Routing Protocol Using Game Theoretic Approach

By

Varma Jitendra G.
(05MCE019)

Department Of Computer Science & Engineering

Institute Of Technology

Nirma University Of Science & Technology

Ahmedabad 382481

May 2007

Routing Protocol Using Game Theoretic Approach

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Varma Jitendra G.
(05MCE019)

Guide
Dr. S.N. Pradhan
Prof. Gaurang Raval

Department Of Computer Science & Engineering

Institute Of Technology

Nirma University Of Science & Technology

Ahmedabad 382481

May 2007

NIRMA UNIVERSITY
INSTITUTE OF TECHNOLOGY

This is to certify that Dissertation entitled

Routing Protocol Using Game Theoretic Approach

Submitted by

Varma Jitendra G.

has been accepted toward fulfillment of the requirement

for the degree of

Master of Technology in Computer Science & Engineering

Prof. (Dr.) S. N. Pradhan	Prof. D. J. Patel
Professor In Charge	Head of The Department

Prof. A. B. Patel
Director, Institute of Technology

CERTIFICATE

This is to certify that the Major Project entitled "Routing Protocol Using Game Theoretic Approach" submitted by Mr. Varma Jitendra G. (05MCE019), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science And Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof.(Dr.) S. N. Pradhan
Guide,
Department of Computer Science &
Engineering,
Institute of Technology,
Nirma University,
Ahmedabad.

Prof. Gaurang Raval
Guide,
Assistance Professor
Department of Information Technology,
Institute of Technology,
Nirma University ,
Ahmedabad.

ACKNOWLEDGEMENT

Behind every successful venture, lie the joint efforts put in by more than one person, co-operatively working as a team, although, they may be involved in the process either directly or indirectly. No one can proceed towards achieving success without the fruitful guidance of the mentors and co-ordinates. Thus, one should never forget their contribution in the success of any task, either big or small.

Hence, I would first of all like to express my humble gratitude towards the mentor and guide of my project dissertation Dr. S. N. Pradhan (Senior Professor and M.Tech. Co-ordinator) who have always inspired all of us to put in the maximum of our efforts into the dissertation work. They have always been willingly helpful to provide the necessary facilities and also to solve our problems, which may have occurred during our ongoing project work.

My sincerest thanks to my guide Prof. Gaurang Raval sir to continuously guide me in my project area and to provide me with all the necessary resources and material which have been extremely useful in my dissertation study. I am highly obliged to him for his constant overwhelming support.

Secondly, I would like to thank Prof. A. B. Patel (Director, NIT) and Prof. D. J. Patel (H.O.D., CSE Dept.) for supervising the entire dissertation program and organizing meetings in order to receive feedback from students as well as the staff-in-charge regarding the problems faced in the program and their efforts to solve them to their best.

Last, but not at all the least, I thank all the people, either directly or indirectly related, involved or concerned in helping me with the project dissertation work through their moral support; be it my family, my classmates or my friends.

— JITENDRA VARMA —
(05MCE019)

ABSTRACT

A recent trend in routing research is to avoid inefficiencies in network level routing by allowing hosts to either choose routes themselves (*e.g.*, source routing) or use overlay routing networks. Such end-to-end route selection schemes are selfish by nature; in that they allow end users to greedily select routes to optimize their own performance without considering the system-wide criteria. Recent theoretical results suggest that in the worst case, selfish routing can result in serious performance degradation due to lack of cooperation. So, the objective of this dissertation work is to apply Game Theoretic Approach on routing protocols for removing inefficiencies and analyzing how it is better than conventional TCP/IP routing protocols.

Game theory is a branch of applied mathematics, which deals with multiperson decision making situations. A routing protocol can be modeled as a minimax game between the network and the routers. And it is analyzed based on different parameters like Delay, Jitter, Processing power etc.

Network Simulator 2 tool is used for coding and simulation purpose of conventional routing protocol and minimax algorithm. rtProtoMIN - a new routing protocol is implemented in NS2 which uses minimax algorithm for finding the path from source to destination. The rtProtoDV (RIP) protocol available in NS2 and the newly implemented rtProtoMIN are being used in this project to analyze their performance in different scenarios and are compared to the already existing routing protocols.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 GENERAL

1.1.1 Routing Protocols

In computer networking the term **routing** refers to selecting paths in a computer network along which to send data. Routing directs forwarding, the passing of logically addressed packets from their source toward their ultimate destination through intermediary nodes (called routers). It facilitate the exchange of routing information between networks, allowing routers to build routing tables dynamically. Traditional IP routing stays simple because it uses **next-hop routing** where the router only needs to consider where it sends the packet, and does not need to consider the subsequent path of the packet on the remaining hops[8]. Traditional IP routing stays simple because it uses **next-hop routing** where the router only needs to consider where it sends the packet, and does not need to consider the subsequent path of the packet on the remaining hops.

1.1.2 Game Theory

Game theory is a branch of applied mathematics, which deals with multiperson decision making situations. The basic assumption is that the decision makers pursue some well defined objectives and take into account their knowledge or expectations of the other decision makers' behavior. Many applications of game theory are related to economics, but it has been applied to numerous fields ranging from law enforcement to voting decisions in European Union [5]. There are two main ways to capitalize game theory. It can be used to analyze existing systems or it can be used as a tool when designing new systems. Existing systems can be modeled as games. The models can be used to study the properties of the systems. For example, it is possible to analyze the effect of different kind of users on the system. The other approach is implementation theory, which is used when designing a new system. Instead of fixing a game and analyzing its outcome, the desired outcome is fixed and a game ending in that outcome is looked for. When a suitable game is discovered, a system fulfilling the properties of the game can be implemented.

1.1.3 Usage

Game theory was invented to provide a mathematical foundation for reasoning about conflict and competition. It has grown into a rich theory, with powerful mathematical and computational tools. It also has the advantage of retaining its intuitive appeal, which is what first attracted us to it. Two insights enabled us to turn this pool of theory and tools into a potentially powerful analytical capability for analyzing routing [3].

The game theoretic techniques are applicable to the analysis of routing protocols. The particular way in which we have applied game theory to this problem is described in following chapters.

1.2 SCOPE OF WORK

The objective of this dissertation work is to apply Game Theoretic approach on routing protocols for removing inefficiencies and analyzing how it is better than conventional TCP/IP routing protocols. Network Simulator 2 tool is used for coding and simulation purpose of conventional routing protocol and minimax algorithm. RIP (rtProtoDV) routing protocol already implemented in NS2, game theory approach is applied on that and see how it improves the rtProtoDV. RtProtoDV uses distance vector technique for finding path to destination node but, here DV is replaced by minimax algorithm which used for finding path to destination node. RtProtoMIN - a new routing protocol is implemented in NS2 which uses minimax algorithm for finding the path from source to destination.

These two protocols rtProtoDV and rtProtoMIN are analyzed on different parameters like delay jitter, delay, convergence etc. These protocols are simulated on different scenarios like varying packet size and fixed network, fixed packet size and varying networks, convergence etc and analyzed how game theory improves rtProtoDV.

1.3 ORGANIZATION OF MAJOR PROJECT

Chapter 2: This chapter contains literature survey and also gives the background information about game theory which is used for understanding how game is modeled as routing and basics of TCP/IP routing protocols.

Chapter 3: This chapter gives the strategy; how routing is to be modeled as game and minimax algorithm is used for finding the path from source to destination.

Chapter 4: This gives the whole information regarding the simulation environment, NS2 (Network Simulator 2) which is used for coding and simulation purpose.

Chapter 5: In this chapter rtProtoDV (RIP) protocol which is already implemented in NS2 is simulated and inefficiency of that protocol is shown.

Chapter 6: This chapter gives the implementation details of rtProtoMIN protocol in that routing is modeled as game. It gives the detailed information regarding data packet, control packet, classes and structure which is used for implementing rtProtMIN.

Chapter 7: In this chapter rtProtoMIN and rtProtoDV is simulated on different scenarios and comparison is made between them and observed which protocol give the better performance and how game theory improves the performance of RIP.

Chapter 8: This chapter contains summary of whole dissertation, conclusion of it and future work.

Appendix A: This contains the pesudocode of minimax algorithm and discusses implementation details of it.

Appendix B: This gives basic information regarding AWK scripts.

2 LITERATURE SURVEY

2.1 BASICS OF GAME THEORY

This section introduces the basic concepts of game theory. The aim is to supply sufficient information to understand the applications in this thesis work. The most common types of games and their solutions are presented.

2.1.1 Introduction

Game theory is a branch of applied mathematics, which deals with multiperson decision making situations. The basic assumption is that the decision makers pursue some well defined objectives and take into account their knowledge or expectations of the other decision maker's behavior. Many applications of game theory are related to economics [7], but it has been applied to numerous fields ranging from law enforcement to voting decisions in European Union [5]. There are two main ways to capitalize game theory. It can be used to analyze existing systems or it can be used as a tool when designing new systems. Existing systems can be modeled as games. The models can be used to study the properties of the systems. For example, it is possible to analyze the effect of different kind of users on the system. The other approach is implementation theory, which is used when designing a new system. Instead of fixing a game and analyzing its outcome, the desired outcome is fixed and a game ending in that outcome is looked for. When a suitable game is discovered, a system fulfilling the properties of the game can be implemented.

Most game theoretical ideas can be presented without mathematics, hence only give some formal definitions. The prisoner's dilemma (classical games) which is used to demonstrate the concepts of game theory [5].

2.1.2 Prisoner's Dilemma

In the prisoner's dilemma, two criminals are arrested and charged against a crime. The police do not have enough evidence to convict the suspects, unless at least one confesses. The criminals are in separate cells, thus they are not able to communicate during the process. If neither confesses, they will be convicted of a minor crime and sentenced for one month. The police offer both the criminals a deal. If one confesses and the other does not, the confessing one will be released and the other will be sentenced for 9 months. If both confess, both will be sentenced for six months. The possible actions and corresponding sentences of the criminals are given in table 2.1.

Table 2.1: Prisoner's Dilemma

		Criminal 2	
		Don't confess	Confess
Criminal 1	Don't confess	$(-1,-1)$	$(-9,0)$
	Confess	$(0,-9)$	$(-6,-6)$

2.1.3 Assumptions And Definitions

2.1.3.1 Game

A game consists of players, the possible actions of the players, and consequences of the actions. The players are decision makers, who choose how to act. The actions of the players result in a consequence or outcome. The players will try to ensure the best possible consequence according to their preferences [7]. The preferences of a player can be expressed either with a utility function, which maps every consequence to a real number, or with preference relations, which define the ranking of the consequences. With mild assumptions, a utility function can be constructed if the preference relations of a player are known.

2.1.3.2 Rationality

The most fundamental assumption in game theory is rationality. Rational players are assumed to maximize their payoff. If the game is not deterministic, the players maximize their expected payoff. The idea of maximizing the expected payoff was justified by the seminal work of von Neumann and Morgenstern in 1944 [5]. It is also assumed that the players are intelligent, which means that they know everything what we know about the game and they can make the same deductions about the situation that we can make.

2.1.3.3 Solution

In game theory, a solution of a game is a set of the possible outcomes. A game describes what actions the players can take and what consequences of the actions are. The solution of a game is a description of outcomes that may emerge in the game if the players act rationally and intelligently [7]. Generally, a solution is an outcome from which no player wants to deviate unilaterally. Solutions to some game types are presented in later sections.

2.1.4 Classification Of Games

Games can be classified into different categories according to their properties [5, 6, 7]. The terminology used in game theory is inconsistent, thus different terms can be used for the same concept in different sources.

2.1.4.1 Noncooperative and cooperative games

Games can be divided into noncooperative and cooperative games according to their focus. Cooperative games are also called coalition games. In noncooperative games, the actions of the single players are considered. Correspondingly, in coalition games the joint actions of groups are analyzed, i.e. what is the outcome if a group of players cooperate. The interest is in what kind of coalitions form. The prisoner’s dilemma is noncooperative games.

2.1.4.2 Strategic and extensive games

In strategic or static games, the players make their decisions simultaneously at the beginning of the game. While the game may last long and there can be probabilistic events, the players can not react to the events during the game. The prisoner's dilemma is strategic game.

On the other hand, the model of an extensive game defines the possible orders of the events. The players can make decisions during the game and they can react to other player's decisions. Extensive games can be finite or infinite. Formal definitions of strategic and extensive games are given in section 2.1.5.

A class of extensive games is repeated games, in which a game is played numerous times and the players can observe the outcome of the previous game before attending the next repetition. A typical example is a repeated prisoner's dilemma in which the same situation is repeated several times [7].

2.1.4.3 Zero-sum games

Games can be divided according to their payoff structures. A game is called zerosum game, if the sum of the utilities is constant in every outcome. Whatever is gained by one player is lost by the other players. [5] Gambling is a typical zero-sum game. Neither of the example discussed are zero-sum games. Zero-sum games are also called strictly competitive games.

2.1.4.4 Games with complete and incomplete information

In games with complete information the preferences of the players are common knowledge, i.e. all the players know all the utility functions. In a game of incomplete information, in contrast, at least one player is uncertain about another player's preferences [5].

A sealed-bid auction is a typical game with incomplete information. A player knows his own valuation of the merchandise but does not know the valuations of the other bidders.

2.1.5 Strategic Games

In strategic games, the players first make their decisions and then the outcome of the game is determined. The outcome can be either deterministic or contains uncertainties.

The actions of the players may take place during a long time period but the decisions are made without knowledge of the decisions of the other players.

Definition 2.1.5.1 A strategic game consists of

- a finite set N (the set of players)
- for each player I $\in$ N a nonempty set Ai (the set of actions available to player i)
- for each player i $\in$ N a utility function Ui on A = $\times_{j\in N} A_j$. [5,7]

The players can choose their actions either from discrete alternatives or from a continuous set. For example, a choice of a route in a network is discrete but the possible transmission powers in a wireless network form a continuous set. If the decisions are discrete, strategic games with two players are usually illustrated with a matrix representation as in tables 2.1

The solution of a strategic game is Nash equilibrium. Every strategic game with finite number of players each with a finite set of actions has an equilibrium point. This Nash equilibrium is a point from which no single player wants to deviate unilaterally [7].

Definition 2.1.5.2: Nash equilibrium of a strategic game (N; (Ai); (Ui)) is a profile $a^* = (a_1^*, \ldots, a_N^*) \in A$ of actions with the property that for every player $i \in N$ we have

$$U_i(a^*) \geq U_i(a_1^*, \ldots, a_{i-1}^*, a_i, a_{i+1}^*, \ldots, a_N^*) \textit{ for all } a_i \in A_i.$$

2.1.6 Extensive Games

The strategic game model is suitable for representing simple real life events such as auctions. Many more complex situations can be abstracted sufficiently to be modeled as a strategic game. However, the limitations of the strategic games are evident in many cases. A more versatile model is needed, when more complex interactions are occurring between the decision makers [6].

Especially the possibility to react to the actions of the other players is essential in many applications, thus a broader model is needed. Extensive games eliminate the limitation of the simultaneous decisions, thus they make possible to model a wider range of real life situations. It should be noted that for simplicity the following formulation does not allow simultaneous actions of the players, i.e. the game has perfect information. An extensive game with imperfect information can be formulated similarly [5].

Definition 2.1.6.1 An extensive game with perfect information has the following components.

- A set N (the set of players)
- A set H of sequences (finite or infinite) of actions that satisfies the following three properties.
- A function P that assigns to each nonterminal history (each member of H\Z) a member of N. (P is the player function, P (h) being the player who takes an action after the history h.)
- For each player $i \in N$ a utility function Ui on Z.

We form an example two-stage extensive game. First, player 1 chooses between actions L and R. After observing player 1's decision, player 2 decides between actions *A* and *B* if player 1 played L and between C and D if player 1 played R. Extensive games with two players can be illustrated with matrices similarly to the strategic games. The example game is given. Instead of the actions, the columns and rows are now the strategies of the players. The utilities of the outcomes are also visible. All the relevant information is available in the matrix, but the chronology of events is hard to perceive. A better option is to form a tree illustrating the game as in Figure 2.1.

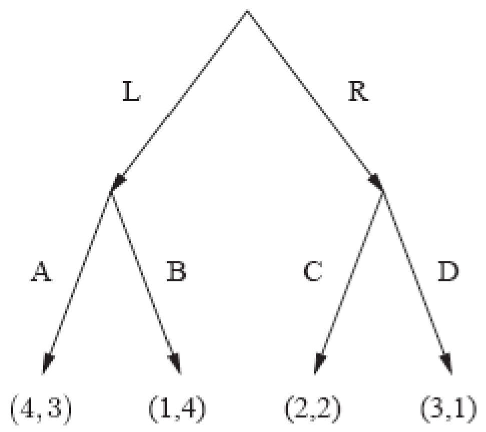


Figure 2.1: Extensive Form

As in the strategic games, the solution of an extensive game is a Nash equilibrium from which no player has an incentive to deviate unilaterally. The solution of the example game can be deducted easily.

2.2 ROUTING PROTOCOLS

2.2.1 Introduction

In computer networking the term **routing** refers to selecting paths in a computer network along which to send data.Routing directs forwarding, the passing of logically addressed packets from their source toward their ultimate destination through intermediary nodes (called routers). It facilitate the exchange of routing information between networks, allowing routers to build routing tables dynamically. Traditional IP routing stays simple because it uses **next-hop routing** where the router only needs to consider where it sends the packet, and does not need to consider the subsequent path of the packet on the remaining hops. Traditional IP routing stays simple because it uses **next-hop routing** where the router only needs to consider where it sends the packet, and does not need to consider the subsequent path of the packet on the remaining hops.

Recent research has lead not only to many routing protocols, but too many routing techniques. By a routing technique we mean the basic strategy that a routing protocol uses to store and propagate routing information. A routing technique captures the following three protocol characteristics [3]:

- Whether routing information is propagated on demand or proactively;
- Whether routing information is propagated by flooding or by propagation on a spanning tree; and
- The format of the routing information stored locally and communicated (whether it is link state data, distance vectors).

2.2.2 Dynamic Routing Protocols

If a designated path becomes unavailable, the existing nodes must determine an alternate route to use to get their data to its destination. They usually accomplish this through the use of a routing protocol using one of two broad classes of routing algorithms: distance vector algorithms and link state algorithms, which together account for nearly every routing algorithm in use on the Internet [9].

2.2.2.1 Distance Vector Routing Algorithm

Distance Vector protocols judge best path based on metric variable. Distance can be hops or a combination of metrics calculated to represent a distance value. The name distance vector is derived from the fact that routes are advertised as vectors of (distance, direction), where distance is defined in terms of a metric and direction is defined in terms of the next-hop router.

Distance vector algorithms use the Bellman-Ford algorithm. This approach assigns a number, the cost, to each of the links between each node in the network. Nodes will send information from point A to point B via the path that results in the lowest total cost (i.e. the sum of the costs of the links between the nodes used) [8 , 9].

Distance-vector routing protocols are simple and efficient in small networks, and require little managaement if any. However, they do not scale well, and have poor convergence properties, which has led to the development of more complex but highly scalable link-state routing protocols for use in large networks. Distance-vector protocols suffer from the count-to-infinity problem.

Distance vector routing protocols include the following:

- Routing Information Protocol (RIP) for IP
- Cisco's Internet Gateway Routing Protocol (IGRP)

2.2.2.2 Link State Routing Protocol

When applying link-state algorithms, each node uses as its fundamental data a map of the network in the form of a graph. To produce this, each node floods the entire network with information about what other nodes it can connect to, and each node then independently assembles this information into a map. Using this map, each router then independently determines the best route from itself to every other node.

Link state protocols, sometimes called shortest path first or distributed database protocols, are built around a well-known algorithm from graph theory, E. W. Dijkstra'a shortest path algorithm [9].
Examples of link state routing protocols are:

- Open Shortest Path First (OSPF) for IP
- The ISO's Intermediate System to Intermediate System (IS-IS) .

Although link state protocols are rightly considered more complex than distance vector protocols, the basic functionality is not complex at all:

- Each router establishes a relationship—an adjacency—with each of its neighbors.
- Each router sends link state advertisements (LSAs), some
- Each router stores a copy of all the LSAs it has seen in a database. If all works well, the databases in all routers should be identical.
- The completed topological database, also called the link state database, describes a graph of the internetwork. Using the Dijkstra algorithm, each router calculates the shortest path to each network and enters this information into the route table [9].

There are different routing protocols for routing data on internet, TCP/IP routing protocols and Adhoc routing protocols.

- TCP/IP Routing
  -RIP
  -OSPF
  -BGP
- Ad hoc network routing protocols
  -Dynamic source routing
  -AODV (Adhoc on Demand Distance Vector)
  -Hierarchical State routing protocol
  -Optimized Link State Routing Protocol
  -Destination sequenced Distance vector

By using Game theory approach inefficiency of Adhoc routing protocols are removed and performance is measured [3]. But the inefficiency in TCP/IP routing protocols are not removed .So here game theory is applied approach on TCP/IP routing protocols and to improve the performance of protocols. The inefficiency of routing protocol is given in detail in following sections.

2.2.3 TCP/IP Routing Protocols

Thesis work mainly concentrates on three routing protocols RIP, OSPF and BGP.

2.2.3.1 Routing Information Protocol (RIP)

The **Routing Information Protocol** (RIP) is one of the most commonly used interior gateway protocol (IGP) routing protocols on internal networks (and to a lesser extent, networks connected to the Internet), which helps routers dynamically adapt to changes of network connections by communicating information about which networks each router can reach and how far away those networks are. Although RIP is still actively used, it is generally considered to have been made obsolete by routing protocols such as OSPF and IS-IS [9]. Nonetheless, a somewhat more capable protocol in the same basic family (distance-vector routing protocols), is the Cisco proprietary (IGRP) **Interior Gateway Routing Protocol**.

RIP is a distance-vector routing protocol, which employs the hop count as a routing metric. The maximum number of hops allowed with RIP is 15. Each RIP router transmits full updates every 30 seconds by default, generating large amounts of network traffic in lower bandwidth networks. It runs *above* the network layer of the Internet protocol suite, using UDP port 520 to carry its data. A mechanism called split horizon with limited poison reverse is used to avoid routing loops. Routers of some brands also use a holddown mechanism known as heuristics, whose usefulness is arguable and is not a part of the standard protocol.

2.2.3.2 Open Shortest Path First (OSPF)

Open Shortest Path First (OSPF) is a routing protocol which was first defined as version 2 in RFC 2328. It is used to allow routers to dynamically learn routes from other routers and to advertise routes to other routers. Advertisements containing routes are referred to as Link State Advertisements (LSAs) in OSPF [9]. OSPF router keeps track of the *state* of all the various network connections (links) between itself and a network it is trying to send data to. This makes it a link-*state* routing protocol. OSPF supports the use of classless IP address ranges and is very efficient. OSPF uses areas to organize a network into a hierarchal structure; it summarizes route information to reduce the number of advertised routes and thereby reduce network load and uses a designated router to reduce the quantity and frequency of Link State Advertisements. OSPF does require the router have a more powerful processor and more memory than other routing protocols.

OSPF selects the best routes by finding the lowest cost paths to a destination. All router interfaces (links) are given a cost. The cost of a route is equal to the sum of all the costs configured on all the outbound links between the router and the destination network, plus the cost configured on the interface that OSPF received the Link State Advertisement on. The **Open Shortest Path First** (**OSPF**) protocol is a link-state, hierarchical interior gateway protocol (IGP) for network routing. Dijkstra's algorithm is used to calculate the shortest path tree. It uses cost as its routing metric. A link state database is constructed of the network topology which is identical on all routers in the area.

2.2.3.3 Boarder Gateway Protocol (BGP)

The **Border Gateway Protocol (BGP)** is the core routing protocol of the Internet. It works by maintaining a table of IP networks or 'prefixes' which designate network reachability between autonomous systems (AS). It is described as a path vector protocol. BGP does not use traditional IGP metrics, but makes routing decisions based on path, network policies and/or rulesets.BGP supports Classless Inter-Domain Routing and uses route aggregation to decrease the size of routing tables [9].

Very large private IP networks can also make use of BGP. An example would be the joining of a number of large Open Shortest Path First (OSPF) networks where OSPF by itself would not scale to size. Another reason to use BGP would be multihoming a network for better redundancy.

2.3 RELATED WORK

2.3.1 Inefficiency In Routing Protocol:

Despite its obvious success, robustness, and scalability, the Internet suffers from a number of end-to-end performances and availability problems. Internet's inefficiencies can be argued as Internet behavior can be improved by spreading intelligent routers at key access and interchange points to actively manage traffic [1].

A routing system is responsible for forwarding traffic between nodes of a network. There are a number of problems this system can be inefficient. It can forward packets along routes that are non-optimal or it can spread load unequally, such that some links are over-utilized while others are idle.

We classify potential sources of routing inefficiencies into four principle categories:

- ***Poor Routing Metrics.*** [1] Today's backbone, or “default-free”, routers generally exchange only connectivity information between each other. In the absence of explicit policy rules, these routers make routing decisions by minimizing the number of independent Autonomous Systems (AS) traversed in getting to the destination. This metric correlates poorly with performance characteristics such as latency or drop rate; it does not change as the performance changes. This is not surprising when one considers that AS's generally correspond to organizational domains and can have enormous scope. For instance, all of MCI's Internet backbone is represented by a single AS number.

- ***Restrictive Routing Policies.*** Policy routing allows each AS to define its own rules for where to send traffic, which routes to advertise, and what traffic to transit. These policies are constructed to support the interests of individual service providers and can negatively affect overall reachability and performance [1]. For instance, the common *early exit* policy attempts to dispatch a packet bound for a host on a foreign network as soon as possible, even if this means sending it in the opposite geographical direction from where it is going. This is suboptimal but, for lack of alternative mechanisms, it is used to limit the amount of traffic one network carries for another. For similar reasons, large providers have established private peering relationships to exchange routing information and traffic, while smaller providers are left at the congested public exchange points. Consequently, packets sent from or destined to smaller networks have less diversity in their choice of routes and poorer connectivity as a result. Finally, some government-funded networks have legal limitations on how they may be used, resulting in policies that only carry traffic meeting some *acceptable use* criteria.

- ***Manual Load Balancing*.** Internet Service Providers and multi-homed organizations generally must pay a fixed fee for the links they use to connect their routers. [1] Consequently, they are interested in balancing the amount of load on their links to take the best advantage of their fixed cost. There is no mechanism for doing this automatically so operators balance load by adding and removing policy rules on a daily basis in response to measured link utilization. While this may keep link utilization high, it does not make for the best routing decisions. In fact, it is extremely likely that there is an alternative assignment of routes to links that would achieve both equal utilization and better overall performance.

- ***Single Path Routing.*** Current Internet routers select a single path to reach a given destination. Alternate paths to the same destination may have underutilized links. This capacity can only be exploited by routing traffic along multiple paths to each destination [1].

2.3.2 Detour Project:

While it is clear that each of these factors contribute to making a less efficient routing system,

- ***Detour:*** - Is the project implemented to remove the inefficiencies in routing [1].

Detour is composed of a set of geographically distributed router nodes interconnected using *tunnels*. A tunnel can be thought of as a virtual point-to-point link. Each packet entering a tunnel is encapsulated into a new IP packet and forwarded through the Internet until it reaches the tunnel's exit point. This same mechanism has previously been used to form the multicast backbone (MBONE) and the experimental IPv6 backbone (6BONE). Tunnels are useful because they allow new routing functionality to be prototyped while using the existing network infrastructure.

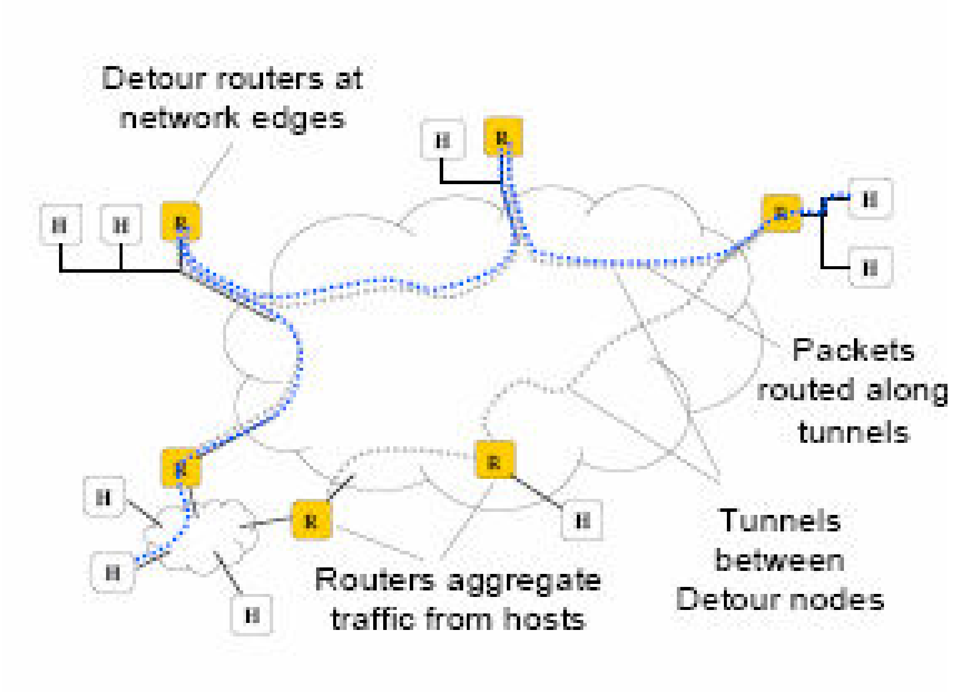

Detour routers at network edges
H
R
Packets routed along tunnels
Tunnels between Detour nodes
Routers aggregate traffic from hosts

Figure 2.2: Detour Architecture

A host wishing to use the Detour network will direct its outbound traffic to the nearest Detour router. Its packets will be forwarded along tunnels within the Detour network and will exit at a point close to the destination. In order that responses return in the same fashion, the system must perform network address translation, so the source address of the packet reflects the exit router and not the actual source. This complication is a necessary consequence of using tunnels to superimpose a new routing framework.

- Such end-to-end route selection schemes are selfish by nature in that they allow end users to greedily select routes to optimize their own performance without considering the system-wide criteria.

- Selfish routing can result in serious performance degradation due to lack of cooperation.

2.3.3 PROBLEM DEFINITION

- Implementing the exciting routing protocol (RIP) using Game approach, by using different algorithms.
- A game-theoretic approach to compute the traffic equilibria of various routing schemes and then evaluate their performance.
- Analyzing the performance of protocol by game theory and the other conventional protocols on different parameters like
    - Delay jitter
    - Convergence
    - Throughput
    - Delay
    - Soundness
    - Processing Time

3 MODELING ROUTING AS GAME

This chapter explains how routing protocols can be modeled as game. And also gives the strategy for implantation of game theory algorithm on routing protocols. Here minimax algorithm is applied on routing protocol for finding path from source to destination.

3.1 MINIMAX ALGORITHM:

Game theory was invented to provide a mathematical foundation for reasoning about conflict and competition [5]. It has grown into a rich theory, with powerful mathematical and computational tools. It also has the advantage of retaining its intuitive appeal, which is what first attracted us to it. Two insights enabled us to turn this pool of theory and tools into a potentially powerful analytical capability for analyzing routing:

- A routing protocol can be modelled as a minimax game between the network and the routers.
- The minimax value of the game can quantify the performance properties.

Minimax (sometimes **minmax**) is a method in decision theory for minimizing the maximum possible loss [13]. Alternatively, it can be thought of as maximizing the minimum gain (**maximin**). It started from two player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves. It has also been extended to more complex games and to general decision making in the presence of uncertainty.

A **simple** version of the algorithm deals with games such as tic-tac-toe, where each player can win, lose, or draw. If player A can win in one move, his best move is that winning move. If player B knows that one move will lead to the situation where player A can win in one move, while another move will lead to the situation where player A can, at best, draw, then player B's best move is the one leading to a draw. Late in the game, it's easy to see what the "best" move is. The Minimax algorithm helps find the best move, by working backwards from the end of the game. At each step it assumes that player A is trying to maximize the chances of A winning, while on the next turn player B is trying to minimize the chances of A winning (i.e., to maximize B's own chances of winning) [13].

Suppose we are considering a two-player game in which, at each turn, a player has a choice of three legal moves. Figure 3.1 might represent possible choices for the first two moves of such a game; it is a tree in which each node represents a state of the game and each branch represents a legal move between two states. Such a structure is called a game tree [3].

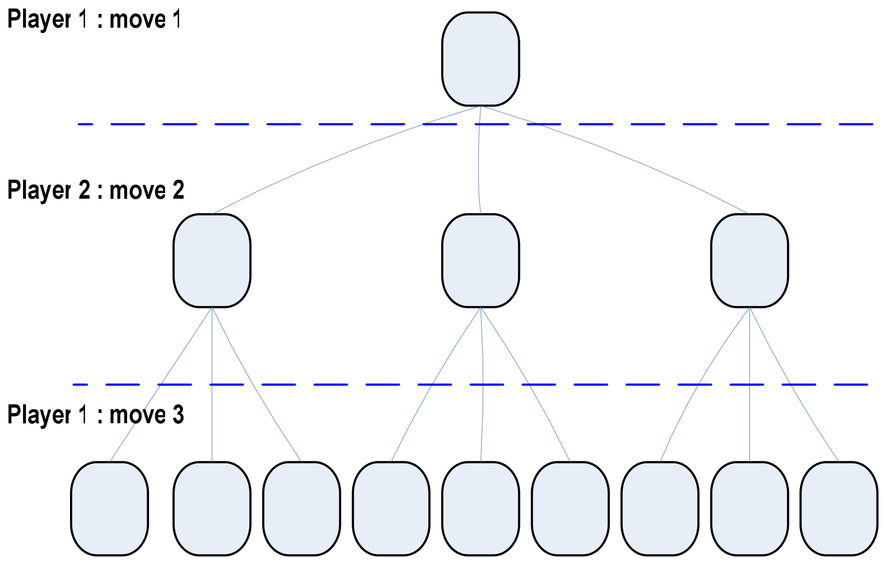


Figure 3.1: A Representation of Game

A Run of the game is a path through the game tree starting at the root node (which represents the initial state) and ending at a leaf node (at which the game has ended). Each run of the game therefore corresponds to a particular sequence of moves chosen by the two players in turn. A cost function is evaluated at the leaves of the game tree [3]. It maps each complete run of the game to a value representing the outcome of the game for this run. The outcome is the cost of this run for a particular player, known as the minimising player. The game must be zero-sum, meaning that the cost for the other player (the maximising player) is minus the cost for the minimising player. As their names suggest, the minimising player tries to minimise the cost function, while the maximising player tries to maximise it. When the game tree can be fully explored, the minimax strategy will find a path that guarantees the best outcome for each player when the other plays as well as possible.

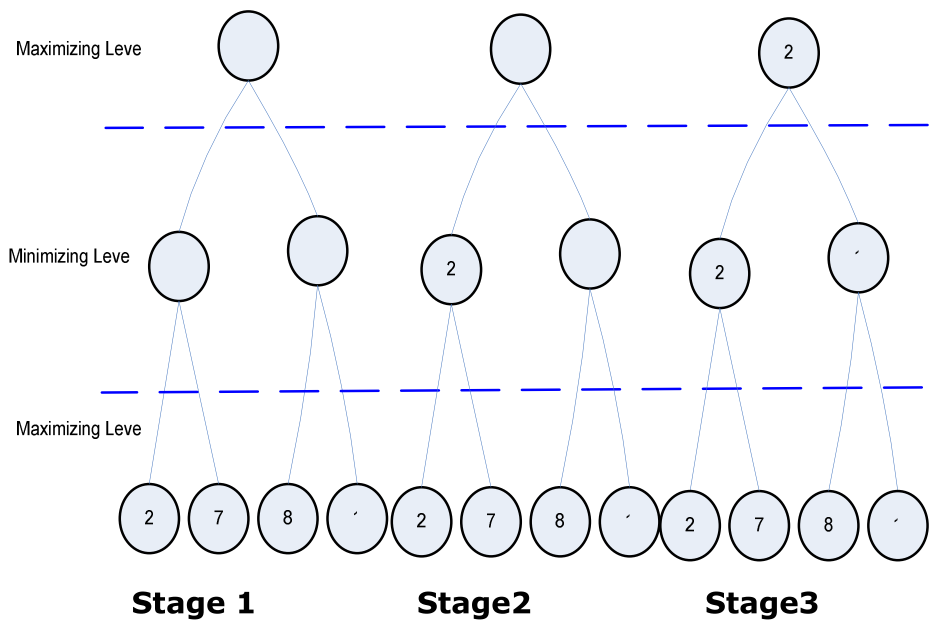


Figure: 3.2 the minimax algorithm at work

For example, consider a game in which each player makes a single move in turn, each choosing from two possible moves. Figure 3.2 shows three stages of a minimax search for such a game. Stage 1 shows the value of the cost function at each leaf node. Stage 2 shows that in the left-hand, middle layer state of the game tree Player II (the minimising player) would choose the move that minimizes the outcome; the game would end in the leaf state with outcome 2. Stage 3 shows Player II's decision in the right-hand state, and also that Player I would choose the move leading to the left-hand state, guaranteeing himself the largest minimal outcome.

3.2 MAPPING ROUTING AS A GAME

The intuition behind modeling routing as a game is to note that the problem of routing can be understood as a contest between the network and the routers. And also see that game theory useful for improving the performance of routing protocols. The routers are, in effect, competing with a network that is trying to outwit them.

- Identify the two players and their initial states, saying which is the minimizing player and which is the maximising player
- Define the game moves for each of the players;
- Specify a cost function that quantifies the outcome for the minimising player; the minimising player chooses moves to minimise this function and the maximizing player chooses moves to maximize it.

In all our uses of game theory the two players will be same. All the routers together form one player, which is referred to henceforth as the set-of-routers player; the other player is the set of links, which is called the network player [3]. Game moves for the set-of-routers player are, in essence to execute the routing protocol. And for the network player game moves are to change the network topology. This is the basic insight behind our mapping to model routing protocols as games. Once the game has been defined the game tree can be constructed and explored.

The minimax strategy searches through the game tree to find the minimax path [14]; the minimax value (or minimax outcome) is the cost function applied to this path. The meaning of the minimax value can be interpreted in the following way: within the constraints provided to the game, if the routers behave optimally, then whatever changes in the network occur, the routers are guaranteed to do no worse than the minimax value.

We define the game as follows:

- The set-of-routers player, representing the set of all the routers, is the minimizing player; the network player, representing the set of all the links, is the maximising player; in the initial state of the game all links are down and each router has a correct view of the network;
- An atomic move for a router is to send all its routing messages, as specified by the protocol (in addition, all routers notice local link changes and process received messages). An atomic move for the network changes the state of one link from up to down or vice versa. A game move, for either player, is a (small) number of atomic moves;

Below figure 3.3 shows how game components is mapped into game component.

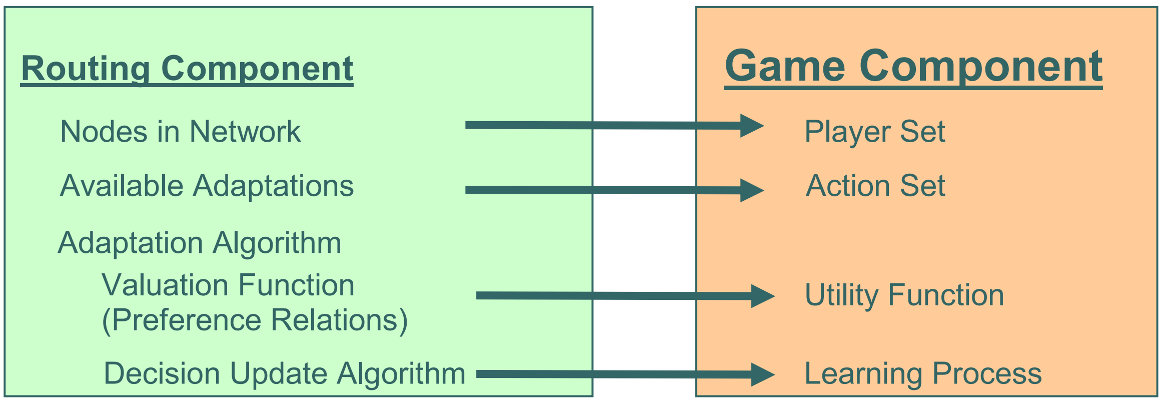


Figure 3.3: Modeling Routing as Game

3.2.1. Detailed Description Of Game Moves:

All the routers together form one player, which is referred to henceforth as the set-of-routers player; the other player is the set of links, which is called the network player. Game moves for the set-of-routers player are, in essence, to execute the routing protocol. And for the network player game moves are to change the network topology.

The minimax algorithm in section 3.1 moves is considered as game moves for finding the path from source to destination.

3.2.2 Detailed Description Of A Set-Of-Routers Atomic Move

The atomic moves and game moves for the network are reasonably clear. The game moves for the set-of-routers are rather more complex, so we elaborate on them in detail.

In each atomic move one router is chosen (the choice being determined by the need to minimise the cost function that records inconsistency and network traffic). Let us say that router n is chosen, and then the atomic move consists of the following sequence of activities:

(i) Each router checks the state of its local links, and updates its own record of the state of those links accordingly.

(ii) Each router performs some processing, as follows:

(a) In link state routing, every router creates its own LSP and puts it on its outgoing queue.
(b) In distance vector routing, each router discards distance vectors it had received from neighbors for which the link has just gone down, and recalculates its own distance vector accordingly.

(iii) Router n performs a broadcast, which consists of the following:

(a) In link state routing, router n processes the LSPs on its holding queue (which is empty the first time when this router is scheduled). For each one, if router n is now linked to the LSP's destination router, that LSP is removed from the holding queue and added to a list of LSPs that are to be broadcast. Router n then processes the LSPs on its outgoing queue (which contains only router n's own LSP the first time this router is scheduled). Each LSP in the outgoing queue is cloned a number of times, with each clone corresponding to a particular destination router other than n itself and the sender of the LSP (if different from n). For each cloned LSP, if n is linked to the destination of that LSP, the LSP is added to the list of LSPs to be broadcast, overwriting any duplicate LSPs already there. If n is not linked to the destination of the LSP, the LSP is added to n's holding queue, as long as its timestamp is more recent than that of any LSP already in the holding queue, originating from the same source and with the same destination. Each LSP in the list to be broadcast is then delivered to its destination.

(b) In distance vector routing, router n broadcasts its own distance vector to all neighbors to which it is linked (as required by the "output always" model).

(iv) All routers that have been sent updates receive them, and act as Follows:

(a) In link state routing, when a router receives an LSP it updates its view of the network accordingly. It then puts the LSP on its outgoing queue, but only if its timestamp is more recent than that of any LSP already in the outgoing queue, originating from the same source and with the same destination. (In reverse-path forwarding, the router does none of this unless it believes the sender of the LSP is on the shortest path between itself and the source of the LSP.)

(b) In distance vector routing, when a router receives a distance vector, it adds this to its list of stored distance vectors, replacing any distance vector previously received from the same neighbor. It then recalculates its own distance vector.

3.3 GAME TREES

When the above defined atomic moves are applied on the network all the routing tables and the game tree are constructed. On that game tree minimax algorithm applied to fine destination.

The game tree consists of all moves available to the current player as children of the root, and then all moves available to the next player as children of these nodes, and so forth, as far into the future of the game as desired. Each branch of the tree represents a possible move that player could make at that point in the game. Evaluating the game at a leaf of this tree yields the projected status of the game after that sequence of moves is made by the players. A deeper search of the game tree provides more information about possible advantages or traps and therefore yields a better move.

3.3.1 Example

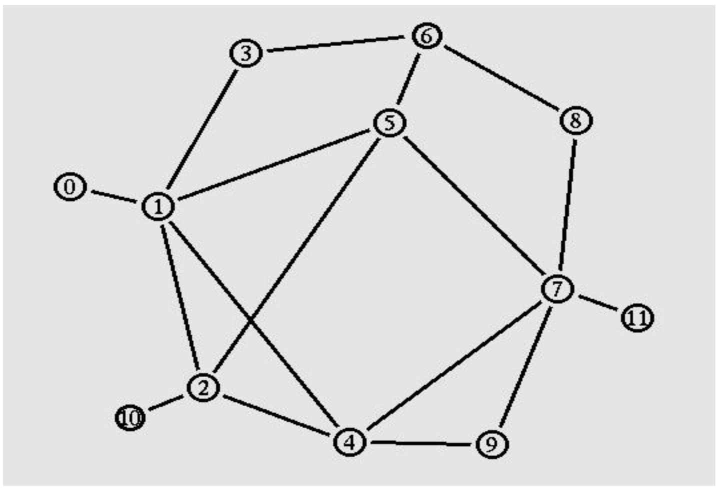


Figure 3.4 Network

The statistics of the network as follows:

Packet sending node = n0, n12
Destination node= n11
Routers= 10 routers

On the above example atomic moves of router and network links are applied and game tree is constructed and finds destinations using minimax algorithm.

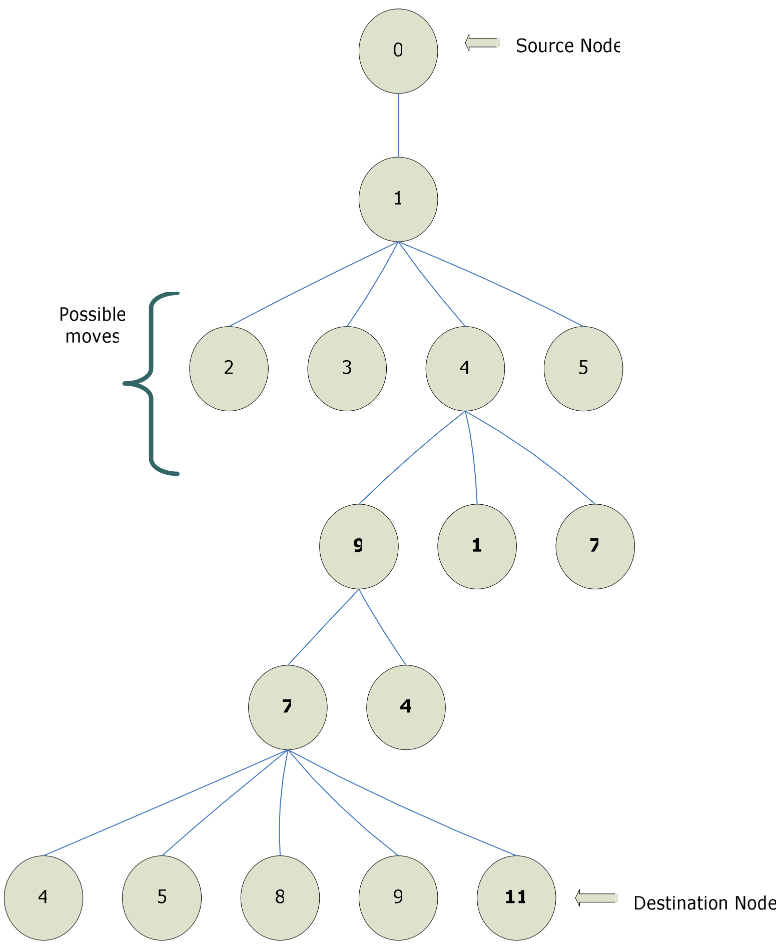


Figure 3.5 Game tree whish gives path from source to destination

The figure.3.4 shows the network for which finds the path from source to destination. When the routers prepare the routing table by using RIP protocol control messages like request and reply message, these are the atomic moves for the game and it prepares the tree which gives the connectivity information of whole network. The tree is used by minimax module for finding the path from source to destination. The above figure.3.5 shows the tree with their possible moves for each router, and finds the path from source to destination.

When in network convergence occur (some links are down) the topology of the whole network will be changing. According to that, the whole tree for network will be changing, figure.3.6 shows how it responses to topology change.

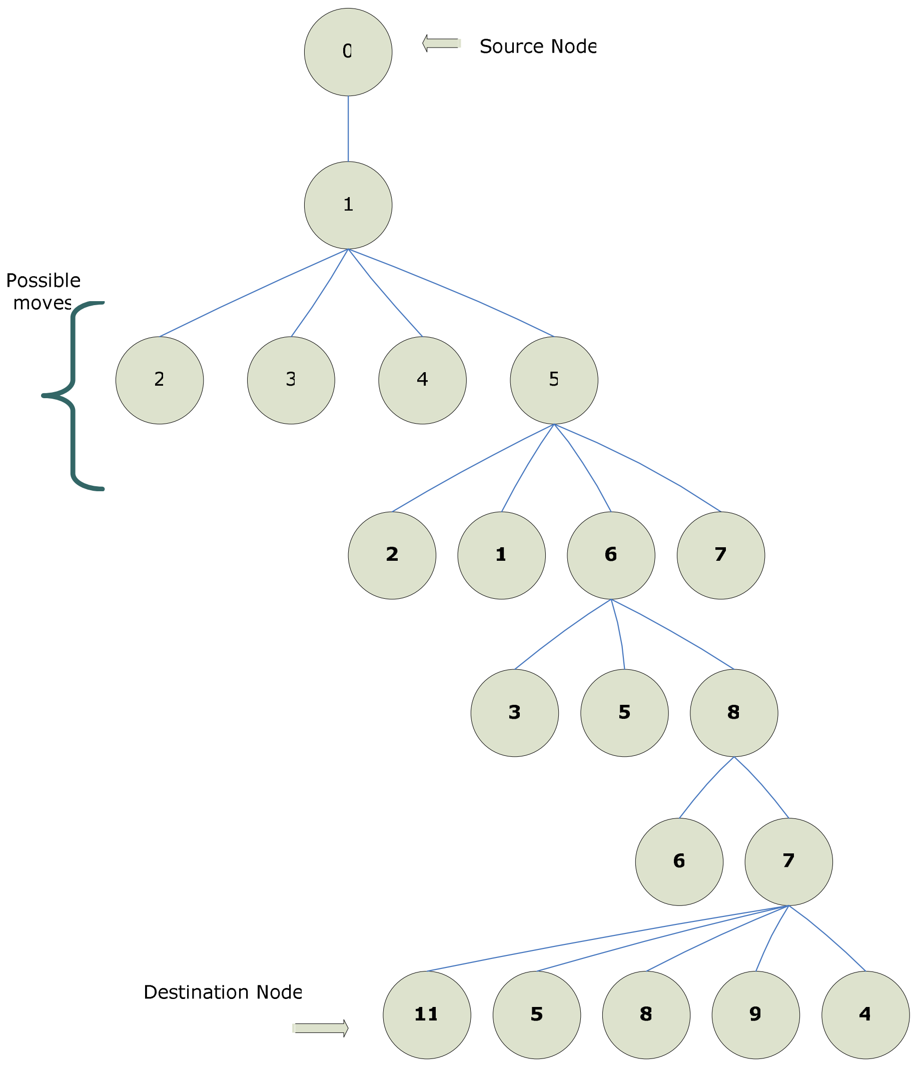


Figure: 3.6 Game tree after change in topology of network

In section 2.1 basics of game theory, when there is change in game or there is any need for backward movement, it takes the backward movement for finding the best move for minimizing the possible loss.

When there is any topology change in the network, minimax algorithm takes the moves backward and prepares the tree and finds the path to destination node.

4 SIMULATION ENVIRONMENT

This chapter gives the basic information about simulation environment or tool used for coding and simulation. The purpose of this chapter is to give a new user some basic idea of how the simulator works, how to setup simulation networks, how to create new network components, routing module etc.

4.1 BASIC OF NETWORK SIMULATOR

NS is an object oriented simulator, written in C++, with an OTcl interpreter as a front-end. The simulator supports a class hierarchy in C++ (also called the compiled hierarchy in this document), and a similar class hierarchy within the OTcl interpreter. The two hierarchies are closely related to each other; from the user's perspective, there is a one-to-one correspondence between a class in the interpreted hierarchy and one in the compiled hierarchy. The root of this hierarchy is the class TclObject. Users create new simulator objects through the interpreter; these objects are instantiated within the interpreter, and are closely mirrored by a corresponding object in the compiled hierarchy. The interpreted class hierarchy is automatically established through methods defined in the class TclClass. User instantiated objects are mirrored through methods defined in the class TclObject. There are other hierarchies in the C++ code and OTcl scripts; these other hierarchies are not mirrored like TclObject [16].

NS uses two languages because simulator has needs to do two different kind of things it.

- Detailed simulations of protocols require a systems programming language which can efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. For these tasks run-time speed is important and turn-around time (run simulation, find bug, fix bug, recompile, re-run) is less important.

- On the other hand, a large part of network research involves slightly varying parameters or configurations, or quickly exploring a number of scenarios. In these cases, iteration time (change the model and re-run) is more important. Since configuration runs once (at the beginning of the simulation), run-time of this part of the task is less important.

NS meets both of these needs with two languages, C++ and OTcl. C++ is fast to run but slower to change, making it suitable for detailed protocol implementation.

4.1.1 Architecture Of NS2

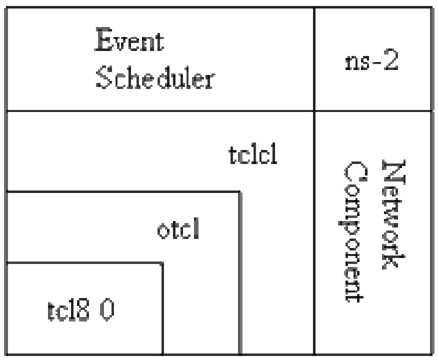


Figure 4.1 Architectural View of NS2

Figure 4.1 shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is an OO extended Tcl interpreter with network simulator libraries [16].

4.1.2 C++ And OTcl Duality:

The event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object. The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to Otcl [16].

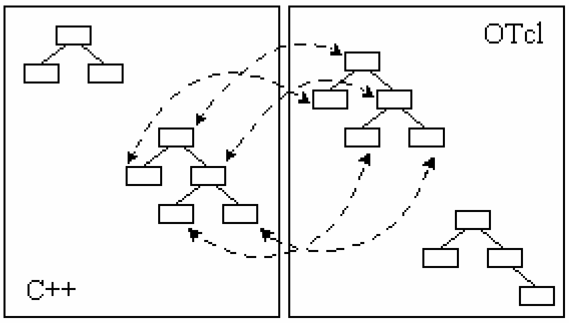


Figure 4.2 C++ and OTcl: The Duality

Figure 4.2 shows an object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

There are a number of classes defined in *~tclcl/*. In NS2 only focus on the six classes that are used in NS2:

- The Class Tcl contains the methods that C++ code will use to access the interpreter.
- The class TclObject is the base class for all simulator objects that are also mirrored in the compiled hierarchy.
- The class TclClass defines the interpreted class hierarchy, and the methods to permit the user to instantiate TclObjects.
- The class TclCommand is used to define simple global interpreter commands.

- The class EmbeddedTcl contains the methods to load higher level built-in commands that make configuring simulations easier.

- Finally, the class InstVar contains methods to access C++ member variables as OTcl instance variables [16].

4.1.3 Simplified User's View:

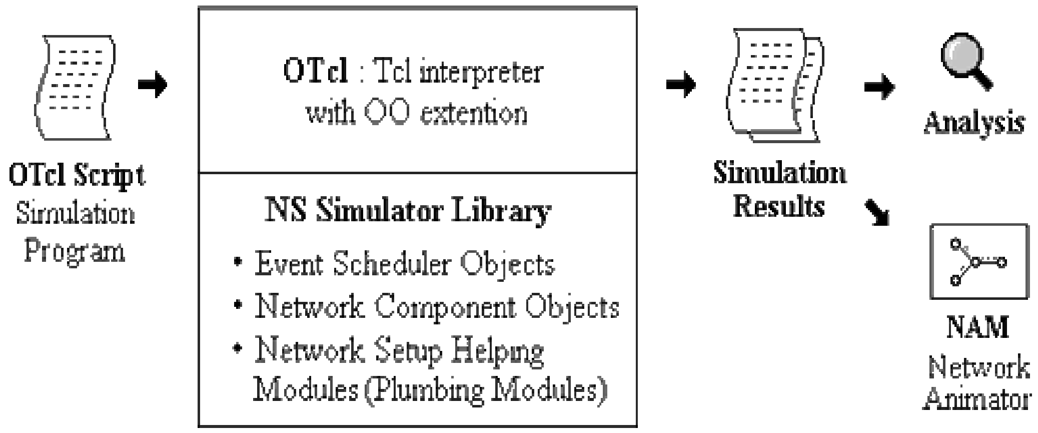

OTcl Script
Simulation Program
OTcl : Tcl interpreter with OO extention
NS Simulator Library
• Event Scheduler Objects
• Network Component Objects
• Network Setup Helping Modules (Plumbing Modules)
Simulation Results
Analysis
NAM
Network Animator

Figure 4.3 Simplified User's View Of Network Simulator 2

As shown in Figure 4.3, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries. To setup and run a simulation network, a user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler [16].

When a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input Tcl script. The data can be used for simulation analysis or as an input to a graphical simulation display tool called Network Animator (NAM) that is developed as a part of VINT project [16]. NAM has a nice graphical user interface similar to that of a CD player (play, fast forward, rewind, pause and so on), and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis.

4.2 ROUTING MODULE IN NETWORK SIMULATOR:

Every routing implementation in NS consists of three functional blocks:

- Routing agent exchanges routing packet with neighbors,
- Route logic uses the information gathered by routing agents (or the global topology database in the case of static routing) to perform the actual route computation,
- Classifiers sit inside a Node. They use the computed routing table to perform packet forwarding.

When implementing a new routing protocol, one does not necessarily implement all of these three blocks. For instance, when one implements a link state routing protocol, one simply implement a routing agent that exchanges information in the manner link state does, and a route logic that applies Dijkstra on the resulting topology database. It can then use the same classifiers as other unicast routing protocols [16].

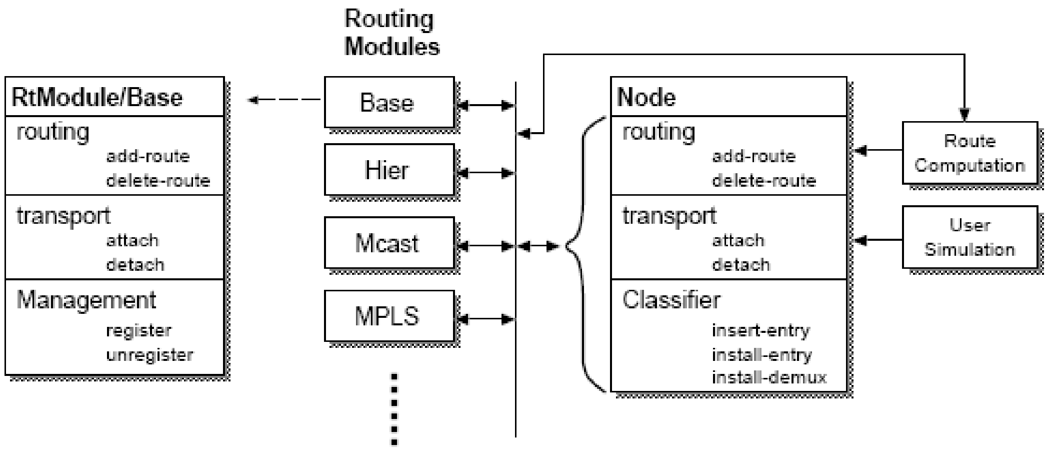


Figure: 4.4 Interaction among node, routing module, and routing .the dashed line shows the details of one routing module

When a new routing protocol implementation includes more than one functional block, especially when it contains its own classifier, it is desirable to have another object, which we call a routing module, that manages all these functional blocks and to interface with node to organize its classifiers. Figure 4.4 shows functional relation among these objects. Notice that routing modules may have direct relationship with route computation blocks, i.e., route logic and/or routing agents. However, route computation may not install their routes directly through a routing module, because there may exist other modules that are interested in learning about the new routes. This is not a requirement, however, because it is possible that some route computation is specific to one particular routing module, for instance, label installation in the MPLS module [16].

A routing module contains three major functionalities:

1. A routing module initializes its connection to a node through register {}, and tears the connection down via unregister{}. Usually, in register {} a routing module

- tells the node whether it interests in knowing route updates and transport agent attachments, and
- Creates its classifiers and install them in the node (details described in the next subsection). In **unregister { }** a routing module does the exact opposite: it deletes its classifiers and removes its hooks on routing update in the node.

2. If a routing module is interested in knowing routing updates, the node will inform the module via **RtModule::add-route {dst, target}** and **RtModule::delete-route{dst, nullagent}.**

3. If a routing module is interested in learning about transport agent attachment and detachment in a node, the node will inform the module via **RtModule::attach {agent, port}** and **RtModule::detach {agent, nullagent}**.

There are several derived routing module examples in ~*ns*/tcl/lib/ns-rtmodule.tcl, which may serve as templates for new routing modules [16].

4.3 EXAMPLE

Figure 4.5 shows example simulate using NS2 tool and observes how it finds route for destination and also the working environment of NS2 tool.

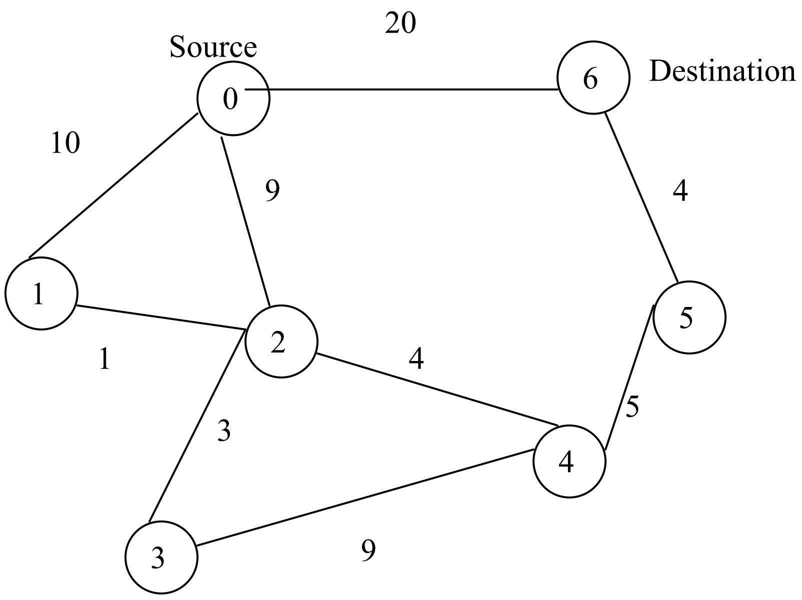
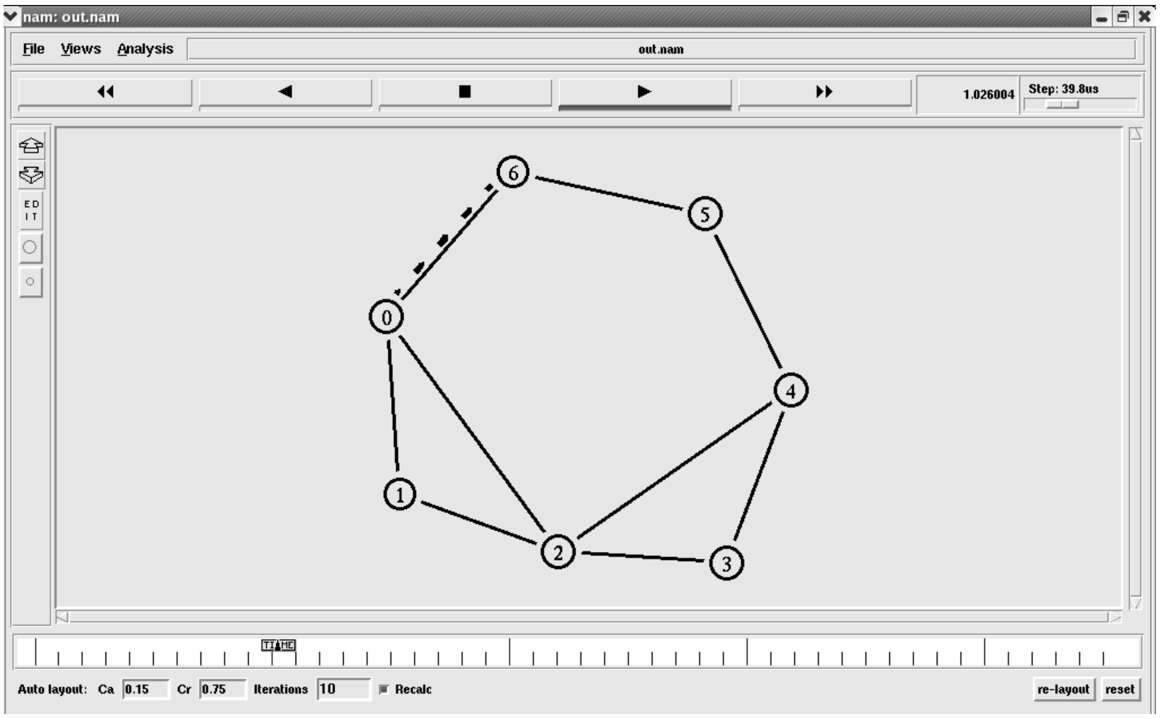


Figure 4.5: Network of nodes and routers

Simulation of above example:



Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc

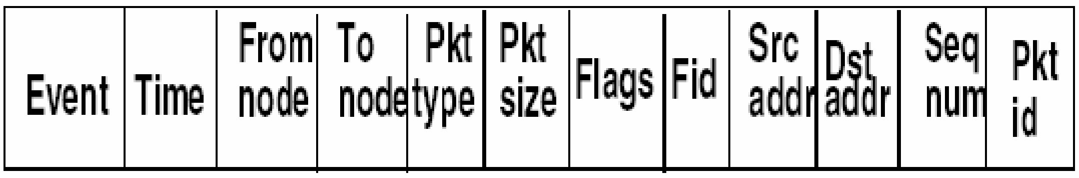
Figure 4.6: Simulation Of Above Network And NS2 Working Environment

The figure 4.6 shows the simulation environment of NS2, it shows how the packet is sent to destination node. When the simulation of any network is executed, NAM file of that particular example will run and also trace file is generated for analysis of the network.

4.4 STRUCTURE OF TRACING:

NS simulation can produce both the visualization trace (for NAM) as well as an ASCII file trace corresponding to the events registered at the networks

When tracing into an output ASCII file, the trace is organized in 12 fields (L to R) as follows in figure.4.7



Event	Time	From node	To node	Pkt type	Pkt size	Flags	Fid	Src addr	Dst addr	Seq num	Pkt id

Figure 4.7: Fields Appearing In Trace

1. The first field is the event type. It is given by one of four possible symbols

   r: receive (at to_node)
   +: enqueue (at queue)
   - : dequeue (at queue)
   d: drop(at queue)

2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type.
6. Gives the packet size.
7. Some flags
8. This is the flow id of IPv6 that a user can set for each flow at input OTcl script. One can use this filed further for analysis purpose; it is also used for specifying stream color for the NAM display.
9. This is the source address given in the form of "node. Port".
10. This is the destination address given in the same form.
11. This network layer protocol packet sequence number. Even thought UDP network in real network do not use sequence number, NS keeps the track of NS packet sequence number for analysis purpose [12].
12. The last field shows the unique id of packet.

Having simulation trace data at hand, all one has to do is to transform a subset of the data of interest into a comprehensible information and analyze it. One can of course write programs in any programming language that can handle data files like awk, pearl etc. Yet several tools that seem particularly adapted for these purpose and that are freely available under various operating systems. [17] Tracegraph is the tool implemented in matlab, and which is freely available .In this tool trace file is given as input, and various graphs will generate according to users requirement.

Trace graph supports the following ns-2 trace file formats:

- Wired
- Satellite
- Wireless (old and new trace)
- Wired-cum-wireless.

Trace file loading stage is divided into 4 stages:

- automatic trace file format recognition using the first *n* file lines, where *n* is specified in Trace graph configuration file, if the format cannot be recognized it can be specified manually
- Trace file parsing to extract necessary simulation data which is saved to a temporary file, trace files can contain much more data than is needed by the system, so unnecessary information is omitted to speed up trace file loading
- Temporary file loading
- Constraints calculations (packets types, packets sizes, flows IDs, trace levels, number of nodes, simulation time) – in order to speed up data processing [17].

5 RIP SIMULATION

Network simulator 2 has unicast support for routing protocols like RIP, OSPF etc. In this chapter rtProtoDV (RIP) which is already implemented in NS2, is simulated and the results are analyzed for inefficiencies in rtProtoDV. Here, to see the behavior of RIP 4 different examples are taken with different parameters like bandwidth, queue, packet size... etc and how it works. According to simulation results the inefficiencies in RIP protocol are found.

5.1 SIMULATION SETUP:

In this simulation 4 examples are taken, the parameters as follows.

Table 5.1 RIP simulation parameters

Parameter	**Value**
Bandwidth	Varied (but same for all example)
Simulation time	20 sec
Nodes	varied
Traffic Type	FTP
Packet size	1040
Traffic	TCP

Following are the examples used for simulation and analyzing RIP protocol.

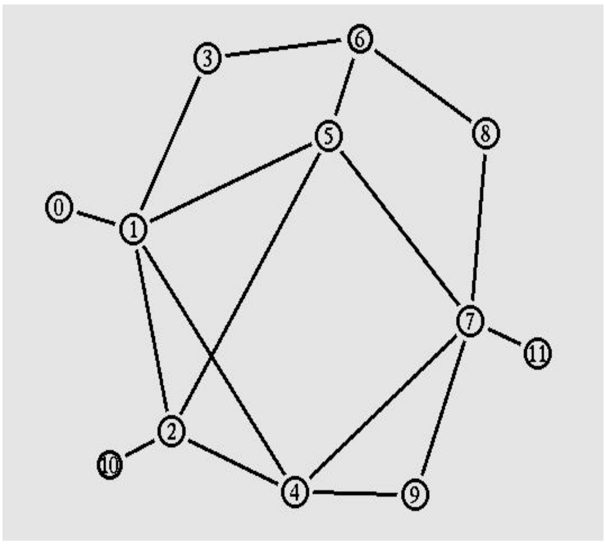


Figure 5.1: Example1 RIP implementation

Packet sending node = n0

Destination node= n11

Routers= 9 routers

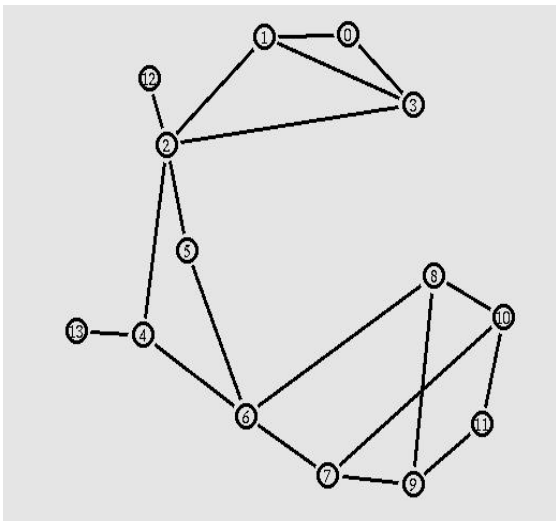


Figure 5.2: Example 2 RIP implementation

Packet sending node = n0,n12,n13

Destination node= n10

Routers= 10 routers

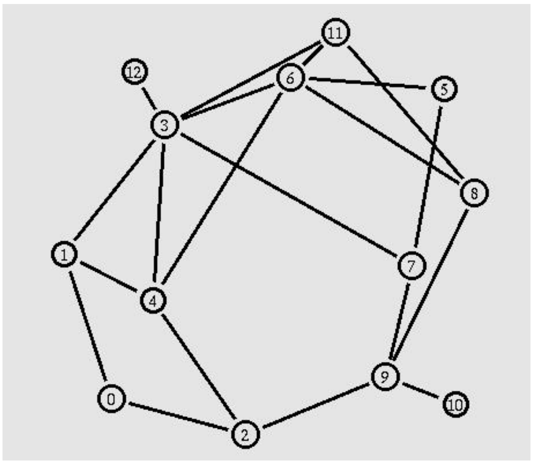


Figure 5.3: Example 3 RIP implementation

Packet sending node = n0, n12

Destination node= n10

Routers= 10 routers

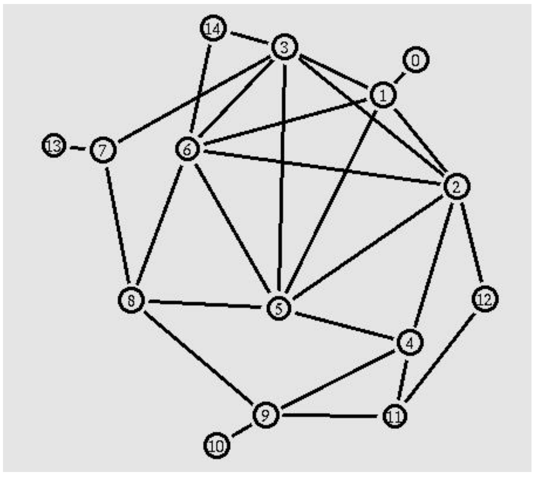


Figure 5.4: Example 4 RIP implementation

Packet sending node = n0, n13

Destination node= n10

Routers= 12 routers

5.2 RESULT

Fallowing table shows results of above 4 examples on different parameters like convergence, throughput, delay, Processing time, delay jitter etc.

Table 5.2 RIP Simulation Result

Parameters	**Example1**	**Example 2**	**Example 3**	**Example 4**
Convergence (Sec)	0.069	0.56	0.43	0.555

Throughput (PKT/TIL)	A= 2.2321
	M=5.0
		A=0.8909
		M=3.0
			A=2.01851
			M=4.0
				A=15.7142
				M=34.0

Jitter {Sec)	A=0.01081
	M=0.2582
		A=0.5119
		M=8186
			A=0.0530
			M=0.7658
				A=0.0442
				M=0.6168

Processing Time (Sec)	A= 0.01934
	M=0.2454
		A=0.111985
		M=0.81864
			A=0.0301
			M=0.3468
				A=0.0267
				M=0.371406

Delay (Sec)	A=0.2524
	M=0.4386
		A=0.6953
		M=1.2206
			A=0.3418
			M=0.7228
				A=0.3361
				M=0.6868

A=Average
M=Maximum
From above results we can say that:

- The timer associated with each entry in routing table much larger than the period of transmission of information.
- Slow Convergence
- Count to infinity Problem
- Difference in the links speed is not reflected in hop count metrics
- Congested links are included in path from source to destination.

As we have seen in chapter 3 how the routing is to be modeled as game. So here that approach is to be used for modeling routing as game and removing the in inefficacies of RIP protocol.

6 IMPLEMENTATION DETAILS OF RTPROTOMIN

6.1 INTRODUCTION:

The RIP (Routing Information Protocol) uses Distance Vector (DV) routing protocol for finding path from source to destination, and sends the packets. DV routing is the implementation of Distributed Bellman-Ford (or Distance Vector) routing. As we seen in above chapter rtProtoDV is the RIP implementation in Network Simulator 2 (NS2) unicast routing. Here replacing Distributed Bellman-Ford (or Distance Vector) with game theory algorithms, like minimax algorithm for finding path. RtProtoMIN is implemented in NS2. The structure, classes and all details of rtProtoMIN are as follows.

6.2 BLOCK DIAGRAM OF RTPROTOMIN:-

Figure 6.1 shows how different files are used when simulating example. There are two main .h and .cc files are modified and one .h and .cc file is added in NS2 for rtProtoMIN.

- rtProtoMIN.h, rtProtoMIN.cc
- route.h, route.cc
- rttable.h , rttable.cc
- bsd-list.h

These are the main files associated with rtProtoMIN, and tcl files are as follows

- Ns-route.tcl
- Route-proto.tcl

These 2 files are useful for implementing rtProtoMIN in tcl script. This mainly concerns with simulation of rtProtoMIN.

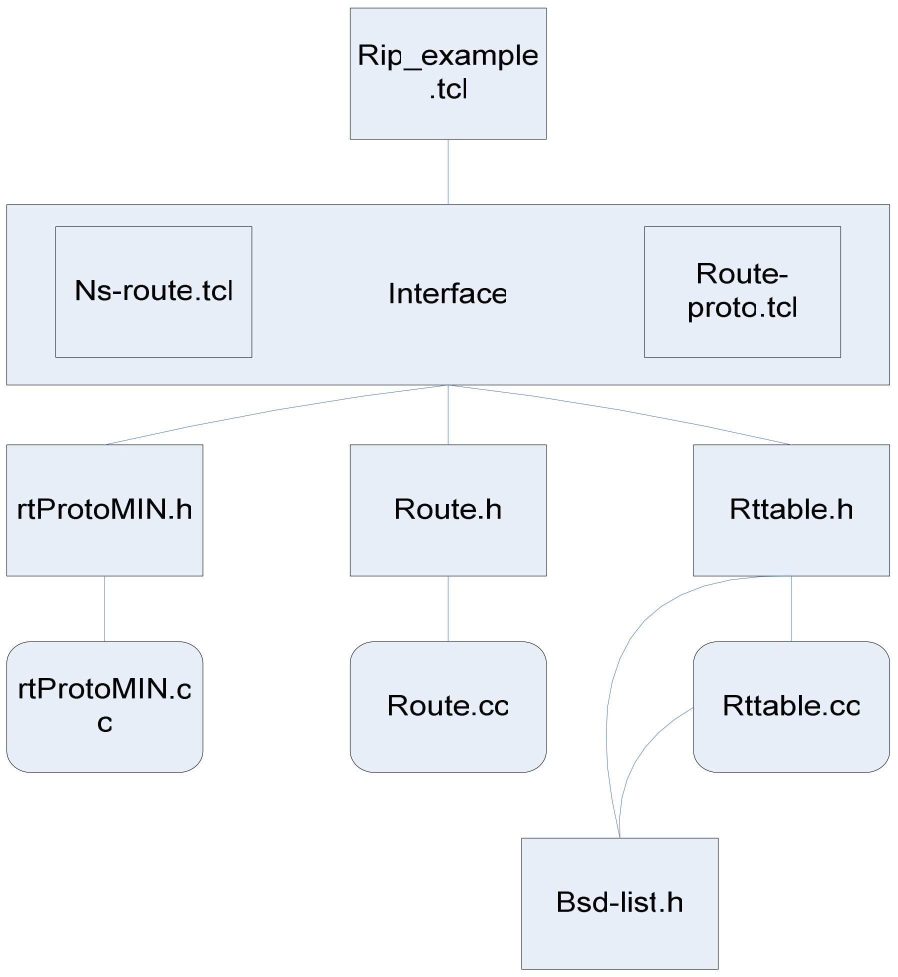


Figure: 6.1 Block Diagram of rtProtoMIN

6.3. INTERNALS AND ARCHITECTURE OF RTPROTOMIN:

Here all the classes associated with rtProtoMIN, and the code path used to configure and execute rtProtoMIN protocols is explained.

The Class rtProtoMIN and the header structure are defined as shown in figure 6.2.

6.3.1 RtProtoMIN:

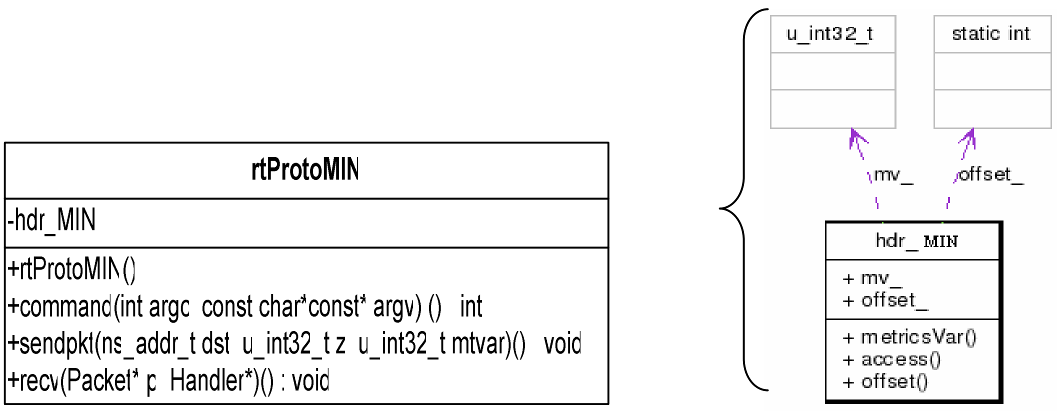

rtProtoMIN
-hdr_MIN
+rtProtoMIN()
+command(int argc const char*const* argv) () int
+sendpkt(ns_addr_t dst u_int32_t z u_int32_t mtvar)() void
+recv(Packet* p Handler*)() : void
u_int32_t
static int
mv_
offset_
hdr_MIN
+ mv_
+ offset_
+ metricsVar()
+ access()
+ offset()

Figure 6.2: RtProtoMIN Class And Header Structure

The above figure 6.2 shows the class rtProtoMIN and header structure which passes the information to other routers. That packet is called as control packet that is rtProtoMIN. The internal methods and call graph of rtProtoMIN is as follows.

6.3.1.1 Hdr_MIN:

The header structure of rtProtoMIN as shown above, it contains the metrics variable. When there are multiple routes to the same destination, a router should have a mechanism to calculate the best path. A metric is a variable assigned to router as a means of ranking them from most preferred to least preferred.

6.3.1.2 Sendpkt ():

This function concerns with sending packet to the destination node. In that three main parameters have to pass destination address, metrics variable.
ns_addr_t: Declares the network address in NS2.
u_int32_t: Unsigned 32 bit integer.

The call graph of sendPkt function is as shown in below figure.

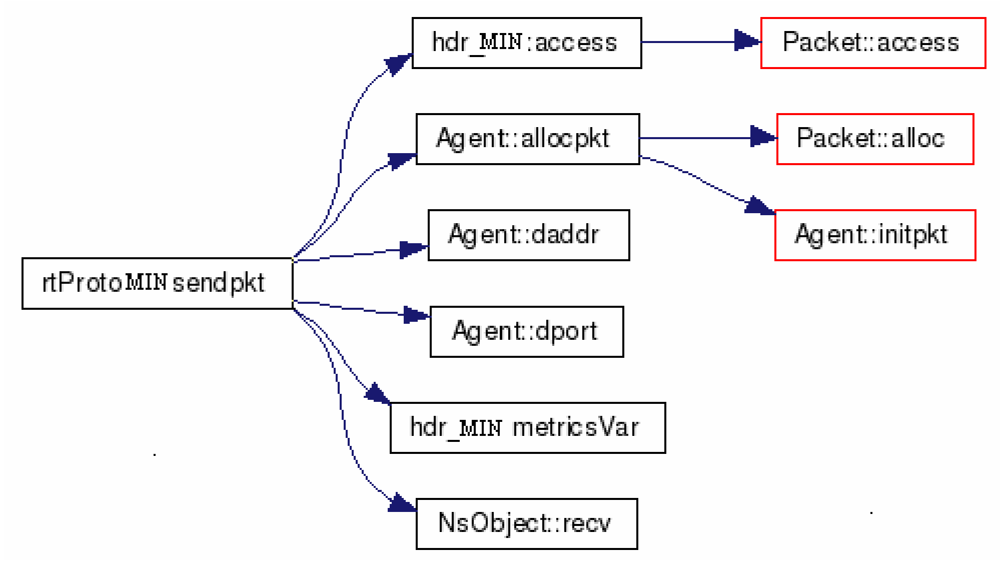


Figure 6.3: Call Graph Of Sendpkt Function.

6.3.1.3 Recv ():

This function deals with receiving the packet to destination node, after receiving it will free that packet.

The call graph of receive function is as follows

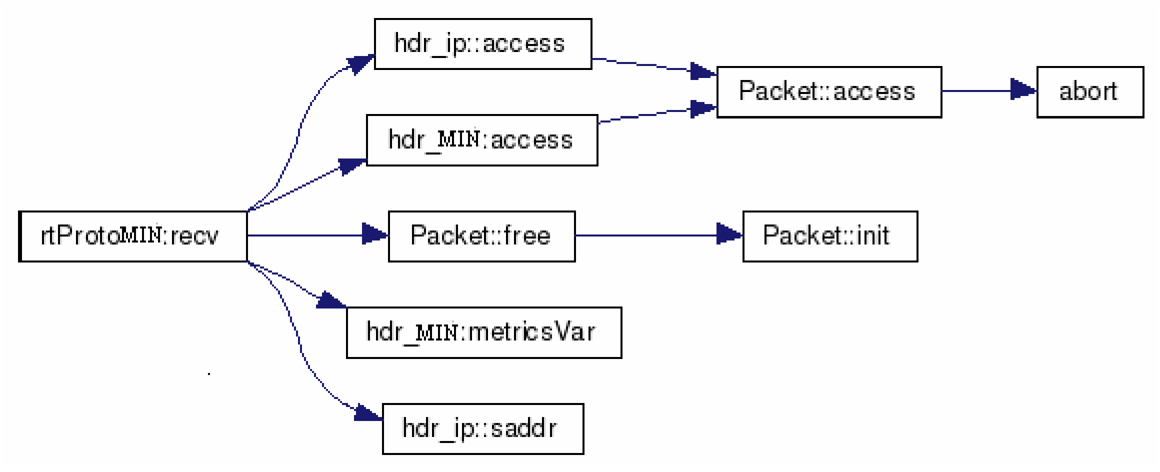


Figure 6.4: Call Graph Of Recv () Function.

There are four main classes, the class RouteLogic, the class rtObject, the class rtPeer, and the base class Agent/rtProto for all protocols. In addition, the routing architecture extends the classes Simulator, Link, Node and Classifier.

6.4 ROUTE.H

Here two structures are defined

- Adj_entry
- Route_Entry

Also all the methods which are useful for finding the routes like , compute routes , lookup, check , alloc , minimax .etc. It cotains main class that is RouteLogic.

6.4.1 Class RouteLogic

This class defines two methods to configure unicast routing, and one method to query it for route information. It also defines an instance procedure that is applicable when the topology is dynamic.

- The instance procedure **register {}** is invoked by **Simulator::rtproto {}.** It takes the protocol and a list of nodes as arguments, and constructs an instance variable, **rtprotos_,** as an array; the array index is the name of the protocol, and the value is the list of nodes that will run this protocol.

- The **configure {}** reads the **rtprotos_** instance variable, and for each element in the array, invokes route protocol methods to perform the appropriate initializations. It is invoked by the simulator run procedure.

- The instance procedure **lookup {}** takes two node numbers, node **Id1** and node **Id2**, as argument; it returns the id of the neighbor node that node **Id1** uses to reach node **Id2**.

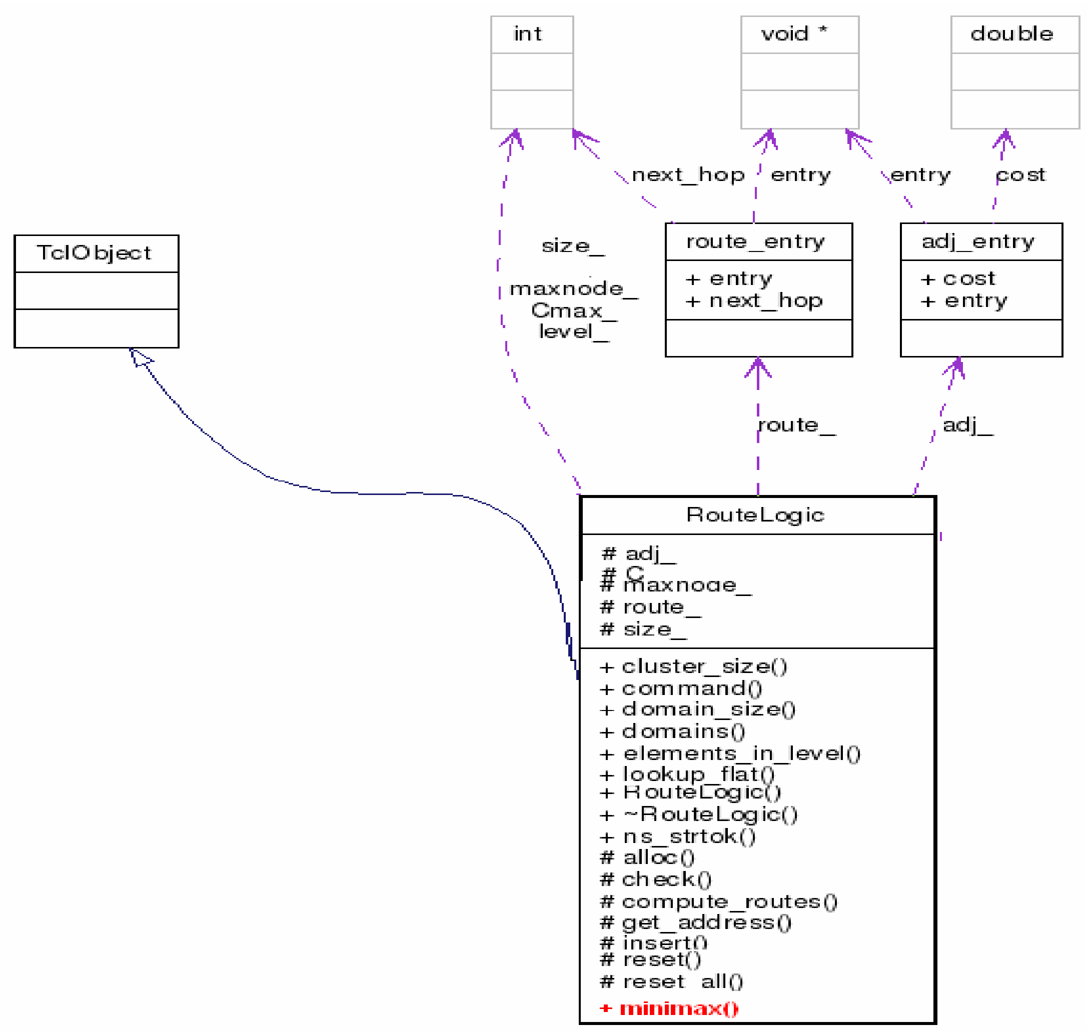


Figure 6.5: Route Logic Class and structure: route_entry and adj_entry

The above diagram shows two structures, which is used in route computation and other member function used for route computation. The new module added in routelogic is **minimax ()** shown by red color in diagram. rtProtoMIN use minimax algorithm for finding route from source to destination.

Route_entry: This structure consists of two variables next_hop and entry. It concerns with route information for network.

Adj_entry: This structure consists of two variables cost and entry. It concerns with cost information for each adjacent entry of router.

RouteLogic class also consists of other function like constructor **Route_Logic()** it initializes all variables to zero , **alloc ():** used for allocating memory for routers , **reset_all():** when route are found or there is convergence all routers are reset again and find the routes , **check ():** checks the maximum hops allowed for the protocol , **insert():** this method adds adjacent routes for each router, **Minimax ():** the implementation details and logic is given in above chapter. It takes tree as argument formed using atomic moves; this tree contains all connectivity information of whole network.

The routine checks the protocol agent's instance variable, **rtsChanged_** to see if any of the routes in that protocol have changed since the protocol was last examined. It then uses the protocol's instance variables **arrays, nextHop_, rtpref_,** and **metric_** to compute its own arrays. The rtObject will install or modify any of the routes as the changes are found.

6.4.2 Class Rtobject

This class is used in simulations that use dynamic routing. Each node has a rtObject associated with it, that acts as a co-ordinator for the different routing protocols that operate at a node. At any node, the rtObject at that node tracks each of the protocols operating at that node; it computes and installs the next route to each destination available via each of the protocols. In the event that the routing tables change, or the topology changes, the rtObject will alert the protocols to take the appropriate action.

The class defines the procedure **init-all {}**; this procedure takes a list of nodes as arguments, and creates a **rtObject** at each of the nodes in its argument list. It subsequently invokes its **minimax ()** method for finding routes.

6.4.3 The Class Rtpeer

This is a container class used by the protocol agents. Each object stores the address of the peer agent, and the metric and preference for each route advertised by that peer. A protocol agent will store one object per peer. The class maintains the instance variable **addr_,** and the instance variable arrays, **metric_** and **rtpref_;** the array indices are the destination node handles. The class instance procedures, **metric{}** and **preference{}**, take one destination and value, and set the respective array variable. The procedures, **metric{}** and **preference{}**, take a destination and return the current value for that destination. The instance procedure **addr {}** returns the address of the peer agent.

6.4.4 Class Agent/rtProto

This class is the base class from which all routing protocol agents are derived. The constructor for the rtProtoMIN agent initializes a number of instance variables; each agent stores an array, indexed by the destination node handle, of the preference and metric, the interface (or link) to the next hop, and the remote peer incident on the interface, for the best route to each destination computed by the agent. The agent creates these instance variables, and then schedules sending its first update within the first 0.5 seconds of simulation start.

6.5 RTTABLE.H:

This file concerns with the routing table building, rttable.h declarer all the methods and definition of these methods are written in rttable.cc.
This file contains main 3 classes which are useful for building routing table.

- Class Neighbour
- Class rt_entry
- Class rttable

Here each move for building routing table is called as atomic move; this is explained in previous chapter 3. This move is useful for building tree, which contains all connectivity information of the network.

Rttable.h uses **bsd-list.h** for storing all information. Bsd-list.h defines the different type of data structure, those are useful for storing information of routing table, neighbor etc. It defines the five types of data structures singly link-list, singly link-list tail queue, lists, tail queues and circular queues.

These data structure used by rttable.h for building routing table and storing neighbor connectivity information.

6.5.1 Class Neighbor:

This class defines the methods for getting all information of neighbors in the network. Below diagram shows class neighbor, it shows attributes and methods. Neighbor method makes the neighbor entry of neighbor node.

LIST_ENTRY (neighbor) concerns with entry of neighbor, which is defined in bsd-list.h.

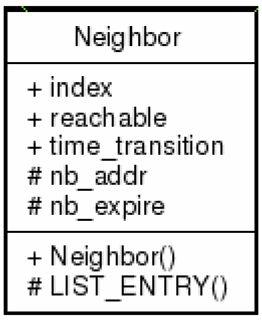


Figure 6.6: Class Neighbor

6.5.2 Class Rt_Entry:

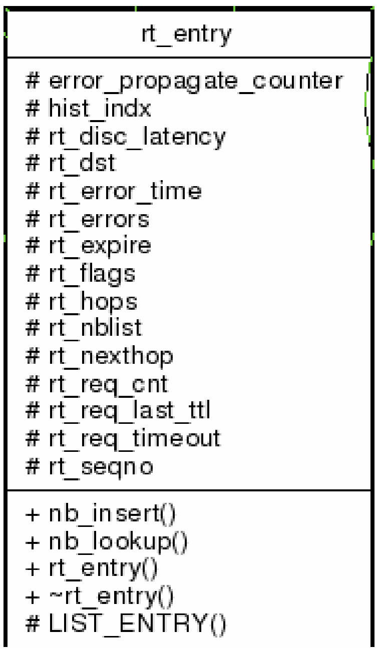


Figure 6.7**:** Class rt_entry

The above diagram shows the route entry class the attributes and methods as shown in figure 6.7. This class concerns with inserting neighbors, neighbor lookup. Rt_entry initializes all the variables to zero which is used for building routing table. LIST_ENTRY () concerns with neighbor route entry.

6.5.3 Class Rttable:

This class concerns with adding the routing information for routers and prepares routing table. There are three main functions of this class are,

Rt_add (): This adds entry into routing table, it adds destination address and metrics variable i.e. cost to reach to the destination node. It uses LIST_HEAD_INSERT () structure for adding routing information.

Rt_lookup (): This method search the whole routing table for particular id i.e destination node. And returns the neighbor node route to destination node.

Rt_delete (): If there is convergence or when the simulation ends the particular entry or whole routing table is deleted by using this method , and it is modified using request and reply messages of protocol. LIST_REMOVE () is used for this purpose.

6.5.4 Other Classes Simulator, Node, Link, And Classifier

The one other method used internally is get-routelogic {}; this procedure returns the instance of routelogic in the simulation.

- The class Node contains these additional instance procedures to support dynamic unicast routing: init-routing {}, add-routes {}, delete-routes {}, and rtObject {}.

- The instance procedure init-routing {} is invoked by the rtObject at the node. It stores a pointer to the rtObject, in its instance variable rtObject_, for later manipulation or retrieval.

- The instance procedure add-routes{} takes a node id, and a list of links. It will add the list of links as the routes to reach the destination identified by the node id. The realization of multiPath routing is done by using a separate Classifier/multiPath.

- The instance procedure delete-routes {} takes a node id, a list of interfaces, and a nullAgent. It removes each of the interfaces in the list from the installed list of interfaces.

In this way all these four files are related to each other. As shown in figure 6.1 the interface between simulation example and the .cc and .h header files. The *.tcl* files like *ns-proto.tcl, route-proto.tcl* that forms the duality objects with the objects of C++ code. As this duality seen in the architecture of network simulator 2.

6.6 CHANGES TO BE MADE WHEN ADDING RTPROTOMIN PROTOCOL IN NS2:-

Following are the steps for adding rtProtoMIN protocol into NS2.

1) rtProtoMIN.h and rtProtoMIN.cc files are put into routing folder. **/usr/src/ns/routing/**

2) Packet.h :-
   Following changes are made in the file. RtProtoMIN is the control packet of the protocol.

   - **PT-RTPROTOMIN** this packet is add in to packet.h
   - P_info() Name_[PT-RTPROTOMIN] = "rtProtoMIN"

3) TCL Library
   When adding new routing protocol we have to add the same changes in tcl library. There are two main files in which we have to make changes.

- Tcl/lib/ns-packet.tcl

  Add "rtProtoMIN"
- TCL/lib/ns-default.tcl

  Agent/rtProto/MIN - set Preference

  Agent/rtProto/MIN - Infinity

  Agent/rtProto/MIN - adveretinterval 2

4) TCL/rtglib/route-protocol.tcl

   Add all the function which is used to connect tcl and .cc files with simulation

5) Add path **/routing/rtProtoMIN.o** in make file. During recompilation NS2 rtProtoMIN.o file will be generated.

6) Recompile whole NS2

   - Make clean
   - Touch packet.h
   - Make

By using these steps we can add rtProtoMIN protocol in NS2.
So now **$ns rtProto MIN** can be used for the simulation of protocol.

7 SIMULATION STUDY AND RESULTS

Network Simulator 2 tool is used for the simulation of unicast routing protocol. As we have seen in above chapters the implementation details of rtProtoMIN which uses minimax algorithm for finding path, and NS2 have rtProtoDV which uses conventional way for finding path.

Now in this chapter these two protocols simulation is done on different scenarios, and analyzed on different parameters.

- Fixed network Varying Packet size: In this scenario network is fixed and packet size is varied from example to example, and it analyzed to find how they responds to delay

- Fixed Packet Size varying network: In this scenario Packet size is same for the entire network but network is changed and its complexity is increased.

- Changing queue to see how it responds to jitter

The above scenarios are simulated for both rtProtoDV and rtProtoMIN, and comparison is made between them to see the performance of routing protocol.

7.1 SCENARIO1: VARYING PACKET SIZE

In this scenario we have taken a network, in that the packet size varied from example to example. The setup of this example as follows.

7.1.1 Simulation setup

Table 7.1 Parameters used during simulation.

Parameter	**Value**
Bandwidth	Varied (but same for all example)
Simulation time	20 sec
Nodes	26
Traffic Type	FTP
Packet size	Varied from example to example
Number of flows	7
Traffic	TCP

Here four examples are taken, in that the network is same for four example and simulation parameters as in table 7.1. But packet size is varies 1040, 1540, 2040 and 2540 from example 1 to example 4. These four examples are simulated using rtProtoDV and rtProtoMIN routing protocols. Here in this simulation bursty (bulky) data transmitted by each flow, to see how these both protocols respond to delay and jitter.

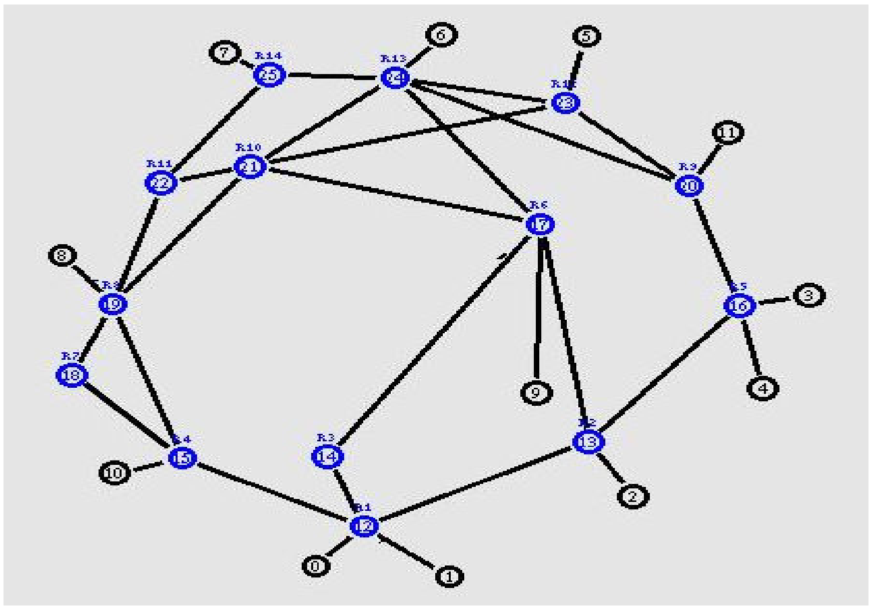


Figure 7.1 Routing Network

Figure 7.1 shows the network which is used for simulation of this scenario. In above network blue circles are routers and black circles are traffic generating nodes in the network. Here there are total 7 flows which are sending data, from that only source node 0 and destination 7 are used for reading purpose and other nodes are generating heavy traffic in network. By this setup delay and jitter of network is analyzed for both protocols and see which protocol gives the good performance.

7.1.2 Statistics Of Scenario 1:

Example 1:

In this example packet size is 1040 and its result is collected in below table 7.2. The statistics of the network as follows.

Simulation length in seconds: 19.715883
Number of nodes: 26
Number of sending nodes: 7
Number of receiving nodes: 5
Number of generated packets: 3337

Number of sent packets: 3337

Number of forwarded packets: 12014

Number of dropped packets: 24

Number of lost packets: 100

Packet size: 1040

Number of sent bytes: 3470480

Number of forwarded bytes: 12494560

Number of dropped bytes: 24960

Packets dropping nodes: 12 13 15 17 19

Example 2:

In this example packet size is 1540 and its result is collected in below table 7.2. The statistics of the network as follows.

Simulation length in seconds: 19.715626

Number of nodes: 26

Number of sending nodes: 7

Number of receiving nodes: 5

Number of generated packets: 2244

Number of sent packets: 2244

Number of forwarded packets: 7995

Number of dropped packets: 23

Number of lost packets: 94

Packet size: 1540

Number of sent bytes: 3455760

Number of forwarded bytes: 12312300

Number of dropped bytes: 35420

Packets dropping nodes: 12 13 15 17 19

Example 3:

In this example packet size is 2040 and its result is collected in below table 7.3. The statistics of the network as follows.

Simulation length in seconds: 19.715632
Number of nodes: 26
Number of sending nodes: 7
Number of receiving nodes: 5
Number of generated packets: 1723
Number of sent packets: 1721
Number of forwarded packets: 6058
Number of dropped packets: 15
Number of lost packets: 80
Packet size: 2040
Number of sent bytes: 3510840
Number of forwarded bytes: 12358320
Number of dropped bytes: 30600
Packets dropping nodes: 12 13 17 19

Example 4:

In this example packet size is 2540 and its result is collected in below table7.4 . The statistics of the network as follows.

Simulation length in seconds: 19.715233
Number of nodes: 26
Number of sending nodes: 7
Number of receiving nodes: 5
Number of generated packets: 1447
Number of sent packets: 1446
Number of forwarded packets: 4939
Number of dropped packets: 7
Number of lost packets: 89
Packet size: 2540
Number of sent bytes: 3672840
Number of forwarded bytes: 12545060
Number of dropped bytes: 17780
Packets dropping nodes: 12 17 19

7.1.3 Results:

The above network is simulated and trace file is generated, these are analyzed by using AWK script for delay, jitter, packet loss, generated packets etc. These awk scripts are in appendix B. The collected results for rtProtoDV and rtProtoMIN have shown in tables 7.2 & 7.3.

Table 7.2 senario1: simulation results

Parameters		Example 1 PS : 1040		Example 2 PS : 1540	
		rtProtoDV	rtProtoMIN	rtProtoDV	rtProtoMIN
Jitter (Sec)	Average	0.020653	0.012710	0.03939	0.027147
	Maximum	0.33002	0.256223	0.80028	0.75631
Delay (Sec)	Average	0.3820	0.2949	0.6966	0.512546
	Minimum	0.1504	0.106971	0.1939	0.145244
	Maximum	0.7957	0.510179	1.332	1.00944
Processing Time(Sec)	Maximum	0.39225	0.448966	0.621822	0.719478

Table 7.3: Scenario 1 Simulation results

Parameters		**Example 3 PS : 2040**		**Example 4 PS : 2540**	
		rtProtoDV	**rtProtoMIN**	**rtProtoDV**	**rtProtoMIN**
Jitter (Sec)	**Average**	0.04815	0.03764	0.047051	0.051136
	Maximum	0.45091	0.40904	1.004249	1.057614
Delay (Sec)	**Average**	0.7471	0.82205	0.957002	0.9819
	Minimum	0.2374	0.21135	0.28093	0.2983
	Maximum	1.1887	1.33852	1.4986	1.56639
Processing Time(Sec)	**Maximum**	0.82807	0.95131	1.21190	1.268051

The above table shows the result of scenario 1. As we see that packet size is increasing from example 1 to example 4 . Here mainly Delay parameter is concerned. From table 7.2 & 7.4 it is observed that maximum and average delay for example 1 & 2 is rtProtoMIN is better as compared to rtProtoDV. As the packet size increased from 2040 to 2540 the delay performance for rtProtoMIN is decreased as compared to rtProtoDV. And same case with jitter for first two examples it gives the best results but for later two examples it does not gives better results as compared to rtProtoDV.

Simulation processing time for rtProtoMIN increases as we increase packet size. But processing time of rtProtoDV is less as compared to rtProtoMIN.

The below graphs shows comparison of Delay between rtProtoDV and rtProtoMIN. The graph is plotted for Packet send time at source node 0 VS Delay between source node 0 and destination node 7. The graphs as follows.

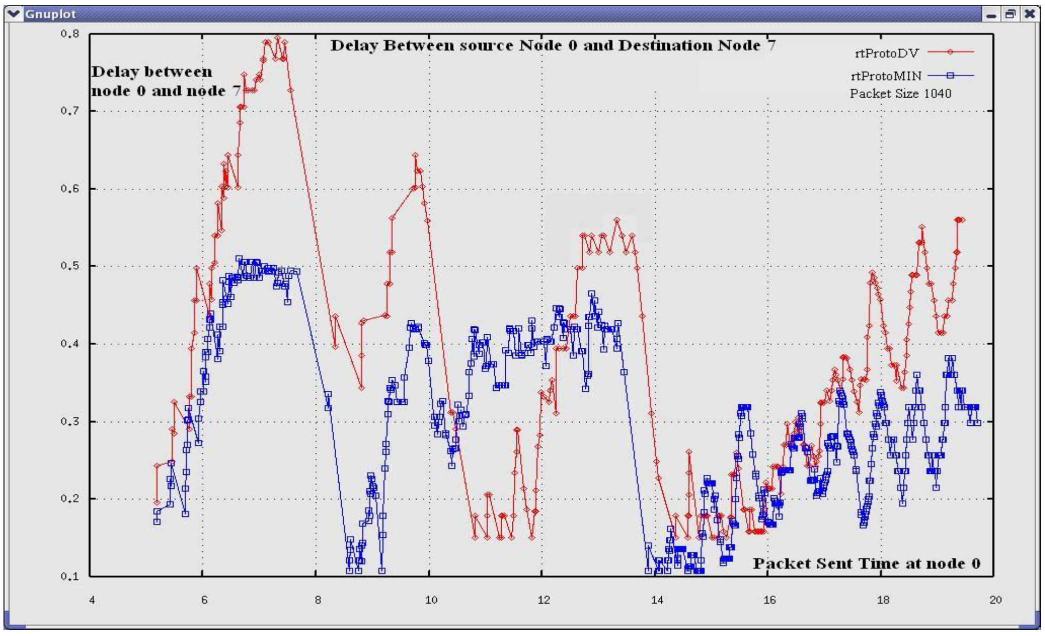


Figure 7.2. Example1 PS 1040: Delay

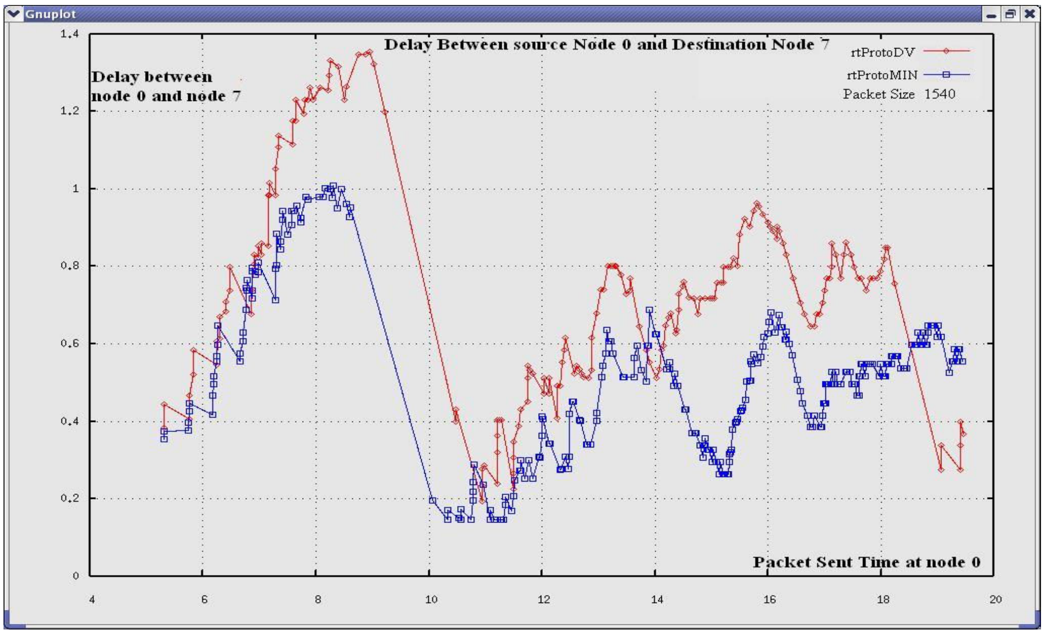


Figure 7.3: Example2 PS 1540: Delay

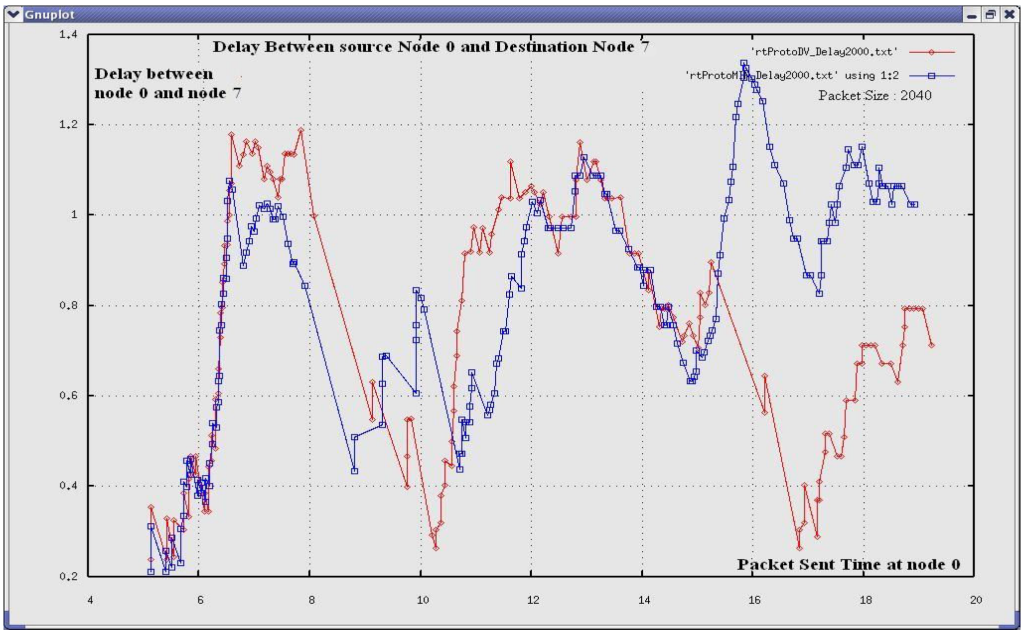


Figure 7.4: Example 3 PS 2040: Delay

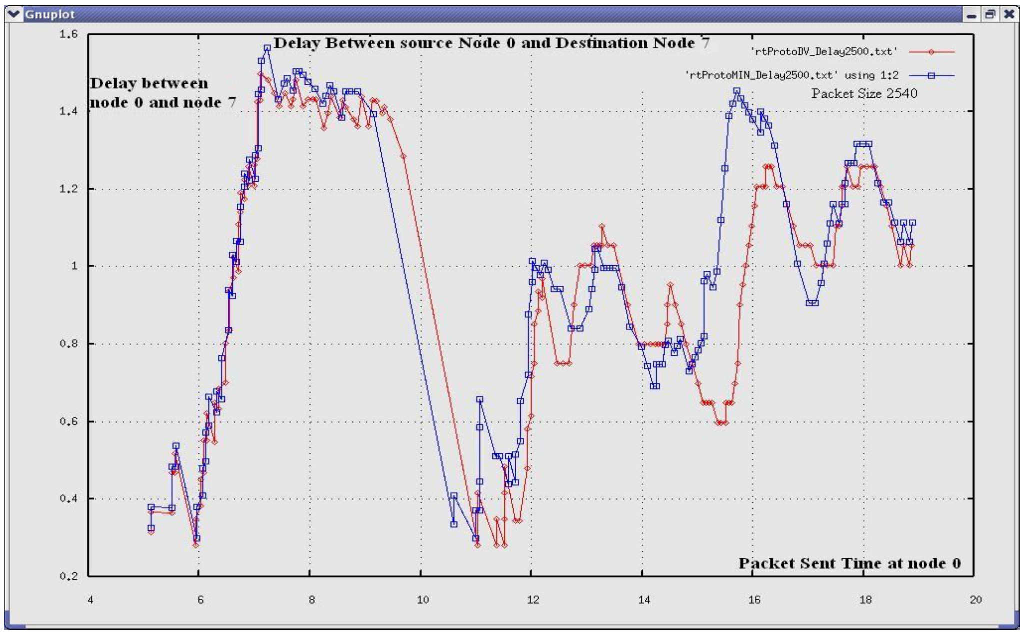


Figure 7.5: Example 4 Ps 2540: Delay

As shown in above graphs the red line indicates delay for rtProtoDV and blue line indicates delay variation for rtProtoMIN. As shown in figure 7.2 delays for rtProtoMIN is less as compared to rtProtoDV but at some places it goes high it is due to the bulky data transmission by different nodes. We got the best performance of rtProtoMIN for pocketsize 1540, as shown figure 7.3 rtProtoMIN shows good performance than rtProtoDV.

As packet size increases almost same delay variation for rtProtoDV is observed and rtProtoMIN as in figure 7.5.

7.2 SCENARIO 2: FIXED PACKET SIZE AND VARYING NETWORK

In this scenario the packet size is fixed for all four networks but varying the network and also increasing the traffic on network. These examples analyzed on different parameters like Jitter, Delay, and Simulation Processing Time etc.

7.2.1 Simulation Setup and Statistics

Table 7.4 Simulation Parameters

Parameter	**Value**
Bandwidth	Varied
Simulation time	20 sec
Nodes	Varied
Traffic Type	FTP
Packet size	1040
Number of flows	Varied
Traffic	TCP
Queue	Drop Tail
Queue Size	20

Here four examples are taken, in that the packet size is same for four examples and simulation parameters as in table 7.4. But network, traffic, flows varies from example 1 to example 4. These four examples are simulated using rtProtoDV and rtProtoMIN routing protocols. Here in this simulation bursty (bulky) data transmitted by each flow, to see how these both protocols respond to delay and jitter.

Examples:

Below examples are used for the simulation purpose. In these examples blue box is the routers, and black circles which generate the traffic on the network. In all the examples the flows are varied from example to example, and busty data is transmitted by senders to see the effect on jitter and how these two protocols handles it. These networks simulated by using rtProtoDV and rtProtoMIN and analyzed on jitter, delay, processing time etc.
The networks and there statistics as follows.

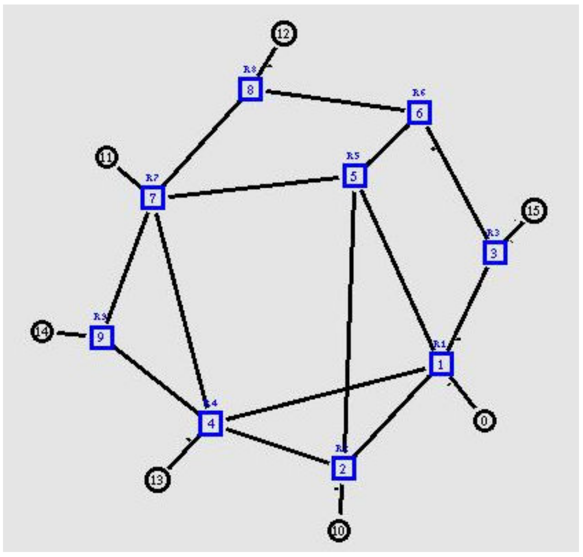


Fig 7.6 : Example 1

Simulation length in seconds:19.733147

Number of nodes: 16

Number of sending nodes: 4

Number of receiving nodes: 4

Number of generated packets: 1016

Number of sent packets: 999

Number of forwarded packets: 3420

Number of dropped packets: 15

Number of lost packets: 23

Packet size: 1040

Number of sent bytes: 1038960

Number of forwarded bytes: 3556800

Number of dropped bytes: 15600

Packets dropping nodes: 2 4 8 9

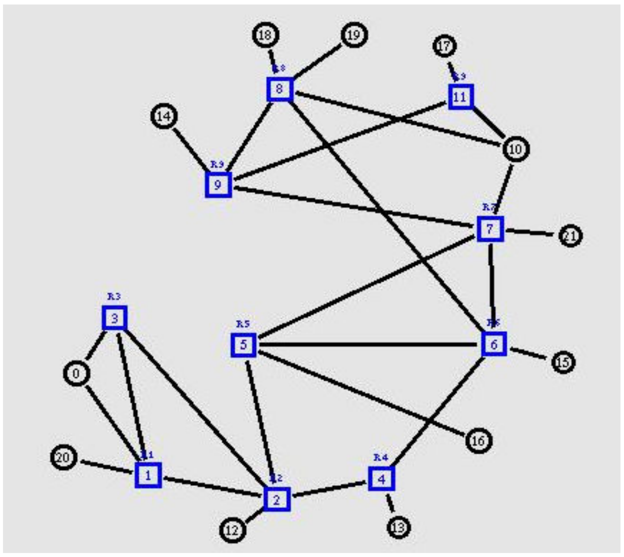


Fig 7.7: Example 2

Simulation length in seconds: 19.8125

Number of nodes: 22

Number of sending nodes: 6

Number of receiving nodes: 6

Number of generated packets: 1657

Number of sent packets: 1649

Number of forwarded packets: 6122

Number of dropped packets: 38

Number of lost packets: 74

Packet size: 1040

Number of sent bytes: 1714960

Number of forwarded bytes: 6366880

Number of dropped bytes: 39520

Packets dropping nodes: 2 5 6 7

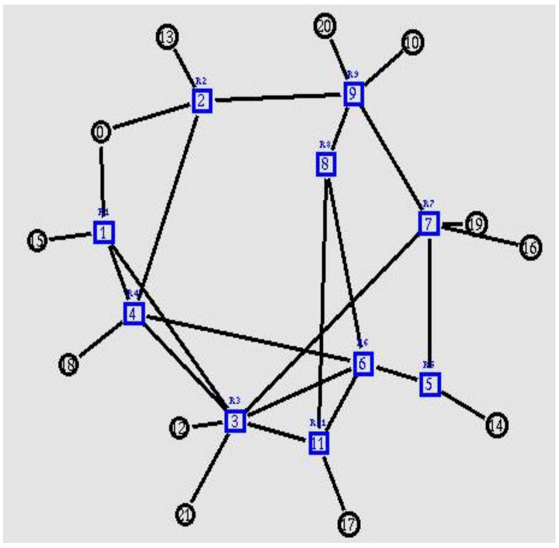


Fig 7.8: Example 3

Simulation length in seconds: 20.41628

Number of nodes: 22

Number of sending nodes: 6

Number of receiving nodes: 6

Number of generated packets: 2179

Number of sent packets: 2179

Number of forwarded packets: 8762

Number of dropped packets: 30

Number of lost packets: 21

Packet size: 1040

Number of sent bytes: 2266160

Number of forwarded bytes: 9112480

Number of dropped bytes: 31200

Packets dropping nodes: 0 4 5 7

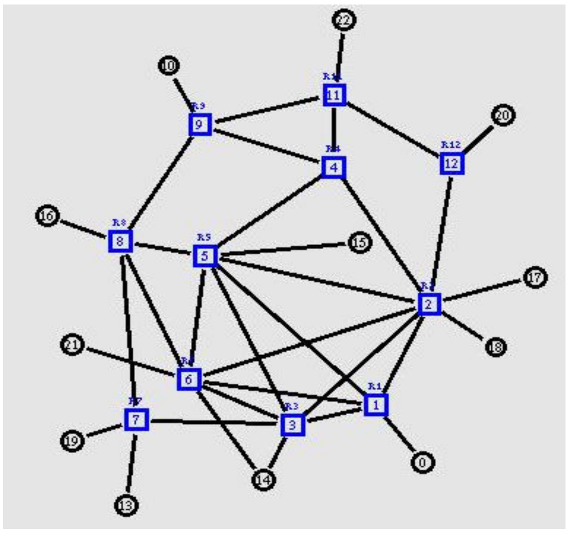


Fig 7.9: Example 4

Simulation length in seconds: 20.59629

Number of nodes: 23

Number of sending nodes: 7

Number of receiving nodes: 6

Number of generated packets: 2264

Number of sent packets: 2264

Number of forwarded packets: 8103

Number of dropped packets: 37

Number of lost packets: 37

Packet size: 1040

Number of sent bytes: 2354560

Number of forwarded bytes: 8427120

Number of dropped bytes: 38480

Packets dropping nodes: 3 11

7.2.2 Results

Below table is collection of results of above simulation setup.

7.5 scenario2: simulation results

Parameters		**Example 1 PS : 1040**		**Example 2 PS : 1040**	
		rtProtoDV	**rtProtoMIN**	**rtProtoDV**	**rtProtoMIN**
Jitter (Sec)	**Range**	0.0 - 0.7488	0.0 - 0.2673	0.0 - 0.37942	0.0 - 0.18400
Delay (Sec)	**Average**	0.44023	0.146320	0.60200	0.48001
	Minimum	0.169253	0.381533	0.288499	0.246480
	Maximum	0.973973	0.890773	1.035539	0.936869
Processing Time(Sec)	**Maximum**	0.712666	0.781893	0.690389	0.770057

7.6 Scenario2: Simulation result

Parameters		**Example 3 PS : 1040**		**Example 4 PS : 1040**	
		rtProtoDV	**rtProtoMIN**	**rtProtoDV**	**rtProtoMIN**
Jitter (Sec)	**Range**	0.0 - 0.41774	0.0 - 0.30850	0.0 - 0.43741	0.0 - 0.30506
Delay (Sec)	**Average**	0.366154	0.32724	0.56660	0.48390
	Minimum	0.236106	0.20224	0.19301	0.17405
	Maximum	0.716453	0.64080	0.75109	0.71884
Processing Time(Sec)	**Maximum**	0.429093	0.621760	0.508267	0.831261

In table 7.5 & 7.6 jitter results are taken in rage from minimum to maximum jitter. As from above table we see that for all four examples jitter is better for rtProtoMIN as compared to rtProtoDV. In case of delay, rtProtoMIN delays less as compare to rtProtoDV delay.

But the processing power is increased for rtProtoMIN as compared to rtProtoDV. Here in case of fixed packet size and varying network, we get the best performance is seen of rtProtoMIN as compare to rtProtoDV.

7.3 SCENARIO 3: CHANGING QUEUE

In this scenario the examples of section 7.2 are used for simulation purpose but, different queues used for simulation and see how it responds to congestion and variation of jitter.

7.3.1 Simulation Setup

Table 7.7: simulation setup for scenario 3

Parameter	**Value**
Bandwidth	Varied
Simulation time	20 sec
Nodes	Varied
Traffic Type	FTP
Packet size	1040
Number of flows	Varied
Traffic	TCP
Queue	DropTail,FQ,RED,DRR
Queue Size	20

Here four examples are taken, in that the packet size is same for four examples and simulation parameters as in table 7.7. But network, traffic, flows and queue varies from example 1 to example 4. These four examples are simulated using rtProtoDV and rtProtoMIN routing protocols. Here in this simulation bursty (bulky) data transmitted by each flow, to see how these both protocols respond to delay and jitter.

In this simulation four queues are used DropTail, FQ (Fair Queuing), RED (Random early detection), DRR (Deficit round robin). For these four queues jitter and delay are found, and range of jitter and delay is in the below table 7.8 & 7.9. This range is between minimum and maximum value of jitter and delay.

7.3.2 Simulation Results

The below table is collection of results for all four examples, jitter and delay range (minimum – maximum) is given in blow table.

Table 7.8: Scenario3: simulation result

Parameters		**Example 1 PS : 1040**		**Example 2 PS : 1040**	
	Queue	**rtProtoDV**	**rtProtoMIN**	**rtProtoDV**	**rtProtoMIN**
Jitter (sec)	**DropTail**	0.0 - 0.7488	0.0 - 0.2673	0.0 - 0.37942	0.0 - 0.18400
	RED	0.0 - 0.3814	0.0 - 0.1819	0.0 - 0.3310	0.0 - 0.0944
	FQ	0.0 - 0.1331	0.0 - 0.9238	0.0 - 0.0480	0.0 - 0.0528
	DRR	0.0 - 0.9984	0.0 - 0.8400	0.0 - 0.0448	0.0 - 0.0369
Delay (Sec)	**DropTail**	0.1692 - 0.7164	0.3815 - 0.6408	0.2884 - 0.7510	0.2464-0.7188
	RED	0.285-0.6013	0.2399 - 0.4602	0.2884 - 0.6628	0.2464 - 0.5904
	FQ	0.1752 - 1.605	0.1463 - 1.2434	0.2884 - 0.4425	0.2464 - 0.4733
	DRR	0.1692 - 1.3906	0.1463 - 1.2066	0.2884 - 0.4427	0.2464 - 0.4070

Table 7.9 Scenario3: Simulation results

Parameters		**Example 3 PS : 1040**		**Example 4 PS : 1040**	
		rtProtoDV	**rtProtoMIN**	**rtProtoDV**	**rtProtoMIN**
Jitter (sec)	**DropTail**	0.0 - 0.41774	0.0 - 0.30850	0.0 - 0.43741	0.0 - 0.30506
	RED	0.0 - 0.2550	0.0 - 0.3797	0.0 - 0.2358	0.0 - 0.2389
	FQ	0.0 - 0.2923	0.0 - 0.1977	0.0 - 0.1153	0.0 - 0.2389
	DRR	0.0 - 0.2439	0.0 - 0.1620	0.0 - 0.7907	0.0 - 0.8454
Delay (Sec)	**DropTail**	0.236 - 0.7164	0.2022 - 0.640	0.1930 - 0.751	0.1740 - 0.718
	RED	0.2361 - .6526	0.146 - 0.460	0.1861 - 0.5004	0.1740 - 0.488
	FQ	0.2315 - 1.6037	0.1752 - 1.6050	0.231 - 1.603	0.174 - 0.488
	DRR	0.2361 - 0.793	0.2022 - 0.404	0.1861- 1.0939	0.175 - 1.270

In above simulation four examples are simulated, by changing their queues and seen how it responds to jitter and delay.

As from above table 7.8 and 7.9 for DropTail, RED queue gives the best performance for rtProtoMIN as compared to rtProtoDV. In case of DRR queue for first three example rtProtoMIN gives good performance than rtProtoDV, but for last example jitter and delay increases as compare to rtProtoDV.

Simulation result with queue FQ rtProtoMIN does not give better results as compared to rtProtoDV. So from above results it can be concluded that rtProtoMIN gives good performance for DropTail, DRR, RED queues but not for FQ.

8 SUMMARY AND CONCLUSION

8.1 SUMMARY

In computer networking the term routing refers to selecting paths in a network along which to send data. Recent research has lead not only to many routing protocols, but to many routing techniques. By a routing technique we mean the basic strategy that a routing protocol uses to store and propagate routing information. Game theory was invented to provide a mathematical foundation for reasoning about conflict and competition. It has grown into a rich theory, with powerful mathematical and computational tools. Two insights enabled us to turn this pool of theory and tools into a potentially powerful analytical capability for analyzing routing.

The objective of this dissertation work is to apply Game Theoretic Approach on routing protocols for removing inefficiencies and analyzing how it is better than conventional TCP/IP routing protocols. The RIP (Routing Information Protocol) uses Distance Vector (DV) routing protocol for finding the path from source to destination, and sends the packets. DV routing is the implements Distributed Bellman-Ford (or Distance Vector) routing. As in chapter 2 rtProtoDV is RIP implementation in Network Simulator 2 (NS2) unicast routing. Distributed Bellman-Ford algorithm is replaced (or Distance Vector) with game theory algorithms, like minimax algorithm for finding path. These two protocols rtProtoDV and rtProtoMIN is simulated on different scenarios and analyzed for different parameters like delay jitter, delay, processing time, throughput, convergence etc. In scenario 1 as packet size increases rtProtoMIN not gives that much better performance than rtProtoDV. Delay graph for four examples is as shown in figure 8.1.

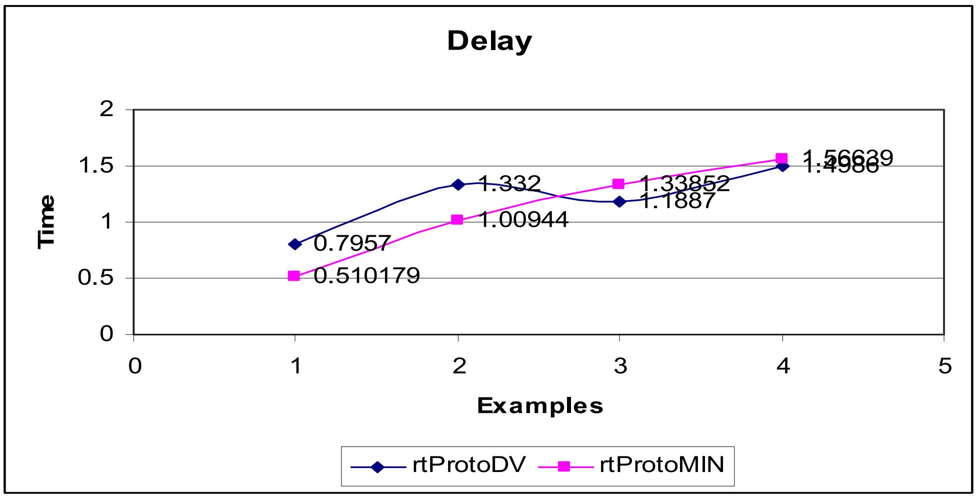
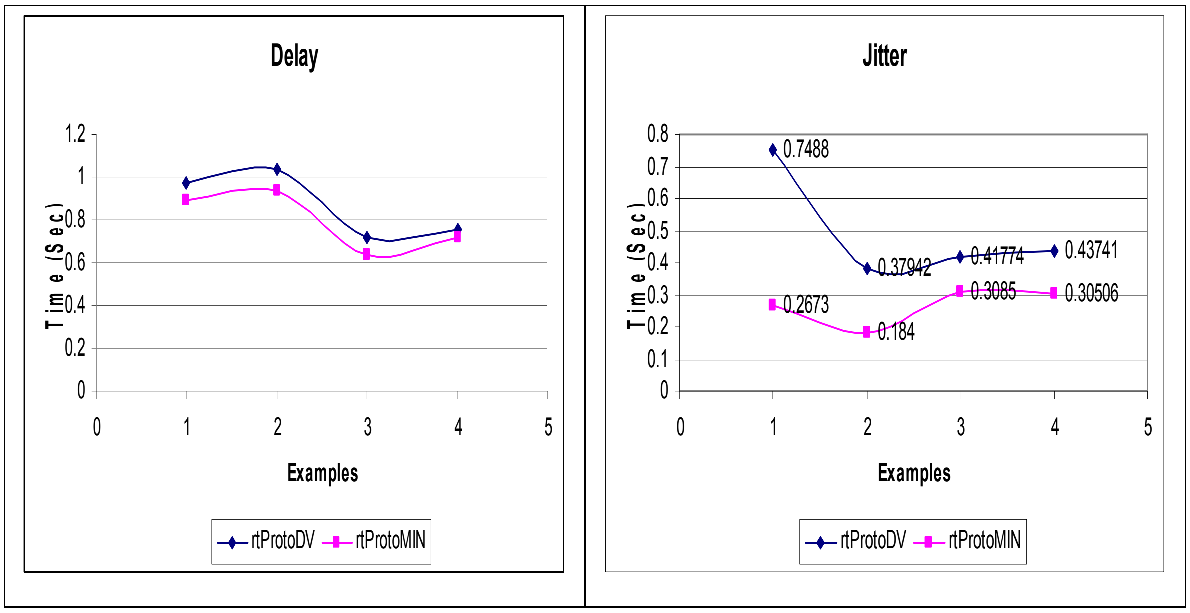


Figure 8.1: Scenario 1: Delay

As from simulation for scenario2 fixed packet size and varying network, rtProtoMIN got good results compared to rtProtoDV shown in figure 8.2. With varied queue like (Droptail, DRR, RED) rtProtoMIN gives the good performance. It means that game theory algorithm used in rtProtoMIN that is minimax algorithm not gives the better performance for each scenario but for specific scenarios it gives good performance.

There are some other algorithms like alpha-beta pruning algorithm which may give the good performance than minimax algorithm.



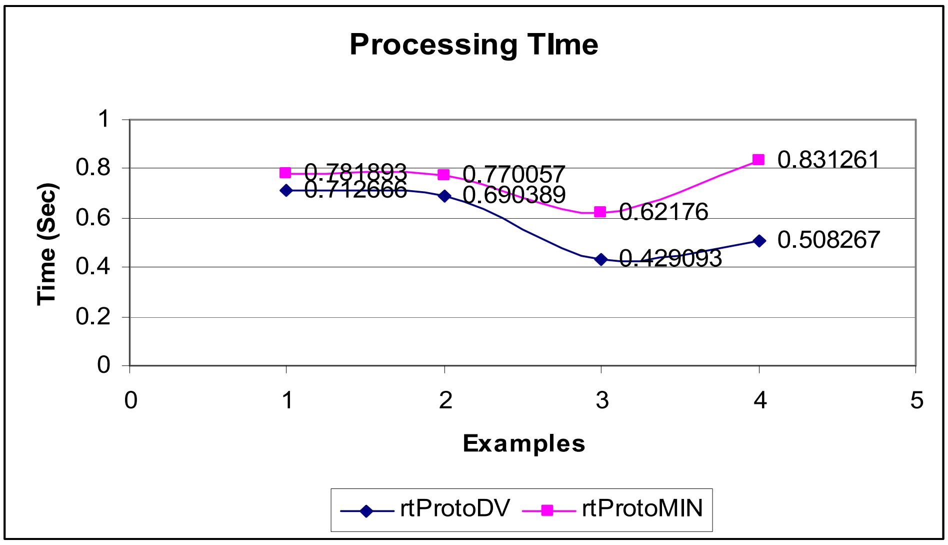


Figure 8.2: Graphs Of Scenario 2

8.2 CONCLUSIONS

In routing networks, game theory has been used to analyze the cooperation of the nodes. The game theoretic approaches try to analyze the problem using a more analytical viewpoint. rtProtoMIN and rtProtoDV is analyzed on different scenarios, and analyzed on different parameters like delay jitter, delay, processing time etc.

In first scenario packet size is varied and network is fixed, in that case as packet size increases the performance of rtProtoMIN decreases as compare to rtProtoDV. For packet size 2540 rtProtoMIN and rtProtoDV gives same performance. In second scenario packet size is fixed (1040) and network is varied in that rtProtoMIN gives good performance as compared to rtProtoDV. But in both the cases the simulation processing time of rtProtoMIN increased as compared to rtProtoDV. In third scenario simulation setup is same as scenario 2 but varying the queue like DropTail, FQ, RED, DRR etc from example to example. In this case for Drop Tail, RED and DRR queues rtProtoMIN gives good performance as compared to rtProtoDV but, for queue FQ it gives the worst results.

From above scenario rtProtoMIN gives the good performance with fixed type of scenario like as second scenario if packet size increases the performance of rtProtoMIN goes on decreases. As game theory achieves the cooperation between routers in network, when best route is found from source to destination the congested link is not included in best path.

8.3 FUTURE WORK

As seen in above chapters on rtProtoDV (RIP) game theory applied and observes how game theory improves performance. rtProtoMIN is uses minimax algorithm for finding path to destination node and, also analyzed on different scenarios. For some specifications rtProtoMIN gives good performance than rtProtoDV.

But there are different game theoretic algorithms like alpha beta pruning (one step ahead of minimax algorithm), repetitive game theory etc. are used in implementation then there are more expectations of performance improvements. The strategy used for implementations of rtProtoMIN, same strategy is applied on other protocols like OSPF.

REFERENCES

1. S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E.Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: a case for Informed Internet routing and transport. In *IEEE Micro*, volume 19(1),

2. H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin. "The impact of routing policy on Internet paths". In *Proceedings of IEEE INFOCOM '01*, Anchorage, AK, Apr. 2001.

3. Irfan Zakiuddin a, Tim Hawkins b, Nick Moffat b."Towards A Game Theoretic Understanding of Ad-Hoc Routing". In Electronic Notes in Theoretical Computer Science

4. A. Akella et al., "Selfish behavior and stability of Internet: A game theoretic analysis of TCP," Proceedings of *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, August 2002, pp. 117-130.

5. Game Theory :Theodore L. Turocy Texas A&M University, Bernhard von Stengel London School of Economics

6. Lecture Notes by Y. Narahari "*Game Theory*" Department of Computer Science and Automation , Indian Institute of Bangalore , India September 2005.

7. Martin J. Osborne, Arial Rubinstin, *A Course in Game Theory*, MIT Press, July 1994.

8. Todd Lammle, *CCIE Professional CCNA Study guide,* fourth edition

9. Jeff Doyle, Jennifer Carroll, *TCP/IP Routing* Volume 1: second edition

10. David Leberknight, "Object oriented programming and design minimax search algorithm" page 7

11. The network simulator – NS-2. http://www.isi.edu/nsnam/ns.

12. Eitan Altman and Tania Jimenez " NS Simulator for beginners" Lecture Notes 2003-2004 , Univ. de Los Andes , Merida , Venezuela and ESSI Sophia-Antipolis , France

13. Minimax pseudo code and implantation details: http://www.pressibus.org/ataxx/autre/minimax/node2.html

14. Robin J. G. Upton "Dynamic Stochastic Control: A New Approach to Tree Search & Game-Playing" PhD thesis Statistics Department University of Warwick, UK 23 April 1998.

15. Francisco J. Ros, Pedro M. Ruiz "Implementation of new unicast routing protocol in ns2 " Dept. of Information and Communications Engineering University of Murcia December, 2004

16. Kevin Fall, Kannan Varadhan "*The ns Manual (formerly ns Notes and Documentation)*" The VINT Project a Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC.

17. TraceGraph tool Description: http://www.tracegraph.com/conference.html

APPENDIX A MINIMAX PSEUDOCODE

The minimax algorithm is a method of selecting the best choice of action in a situation, or game, where two opposing forces, or players, are working toward mutually exclusive goals, acting on the same set of perfect information about the outcome of the situation. It is specifically applied in searching game trees to determine the best move for the current player of a game. It uses the simple principle that at each move, the moving player will choose the best move available.

A.1 MINIMAX PSEUDO CODE

```
minimax (in game board, in int depth, in int max_depth,
        out score chosen_score, out score chosen_move)
begin
    if (depth = max_depth) then
        chosen_score = evaluation (board);
    else
        moves_list = generate_moves (board);
        if (moves_list = NULL) then
            chosen_score = evaluation (board);
        else
            for (i = 1 to moves_list.length) do
                best_score = infinity;
                new_board = board;
                apply_move (new_board, moves_list[i]);
                minimax (new_board, depth+1, max_depth, the_score, the_move);
                if (better (the_score, best_score)) then
                    best_score = the_score;
                    best_move = the_move;
                endif
            enddo
            chosen_score = best_score;
            chosen_move = best_move;
         endif
      endif
   end.
```

A.2 FUNCTIONS:

generate_moves (board):

This function generates all possible moves for the current player. It takes the current board situation as the argument and returns the all possible moves at particular situation.

Evaluation(board):

Evaluation is called just before exiting each node in the minimax tree. It should return a value representing the strength of the board in chosen_score. best_score holds the best value returned by any children of the current node in the minimax tree.

apply_move :

This function takes a board and a move, returning the board with all the updates required by the given move.

better (the_score, best_score) :

This function takes 2 scores to compare and a player, returning the score that is more advantageous for the given player. If scores are stored as simple integers, this function can be the standard < and > operators.

Run ():

This function gives the path from source to destination.

Cost_fuction ():

This function takes the Run output as argument and finds the cost of reaching destination.

APPENDIX B AWK SCRIPTS

As seen in chapter 4 when simulating any tcl script of NS2 in that case two files are generated, one is NAM visualization file which shows the animation of network and other is trace file which contains the trace of every millisecond of simulation. The trace format of trace file is explained in chapter 4. This trace file contains only textual data, we cannot visualize form that data. So for calculating parameters likes jitter, delay, throughput, processing time etc awk scripts are used.

B.1 BASICS OF AWK

Awk is a full-featured text processing language with syntax reminiscent of **C**. While it possesses an extensive set of operators and capabilities. Awk breaks each line of input passed to it into fields. By default, a field is a string of consecutive characters delimited by white spaces, though there are options for changing this. Awk parses and operates on each separate field. This makes it ideal for handling structured text files -- especially tables -- data organized into consistent chunks, such as rows and columns.

- The structure of awk commands
  - Each awk command consists of a selector and/or an action; both may not be omitted in the same command. Braces surround the action.
  - selector [only] -- action is print
  - {action}[only] -- selector is every line
  - selector {action} -- perform action on each line where selector is true
  - Each action may have multiple statements separated from each other by semicolons or \n

- Line selection
    - A selector is either zero, one, or two selection criteria; in the latter case the criteria are separated by commas
    - A selection criterion may be either an RE or a boolean expression (BE) which evaluates to true or false
    - Commands which have no selection criteria are applied to each line of the input data set
    - Commands which have one selection criterion are applied to every line which matches or makes true the criterion depending upon whether the criterion is an RE or a BE
    - Commands which have two selection criteria are applied to the first line which matches the first criterion, the next line which matches the second criterion and all the lines between them.
    - Unless a prior applied command has a next in it, every selector is tested against every line of the input data set.

- Processing
    - The BEGIN block(s) is(are) run (mawk's -v runs first)
    - Command line variables are assigned
    - For each line in the input data set
        - It is read and NR, NF, $I, etc. are set
        - For each command, its criteria are evaluated
        - If the criteria is true/matches the command is executed
    - After the input data set is exhausted, the END block(s) is(are) run
- Fields
    - Each record is separated into fields named **$1**, **$2**, etc
    - **$0** is the entire record
    - **NF** contains the number of fields in the current line
    - **FS** contains the field separator RE; it defaults to the white space RE, **/[<SPACE><TAB>]*/**
    - Fields may be accessed either by **$n** or by **$var** where **var** contains a value between **0** and **NF**

So by using awk script processes the trace file data and calculates different parameters of network.

- Delay = packet receive time at destination node – packet send time at source node.
- Round Trip Time (RTT) = acknowledge (ACK) packet receive time at source node – packet send time at source node.
- Processing time = packet forward time at intermediate node – packet receive time at intermediate node.
- Jitter = absolute value of (delay of packet i+1 – delay of packet i), where i = 1.number of sent packets – 1.
- Throughput = number of generated/sent/received/forwarded/dropped packets or bits in a certain time interval.

For Example:

Generated Packets:

```
Awk ‘BEGIN

          {SendPack =0}

          $1 = = "+" && $5 = = "tcp" && $3 = = "0"

          {SendPack = SendPack + 1}

     END {print SendPack} ' rip_example.tr
```

Drop Packets:

```
Awk  'BEGIN

          {DropPack =0 }

          $1 = = "d" && $5 = = "tcp" && $8 = = "0" && $6 = = "1040"

          {DropPack = DropPack + 1}

     END {print DropPack } ' rip_example.tr
```

In this way by using awk script different parameters of of routing network is calculated.