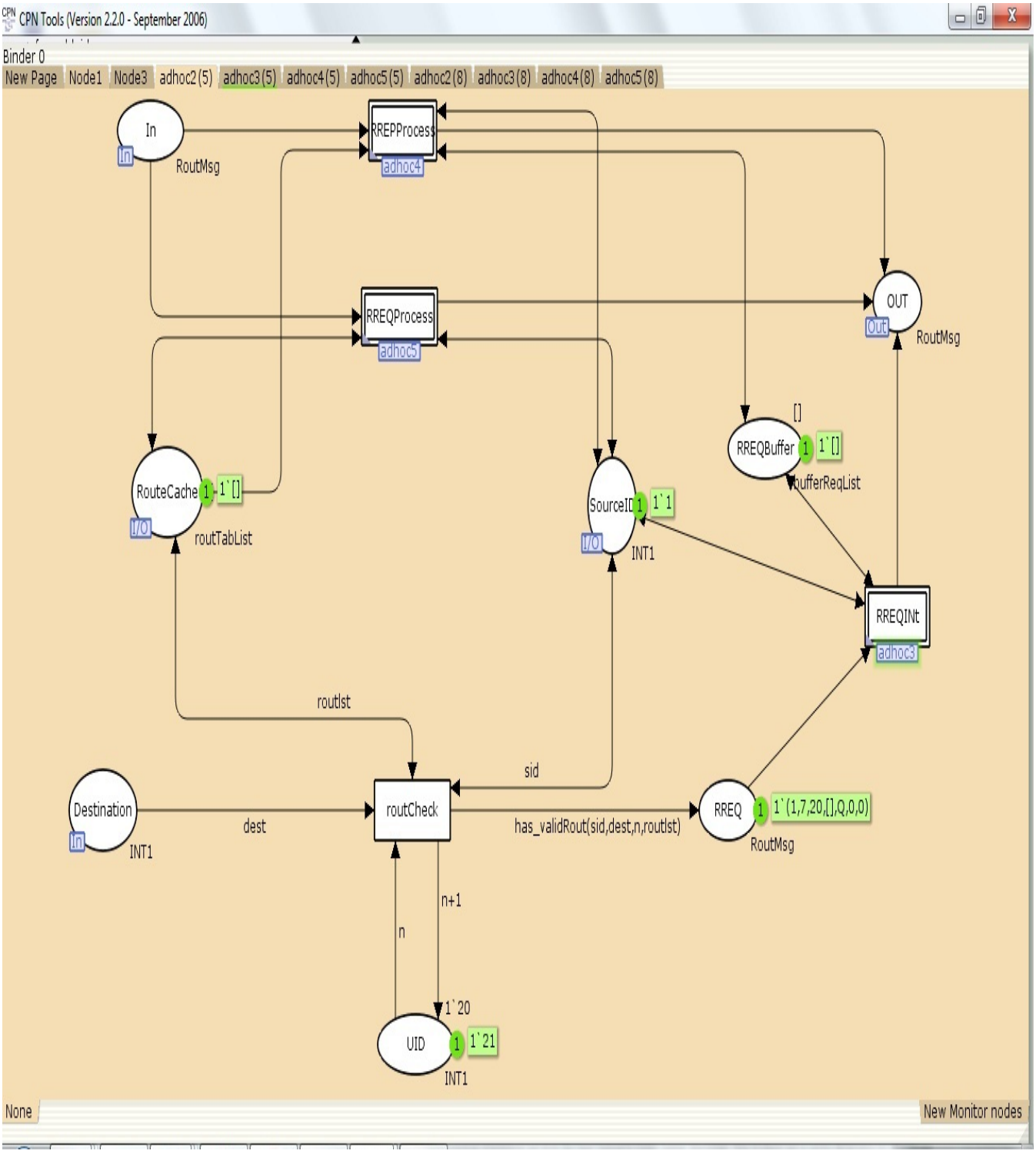


Fig 5.7

RREPProcess

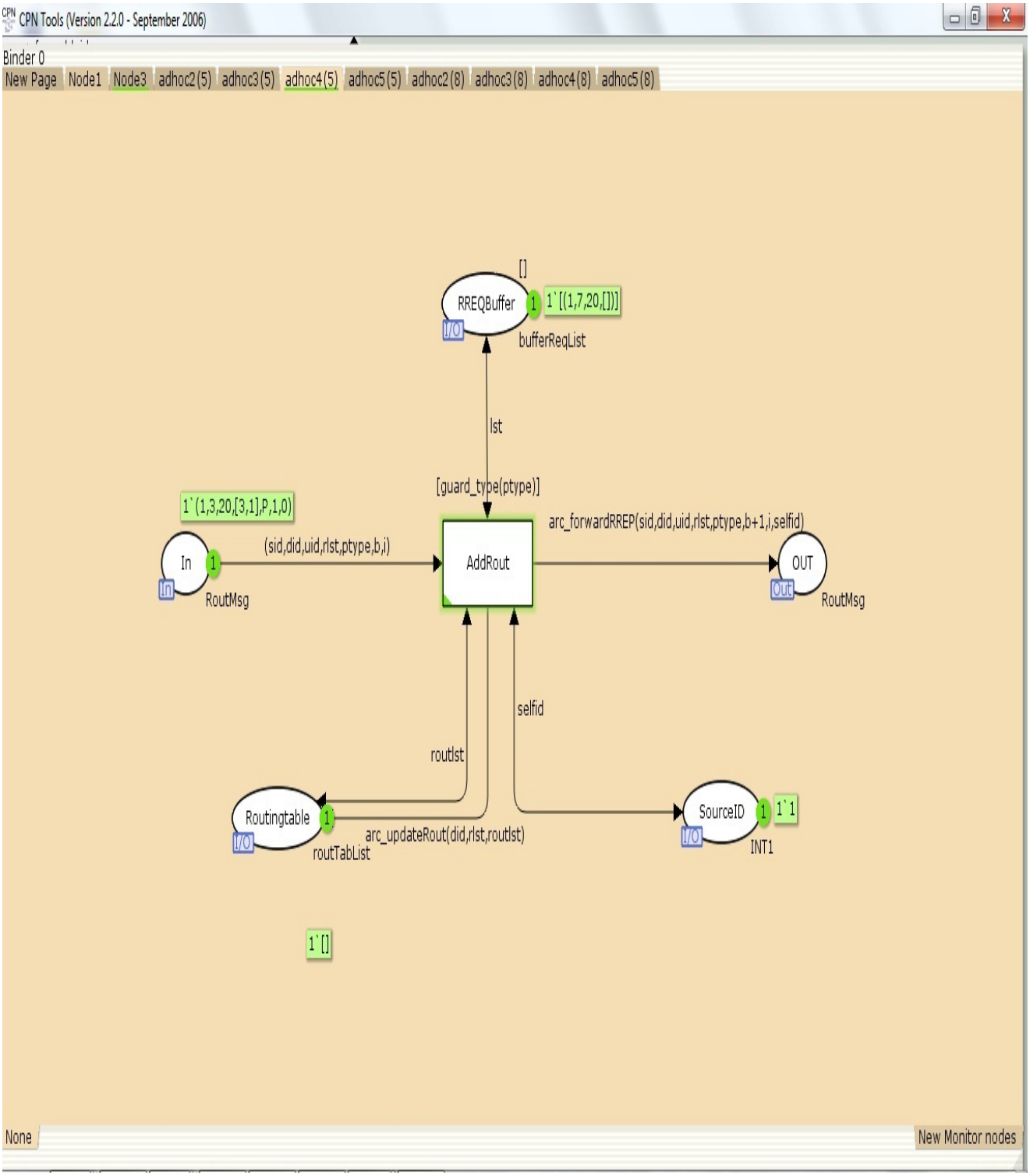


Fig 5.8

RREQInit

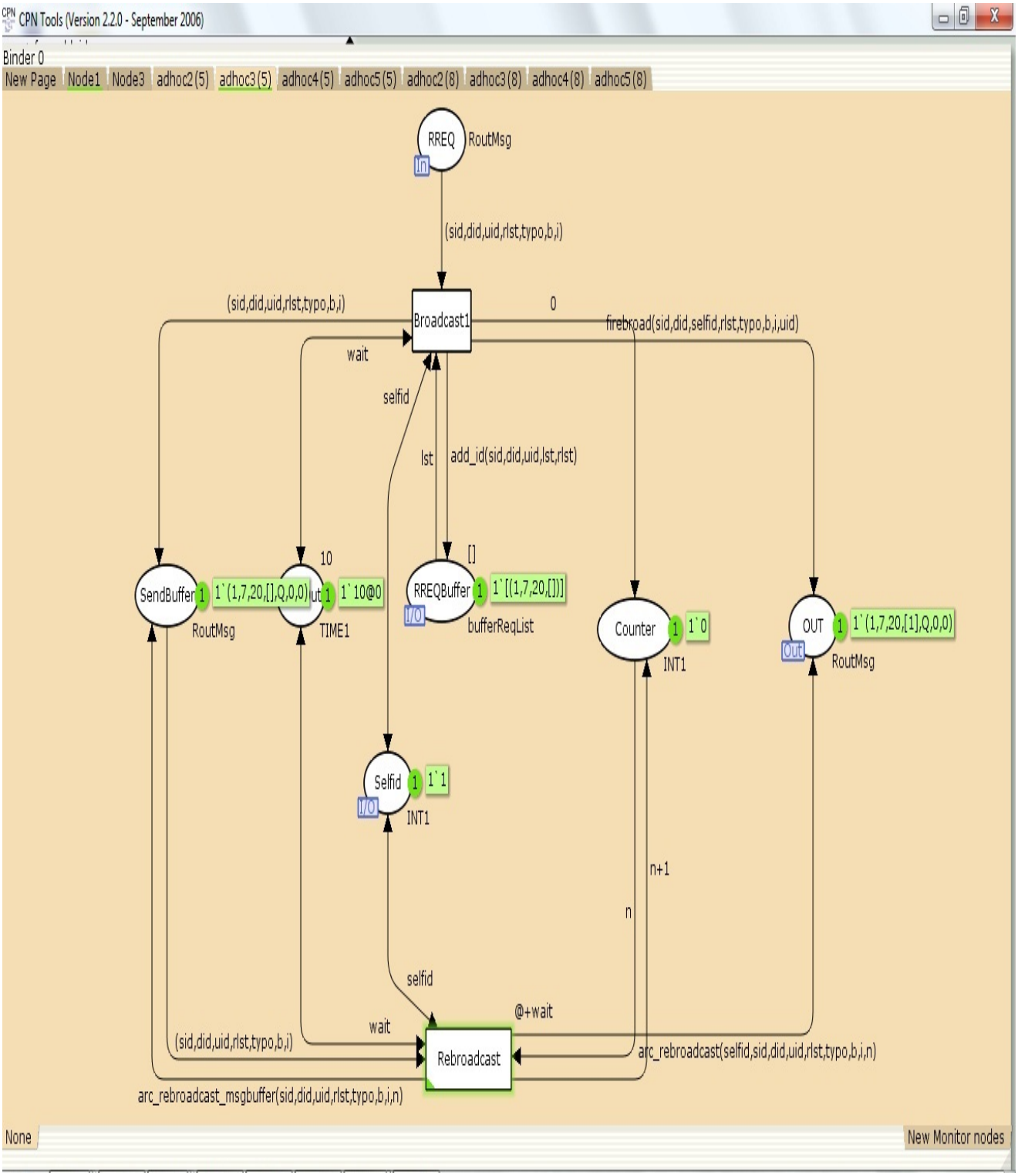


Fig 5.9

RREQProcess

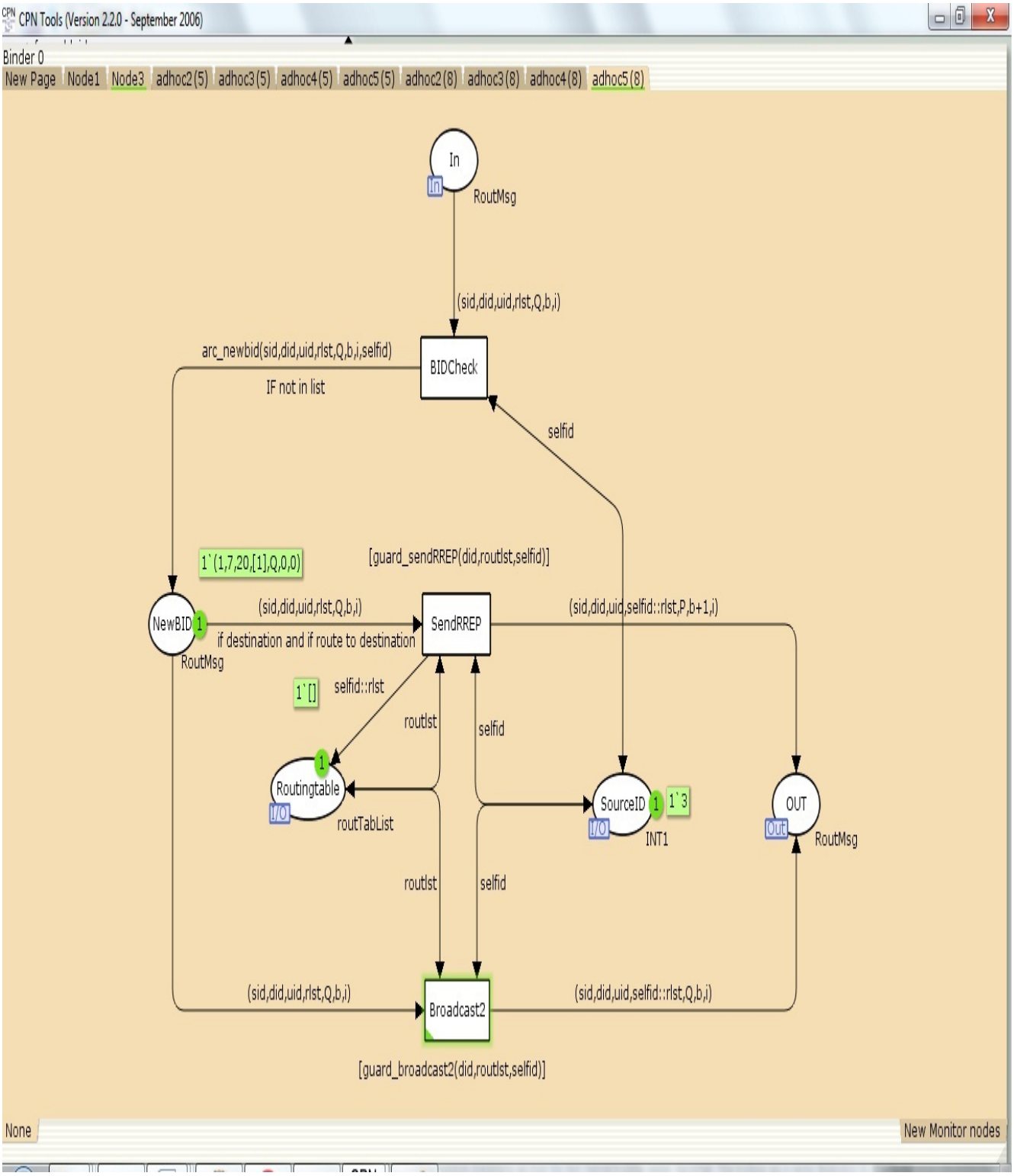


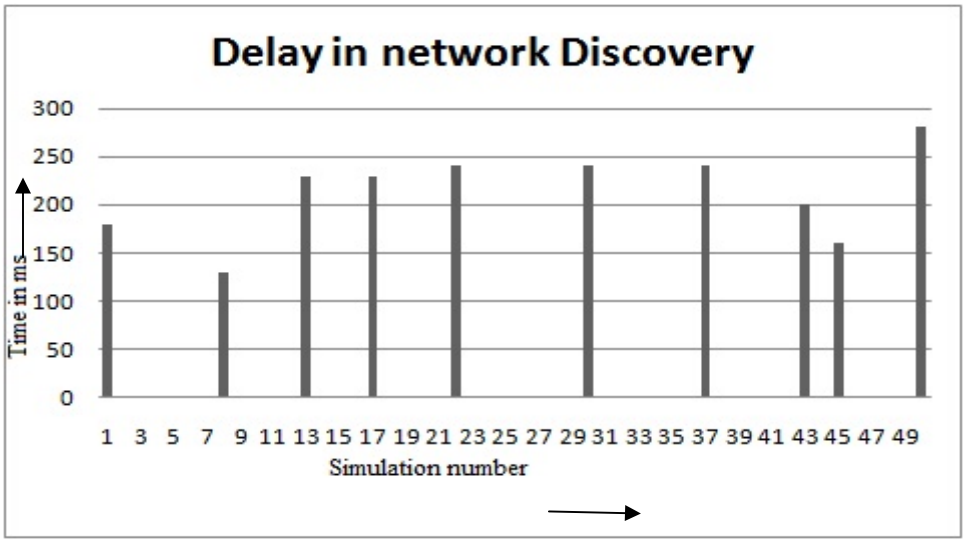
Fig 5.10

CHAPTER 6

Comparison Study & Conclusion

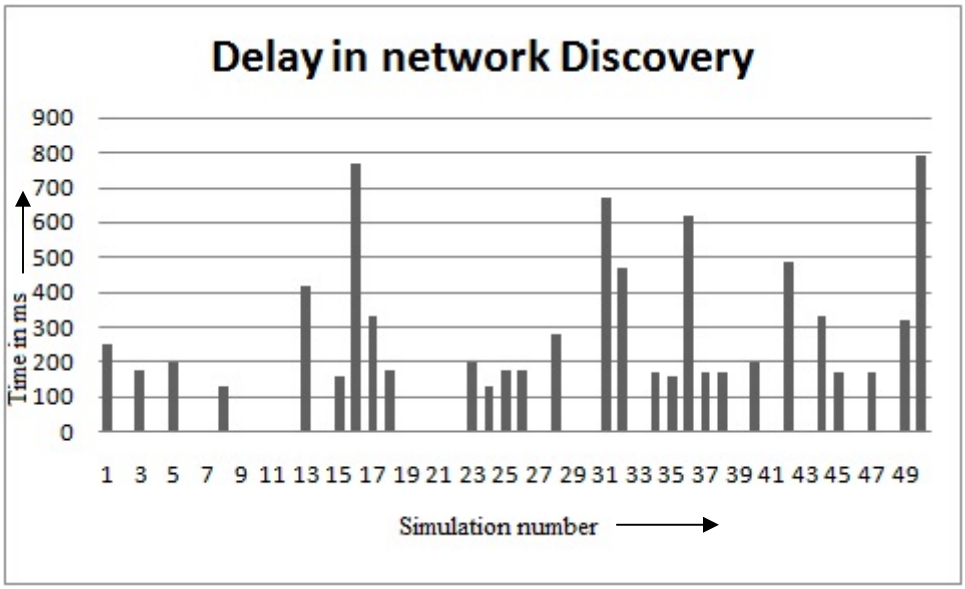
6.1 Comparison of the AODV and DSR algorithm Based on the simulation:-

AODV:-



1. Throughput:-20%
2. Average Delay in Network Discovery:- 209ms.

DSR:-



1. Throughput:-56%
2. Average Delay in Network Discovery:- 303ms.

6.2 Conclusion:-

So from above results it can be concluded that DSR shows better efficiency with lesser number of nodes present in the network but has a higher delay in network discovery which can be accounted due to the source route present in every packet. While AODV has a low delay in network discovery for a small network so based on these factors there is a trade-off between delay due to network discovery and efficiency of the network or throughput.

CHAPTER 7

Bibliography

[1]. Chaoyue Xiong, Tadao Murata, Jeffery Tsai, "*Modeling and Simulation of Routing Protocol for Mobile Ad Hoc Networks Using Colored Petri Nets*", Department of Computer Science, University of Illinois at Chicago,USA,2005.

[2]. K. Jensen, L. M. Kristensen and Lisa Wells "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems" Springer-Verlag. Textbook, in preparation.

[3]. CPN Tools. http://wiki.daimi.au.dk/cpntools/cpntools.wiki

[4]. David B. Johnson David A. Maltz Josh Broch "DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks" , Department of Computer Science, University of Carnegie Mellon University, Pittsburgh, PA 15213-3891.

[5]. Lars M. Kristensen, S_ren Christensen, Kurt Jensen. "The practitioner's guide to coloured Petri nets". CPN Group, Department of Computer Science, University of Aarhus, Denmark, 1998.

[6]S.Basagni, M.Conti, S.Giordno, I.Stojmenovic. "Mobile Ad-HOC networking" by Wiley-Interscience.

[7]. http://en.wikipedia.org/wiki/Petri_net#Petri_nets_basics

[8] K. Jensen. "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use"*. Volume 1&2: Basic Concepts. Springer-Verlag, 1992.

Appendix

```
1) fun intTime() =
IntInf.toInt (time());

globref fileid = (NONE : TextIO.outstream option);

fun getfid () =
      (* this will raise Option exception
       * if fileid = NONE *)
   Option.valOf(!fileid)

fun initfile () =
let
   val filename = OS.Path.concat (Output.getSimOutputDir(),
"Log_File")
in
  Output.initSimOutputDir();
  fileid := SOME (TextIO.openOut filename)
end

2)fun expTime (mean: int) =
  let
 val realMean = Real.fromInt mean
 val rv = exponential((1.0/realMean))
  in
    floor (rv+0.5)
  end;

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

4)fun guard_type(typo)=
if(typo=P)
then true
else false;

5)fun get_Pid(bid,sid,[])=false|get_Pid(bid,sid,(b,s,h,p)::rest)=
if(bid=b andalso sid=s) then p
else get_Pid(bid,sid,rest);

6)fun add_id(sid,did,uid,[],rlst)=[(sid,did,uid,rlst)]
| add_id(sid,did,uid,lst,rlst)=
(sid,did,uid,rlst)::lst;
```

```
7)fun see1(did,rlst)=
if(mem rlst did)
then false
else true;

8)fun has_validRout(sid,did,n,rlst)=
if(rlst=[])
then 1`(sid,did,n,rlst,Q,0,0)
else if(see1(did,rlst))
then  1`(sid,did,n,rlst,Q,0,0)
else
empty;

9)fun fin(rlst,did)=
if(List.nth(rlst,(length rlst))=did)
then true
else
false;

10)fun see(did,rlst)=
if(mem rlst did)
then true
else false;

11)fun take(routlst)=
List.nth(routlst,1);

12)fun guard_sendRREP(did,routlst,selfid)=
if(did=selfid)
then true
else if(see(did,routlst))
then true
else false;

13)fun guard_broadcast2(did,routlst,selfid)=
(did<>selfid andalso see1(did,routlst));

14)fun arc_updateRout(did,rlst,routlst)=
if(see(did,rlst))
then routlst++(rev rlst)
else routlst++(rev (did::rlst));

15)fun arc_forwardRREP(sid,did,uid,rlst,ptype,b,i,selfid)=
if(sid<>selfid)
then 1`(sid,did,uid,rlst,P,b,i)
else empty;
16)fun arc_newbid(sid,did,uid,rlst,typo,b,i,selfid)=
```

```
if(mem rlst selfid) then empty
else 1`(sid,did,uid,rlst,typo,b,i);

17)fun arc_rebroadcast(selfid,sid,did,uid,rlst,typo,b,i,n)=
if(n<3 andalso see1(selfid,rlst))
then 1`(sid,did,uid,selfid::rlst,typo,b,i)
else
empty;

18)fun arc_rebroadcast_msgbuffer(sid,did,uid,rlst,typo,b,i,n)=
if(n<3)
then 1`(sid,did,uid,rlst,typo,b,i)
else
empty;

19)fun getid(i,rlst)=
List.nth(rlst,i);

20)fun firebroad(sid,did,selfid,rlst,typo,b,i,uid)=
if(see1(selfid,rlst))
then 1`(sid,did,uid,selfid::rlst,typo,b,i)
else
empty;

21)fun guard_Msgselect(ptype,selfid,b,rlst,sid)=
(ptype=P andalso selfid=getid(b,rlst))
orelse
(ptype=Q andalso sid<>selfid andalso see1(selfid,rlst));

22)fun arc_probreceive(ptype,selfid,i,sid,did,uid,b,rlst)=
if(ptype=P)
then
1`(sid,did,uid,rlst,ptype,b,i)
else if(ptype=Q andalso sid<>selfid)
then 1`(sid,did,uid,rlst,ptype,b,i)
else empty;


23)fun arc_send_PorQ(ptype,sid,did,uid,pid,i,MAXNeighbor,rlst)=
if(ptype=Q andalso 62>discrete(0,100))
then MAXNeighbor`(sid,did,uid,rlst,ptype,pid,i)
else if(ptype=P andalso 62>discrete(0,100))
then 1`(sid,did,uid,rlst,ptype,pid,i)
else empty;
```