CS 6385

Algorithmic Aspects of Telecommunication Networks

SUMMER 2017

Project 2
Network Reliability Using Exhaustive Enumeration

Submitted

By

Aravind R

(**Net ID: axr156530**)

Instructor:

Andras Farago

Objective

In the project, study experimentally how the network reliability depends on the individual link reliabilities. The Experimental setting consists of a complete undirected graph on n = 5 nodes. The graph being complete has 10 total links. In the current setting, the nodes never fail but the links connecting the nodes have a probability of failure. The failure probability for the network is calculated by the following method. Index the links with the numbers 1,2,3,4...,10 (in arbitrary order). Take a parameter p, $0 \leq p \leq 1$, the same for every link. The parameter pi will take different values in the experiments calculated based on ID of the students. The system is considered operational, if the network is connected. The simulations have to determine the network reliability of the given five nodes. The project furthur explores whether random errors within the readings results in drastic variations in the reliability readings of the network.

Program Methodology

The project requires us to find the overall reliability of a fully connected 5 node network. The network is fully connected means all nodes in the network are connected to every other node in the network. The reliability of this network depends on the reliability of the links of the network. The experimental setting takes into account that the nodes are always up but each and every links are prone to failure. The reliability of the individual links are defined by the formula

Reliability = $p_i = p^{\lceil d_i/3 \rceil}$**;**

Where pi is the reliability factor for the ith link

P is a constant value that varies between 0.05 and 1 .

di is the randomness factor that is the ith digit of the individual utd id.

For finding the network reliability, we use Exhaustive enumeration. The methodology is a brute force approach in which we find the sum of individual reliabilities of all possible topological combination of the given number of nodes and links. In the current case there are 10 links in total. Each of the links are either up or down. There are 2^10 network topology combinations. The reliability of the fully connected network is calculated as the sum of the individual topologies that are connected. The reliability of individual topologies is calculated as the product of the reliability probability for each of the individual links of the topology.

Reliability of Topology i = f(p1,p2,…p10).

The overall reliability of the network is defined as the sum of reliability for each the individual topologies.

Reliability of the network = $\sum_{i=0}^{n} Ri$

Where Ri is the reliability of the individual topologies.

Implementation

Representation

1) Network topology representation

In order to simulate the various combination (2^10) of the network, we represent the state of each network as the binary form of numbers from 0 to 1023. So in state 0 (0000000000) ,all the links are down. In state 1 (0000000001), only link 1 is up and so on to state 1023 (1111111111) fully connected state. This allows us to manually debug the code for verifying the calculated value from the program. For example in topology configuration 1023(10 ones) all links are up. If the individual reliability value of the links is p=0.5, to calculate the overall reliability R10= (1*0.5)^10 = 0.000976525. This representation of the network ensures we can debug the system in case of any errors and check the overall correctness of the system. The topology (graph) representation is represented using adjacency list. Each node of the graph will have an entry and a list of neighbors that it is connected to in the list. The adjacency list ensures that no node is omitted from the overall topology even if it is not connected. It helps in finding if the network is connected or not and thereby avoiding potential error while calculating the reliability values. The representation of network as adjacency list also facilitates the verification of connectedness of the graph using another algorithm(DFS, Tarjan's Algorithm etc) to verify the result of the program.

2)Finding the connectedness of the graph

To find the reliability of a network, we have to simulate the various combinations of the network. Each link of the network can be in two states, either up(1) or down(0). The whole network is up if the network is connected. I.e. each and every node in the network is reachable. In order to find whether the given topology is connected or not, we use the Breadth First Search algorithm. The BFS algorithm, we take an arbitrary node and visit all its immediate neighbors first. We store the neighbors in a queue as they are visited. After visiting all the neighbors of the current node, we then pop the next node from the queue and visit all the neighbors of that node. The termination condition is when there are no mode nodes in the queue. For a connected graph, each and every node is reachable from every other node in the network. This means that at the end of BFS, every node in the network must have been visited. In case of a unconnected graph, BFS execution will have nodes which are not visited. This ensures that miscounting of reliabilities is avoided. The BFS module for the program can also be replaced by other algorithm as it is implemented as a function module in the overall program.

Technologies used

Programming Language	Java
Operating System	Windows 10
IDE	NetBeans
Visualization Tools	MS Excel,MS Visio

Program Details

Main Program	NW_Reliability.java

- The Main program of the system is implemented in NW_Reliability .java
- We represent the graph for the network by storing the link details such as source, destination and state of the link : Up or Down.
- The idea is to generate the entire set of topology combination with the given nodes.
- Each of the possible topologies is represented as an adjacency list for each of the 5 nodes.
- The adjacency list is generated for each of the network topology .
- The system then uses Breadth First Search for checking if the given topologies are connected or not.
- BFS algorithm checks for each nodes adjacent nodes and then goes to the next node.
- At the end of the breadth first search algorithm, we check if all the nodes are visited. If all the nodes are visited then it is a connected network.
- If the given topology is connected, then we calculate the reliability of the topology based on the formula mentioned.
- For each of the connected topologies, we calculate the overall reliability of the network.
- The individual reliability of the links are determined as: If the link state is up then reliability value =p if the link state is down then the value is (1-p).
- The overall reliability of the network is calculated as the sum of the individual connected topologies that the network can form.
- Simulating random error in the question involves generating random numbers within the 1024 intervals and flipping the state (connected or unconnected) of the topologies.
- This ensures that random errors are introduced in the network.

Supporting Program	Links.java

- The Links.java program implements the class that describes a link.
- It stores the details of the links such as the two nodes it connects and the state of the link in the current topology.

Algorithm

The simulation can be done as follows:

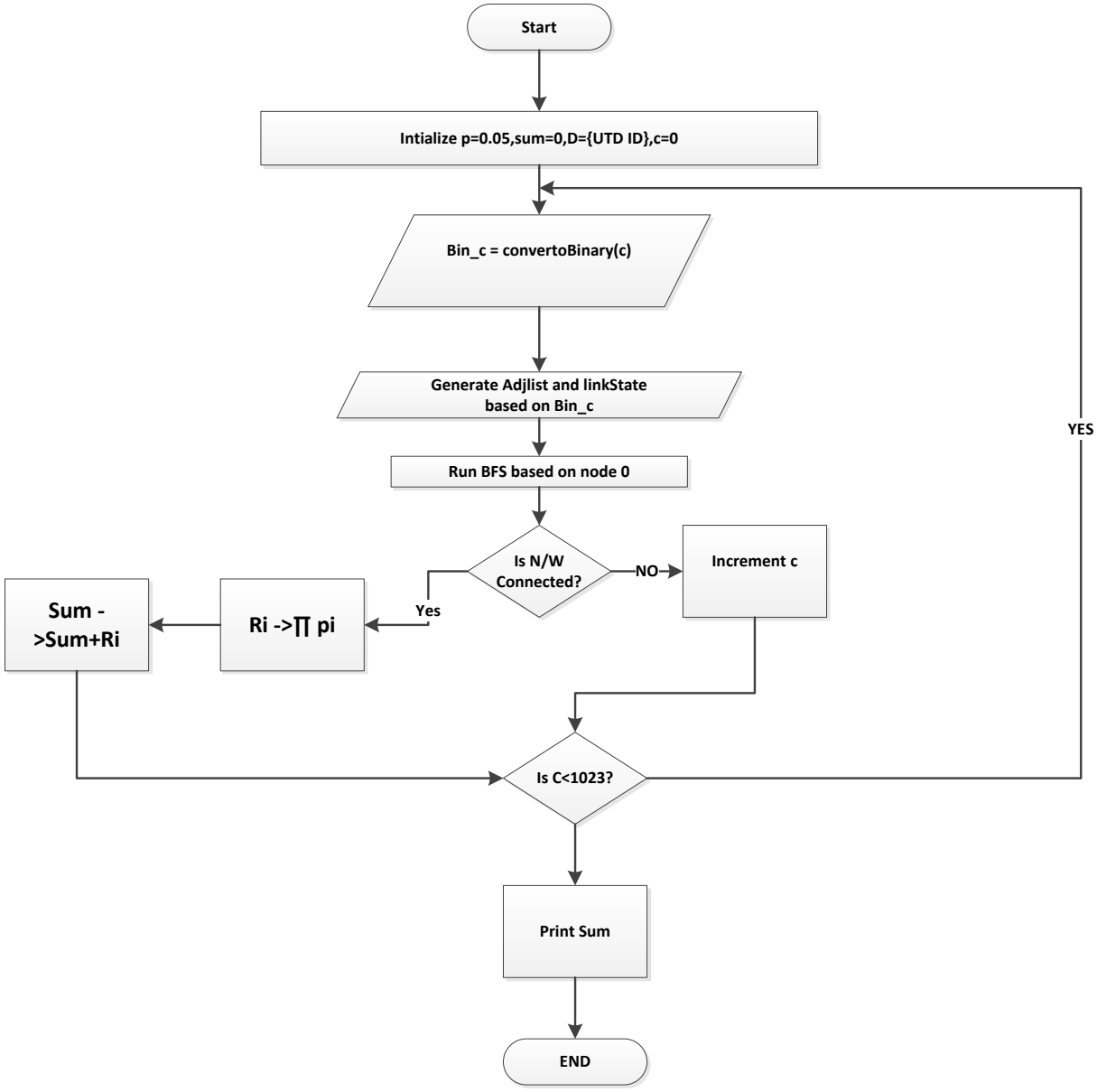
1. Initialize values of number of nodes and number of links.
2. Initialize p->0.05, Sum->0
3. Initialize array d as list of digits of the UTD id.
4. Start counter for c -> 0 to 1023 .
5. binaryc -> binary value of counter.
6. Generate the link states of the topology represented by the binary value of c.
7. Generate the link reliability Pi-> p ^di/3
8. Assign state of link i as down if ith digit of binary c is 0 else assign up.
9. Generate adjacency list of each nodes based on the current topology.
10. Run Breadth First Search algorithm on the adjacency list of the topology.
11. If the graph is connected, **Ri ->∏ pi.**
12. Calculate overall Network Reliability : **Sum -> sum+Ri.**
13. if c<1023 increment c goto step 5
14. Print overall Reliability value = Sum.
15. if p<1.05 then p->p+0.5 goto step 4
16. End

For Random error generation

1. Initialize values of number of nodes and number of links.
2. Initialize k->0
3. Initialize p->0.9, Sum->0,round->0
4. Initialize array d as list of digits of the UTD id.
5. Generate k random numbers and store in Rand_arry
6. Start counter for c -> 0 to 1023 .
7. binaryc -> binary value of counter.
8. Generate the link states of the topology represented by the binary value of c.
9. Generate the link reliability Pi-> p ^di/3.
10. Assign state of link i as down if ith digit of binary c is 0 else assign up.
11. Generate adjacency list of each nodes based on the current topology.
12. Run Breadth First Search algorithm on the adjacency list of the topology.
13. If c in Rand_arry then flip isconnected state.
14. Calculate **Ri ->∏ pi.** based on flipped state condition.
15. Calculate overall Network Reliability: **Sum -> sum+Ri.**
16. if c<1023 increment c goto step 5.
17. Print overall Reliability value = Sum
18. if round <5 then round->round+1 goto step 4
19. if k<21 then k->k+1 goto step 3.
20. End.

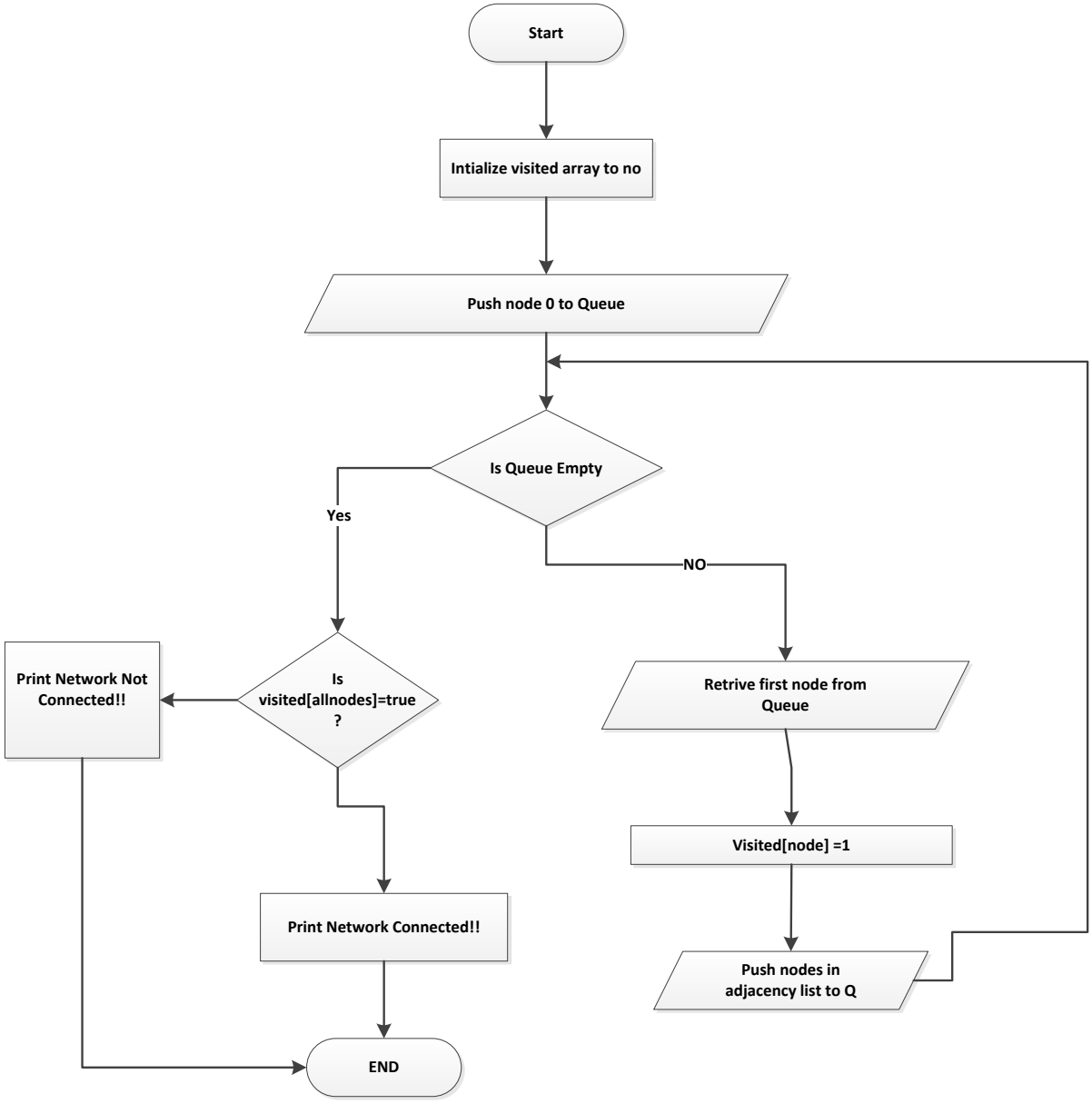
Flow Chart

1) Simulation



Generates the all possible network topologies and calculates the Network reliability.

2) Breadth First Search



Checks if the current Network Topology is connected or not using Graph adjacency list and Link state information.

Running the Program

1. Copy the NW_Reliability.java, Links.java program to the location to run.
2. To Compile Javac NW_Reliability.java Links.java.
3. To run the program Java NW_Reliability.

Program

NW_Reliability.java

```
package nw_reliability;


import java.util.ArrayList;
import java.util.Random;


public class NW_Reliability
{
  static double reliability=0.0;
  double p=0.05;
  String NW_topo;
  Links links_desc[];
  ArrayList<Integer> []nodes=new ArrayList[5] ;                    //adjacency list for nodes
  static ArrayList<Integer> rand_nos;
  public NW_Reliability()
  {
     links_desc=new Links[10]; //Generating the links
     for(int i=0;i<5;i++)
         nodes[i]=new ArrayList();
  }
  private boolean isconnected() //bfs
  {
```

```
    ArrayList<Integer> q = new ArrayList();
    q.add(0);
    int visited[]=new int[5], sum=0;
    boolean flag=true;
    while(!q.isEmpty())
    {
       int curr=q.remove(0);
       for(int i=0;i<nodes[curr].size();i++)
       {
          int k=nodes[curr].get(i);
          if(visited[k]!=1)
          {
              visited[k]=1;
              sum+=1;
              q.add(nodes[curr].get(i));
          }
       }
    }
    if (sum!=5)
       flag=false;
    return flag;
}
public double relib_clac(int flag)
{
    int index=0;
    int d[]={2,0,2,1,2,7,7,2,5,6};
    double p_l,x,local_reli=1;
     while(index<10)
       {
         if(flag==0)
```

```
        {
           p_l=Math.pow(p, Math.ceil((double)d[index]/3.0));
        }
        else
           p_l=p;
        x=1-p_l;
        if(links_desc[index].state==1)
            local_reli*=p_l;
        else
           local_reli*=x;
        index++;
       }
     return local_reli;

  }
  public void calc_reliability()
  {
     if(isconnected())
     {
        reliability+=relib_clac(0);
     }
     if(NW_topo.equals("1111111111"))
     {
        System.out.println("P="+p+" Network Reliability of "+NW_topo+" :"+reliability);
        reliability=0.0;                    //Resetting the reliability value to zero for next p value calculation
     }
  }
  public void calc_reliability(int flg)
  {
```

```
    if(rand_nos.contains(flg))
    {
        if(!isconnected())
        {
           reliability+=relib_clac(1);
        }
    }
    else
    {
       if(isconnected())
        {
           reliability+=relib_clac(1);
        }
    }
    if(NW_topo.equals("1111111111"))
    {
       System.out.println("P="+p+" Network Reliability of "+NW_topo+" :"+reliability);
       reliability=0.0;     //Resetting the reliability value to zero for next p value calculation
    }
}
public void random_gen(int k) //generating random numbers
{
   rand_nos=new ArrayList ();
   Random r=new Random();
   for(int i=0;i<k;i++)
   {
      int next=r.nextInt(1024);
     rand_nos.add(next);
   }
}
```

```
public void links_details(String topo)
{
  int n1=0,link_count=0;
  int n2=n1+1;
  NW_topo=topo;                          //taking backup of the network topography
  while(link_count<10)
  {
   int st=0;
   if(topo.charAt(link_count)=='1')     //generating the adjacency list
   {
      st=1;
      nodes[n1].add(n2);
      nodes[n2].add(n1);
   }
   links_desc[link_count]=new Links();
   links_desc[link_count].set_links_state(n1,n2,st);
   link_count++;
   n2+=1;
   if(n2==5)
   {
      n1+=1;
      n2=n1+1;
   }
  }
}
public static void main(String[] args)
{
   int topology=1000;
    try
     {
```

```
 for(double k=0.05;k<1.05;k+=0.05)
 {
  for(int i=0;i<1024;i++)
  {
      NW_Reliability n = new NW_Reliability();
      n.p=k;
      String str=Integer.toBinaryString(i);
      n.links_details((("0000000000")+str).substring(str.length()));
      n.calc_reliability();
   }
 }
for(int k=0;k<21;k++)
{
   for(int round=0;round<5;round++)
   {
        System.out.println("K="+k+" Round ="+round);
        NW_Reliability n = new NW_Reliability();
        n.random_gen(k);
        for(int i=0;i<1024;i++)
        {
            NW_Reliability n1 = new NW_Reliability();
            n1.p=0.9;
            String str=Integer.toBinaryString(i);
            n1.links_details((("0000000000")+str).substring(str.length()));
            n1.calc_reliability(i);
        }
   }
 }
}
catch(Exception e){System.out.println(e);}
```

```
  }
}
```

Links.java

```
package nw_reliability;
public class Links
{
   int n1;
   int n2;
   int state;
   double reli_fact;
   void set_links_state(int s,int d,int stat)
   {
      n1=s;
      n2=d;
      state=stat;
   }
}
```

Analysis

P	Network Reliability
0.05	5.35E-04
0.1	0.00451354
0.15	0.015846269
0.2	0.038527119
0.25	0.076061589
0.3	0.130837447
0.35	0.203544955
0.4	0.292776306
0.45	0.394924185
0.5	0.504455566
0.55	0.614565409
0.6	0.718129657
0.65	0.808796879
0.7	0.882004111
0.75	0.935693124
0.8	0.970548515
0.85	0.989674919
0.9	0.997755501
0.95	0.999846178
1	1

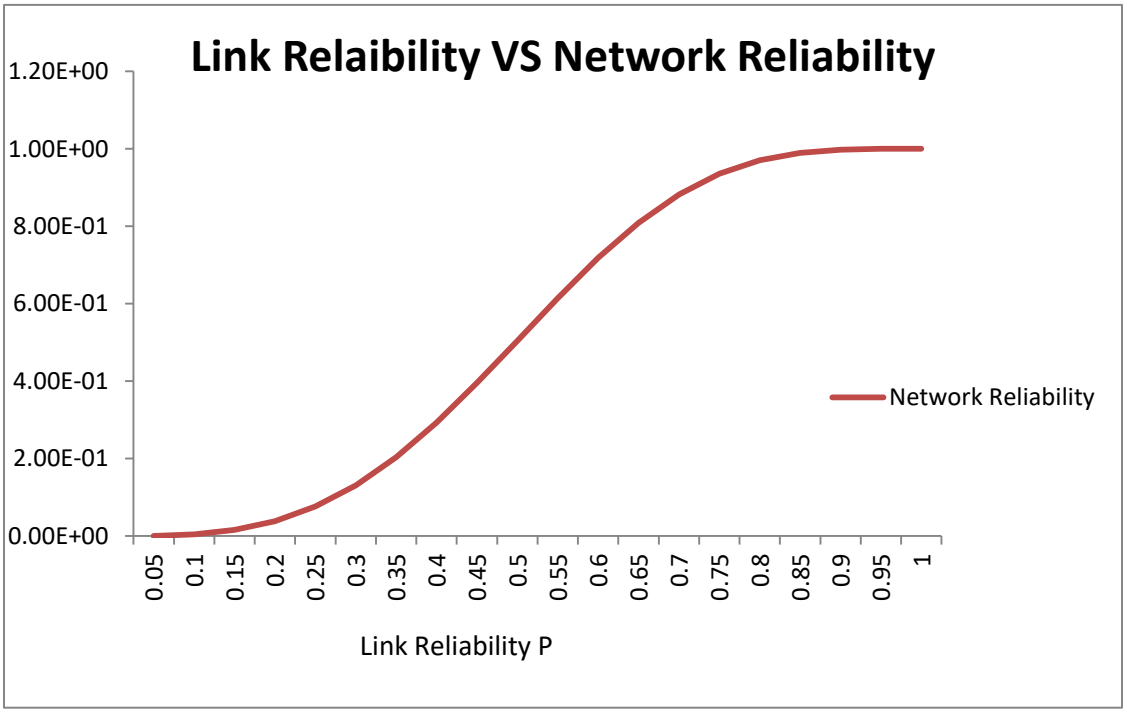
- In the observations, we can see that the reliability of the network increases as the reliability of the each of the links increases.
- The value of reliability tends to 1 as the reliability value for each of the links is closer to 1.
- The highest value of reliability is always for a fully connected network as there are multiple paths to react each of the nodes.

K	Round 1	Round 2	Round 3	Round 4	Round 5	Average
0	0.999492	0.999492	0.999492	0.999492	0.999492	0.999492
1	0.999486	0.999486	0.999014	0.999486	0.999498	0.999394
2	0.995187	0.999013	0.999486	0.990883	0.999386	0.996791
3	0.999433	0.994709	0.990877	0.998955	0.999013	0.996597
4	0.994703	0.990883	0.999386	0.995187	0.99902	0.995836
5	0.995171	0.999013	0.999433	0.998908	0.999386	0.998382
6	0.995169	0.999386	0.998959	0.995081	0.994178	0.996555
7	0.999393	0.998891	0.998424	0.994703	0.999428	0.998168
8	0.999368	0.998376	0.999374	0.999367	0.994709	0.998239
9	0.999332	0.998471	0.998906	0.998848	0.998849	0.998881
10	0.960152	0.994544	0.999274	0.998529	0.995112	0.989522
11	0.994118	0.994915	0.999483	0.999428	0.990293	0.995648
12	0.997509	0.990747	0.994106	0.994657	0.994581	0.99432
13	0.998401	0.998784	0.959835	0.998689	0.997401	0.990622
14	0.998901	0.995012	0.994956	0.994632	0.994164	0.995533
15	0.994526	0.994059	0.986295	0.993587	0.992636	0.99222
16	0.998257	0.990287	0.997499	0.98477	0.955742	0.985311
17	0.994	0.955412	0.985397	0.998342	0.959681	0.978566
18	0.994479	0.996982	0.951526	0.916574	0.960048	0.963922
19	0.989266	0.993535	0.998825	0.992565	0.996935	0.994225
20	0.99496	0.993467	0.959935	0.993936	0.994402	0.98734

- The value of Network reliability falls between the range of 0.99 to 0.96.
- As the k value increases the variation from the normal (1) also increases.
- The increase or decrease of reliability values are completely random to the k value.

Conclusions

1) Relation between link reliability and Network Reliability.

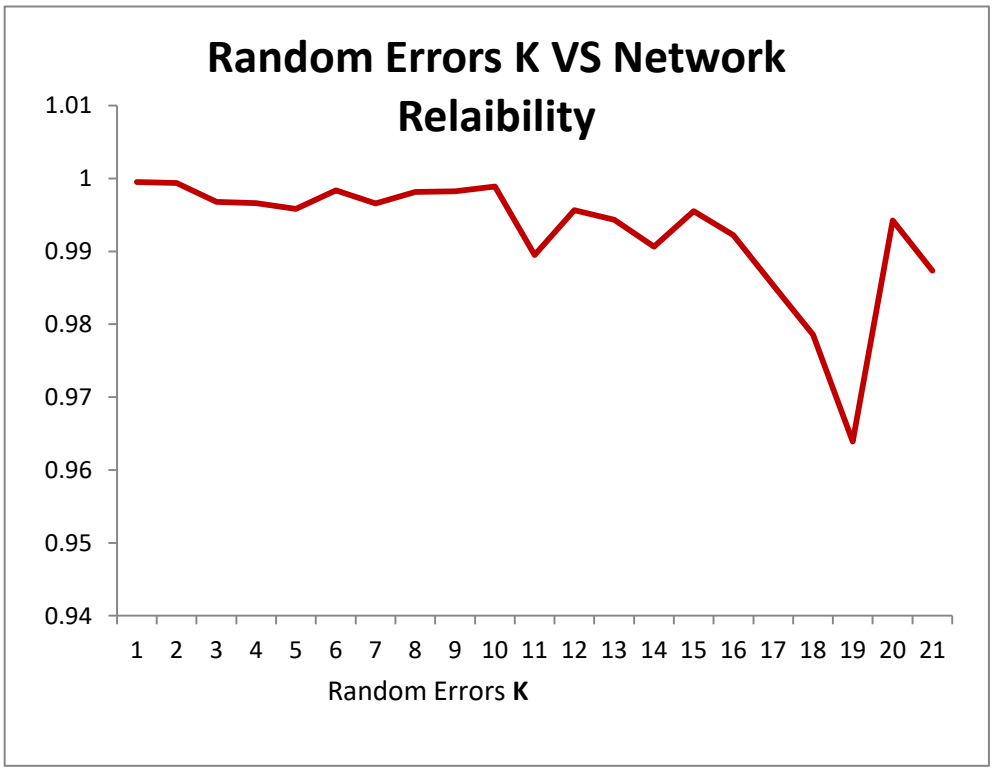


The graph obtained from varying values of link reliability p shows that the overall Network reliability depends on the individual link reliability. The reliability of the network value increases in a nonlinear fashion with the increase in individual link reliabilities. The effect of link reliabilities can be described in terms of three intervals. The first interval is between 0.05 and 0.4. In this interval, an increase in link reliability shows a small increase in the overall Network reliability. In the interval between 0.4 and 0.8 the network reliability shows steady steep increase. This levels out to 1 in the interval between 0.8 to 1. After this the Network reliability remains constant. We can infer from this data that in order to ensure high reliability network, the link reliability must be at least 0.7-1 range.

2) Network Reliability based on error randomness K

K	Average
0	0.999492
1	0.999394
2	0.996791
3	0.996597

4	0.995836
5	0.998382
6	0.996555
7	0.998168
8	0.998239
9	0.998881
10	0.989522
11	0.995648
12	0.99432
13	0.990622
14	0.995533
15	0.99222
16	0.985311
17	0.978566
18	0.963922
19	0.994225
20	0.98734



The range of the Network reliability value fluctuates between 0.99 and 0.96 as the value of k increases. To generate the graph , we kept the value of p as constant 0.9 and calculated the value of Network reliability. For every value of k , we run the experiment 5 times and average out the result of network

reliability to reduce the effect of randomness . In the graph k value and Network reliability values show no relationships between each other. The changes in the Network reliability randomly increases or decreases with respect to the increase in k value. The only constant factor is the rate interval which is between 0.99 to 0.96. To summarize the random error introduces random changes in the network reliability values. There are no relationship between k and the NW reliability.

Output

```
run:
P=0.05 Network Reliability  :5.349165453940988E-4
P=0.1 Network Reliability  :0.004513540366306006
P=0.15000000000000002 Network Reliability  :0.015846269178084408
P=0.2 Network Reliability  :0.03852711955660804
P=0.25 Network Reliability  :0.07606158964335918
P=0.3 Network Reliability  :0.13083744715054185
P=0.35 Network Reliability  :0.20354495591377345
P=0.39999999999999997 Network Reliability  :0.29277630654054404
P=0.44999999999999996 Network Reliability  :0.3949241851041293
P=0.49999999999999994 Network Reliability  :0.5044555664062499
P=0.5499999999999999 Network Reliability  :0.6145654096911449
P=0.6 Network Reliability  :0.7181296571842553
P=0.65 Network Reliability  :0.8087968794170005
P=0.7000000000000001 Network Reliability  :0.8820041119791576
P=0.7500000000000001 Network Reliability  :0.9356931243091822
P=0.8000000000000002 Network Reliability  :0.9705485154385916
P=0.8500000000000002 Network Reliability  :0.9896749194065627
P=0.9000000000000002 Network Reliability  :0.9977555010749938
P=0.9500000000000003 Network Reliability  :0.9998461782591302
P=1.0000000000000002 Network Reliability  :1.0
```

```
K=0
        0.9994922424000018      0.9994922424000018      0.9994922424000018      0.9994922424000018      0.9994922424000018
K=1
        0.9994863375000018      0.9994863375000018      0.9607501935000019      0.9994922505000018      0.9994863375000018
K=2
        0.9994922424000018      0.9994384422000018      0.9994331934000018      0.9989608014000018      0.9994331934000018
K=3
        0.9951823215000019      0.9990073845000018      0.9951337701000018      0.9990199233000019      0.9994909383000018
K=4
        0.9951758334000018      0.9994858272000018      0.9994213836000018      0.9606971304000014      0.9994325454000018
K=5
        0.9990088425000018      0.9994732155000018      0.9994686309000018      0.9990671625000018      0.9989550423000018
K=6
        0.9993872664000019      0.9989490726000018      0.9907766100000018      0.9990152658000018      0.9985403142000018
K=7
        0.9980567685000018      0.9984176235000017      0.9951213771000018      0.9941188563000017      0.9989484093000018
K=8
        0.9989965224000018      0.9946378314000017      0.9988433676000018      0.9989477532000018      0.9988840386000019
K=9
        0.9602542549000018      0.9994260573000018      0.9951633837000018      0.9945865827000018      0.9907588953000018
K=10
        0.9941181354000017      0.9951345720000019      0.9993670812000017      0.9936464724000018      0.9983684160000018
K=11
        0.9903332331000019      0.9897550623000018      0.9946857267000018      0.9979406469000017      0.9993270591000019
K=12
        0.9979931268000017      0.9979287561000018      0.9941117202000018      0.9950210748000018      0.9936978588000018
K=13
        0.9989385669000018      0.9974852415000018      0.9978815169000017      0.9982936215000018      0.9988553961000017
K=14
        0.9936930474000017      0.9993245004000019      0.9983527425000017      0.9940126410000018      0.9984636234000018
K=15
        0.9944849691000017      0.9940565997000018      0.9987824151000018      0.9950598495000018      0.9945986193000018
K=16
        0.9939896127000017      0.9986775120000018      0.9982733544000018      0.9935743662000017      0.9987307290000018

K=17
        0.9854073063000017      0.9986762079000018      0.9977785902000017      0.9977844303000017      0.9903901761000018
K=18
        0.9945330093000018      0.9940362688000017      0.9553539906000017      0.9982997442000018      0.9164386584000014
K=19
        0.9556721415000019      0.9944786187000018      0.9945257193000018      0.9896875659000017      0.9936210303000018
K=20
        0.9935854632000019      0.9982452240000017      0.9562043034000018      0.9978861924000018      0.9945053802000018BU
```

Reference

1. Class notes
2. Breath first Search Algorithim https://en.wikipedia.org/wiki/Breadth-first_search
3. Graph Connectivity : https://en.wikipedia.org/wiki/Connectivity_(graph_theory)
4. https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs
5. https://www.computer.org/csdl/proceedings/afips/1972/5080/00/50800049.pdf