CHAPTER III

Edge-loading

in

Modified De Bruijn Graphs

and its Performance Measures
3.1 INTRODUCTION

In the section (1.5) of the first chapter, we have described De Bruijn graphs which are presented as logical topologies for multihop lightwave networks [75], [76]. The two metrics by which we shall evaluate the performance of a logical topology are the average end-to-end packet delay and the network throughput. The throughput of the network is determined by the edge with the maximum loading, assuming all edges have the same capacity. As the edge loading is small, the throughput becomes larger. Thus, delay and throughput are depending mainly on the routing scheme used in the network. The proposed Modified De Bruijn graphs G*(Δ,d) in this thesis (in chapter 2) are logical topologies on multihop lightwave networks. Now we concentrate on developing algorithms for determining edge loading in the two multihop lightwave networks, G(Δ,d) and G*(Δ,d). Using these algorithms, the various parameters related to edge loading, namely, average edge loading, maximum edge loading, average network utilization, edge load distribution, are derived for different networks based on G(Δ,d) and G*(Δ,d). Also a comparison of these results of De Bruijn and Modified De Bruijn networks of different sizes are presented in this chapter.
In the following sections edge loading in De Bruijn and Modified De Bruijn graphs and their performance measures are presented.

3.2. Algorithm for Edge Loading in De Bruijn graphs (Networks)

In this section, we define some of the notations and definitions used in the algorithm. Then, the development of the algorithm for edge loading in De Bruijn graphs is given.

3.2.1. Notations and Definitions

Definition of Global variables used for describing the Network:

$\Delta$ -- degree of the network

d -- string length of each node label, which denotes the max. distance in Hops between any two nodes when static routing applied.

$N = \Delta^d$, which is the total number of Nodes in the Network.

Src, Dest --- be the any source and destination in the network and denoted as

$A = (a_1, a_2, a_d)$ and $B = (b_1, b_2, b_d)$ where $a(i's)$ and $b(i's)$ can take values '0', '1', '2', ..., 'A-1'.

Spath --- gives the path between source and destination as a sequence of node labels.

Edge $(M), M=1,...,\Delta N$ --- Edge is an array of $M$ edges in the network. An edge connection between two edges. Their label consists of the sequence of slen characters in the bottom node label plus the last character in the top edge label as follows:
Input : Δ,d
The possible edges in the network are as follows:
The total no of edges = ΔN
Eg. Δ=2, d=3
Then tns = 8
total Edges=M =2*8=16.
Since ns=2, there 2 symbols namely '0' and '1' are used for labelling the network nodes. Hence the nodes of the network are '000','001', '010','011','100','101','110','111'.

Edge (M).M=0.1,...,15: Edge is an array of 16(=2*8) edges in the network.
They form 16 counters corresponding to 16 edges used for determining the loading of each of the edges.

--- An Edge from two nodes say '000' to '001' is '0001'. So the 16 possible edges can be written as '0000', '0001', '0010', '0011', '0100', '0101', '0110', '0111', '1000', '1001', '1010', '1011', '1100', '1101', '1110', '1111'.

Let Edge (i) = 0, i=1 to 16 are the counters Edge loading of each initially.

3.2.2 Algorithm for Edge Loading in G(Δ,d) De Bruijn Networks

Procedure DeBruijnRoute (Src,Dest,Δ,d)
begin
for (i=1 to d)
begin
if (ShiftMatch(i,Src,Dest) = TRUE)
break;

Function ShiftMatch(i, Src, Dest)
begin
  flag=True;
  i=1;
  for(j=i to d-1)
  begin
    if (Src(j) NOT= Dest(j)) then
    begin
      flag=False;
      break;
    end;
    i=i+1;
  end;
  return flag;
end;

Procedure merge(Src, Dest, len, spath)
begin
  for(i=1 to d)
  spath(i)=Src(i);
  for(k=d-len+1 to d)
  begin
    i=i+1;
    spath(i)=Dest(k);
  end;
end:
Procedure EdgeLoad(spath, edge)
begin
pathlength = length(spath):
while (TRUE)
begin
if (pathLength = d) then
break:
    ei = 0;
for (j = 0; j <= d)
    ei = ei + spath[i-j] * Δ^j;
edge[ei] = edge[ei] + 1;
pathlength = pathlength - 1;
end;
end;

3.3. Algorithm for Edge Loading in Modified De Bruijn Graphs (Networks)

In this section, we first consider the definition of some variables and notations used in this algorithm. Next, the algorithm for the edge loading in Modified De Bruijn graphs is presented.

3.3.1. Notations and Definitions:

Δ--Degree of the network.

d--String length of each network node label, this length is the diameter, maximum distance in number of Hops between any two nodes.
N = $A^d$ is the total number of nodes in the Network.

Let Src be the Source node and Dest be the destination node. They are represented as $a_1a_2a_3...a_d$ i.e., $a_i$'s can take values 0, 1, 2, ..., $d$ - 1.

Edge(M), $M = 1, ..., A^d$ -- Edge is an array of M edges in the network. An edge connection between two edges. Their label consists of the sequence of d characters in the bottom node label plus the last character in the top edge label as follows:

- $S$ path -> List of nodes on the shortest path from Src to Dest
- IsSelf(Node) -> finds whether it is a Self node or not.

DistBothSelfload -> finds the shortest distance in number of hops between a Self-node(source) is one in which all the symbols in its address label are same.

PreSelfToDest -> refers to the preceding self node to the destination node.

AdjselfToSrc -> refers to adjacent node to the source node

DistPreselfToDest -> finds the preceding self node to a Nonself node (destination) and also returns the shortest distance between the PreselfToDest(self node) and destination.

DistOfAdjselfToSrc -> finds the adjacent self node to source(Nonself node) and returns the shortest distance between the source and AdjselfToSrc.

Shiftmatch(i, Src, Dest, $0 \leq i \leq slen$) -- An operation on the two strings

- Src and Dest to TRUE iff $(b_1, b_2, ..., b_d) = (a_{i+1}, a_{i+2}, ..., a_d)$
- and FALSE otherwise.
Merge(i.Src.Dest).0<=i<=d --> is a string(or sequence) of length slen+i given by = (a_1,a_2,...,a_d b_{d+1},...,b_d)

Edgeload(sp.edge)---> finds the edges on reading the shortest path, identifies the each edge of length (d+1) and increment the occurrence of each identified edge in the spath.

SelfLoad: Find the path followed and distance from a source node(self-node) to adjacent self-node, increments the counters of the corresponding edges on the path taken.

Transmit next node: refers to the adjacent node from source node to which the packet has to be transmitted.

Findnormallength: finds the distance between source node and the destination node using the De Bruijn routing.

Input :Δ,d

The possible edges in the network are as follows:
The total no of edges = ΔN

Eg. Δ=2,d=3

Then N = 8

total Edges=ΔN =2 x 8 =16.

Since Δ=2 ,there 2 symbols namely '0'and '1' are used for labelling the network nodes.Hence the nodes of the network are '000','001', '010','011','100','101','110','111'.

Edge(M),M=0,1,...,15;Edge is an array of 16(=2 x 8) edges in the network.

They form 16 counters corresponding to 16 edges used for determining the loading of each of the edges.

An Edge from two nodes say '000' to '001' is'0001'.So the 16 possible edges can be written as '0000', '0001', '0010', '0011', '0100', '0101', '0110', '0111', '1000', '1001', '1010', '1011', '1100', '1101', '1110', '1111'.

3.3.2. Algorithm for Edge Loading in Modified De Bruijn networks ($G^*(\Delta, d)$).

Procedure ModifiedDeBruijnEdgeLoad($Src, Dest, \Delta, d$)

begin
if (IsSelf($Src = FALSE$) AND IsSelf($Dest = FALSE$)) then
    Length = BothNonSelfLoad($Src, Dest$);
else if (IsSelf($Src = TRUE$) AND IsSelf($Dest = TRUE$)) then
    Length = BothSelfLoad($Src, Dest$);
else if (IsSelf($Src = TRUE$) AND IsSelf($Dest = FALSE$)) then
    Length = SourceSelfLoad($Src, Dest$);
else if (IsSelf($Src = FALSE$) AND IsSelf($Dest = TRUE$)) then
    Length = DestSelfLoad($Src, Dest$);

Transmit Next node:
return Length;
end;

Procedure BothSelfLoad($Src, Dest$)

begin
Length = FindNormalLength($Src, Dest$);
if (Length $\geq d$) then
begin
    count = DistBothSelf($Src, Dest$);
    if (Length $\geq count$) then
    begin
        selfLoad($Src, count$);
        return count;
    end;
edgeload(spath);
end;
Procedure BothNonSelfload(Source,Destination)
begin
    Length = FindNormalLength(Source, Destination)
    if (Length >= d) then
        begin
        Length1 = DistSrcToAdjself(Source, AdjselfToSource) +
                 DistPreSelfToSrc(PreselfToDestination, Destination) +
                 DistBothSelfAdjselfToSource, PreselfToDestination);
        if (Length > Length1) then
            begin
            DestSelfLoad(Source, AdjselfToSource);
            SourceSelfLoad(PreselfToDestination, Destination);
            SelfLoad(Adjself, DistBothself);
            return Length1;
            end;
        end;
    edgeLoad(spath);
    return Length;
end;

Procedure SourceSelfLoad(Source, Destination)
begin
    Length = FindNormalLength(Source, Destination);
    tsp = spath;
    if (Length >= d) then
        begin
        Length1 = DistBothSelfLoad(Source, PreselfToDestination) +
                  DistOfPreselfToDestination(PreselfToDestination, Destination);
        end;
end;
if(Length >= Length1) then
    
    begin
    BothNonselfLoad(PreselfToDest.Dest);
    SelfLoad(Src.PreselfToDest);
    return Length1;
    
    end;
end;

spath=tsp;
EdgeLoad(spath);
return Length;
end;

Procedure DestSelf(Src,Dest)
begin
    Length=FindNormalLength(Src,Dest);
    tsp = spath;
    if(Length>=d) 
    begin
    Length=DistAdjselfToSrc(Src,AdjselfToSrc) 
    + DistBothself(Adj selfToSrc, Dest);
    if(Length >= Length1) then
    begin
    BothNonself(Src,adjself);
    SelfLoad(adjselfToSrc,Dest);
    return length1;
    
    end;
    
    end;
end;

spath=tsp;
edgeLoad(spath,d+i):
return(Length):
end:

Procedure EdgeLoad(spath,edge)
begin
  pathlength=length(spath);
  while(TRUE)
  begin
    if(pathLength = d) then
      break;
    ei=0;
    for(j=0;j<=d)
      ei=ei+spath[i-j]*(ns^j);
    edge[ei]=edge[ei]+1;
    pathlength = pathlength -1;
  end; end.

3.4 Performance measures of Edge loading in \(G(\Delta,d)\) and \(G^*(\Delta,d)\)

In this section, we define the various performance measures which could be computed using the edge loading algorithms (presented in sections (3.2) and (3.3) of this chapter). we present also the comparison their results for De Bruijn and Modified De Bruijn graphs.

The definition and explanation of different performance measures of edge loading are as follows:
3.4.1 Average edge loading (\( \bar{L} \))

The loading on edge \( I \) is defined as the number of source-destination pairs that use that edge \( I \) for communication. The average edge load is defined as the edge load averaged over all edges in the network.

Let \( M \) be the number of edges in the network. Hence, the average edge load which is denoted by \( \bar{L} \) and is given as

\[
\bar{L} = \frac{1}{M} \sum_{i=1}^{M} L_i \quad \text{...(3.4.1)}
\]

For the De Bruijn graph, \( G(\Delta,d) \) there are

\[
N = \Delta^d \text{ nodes} \quad \text{...(3.4.2)}
\]

Hence, the number of edges in the De Bruijn graph = \( 2N \cdot \Delta \) edges \quad \text{...(3.4.3)}

From the eqns. (3.4.2) and (3.4.3) and using the definition for average edge loading in \( G(\Delta,d) \), we have

\[
\bar{L} = \frac{1}{(2N \cdot \Delta)} \sum_{i=1}^{(2N \cdot \Delta)} L_i \quad \text{...(3.4.4)}
\]

Now consider the average edge loading in Modified De Bruijn graph, \( G(\Delta,d) \). Here for \( G^*(\Delta,d) \) the number of edges is \( 2N \) (as per the
definition $G^*(\Delta,d)$ in section 2.2). On using eqn. (3.4.1) and (3.4.3) the average edge loading on $G^*(\Delta,d)$ can be written as

$$\bar{L} = \frac{1}{2N} \sum_{i=1}^{2N} L_i \quad \text{(3.4.5)}$$

A comparison of the average edge loading values for $G(\Delta,d)$ and $G^*(\Delta,d)$ using the edge loading algorithms presented in sections (3.2) and (3.3) respectively is shown in Table (3.4.1).

### 3.4.2 Maximum edge loading ($L_{\text{max}}$):

The maximum edge loading is defined as the load of the edges which has the maximum load, which is denoted as $L_{\text{max}}$. This parameter is computed for $G(\Delta,d)$ and $G^*(\Delta,d)$ using their edge loading algorithms (presented in sections (3.2) and (3.3)).

A comparison maximum edge load values for $G(\Delta,d)$ and $G^*(\Delta,d)$ is shown in table (3.4.1).

### 3.4.3 Edge load distribution:

The edge load distribution is defined as the distribution of loading on all the edges as $N(N-1)$ source-destination pairs that communicates. This is an important network parameter for determining the network throughput. For instance, comparison of the
edge loading distribution as shown in fig (3.4.1) and fig (3.4.2) for \( G(4,5) \) and \( G^*(4,5) \).

### 3.4.4 Average network utilization:

The utilization of an edge \( i \) defined as the proportion of the loading on edge \( i \) to an edge with maximum loading \( (L_{\text{max}}) \) and it can be written as

\[
U_i = \frac{L_i}{L_{\text{max}}} \quad \ldots (3.4.4)
\]

The average network utilization is defined as the edge utilization averaged over all the edges in the network which is denoted by \( U_{\text{avg}} \) and is given by

\[
U_{\text{avg}} = \frac{1}{M} \sum_{i=1}^{M} \frac{L_i}{L_{\text{max}}} \quad \ldots (3.4.5)
\]

Where \( M = \) total number of edges in the graph.

A comparison of the average edge utilization for De Bruijn and Modified De Bruijn networks of different networks is shown in Table (3.4.2).
Fig 3.4.1  Edge Loading in De Bruijn Graph G(4,5)
fig 3.4.2  Edge Loading in Modified De Bruijn Graph $G^*(4,5)$
TABLE 3.4.1: Comparison of Edge Loading Values for De Bruijn & Modified De Bruijn Graphs as a function of $\Delta$ and $d$ for $N$ nodes

<table>
<thead>
<tr>
<th>$\Delta$</th>
<th>$d$</th>
<th>$N$</th>
<th>De Bruijn</th>
<th>Modified De Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$L_{\text{max}}$</td>
<td>$L$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>8</td>
<td>11</td>
<td>8.428572</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>16</td>
<td>29</td>
<td>22.6666</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>32</td>
<td>81</td>
<td>58.387096</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>64</td>
<td>208</td>
<td>145.0331754</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>128</td>
<td>503</td>
<td>349.448822</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>256</td>
<td>1151</td>
<td>821.372559</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>512</td>
<td>2789</td>
<td>1892.273926</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1024</td>
<td>6309</td>
<td>4289.117188</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>27</td>
<td>31</td>
<td>22.307692</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>81</td>
<td>138</td>
<td>91.425003</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>243</td>
<td>535</td>
<td>351.867767</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>729</td>
<td>1945</td>
<td>1294.247192</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2187</td>
<td>6997</td>
<td>4606.407227</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>16</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>64</td>
<td>57</td>
<td>42.238094</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>256</td>
<td>313</td>
<td>230.305878</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1024</td>
<td>1589</td>
<td>1173.618774</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>25</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>125</td>
<td>86</td>
<td>68.19355</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>625</td>
<td>586</td>
<td>463.237183</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3125</td>
<td>3711</td>
<td>2937.477539</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>36</td>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>216</td>
<td>121</td>
<td>100.162788</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1296</td>
<td>985</td>
<td>814.193054</td>
</tr>
</tbody>
</table>

*Note: $L_{\text{max}}$ represents the maximum edge loading value, while $L$ represents the actual edge loading value.*
<table>
<thead>
<tr>
<th>A</th>
<th>d</th>
<th>N</th>
<th>De Bruijn</th>
<th>Modified De Bruijn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>U_{avg}</td>
<td>U_{avg}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>8</td>
<td>0.766234</td>
<td>0.859375</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>16</td>
<td>0.781609</td>
<td>0.788462</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>32</td>
<td>0.720829</td>
<td>0.694531</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>64</td>
<td>0.697268</td>
<td>0.686969</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>128</td>
<td>0.694730</td>
<td>0.691821</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>256</td>
<td>0.713615</td>
<td>0.710543</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>512</td>
<td>0.678477</td>
<td>0.676679</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1024</td>
<td>0.679842</td>
<td>0.679036</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>2048</td>
<td>0.676239</td>
<td>0.675878</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9</td>
<td>0.714286</td>
<td>0.619048</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>27</td>
<td>0.719603</td>
<td>0.684588</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>81</td>
<td>0.662499</td>
<td>0.651369</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>243</td>
<td>0.657697</td>
<td>0.653951</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>729</td>
<td>0.665424</td>
<td>0.664148</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2187</td>
<td>0.658342</td>
<td>0.657918</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>16</td>
<td>0.777778</td>
<td>0.722220</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>64</td>
<td>0.741020</td>
<td>0.726974</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>256</td>
<td>0.731804</td>
<td>0.732217</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>1024</td>
<td>0.738580</td>
<td>0.737671</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>25</td>
<td>0.818182</td>
<td>0.781818</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>125</td>
<td>0.792950</td>
<td>0.785583</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>625</td>
<td>0.790515</td>
<td>0.789015</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3125</td>
<td>0.791579</td>
<td>0.791283</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>36</td>
<td>0.846154</td>
<td>0.820513</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>216</td>
<td>0.827795</td>
<td>0.823466</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1296</td>
<td>0.825629</td>
<td>0.825883</td>
</tr>
</tbody>
</table>
3.5 RESULTS AND CONCLUSIONS

A critical comparison of the algorithms presented (vide section (3.2) & (3.3), Tables (3.4.1) and (3.4.2), and the graphs in Fig. (3.4.1) and (3.4.2), one can draw the following conclusions:

The edge loading algorithm presented in Section (3.2) and (3.3) and Table (3.4.1) (vide column (4) and (6)) the following observations are made

(i) for second degree Modified De Bruijn graphs (networks) of sizes 64, 128, 256, 512 and 1024 nodes, the maximum edge loading ($L_{\text{max}}$) values of $G^*(\Delta, d)$ are smaller than that of $G(\Delta, d)$.

(ii) For 3rd, 4th, 5th and 6th degree Modified De Bruijn graphs (networks) of all sizes (vide columns 3 of Table (3.4.1)), the maximum edge loading ($L_{\text{max}}$) values are same for both $G^*(\Delta, d)$ and $G(\Delta, d)$.

From the Table (3.4.1) (vide column (5) and (7)), the following observations can be made regarding average edge loading ($\bar{L}$).

For Modified De Bruijn graphs (networks) of all sizes, $\bar{L}$ values (vide columns (5) and (7) of Table (3.4.1)) are smaller than that of De Bruijn graphs.
The edge loading values are more evenly distributed in $G^* (\Delta, d)$ than $G(\Delta, d)$. For instance, from the graphs shown in Fig. (3.4.1) and (3.4.2), we find that the edge load distribution of $G^*(4, 5)$ is more evenly distributed than that of $G(4, 5)$. From Table (3.4.2), (vide column (4) and (5), we observe that the average network utility ($U_{avg}$) values of $G^*(\Delta, d)$ are slightly smaller than that of $G((\Delta, d))$. However, the edge loading algorithm presented in section (3.3) facilitates in providing higher link utilisation for individual links in $G^*(\Delta, d)$ to that of $G(\Delta, d)$ because from Fig.(3.4.1) and (3.4.2.), where the results of $G^*(4,5)$ are more evenly distributed over all edges than that of $G(4,5)$. 