

CS 6385.0W1 – Summer 2017
Algorithmic Aspects of Telecommunication Network

Project:3
Effect of the Individual Link Reliabilities on the Network
Reliability

by

Bhakti Khatri [Net ID: brk160030]

➤ Project Description:

The theme of this project is to study experimentally and test how the network reliability depends on the individual link reliabilities. In order to compute the network reliability as a numerical measure, the method of exhaustive enumeration is used.

➤ Network Description:

➤ Network topology:

A complete undirected graph on $n = 5$ nodes. This means, every node is connected with every other one (parallel edges and self-loops are excluded in this graph). As a result, this graph has $m = 10$ edges, representing the links of the network.

➤ Components that may fail:

The links of the network may fail, the nodes are always up. The reliability of each link is p , the same for every link. The parameter p will take different values in the experiments.

➤ Reliability configuration:

The system is considered operational, if the network topology is connected.

➤ Inputs and Outputs:

Input is a complete undirected graph with 5 vertices which is auto generated by the program. There will be 10 edges (number of edges = $(n * (n - 1)) / 2$; $n=5$) in the graph.

Output is always a reliability value of the entire network calculated using exhaustive enumeration.

➤ Scenarios:

1. The first scenario is to run the program for different values of ' p '. The parameter ' p ' runs over the $[0.05, 1]$ interval in steps of 0.05. Show graphically in a diagram how the obtained network reliability values depend on p .

2. The second scenario is to fix the parameter 'p' at 0.9 and do the following experiment:

Among the $2^{10} = 1024$ possible combinations of component states pick k of the combinations randomly, and flip the corresponding system condition. That is, if the system was up, change it to down, if it was down, change it to up. This models the situation that there are some random errors in the computation. Show in a diagram, how the reliability of the system changes due to this alteration. With this we want to investigate how the result depends on random errors. Specifically, show in a diagram how the change depends on k, in the range $k = 0, 1, 2, 3, \dots, 20$. During this experiment keep the value of the parameter p fixed at $p = 0.9$. To reduce the effect of randomness, run several experiments for each value of k, and average them out.

➤ Assumptions:

Network topology is a complete undirected graph. We are assuming that the nodes are always up (operational), but links may fail. Hence, some links or edges may disappear but remaining links will keep the network either connected or disconnected. Hence, the network reliability only depends on individual link reliability.

➤ Goal:

Given a complete undirected graph as input with N nodes design an algorithm that computes the network reliability using Exhaustive Enumeration Method.

Thus, we have to design an algorithm to evaluate the reliability of a given network configuration. Consider the probability of up and down states of the network links to determine the network probability. If there exists an edge between given two nodes, then that link is said to be UP otherwise DOWN. The process starts by assuming a link reliability for all the links in the network and then consider all the possible combinations in which the given network can be connected with the same number of links. For all these combinations, we calculate the overall reliability of the network. This kind of approach is called exhaustive enumeration and is useful for small network topologies without any particular connectivity pattern like series or parallel. We also experiment by flipping the states of some links in the network

from up to down. Keeping the link reliabilities same, we study the reliability of the network under these network changes.

➤ Pseudocode:

N: number of nodes

p: probability that link is up

r: reliability of a network state

R: Reliability of a whole network

INPUT: Network topology: an undirected complete graph G with n nodes,

$$[\text{total edges} = (n * (n - 1)) / 2]$$

Link probability p : probability that link is up

OUTPUT: Reliability of a network, R

Generate all the states of the graph

Set reliability = 1

For each state that belong to all states (every combination)

do if G is connected

then find the reliability r for each state

Calculate and return R by summing up all r (by adding the reliabilities of each network state)

➤ Algorithm:

(1)Generate all possible 1024 combinations with the help of 10-digit binary number.

(2)Create an adjacency graph matrix as graphMatrix and map that matrix to link by mapMatrixLink.

The graph matrix contains data that if there is an edge between any nodes and mapToLink matrix contains the edge information.

Develop depth first search to find if the connected components contains all the nodes in the network to check the network is connected.

(3) For each combination, calculate the reliability.
 Set reliability 1
 For each edge,
 if edge is up
 $\text{reliability} = \text{reliability} * p$ if
 edge is down
 $\text{reliability} = \text{reliability} * (1 - p)$

where,

p : probability that link is up $(1 - p)$:
 probability that link is down

(4) Calculate the total network reliability by summing up the reliability for all 1024 possible network combinations.

(5) Repeat the steps 2 to 5 for p values from $[0.05, 1]$ in the steps of 0.05.

Flip Experiment:

(1) Generate all possible 1024 combinations with the help of 10-digit binary number.

(2) From the 1024 combinations obtained from step 1, pick up k combinations randomly and flip their system states to change the state of the network topology

Set reliability 1

For each edge,

 if edge is up
 $\text{reliability} = \text{reliability} * p$ if
 edge is down
 $\text{reliability} = \text{reliability} * (1 - p)$

where,

p : probability that link is up $(1 - p)$:
 probability that link is down

(3) For each connected combination, calculate the reliability

(4) Calculate the total network reliability by summing up the reliability for all 1024 possible network combinations.

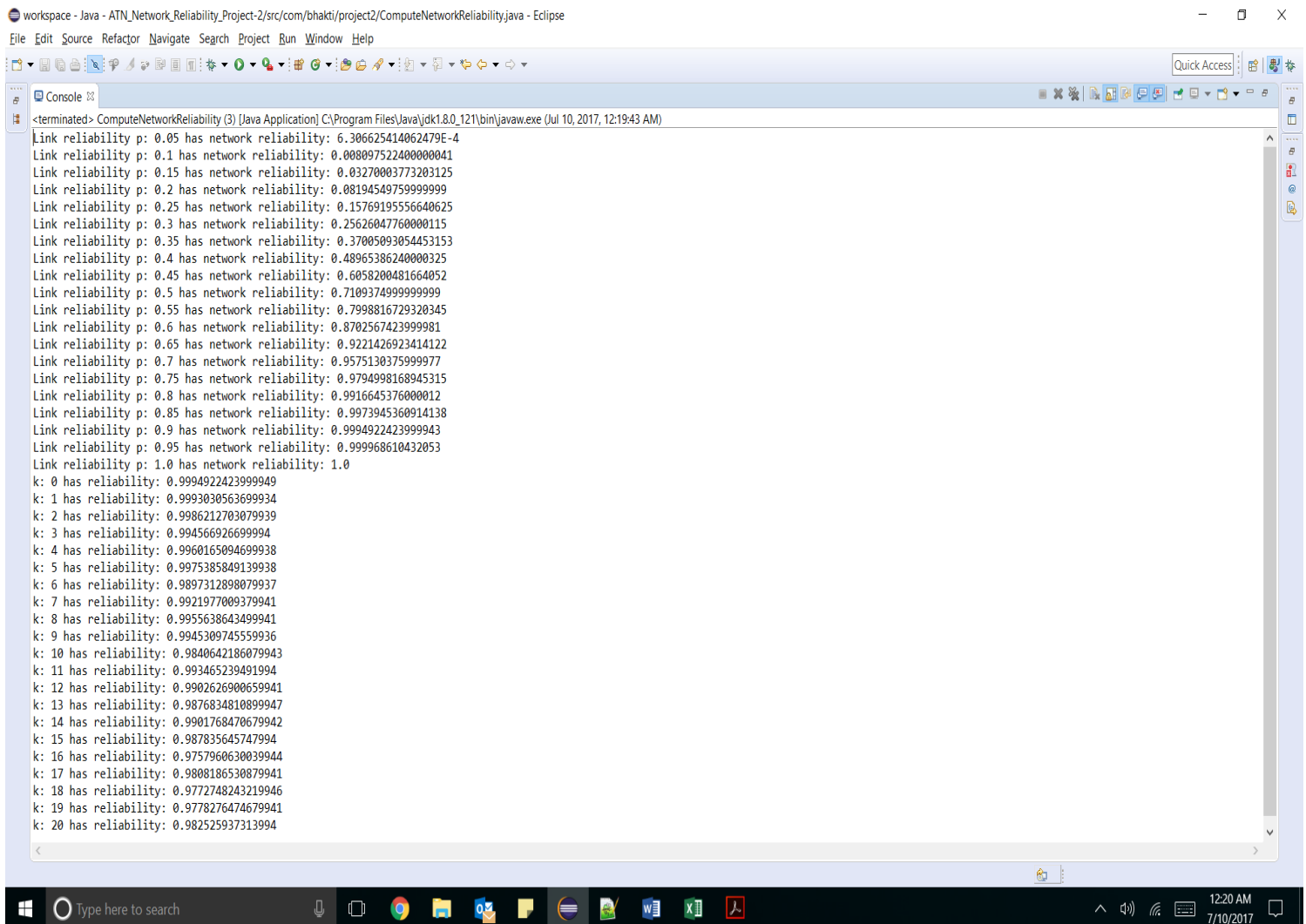
(5) Repeat the steps 2 to 5 for $p = 0.9$ and k values from $[0, 20]$ in the steps of 1.

➤ Program Details:

It consists of a Java file `ComputeNetworkReliability.java` which calls the function **`computePKReliability()`** that returns reliability for a given value of p and k for all the network states. In the regular experiment, we are incrementing the values of p in the steps of 0.05. In flip experiment, we are fixing the value of p to 0.9 and selecting 20 random states out of 1024 states and changing their system state.

➤ ReadMe:

❖ Run `ComputeNetworkReliability.java` which gives the following output in console window:



```
<terminated> ComputeNetworkReliability (3) [Java Application] C:\Program Files\Java\jdk1.8.0_121\bin\javaw.exe (Jul 10, 2017, 12:19:43 AM)
Link reliability p: 0.05 has network reliability: 6.306625414062479E-4
Link reliability p: 0.1 has network reliability: 0.008097522400000041
Link reliability p: 0.15 has network reliability: 0.03270003773203125
Link reliability p: 0.2 has network reliability: 0.08194549759999999
Link reliability p: 0.25 has network reliability: 0.15769195556640625
Link reliability p: 0.3 has network reliability: 0.25626047760000115
Link reliability p: 0.35 has network reliability: 0.37005093054453153
Link reliability p: 0.4 has network reliability: 0.48965386240000325
Link reliability p: 0.45 has network reliability: 0.6058200481664052
Link reliability p: 0.5 has network reliability: 0.71093749999999999
Link reliability p: 0.55 has network reliability: 0.7998816729320345
Link reliability p: 0.6 has network reliability: 0.8702567423999981
Link reliability p: 0.65 has network reliability: 0.9221426923414122
Link reliability p: 0.7 has network reliability: 0.9575130375999977
Link reliability p: 0.75 has network reliability: 0.9794998168945315
Link reliability p: 0.8 has network reliability: 0.9916645376000012
Link reliability p: 0.85 has network reliability: 0.9973945360914138
Link reliability p: 0.9 has network reliability: 0.9994922423999943
Link reliability p: 0.95 has network reliability: 0.999968610432053
Link reliability p: 1.0 has network reliability: 1.0
k: 0 has reliability: 0.9994922423999949
k: 1 has reliability: 0.9993030563699934
k: 2 has reliability: 0.9986212703079939
k: 3 has reliability: 0.994566926699994
k: 4 has reliability: 0.9960165094699938
k: 5 has reliability: 0.9975385849139938
k: 6 has reliability: 0.9897312898079937
k: 7 has reliability: 0.9921977009379941
k: 8 has reliability: 0.9955638643499941
k: 9 has reliability: 0.9945309745559936
k: 10 has reliability: 0.9840642186079943
k: 11 has reliability: 0.993465239491994
k: 12 has reliability: 0.9902626900659941
k: 13 has reliability: 0.9876834810899947
k: 14 has reliability: 0.9901768470679942
k: 15 has reliability: 0.987835645747994
k: 16 has reliability: 0.9757960630039944
k: 17 has reliability: 0.9808186530879941
k: 18 has reliability: 0.9772748243219946
k: 19 has reliability: 0.9778276474679941
k: 20 has reliability: 0.982525937313994
```

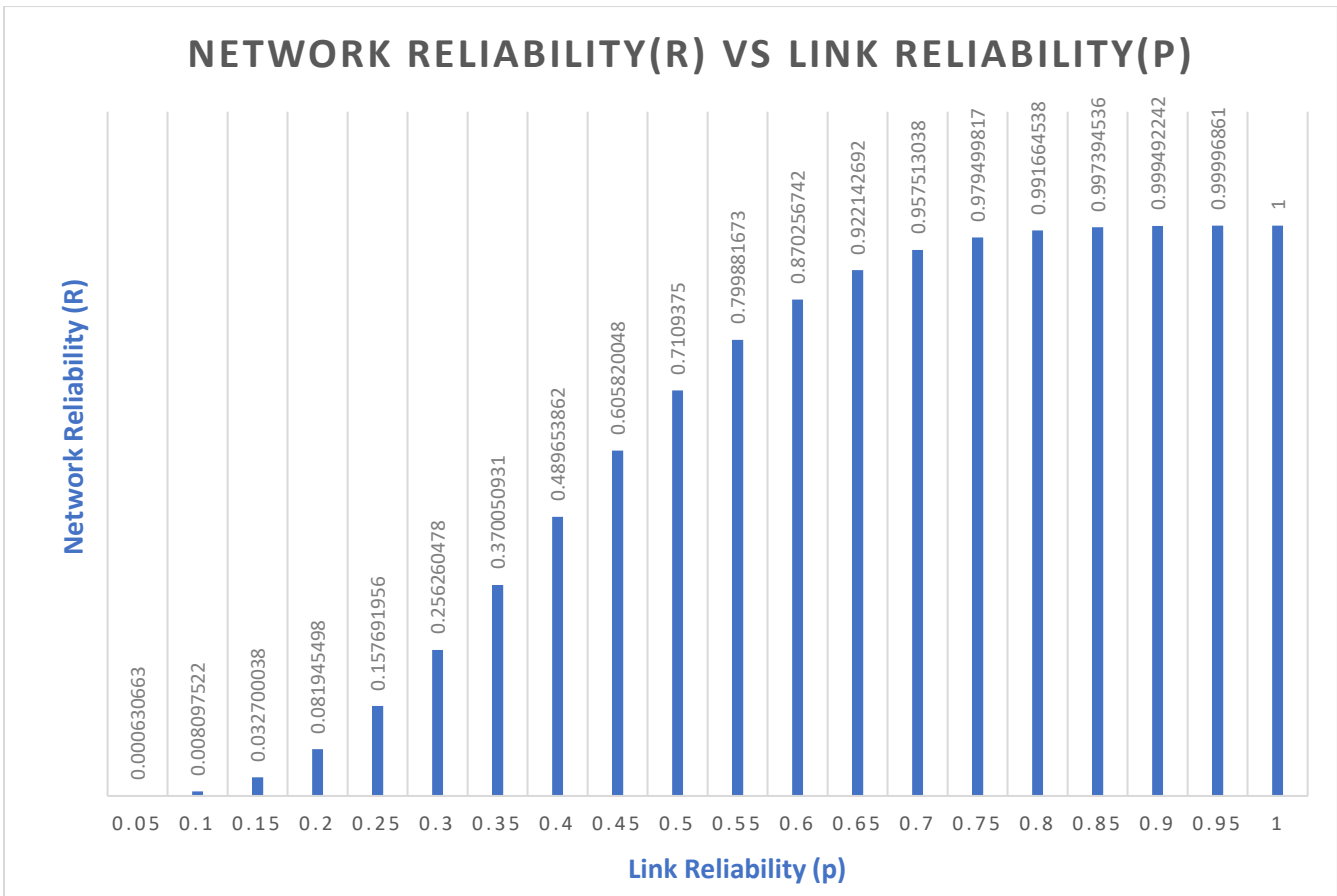
➤ Output values for p and k:

Sr No.	LINK RELIABILITY (p)	NETWORK RELIABILITY (R)
1	0.05	6.306625414062479E-4
2	0.1	0.008097522400000041
3	0.15	0.03270003773203125
4	0.2	0.08194549759999999
5	0.25	0.15769195556640625
6	0.3	0.25626047760000115
7	0.35	0.37005093054453153
8	0.4	0.48965386240000325
9	0.45	0.6058200481664052
10	0.5	0.7109374999999999
11	0.55	0.7998816729320345
12	0.6	0.8702567423999981
13	0.65	0.9221426923414122
14	0.7	0.9575130375999977
15	0.75	0.9794998168945315
16	0.8	0.9916645376000012
17	0.85	0.9973945360914138
18	0.9	0.9994922423999943
19	0.95	0.999968610432053
20	1.0	1.0

k	R (Round-1)	R (Round-2)	R (Round-3)	AVERAGE NETWORK RELIABILITY
0	0.9994922423999949	0.9994922423999949	0.9994922423999949	0.999492242
1	0.9989601018119934	0.998739660869993	0.9988494198479932	0.998849728
2	0.998278396227994	0.9983886462539937	0.9943160763239941	0.996994373
3	0.9947732913179941	0.9974960055459939	0.9974153863079933	0.996561561
4	0.9980859122839935	0.9935523226439941	0.990566761787994	0.994068332
5	0.9968143922459943	0.9975017893499942	0.9978031802159939	0.997373121
6	0.9966367014839943	0.9930846656619937	0.9968798730519939	0.995533747
7	0.9908777954179945	0.9969620711779937	0.9966996165339941	0.994846494
8	0.9938677663979938	0.9915224816619941	0.9949047471479936	0.993431665
9	0.9804792386079941	0.9936070994519942	0.9945843628239938	0.9895569
10	0.9913610604519938	0.987898012227994	0.991951618967994	0.990403564
11	0.9902215092159942	0.9889726644379938	0.9927595398819942	0.990651238
12	0.9926620847919944	0.9777516143759948	0.9903271269519941	0.986913609
13	0.9907258605659939	0.9865141050059938	0.991859883769994	0.98969995
14	0.9853824604599941	0.9885411775179941	0.9913060885239937	0.988409909
15	0.9833853206079946	0.9785752758539941	0.9833181713639942	0.981759589
16	0.9828863380899943	0.9821401419259941	0.9865646366679937	0.983863706
17	0.9848043508239941	0.9815374808219943	0.988762886607994	0.985034906
18	0.9840058431959937	0.9871559163379939	0.9897124372499942	0.986958066
19	0.9842839489359946	0.9806936337099943	0.9857157162319942	0.983564433
20	0.9784185291019942	0.976015735003994	0.9819994082059946	0.978811224

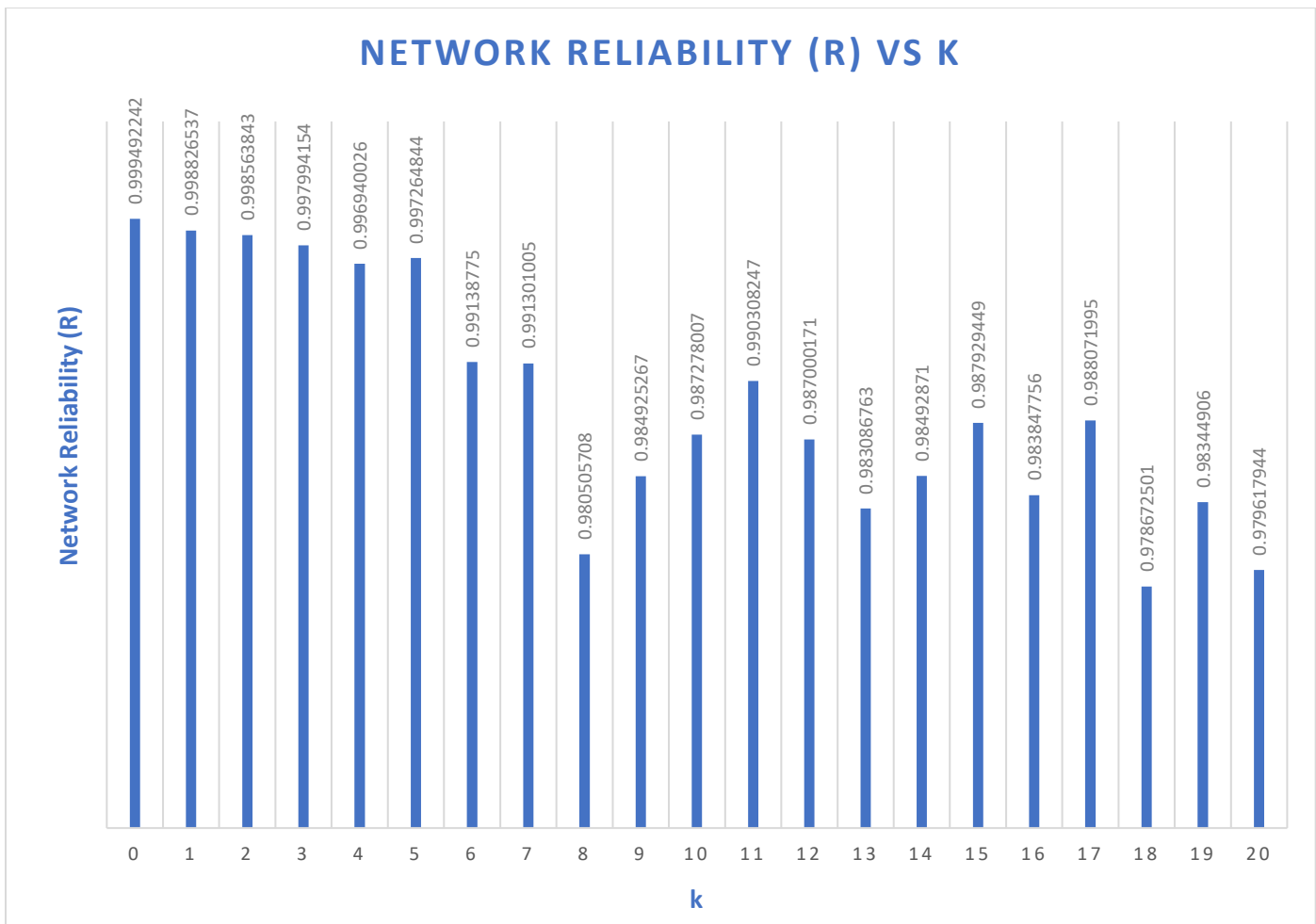
➤ Observation:

- ❖ The chart below shows the relationship between the Link Reliability (p) and the Network Reliability (R).



From the above chart, we can observe that there is strong relationship between link reliability (p) and network reliability (R). For smaller values of p , the network reliability is incrementing slowly and after a certain point it has a steady increase. We also observe that for higher values of p from 0.9, the reliability becomes almost constant. Thus, we can infer that higher the probability of up individual links or operational links, better is the reliability of a network. In order to achieve high reliable network, we should make sure that individual links are configured such that their link probability is at least 0.9.

- ❖ The chart below shows the relationship between the k and the Network Reliability (R) when $p = 0.9$.



By observing the above graph, we can infer that for fixed $p = 0.9$, the Network Reliability (R) increases for some values of k and decreases for some values of k as k varies from 0 to 20. Thus, the values change arbitrarily. Hence the relationship between k and the Network Reliability is random.

➤ Conclusion:

Thus, in this project we conducted an experimental study to test the dependence of network reliability on the individual link reliabilities. From the studies performed, we have concluded that:

- ✓ Higher the reliability of individual links, higher the network reliability.
- ✓ Flipping more number of system states will decrease the network reliability.

➤ References:

- (1) Professor Andras Farago's Lecture Notes
- (2) www.wikipedia.com

➤ Appendix:

[Source Code:]

ComputeNetworkReliability.java

```
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Random;

public class ComputeNetworkReliability
{
    public static double linkrel;
    public static int graphMatrix[][];
    public static int mapMatrixLink[][];
    public static double networkReliability;

    public static void main(String[] args)
    {
        /*p- reliability of every link; k- number of combinations*/
        double p;
        int k=0;
        DecimalFormat df = new DecimalFormat("#.##");

        for(p=0.05;p<1.05;p+= 0.05)
        {
            //System.out.println("Link reliability p:
"+Double.valueOf(df.format(p))+ " has network reliability: "+computePKReliability(p,k));
        }

        for(k=0;k<21;k++)
        {
            double reliability = 0;
            for(int j=0;j<100;j++)
            {
                reliability = reliability +computePKReliability(0.9, k);
            }
            reliability = reliability/100;
            //System.out.println("k: "+k+" has reliability: "+reliability);
            System.out.println(reliability);
        }

        /* To compute reliability for value of p (reliability of every link) and k (number of
        combinations) */
        public static double computePKReliability(double p, int k)
        {
            ArrayList<Integer> randomStates= new ArrayList<Integer>();
            double reliability=0;

            for(int j = 0; j < k; j++)
            {
                Random rnum = new Random();
                int randomnum = rnum.nextInt(1024);

                while(randomStates.contains(randomnum))
                {
                    randomnum = rnum.nextInt(1024);
                }
            }
        }
    }
}
```

```

        randomStates.add(randomnum);
    }

    for(int i=0;i<=1023;i++)
    {
        buildMatrix();
        linkrel=p;
        String state = String.format("%10s",
Integer.toBinaryString(i)).replace(" ", "0");

        for(int s=0;s<10;s++)
        {
            if (state.charAt(s) == '1')
            {
                graphMatrix[mapMatrixLink[s][0]][mapMatrixLink[s][1]]= 0;
                graphMatrix[mapMatrixLink[s][1]][mapMatrixLink[s][0]]= 0;
            }
        }

        if(randomStates.contains(i))
        {
            if(!checkGraphConnection())
            {
                reliability = reliability +
computeGraphReliability(linkrel);
            }
        }
        else
        {
            if(checkGraphConnection())
            {
                reliability = reliability +
computeGraphReliability(linkrel);
            }
        }
    }
    return reliability;
}

//To build a graph matrix
public static void buildMatrix()
{
    linkrel = 0;
    int key = 0;
    networkReliability = 1;
    //given n=5 nodes for complete undirected graph
    graphMatrix= new int[5][5];
    /* no of edges- [n * (n-1)]/2 so initialize matrix with 10 edges and 2 nodes
*/

    mapMatrixLink = new int[10][2];

    for(int i=0;i<=4;i++)
    {
        for(int j=0;j<=4;j++)
        {
            if (i!=j)
            {
                if(i>j)
                {
                    mapMatrixLink[key][0]= i;
                    mapMatrixLink[key][1]= j;

```

```

        key++;
    }
    graphMatrix[i][j]=1;
}
else
{
    graphMatrix[i][j]=0;
}
}
}

//To compute graph reliability
public static double computeGraphReliability(double p)
{
    for(int i=0;i<=4;i++)
    {
        for(int j=0;j<=4;j++)
        {
            if (i>j)
            {
                if (graphMatrix[i][j]==1)
                {
                    networkReliability = networkReliability * p;
                }
                else
                {
                    networkReliability = networkReliability * (1 - p);
                }
            }
        }
    }
    return networkReliability;
}

//To compute DFS for all n=5 nodes
public static void depthFirstSearch(int i, boolean visitedNode[])
{
    visitedNode[i] = true;
    for(int n=0;n<=4;n++)
    {
        if(graphMatrix[i][n] == 1)
            if(!visitedNode[n])
                depthFirstSearch(n,visitedNode);
    }
}

//To check if graph is connected
public static boolean checkGraphConnection()
{
    boolean connectedFlag = true;
    boolean visitedNode[] = new boolean[5];

    for(int i=0;i<=4;i++)
        visitedNode[i] = false;

    depthFirstSearch(0,visitedNode);

    for(int i=0;i<=4;i++)
        if(!visitedNode[i])
            connectedFlag = false;
}

```

```
        }  
        return connectedFlag;  
    }  
}
```