A SIMULATION APPROACH TO AN OPTIMAL DAILY BANDWIDTH ALLOCATION IN A 250 NODES VIRTUAL PRIVATE NETWORK (VPN)

J.F. Opadiji,
jopadiji@yahoo.com
Dept. of Electrical Engineering,
University of Ilorin, Ilorin, Nigeria
&
T.A. Abdul – Hameed
taoabdulhameed@yahoo.co.uk
Dept. of Electrical/Electronics Engineering
Federal Polytechnic, Ede, Nigeria

All correspondence to the 2nd Author

Abstract
In order to make better use of available network resources, there is a need for planning the bandwidth allocation to communication demands. In this study a bandwidth allocation problem was formulated with three major constraints. A dynamic mathematical model was developed and a Genetic Algorithm method was adopted for an optimization solution. The GA algorithm was implemented with Java programming language. The model was simulated for ten thousand (10,000) generations. Two hundred and fifty (250) nodes were simulated differently under varying bandwidths values of 64Kbps, 128Kbps, 256Kbps, 512Kbps, 1Mbps, 2Mbps, 4Mbps, and 8Mbps for a period of twenty four (24) hours. Simulation results show that utilization factors can be as high as ninety nine percent if optimization conditions are scrupulously observed.

Key Words: Bandwidth, Genetic Algorithm (GA), Simulation, Model, Virtual Private Network (VPN)

1.0 Introduction

A virtual private network (VPN) is an extended network within a network that uses a public telecommunication infrastructure and their technology such as the Internet, to provide remote offices or individual users with secure access to their organization's network. It aims to avoid an expensive system of owned or leased lines that can be used by only one organization. The goal of a VPN is to provide the organization with the same secure capabilities but at a much lower cost. Without proactive management, network capacity fills with inappropriate traffic and viruses, and the connection becomes ineffective. [1, 2, 3]

Virtual Private Network (VPN) combines two concepts: virtual networking and private networking. In a virtual network, geographically distributed and remote nodes can interact with each other the way they do in a network where the nodes are collocated. The topology of the virtual network is independent of the physical topology of the facilities used to support it. A virtual network is managed as a single administrative entity [4, 5,6].

Bandwidth optimization is one of many concerns of networking engineers. With the exponential growth of digitally rich contents and Internet computing demands for the last few years, the users often perceive that there is insufficient bandwidth available to completely satisfy their needs whereas the problem lies at the end of management who fails to identify certain bandwidth eating unproductive applications [5]. Even the simplest bandwidth optimization techniques can reduce bandwidth costs significantly.

Bandwidth in developing countries is so expensive that most organizations and institutions cannot afford to purchase a sufficient quantity for the users.

2.0 Bandwidth Constrained Optimization Problems

Several studies make use of Genetic Algorithm (GA) based techniques to solve network problems. The motivation behind GA's in nonlinear optimization problems is that the problem can be expressed such that natural evolution, as reported, can provide an attractive paradigm for implementing general nonlinear searches [7, 8, 9].

3.0 System Modeling & Problem Formulation

A model is a simplified representation of the relevant aspects of an actual system or a process. A mathematical model is the characterization of a process, concept or object in terms of explicit mathematical forms. In a mathematical model, the components of an object or system and the relationships of its parts are expressed as mathematical symbols. A dynamic model explains how a situation or system changes. [10s]

Consider a virtual network consisting of "n" nodes represented by n_1, n_2, n_3, --------, n_n as shown in Fig.3.1 below.

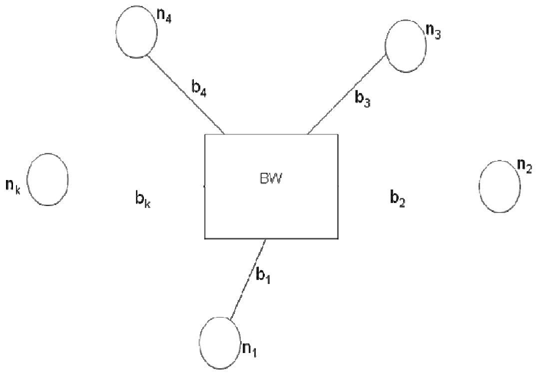

Fig.3.1: A schematic of a typical virtual private network

Assuming the total available bandwidth for all the nodes at any instant is "BW". We try to find an optimal VP bandwidth assignment, which maximizes the total expected network throughput, given the network topology; expected Origin-Destination (OD) traffic loads; and link capacities. An optimal allocation of bandwidth among all VPs such that all demands across each node will always be satisfactorily met is the goal. Each node is considered to behave as a selfish overlay network.

Terms	Notation
Minimum Required Bandwidth at node n_i.	b_{mi}
Available Bandwidth at node n_i	b_i
Throughput Request at node n_i.	t_{ri}
Allocated Throughput at node n_i.	t_{ai}
Total Available Bandwidth for all the nodes at a given time.	BW
time (evaluation period)	t

In this project work we assumed a single VP's between an OD pair.

$$\max \sum_{i=1}^{k} t_{ai} \qquad (1) \qquad \text{s.t.} \quad \sum_{i=1}^{k} b_i \le BW \qquad (2)$$

$$t_{ai} \ge t_{ri} \quad \forall i = 1,2,3,\text{-------},k \qquad (3) \qquad b_i \ge b_{mi} \quad \forall i = 1,2,3,\text{-------},k \qquad (4)$$

To solve the above mathematical model taking the three identified constraints into consideration, statistical characteristics of the bandwidth demand functions from each node at any instant (t_{ri} – Throughput Request at node n_i) should be known. We assume that each function t_{ri} is derived from an appropriate probability density function for bandwidth demand. By considering throughput as "fluid flow", an optimal solution can be searched for.

4.0 Model Simulation and Results

The implementation tool for the model was Java program. (The source code is not included in this paper for brevity purpose). The number of generations matters a lot in deciding the fitness value and subsequently in arriving at a value very close to or equal to the optimal value. The model was simulated for ten thousand (10,000) generations in order to obtain a better solution. Two hundred and fifty (250) nodes were simulated differently under varying bandwidths values of 64Kbps, 128Kbps, 256Kbps, 512Kbps, 1Mbps, 2Mbps, 4Mbps, and 8Mbps for a period of twenty four (24) hours.The results obtained from the model simulation on daily bandwidth consumption and utilization are as tabulated in tables 1 and 2 whilst the utilization curve is as in figure 1.

Table 1 : Daily Bandwidth Consumption for 250 Nodes

Period	@ 64Kbps	@ 128Kbps	@ 256Kbps	@ 512Kbps	@ 1Mbps	@ 2Mbps	@ 4Mbps	@ 8Mbps
12 midnight	65280	130020	258480	523080	1047900	2095620	4192440	8387760
1a.m	65280	130020	261180	524280	1047900	2097000	4192800	8387760
2a.m	65280	130020	261960	524280	1047900	2097000	4193280	8388300
3a.m	65280	130020	261960	524280	1047900	2097000	4194000	8388420
4a.m	65280	130440	261960	524280	1047900	2097000	4194240	8388420
5a.m	65280	130440	261960	524280	1047900	2097000	4194240	8388420
6a.m	65280	130440	261960	524280	1047900	2097000	4194240	8388600
7a.m	65280	130860	262080	524280	1047900	2097000	4194240	8388600
8a.m	65280	130860	262080	524280	1048200	2097000	4194240	8388600
9a.m	65280	130860	262080	524280	1048200	2097000	4194240	8388600
10a.m	65280	130860	262080	524280	1048200	2097000	4194240	8388600
11a.m	65280	130860	262080	524280	1048200	2097000	4194240	8388600
12noon	65280	130860	262080	524280	1048320	2097000	4194240	8388600
1p.m	65280	130860	262080	524280	1048320	2097060	4194240	8388600
2p.m	65280	130860	262080	524280	1048320	2097060	4194240	8388600
3p.m	65280	130860	262080	524280	1048320	2097060	4194240	8388600
4p.m	65280	130860	262080	524280	1048320	2097060	4194240	8388600
5p.m	65280	130920	262080	524280	1048320	2097060	4194240	8388600
6p.m	65280	130920	262140	524280	1048320	2097060	4194240	8388600
7p.m	65280	130920	262140	524280	1048320	2097060	4194240	8388600
8p.m	65280	130920	262140	524280	1048320	2097060	4194240	8388600
9p.m	65280	130920	262140	524280	1048320	2097060	4194240	8388600
10p.m	65280	130920	262140	524280	1048320	2097060	4194240	8388600
11p.m	65280	130920	262140	524280	1048320	2097060	4194300	8388600

Table 2: Daily Bandwidth Utilization for 250 Nodes

Period	@ 64Kbps	@ 128Kbps	@ 256Kbps	@ 512Kbps	@ 1Mbps	@ 2Mbps	@ 4Mbps	@ 8Mbps
12 midnight	0.996094	0.991974	0.986023	0.997696	0.999355	0.999269	0.999556	0.999899
1a.m	0.996094	0.991974	0.996323	0.999985	0.999355	0.999928	0.999641	0.999899
2a.m	0.996094	0.991974	0.999298	0.999985	0.999355	0.999928	0.999756	0.999963
3a.m	0.996094	0.991974	0.999298	0.999985	0.999355	0.999928	0.999928	0.999978
4a.m	0.996094	0.995178	0.999298	0.999985	0.999355	0.999928	0.999985	0.999978
5a.m	0.996094	0.995178	0.999298	0.999985	0.999355	0.999928	0.999985	0.999978
6a.m	0.996094	0.995178	0.999298	0.999985	0.999355	0.999928	0.999985	0.999999
7a.m	0.996094	0.998383	0.999756	0.999985	0.999355	0.999928	0.999985	0.999999
8a.m	0.996094	0.998383	0.999756	0.999985	0.999641	0.999928	0.999985	0.999999
9a.m	0.996094	0.998383	0.999756	0.999985	0.999641	0.999928	0.999985	0.999999
10a.m	0.996094	0.998383	0.999756	0.999985	0.999641	0.999928	0.999985	0.999999
11a.m	0.996094	0.998383	0.999756	0.999985	0.999641	0.999928	0.999985	0.999999
12noon	0.996094	0.998383	0.999756	0.999985	0.999756	0.999928	0.999985	0.999999
1p.m	0.996094	0.998383	0.999756	0.999985	0.999756	0.999956	0.999985	0.999999
2p.m	0.996094	0.998383	0.999756	0.999985	0.999756	0.999956	0.999985	0.999999
3p.m	0.996094	0.998383	0.999756	0.999985	0.999756	0.999956	0.999985	0.999999
4p.m	0.996094	0.998383	0.999756	0.999985	0.999756	0.999956	0.999985	0.999999
5p.m	0.996094	0.99884	0.999756	0.999985	0.999756	0.999956	0.999985	0.999999
6p.m	0.996094	0.99884	0.999985	0.999985	0.999756	0.999956	0.999985	0.999999
7p.m	0.996094	0.99884	0.999985	0.999985	0.999756	0.999956	0.999985	0.999999
8p.m	0.996094	0.99884	0.999985	0.999985	0.999756	0.999956	0.999985	0.999999
9p.m	0.996094	0.99884	0.999985	0.999985	0.999756	0.999956	0.999985	0.999999
10p.m	0.996094	0.99884	0.999985	0.999985	0.999756	0.999956	0.999985	0.999999
11p.m	0.996094	0.99884	0.999985	0.999985	0.999756	0.999956	0.999999	0.999999

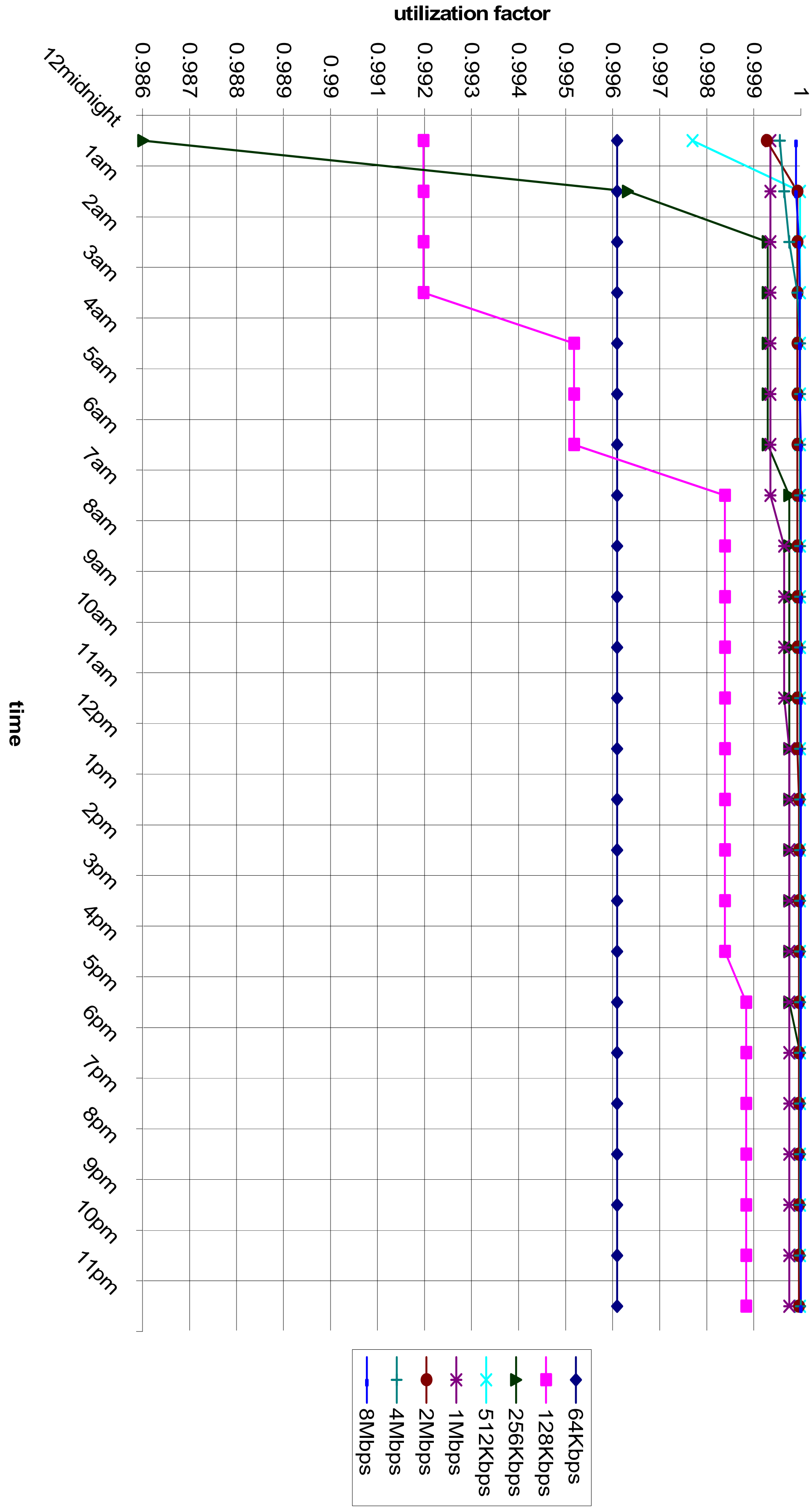

FIGURE 1: DAILY BANDWIDTH UTILIZATION FOR 250 NODES

From tables 1 and 2 as well as figure 1, the following inferences could be inferred for various nodes at different bandwidths.

(i) The bandwidth consumption varies randomly with time.

(ii) With the 250 nodes considered in this work, it gets to a point where there is a considerable increase in the bandwidth consumption and the curve becomes non-linear. The implication is that bandwidth is wasted whenever there is no proportional increment in available nodes.

(iii) There is always a critical point in bandwidth availability at which further increase in available bandwidth (BW) did not improve the utilization factor for the nodes. In actual fact the utilization factor starts decreasing. If the cost of purchase of bandwidth is to be minimized and available bandwidth optimized, VPN must not be operated above the point.

(iv) Each node acts adaptively and optimally to the dynamics of the external environment so that the available bandwidths are shared optimally for each node despite the fact that each node behaves as a selfish node.

(v) The utilization factor at the optimal condition can be as high as 99.99% (250 nodes @ 4096/8192kbps).

Conclusion

This study was on bandwidth optimization for virtual private network. Based on experience, a problem was formulated taken cognizance of the constraints. A dynamic mathematical model was developed. The study adopted a Genetic Algorithm method for the optimization solution in allocation of bandwidth. The GA algorithm was implemented with Java programming language. The model was simulated for ten thousand (10,000) generations in order to obtain a better solution. Two hundred and fifty (250) nodes were considered. Each of the nodes were simulated differently under varying bandwidths values of 64Kbps, 128Kbps, 256Kbps, 512Kbps, 1Mbps, 2Mbps, 4Mbps, and 8Mbps for a period of twenty four (24) hours.

Simulation results show that if the cost of purchase of bandwidth is to be minimized and the available bandwidth optimized, the number of nodes and utility must be commensurate with the quantity of bandwidth purchased by operators of Virtual Private Network. The utilization factors can be as high as ninety nine percent if the method proposed in this study is carefully observed and implemented.

REFERENCES

1. Benvenutti, C. (2007): “Bandwidth Optimization” AfREN, Abuja, Nigeria.

2. Jaffar, J. (1999): “Resource allocation in Networks using Abstraction and Constraint Satisfaction Techniques”, CP’99, LNCS 1713, pp. 204–218.

3. Tanterdtid, S., Steanputtangaul, W., and Benjapolakul, W. (1997): “Optimizing ATM network throughput based on Virtual Paths concept by using Genetic Algorithm”, Proc. IEEE ICIPS’97, Beijing, 1634–1639.

4. VMware White Paper (2006): “Network Throughput in a Virtual Infrastructure”, V Mware Inc., Palo Alto, U.S.A.

5. Sharma, V.; Kumar, V. & Thakur, B.S.: “Need of Bandwidth Management and Formulation of Policy Framework for Effective Utilisation of Internet Services within a University Campus”, International Journal of Computer Science and Communication, Vol. 2, No. 1, January-June 2011, pp. 173-178

6. Banga, V.K., Singh, Y. and Kumar, R.(2007): “Simulation of Robotic Arm using Genetic Algorithm and Analytical Hierachy Process (AHP)”, World Academy of Science, Engineering and Technology.

7. Goldberg, D. E. (1991): Genetic Algorithm in search, “optimization and machine learning”, New York, Addison Wesley.

8. Heitkoetter, J. and Beasley, D. (1994): Eds. The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ). USENET: comp.ai.genetic. rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic/.

9. Podnar, H. and Skorin-Kapov, J. (2002): “An application of a genetic algorithm for throughput optimization in non-broadcast WDM optical networks with regular topologies”, Mathematical Communications 7, p.45-59.

10. Osuagwu, O. E. (2007): Computer – Modelling, Research Analysis and Designing Support System: Oliverson Industrial Publishing House, Owerri, Nigeria. P 4-6.

APPENDIX
SOURCE CODE

```
/*
 * dynamicbandwidth.java
 *
 * Created on 6 May 2010, 19:32
 *
 * To change this template, choose Tools | Template
Manager
 * and open the template in the editor.
 */
package dynamicbandwidth;
import java.util.*;
import java.io.*;
import java.lang.*;
/**
 *
 * @author Taofeek
 */
public class dynamicbandwidth {
   /** Creates a new instance of dynamicbandwidth
*/
   public void dynamicbandwidth() {
   }

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args)
   {
    boolean generate_throughput = false;
      RandomAccessFile solution_matrix = null;
     long start_time = System.currentTimeMillis();
     long run_time = 0;
      int cpoint = 0;
     int no_of_solutions=8;
     int no_of_nodes = 80;
     int[] max_Throughput_request =new
int[no_of_nodes]; //{5,1,5,2,7,7,8,9,8,1};
     int Max_BW=8388608; int time=60;int mm=0;
0,0};
     int[] dummy_solution=new int[no_of_nodes];
//= {0,0,0,0,0};
     for (int j=0; j<no_of_nodes;j++)
     {
     dummy_solution[j]=0;
//System.out.print(dummy_solution[j]);
     }
      //int[][] initial_population =
{{10,10,10,10,10},{6,5,4,6,5},{7,6,4,3,5},{5,4,3,7,
6}};
      int[][] initial_population = new
int[no_of_solutions][no_of_nodes];

      for (int w=0 ; w < no_of_solutions; w++)
      {
      for (int i = 0; i < no_of_nodes; i++)
      {
      initial_population[w][i] =
(int)(10*Math.random());
      }
      }
      for(int y=0;y<no_of_solutions;y++)    //code
for printing
       {
          for(int i=0;i<no_of_nodes;i++)
             {

System.out.print(initial_population[y][i]+"\t");
             }
             System.out.print("\n");
       }
      int[] min_BW_per_node =
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,

      String filename = new String();
      RandomAccessFile Nodes_throughput_file =
null;
      Random fs = new Random();
      filename = String.valueOf(no_of_nodes);
      filename = filename.concat("nodes.dat");
      int hours = 1;
```

```
    int states = 24;
    int no_of_hours = 24;
    if (generate_throughput == true)
    {
     try
     {
        Nodes_throughput_file = new
RandomAccessFile(filename,"rw");
     } catch (FileNotFoundException ffe) {
        System.out.println("Error: File not found");
System.exit(0);
     }
    for(int i = 0; i < no_of_hours; i++)
    {
       try
          {
          for (int s=0;s<no_of_nodes;s++)
          {

Nodes_throughput_file.writeBytes(String.valueOf(f
s.nextInt(850)));
          Nodes_throughput_file.writeByte(13);
          }
          Nodes_throughput_file.writeByte(13);

          } catch (IOException ioe) {};

    }
    }
    if (generate_throughput == true)
     try
     {
        Nodes_throughput_file.close();
     } catch (IOException ioe) {
        System.out.println("File not found");
System.exit(0);
     }

    do
    {
       int generations=1000;
       int iteration = 0;
       Random p = new Random();
       //filename = "nodes1.dat";


       //filename =
filename.concat(String.valueOf(hours)).concat(".dat
");
       /*if (hours > 1){
           try
           {
              Nodes_throughput_file.close();  // =
new RandomAccessFile(filename,"r");
           }
           catch (IOException ioe) {
              System.out.println("File not found");
System.exit(0);
           }
         }*/

        try
        {
           Nodes_throughput_file = new
RandomAccessFile(filename,"r");
        } catch (FileNotFoundException ffe) {
           System.out.println("File not found");
System.exit(0);
        }
       int[][] mmax_Throughput_request=new
int[24][no_of_nodes];
       for (int u = 0; u < 24; u++)
         {
          String empty = "";
          for (int i = 0; i < no_of_nodes; i++)
          {
            try
            {
            mmax_Throughput_request[u][i] =
Integer.parseInt(Nodes_throughput_file.readLine())
;
            }catch (IOException ioe) {
               System.out.println("Unable to read
data");
               System.exit(0);
            }

          }

          try
            {
            empty =
(Nodes_throughput_file.readLine());
            }catch (IOException ioe) {
               System.out.println("Unable to read
data");
               System.exit(0);
            }
        }

        try
        {
           Nodes_throughput_file.close();  // = new
RandomAccessFile(filename,"r");
        } catch (IOException ioe) {
           System.out.println("File not found");
System.exit(0);
        }
```

```
    for (int j=0;j<no_of_nodes;j++)
    {
max_Throughput_request[j]=mmax_Throughput_re
quest[hours-1][j];
    }
    for(int f=0;f<no_of_nodes;f++)
           {
              if(max_Throughput_request[f]>mm)
              {
              mm=max_Throughput_request[f];
              }
           }
     //System.out.println(mm);
     int current_population =
initial_population.length;
    chromosome[] solution = new
chromosome[100000];
     for (int i = 0; i < current_population; i++)
        {
         solution[i] = new
chromosome(initial_population[i]);
        }
     for (int i = current_population; i < 100000; i++)
        {
         solution[i] = new
chromosome(dummy_solution);
        }
     population solution_space = new
population(solution);
     double rate = 0.3;
    while (iteration < generations)
     {
         do
         {
         cpoint =
(int)(no_of_nodes*Math.random());
         } while (cpoint == 0 || cpoint ==
(no_of_nodes-1));
       //System.out.println(cpoint);
     for (int j = 0; j < current_population; j++)
     {


solution_space.individual[j].eligibility(Max_BW,ti
me,max_Throughput_request,no_of_nodes,min_B
W_per_node);
   if (solution_space.individual[j].validity == false)
          solution_space.individual[j].fitness = -1;
    if (solution_space.individual[j].validity == true)
        {

solution_space.individual[j].calculate_cost();

solution_space.individual[j].calculate_fitness(no_of
_nodes);
          }

      }

            solution_space.ranking(current_population);

            current_population =
solution_space.mating(current_population,cpoint,no
_of_nodes);
            for (int j = 0; j < current_population; j++)
            {

solution_space.individual[j].eligibility(Max_BW,ti
me,max_Throughput_request,no_of_nodes,min_B
W_per_node);
          if (solution_space.individual[j].validity ==
false)
             solution_space.individual[j].fitness = -1;
          if (solution_space.individual[j].validity ==
true)
          {

solution_space.individual[j].calculate_cost();

solution_space.individual[j].calculate_fitness(no_of
_nodes);
          }

//System.out.println(solution_space.individual[j].val
idity+"\t");

            }
            solution_space.ranking(current_population);

solution_space.mutation(rate,current_population,no
_of_nodes,mm);
             for (int j = 0; j < current_population; j++)
             {


solution_space.individual[j].eligibility(Max_BW,ti
me,max_Throughput_request,no_of_nodes,min_B
W_per_node);
          if (solution_space.individual[j].validity ==
false)
             solution_space.individual[j].fitness = -1;
          if (solution_space.individual[j].validity ==
true)
          {

solution_space.individual[j].calculate_cost();
```

```
solution_space.individual[j].calculate_fitness(no_of
_nodes);
          }

//System.out.println(solution_space.individual[j].val
idity+"\t");

          }
          solution_space.ranking(current_population);
          for (int k = 0; k < current_population; k++)

System.out.println(solution_space.individual[k].fitn
ess + "\t");
          iteration++;
          System.out.print("\n");

      }
      for(int u=0;u<no_of_nodes;u++)
      {
      System.out.print("ta_"+(u+1)+"=
"+solution_space.individual[0].ta[u]+";"+ "\t");
      }
      System.out.print("\n");

      for(int i=0;i<no_of_nodes;i++)
      {
         System.out.print("bi_"+(i+1)+"=
"+solution_space.individual[0].bi[i]+";" + "\t");
      }
      System.out.print("\n");
      System.out.print("Total bi=
"+solution_space.individual[0].total_bi+
"\t");hours++;
     }while (hours <= states);
   }
}
class chromosome
{
  boolean validity = false;
  double cost = 0.0;  int [] bi;
  double fitness = 0.0;int total_bi = 0;
  int [] ta;
   public chromosome(int[] gene_val)
   {
      ta = gene_val;
   }
   public void calculate_cost()
   {

   }
   public void calculate_fitness(int nn)
   {
      fitness=0;
      for(int i=0;i<nn;i++)
      {
      fitness=fitness+ta[i];
      }

   }
public void eligibility(int M_BW,int t,int[]
max_tr,int nn,int[] min_BW_p_node)
   {
    int ttcount = 0; int tcount = 0; int count = 0;  bi=
new int[nn];
    total_bi=0;
   /*for(int i = 0; i < nn; i++)
   {
      if(ta[i] > max_tr[i])
//System.out.println("ta=
"+ta[i]+"\t"+"tr="+max_tr[i]);
      {
         count++;
      }
      //System.out.println(count);
   }*/
   for(int w=0;w<nn;w++)
   {
      bi[w]=(ta[w]*t);
      if(bi[w]<min_BW_p_node[w])
        {
         tcount++;
        }
   }
  for(int i=0;i<nn;i++)
   {
      total_bi=total_bi+bi[i];
       if(total_bi>M_BW)
       {
         ttcount++;
       }
   }
if ( count > 0 ||tcount > 0 || ttcount > 0)
   validity = false;
else
   validity = true;
}

}
class population
{
   chromosome[] individual;
   public population(chromosome[] citizen)
   {
      individual = citizen;
   }
   public void ranking(long pop)
   {
```

```
        int i = 0; long p = pop;
        chromosome pivot;
        do
        {
            if (individual[i].fitness <
individual[i+1].fitness)
            {
                pivot = individual[i];
                individual[i] = individual[i+1];
                individual[i+1] = pivot;
                i = 0;
            }
            else
                i++;

        } while(i < (p-1));
    }

    public int mating(int ns, int cp, int nn)
    {
        //codes for mating
        int ce,e;
        int pop=0; int i;
        int[] fp = new int[nn], sp = new int[nn];
        int[] fc = new int[nn], sc = new int[nn];
        for (i = 0; i < ns/2; i+=2)
        {
            //System.out.println(ns);
            for (int r = 0; r < nn; r++)
            {
                fp[r] = individual[i].ta[r];
                sp[r] = individual[i+1].ta[r];
            }

            //x = (int)(cp/16);
            //if((cp%16) > 0) y = 1;
            //ce = cp; //x + y;
            e = 0;
            while (e < cp)
            {
                fc[e] = fp[e];
                sc[e] = sp[e];
                e++;
            }
            //e=ce;
            while (e < nn)
            {
                fc[e] = sp[e];
                sc[e] = fp[e];
                e++;
            }
            for (int r = 0; r < nn; r++)
            {
                individual[(ns/2)+i].ta[r] = fc[r];
//individual[i+(int)(ns/2+0.5)].time = fc;
                individual[(ns/2)+i+1].ta[r] = sc[r];
//individual[(int)((ns+i)/2+0.5)+1].time = sc;
            }
        }
        //pop=i;
        pop = ns; //ns+(int)(ns/2);
        return pop;
    }
    public void mutation(double rt, int pop, int nn,int
m)
    {
        int mutation = (int)(rt*(pop-1)*nn);
        int[] mrow = new int[mutation];
        int[] mcol = new int[mutation];
        int k=0;
        for(int i=0; i<mutation; i++)
        {
            k = (int)((pop)*Math.random());
            if (k == 0)
            {
                i--;
            }
            else
            {
                mrow[i] = k;
            }
        }
        for(int i=0; i<mutation; i++)
        {
            k = (int)((nn)*Math.random());
            mcol[i] = k;
        }
        for (int i = 0; i < mutation; i++)
        {
            individual[mrow[i]].ta[mcol[i]] =
(int)((m+1)*Math.random());
        }
}
}
```