Efficient Routing and Broadcasting Algorithms in de Bruijn Networks

Ngoc Chi Nguyen, Nhat Minh Dinh Vo and Sungyoung Lee

Computer Engineering Department, Kyung Hee Univerity
1, Seocheon, Giheung, Yongin, Gyeonggi 449-701 KOREA
{ncngoc, vdmnhat, sylee}@oslab.khu.ac.kr

Abstract. Recently, routing on dBG has been investigated as shortest path and fault tolerant routing but investigation into shortest path in failure mode on dBG has been non-existent. Furthermore, dBG based broadcasting has been studied as local broadcasting and arc-disjoint spanning trees based broadcasting. However, their broadcasting algorithms can only work in dBG(2,k). In this paper, we investigate shortest path routing algorithms in the condition of existing failure, based on the Bidirectional de Bruijn graph (BdBG). And we also investigate broadcasting in BdBG for a degree greater than or equal to two[1].

1 Introduction

For routing in dBG, Z. Liu and T.Y. Sung [1] proposed eight cases shortest paths in BdBG. Nevertheless, Z. Liu's algorithms do not support fault tolerance. J.W. Mao [4] has also proposed the general cases for shortest path in BdBG (case RLR or LRL). For fault tolerance issue, he provides another node-disjoint path of length at most $k + log_2 k + 4$ (in dBG(2,k)) beside shortest path. However, his algorithm can tolerate only one failure node in binary de Bruijn networks and it cannot achieve shortest path if there is failure node on the path. Broadcasting problems on dBG have been investigated as local broadcasting[6] and arc-disjoint spanning trees[7][8]. Nonetheless, the above can only work in a binary de Bruijn network (dBG(2,k)).

Considering limitations of routing and broadcasting in dBG, we intend to investigate shortest path routing in the condition of failure existence and broadcasting in BdBG with a degree greater than or equal to two. Two Fault Free Shortest Path (FFSP) routing algorithms and one broadcasting algorithm (for one-to-all broadcasting in the all port communication model) are proposed. Time complexity of FFSP2 in the worst case is $0(2^{\frac{k}{2}+1}d)$ in comparison with $0((2d)^{\frac{k}{2}+1})$ of FFSP1 (in dBG(d,k) and k=2h). Our study shows that the maximum time steps to finish broadcast procedure is k regardless of the broadcast originator, time complexity at each node is $0(\frac{3}{2}d)$, and no overhead happens in the broadcast message.

[1] This research was partially supported by ITRC project of Sunmoon University

The rest of this paper is organized as follows. Background is discussed in section 2. In section 3, FFSP routing algorithms are presented. Performance analysis for FFSP routing algorithms is carried in section 4. Section 5 discuss about broadcasting algorithm in dBG(d,k). Finally, some conclusions will be given in Section 6.

2 Background

The BdBG graph denoted as BdBG(d,k)[1] has N=d^k nodes with diameter k and degree 2d. If we represent a node by $d_0d_1...d_{k-2}d_{k-1}$, where $d_j \in 0, 1, ..., (d-1)$, 0≤j≤(k-1), then its neighbor are represented by $d_1...d_{k-2}d_{k-1}p$(L neighbors, by shifting left or L path) and $pd_0d_1...d_{k-2}$(R neighbors, by shifting right or R path), where $p = 0, 1, ..., (d-1)$. We write if the path $P = R_1L_1R_2L_2$ consists of an R-path called R_1, followed by an L-path called L_1, an R-path called R_2, an L-path called L_2, and so on, where subscripts are used to distinguish different sub-paths. Subscripts of these sub-paths can be omitted if no ambiguity will occur, e.g., $P = R_1LR_2$ or P=RL. Shift string of a node A is a binary string (0 for left shift and 1 for right shift) which represents path from originator to A.

For simplest broadcasting mechanism, the originator initiates the process by making a "call" to other neighboring vertices in the graph informing them of the message. Subsequently, the informed vertices call their neighboring vertices and the process continues until all vertices in the graph are informed. Basically, this mechanism is like flooding phenomenon. Note that the interval during which a call takes place will be referred to as a time step or simply step. In flooding broadcasting (FB), level of a node A is the number of steps by which a message from originator reaches A (or shortest path length between A and originator).

The following fig. 1a shows us an example for BdBG(2,4). Fig. 1b shows us eight cases of shortest path routing on BdBG. The gray areas are the maximum substring between source (s) and destination (d). The number inside each block represents the number of bits in the block.

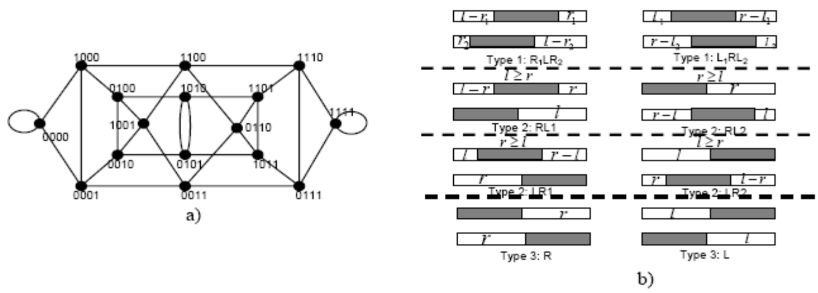

Fig. 1. a)The BdBG(2,4); b)Shortest path types[1].

3 Fault Free Shortest Path Routing Algorithms

In order to provide shortest path in the condition of failure existing, several paths of a specific source destination pair must be provided. Then FFSP is found among those paths. Therefore, the following concepts are proposed. For those, we assume that there is a separately protocol which detects failure nodes and then let other nodes know in periodically.

Definition 1: *the level m^{th} discrete set (DS_m) is a set which contains all neighbors of each element in discrete set level m-1; in the constraint that there is no existent element of discrete set level m coincides with another element of discrete set level q^{th} ($q \leq m$) or failure node set.*

Lemma 1: *DS_m is fault free.*

Lemma 2: *all the neighbors of a node belong to DS_m are in DS_{m-1}, DS_m and DS_{m+1}, except failure nodes.*

Proof: obviously we see that DS_1 and DS_2 contain all the neighbors of DS_1 except failure nodes; DS_1, DS_2 and DS_3 contain all the neighbors of DS_2 except failure nodes. So Lemma 2 is right at m=1,2. Assuming that lemma 2 is right until p, now we prove it is right at p+1. Suppose it's wrong at p+1. That means there exist a neighbor A of an element B$\in DS_{p+1}$, and A $\in DS_i, i < p$. Because lemma 2 is right until p, hence all the neighbors of A are in DS_{i-1}, DS_i and DS_{i+1} except failure nodes. Therefore, there exists an element B'$\in DS_{i-1}, DS_i$ or DS_{i+1}, and B'=B. It contradicts with definition 1. So Lemma 2 is right at p+1. Following inductive method, lemma 2 is proved.

Lemma 3: *there exists no neighbor of any element of DS_m, which is a duplicate of any element of DS_h, $\forall h \leq m$-2.*

Proof: suppose there is a neighbor A of an element B $\in DS_m$ duplicates with an element A' of DS_h (h $\leq$ m-2). Following Lemma 2, all the neighbors of A' are in DS_{h-1}, DS_h and DS_{h+1}. Therefore, there must exist a neighbor B' of A' in level h-1 or h or h+1, and B'=B. It contradicts with definition 1.

Corollary 1: *for duplicate checking at the next level of DS_q, it is not necessary to check with any element of DS_m,$\forall m \leq q$-2.*

By assigning source node S to DS_1, then expanding to the higher level, we have the following theorem.

Theorem 1: *in BdBG(d,k), we can always find a FFSP from node S$\in DS_1$ to node $A_x \in DS_x$($\forall x \leq k$), if it exists.*

Proof: we use inductive method to prove this theorem. When x=1, 2, theorem 1 is right. Assuming that theorem 1 is right until m, m $\leq$ k. Now we prove it is right until m+1. Suppose that path from S to A_{m+1} is not the FFSP. Then we have the following cases,

- There exist $A_p \in DS_p$, $A_p = A_{m+1}$and $p < m+1$.It contradicts definition 1.
- There exists a FFSP, S$\rightarrow B_1 \rightarrow B_2 \rightarrow ... \rightarrow B_k \rightarrow ... \rightarrow B_z \rightarrow ... \rightarrow A_{m+1}$, and B_k, B_{k+1},..., B_z not belonging to any DS_i ($\forall$ i$\leq$m+1). Because $B_{k-1} \in DS_j$ (j$\leq$m+1). Following Lemma 2, all the neighbors of B_{k-1} are in $DS_{j-1} or DS_j or DS_{j+1}$, except failure nodes. Therefore, B_k must be a failure node.

$\rightarrow$Theorem 1 is right at m+1. Theorem 1 is proved.

Corollary 2: *path length of a path from S$\in DS_1$ to $A_x \in DS_x$ is x-1.*

Fault free shortest path algorithm 1 (FFSP1) is proposed as a result of theorem 1 (shown in fig. 2a). It can always find FFSP in all cases (fault free mode, arbitrary failure mode) if the network still remain connected.

Proof of FFSP1: suppose path s→...→a_{ip}→b_{jk}→...→d is not FFSP, and then we have the following cases,

• There exist a FFSP s→...→$a_{i'p'}$→$b_{j'k'}$→...→d (i'≤i, j'≤j). It contradicts with the above assumption that a_{ip} and b_{jk} are the first neighbors between discrete sets A and B.

• There exist a FFSP s→...→$a_{i'p'}$→c_1→...→c_m→$b_{j'k'}$→...→d ($i' < i, j' < j$), and $c_1, c_2, ..., c_m$ do not belong to any discrete set A_p or B_q (p≤i, q≤j). Due to $a_{i'p'}$ ∈$A_{i'}$ and following lemma 2, all the neighbors of $a_{i'p'}$ are in $A_{i'-1}, A_{i'}$ and $A_{i'+1}$ except failure nodes. Therefore c_1 must be a failure node.

Example 1: we want to find a FFSP from source 10000 to destination 01021, failure node 00102 (dBG(3,5)).

Applying FFSP1, we have, A_1 = (10000) B_1 = (01021) A_2 = (00000, 00001, 00002, 01000, 11000, 21000) B_2 = (10210, 10211, 10212, 00102, 10102, 20102).

However, 00102 is a failure node. So B_2=(10210, 10211, 10212, 10102, 20102).

A_3 = (20000, 00010, 00011, 00012, 00020, 00021, 00022, 10001, 10002, 00100, 10100, 20100, 01100, 11100, 21100, 02100, 12100, 22100)

Then we find that 02100 and 10210 in A_3 and B_2 are the first neighbors. FFSP is found by tracking back from 02100 to 10000 and 10210 to 01021. We have FFSP 10000 → 21000 → 02100 → 10210 → 01021. In this example, FFSP1 can provide 2 shortest paths (in the case of no failure node) 10000 → 21000 → 02100 → 10210 → 01021 and 10000 → 00001 → 00010 → 00102 → 01021. We pick up one FFSP 10000→21000→02100→10210→01021 (node 00102 is fail).

Furthermore, we shall see that other elements like 00000, 00002, 01000, 11000 in A_2 are useless in constructing a FFSP. So, eliminating these elements can reduce the size of A_3 (reduce the cost at extending to next level) and improve the performance of our algorithm. It shows the motivation of FFSP2. Before investigating FFSP2, we give some definition and theorem.

Definition 2: *a dominant element is an element which makes a shorter path from source to a specific destination, if the path goes through it.*

Example 2: from the above example 1 we have 2 shortest paths (in the case 00102 is not a failure node) 10000 → 21000 → 02100 → 10210 → 01021 and 10000 → 00001 → 00010 → 00102 → 01021. Thus 00001 and 21000 are dominant elements of A_2, because they make shorter path than others of A_2.

Therefore, by eliminating some non-dominant elements in a level, we can reduce the size of each level in FFSP1 and hence, improve the performance of FFSP1. A question raised here is how we can determine some dominant elements in a DS_k and how many dominant elements, in a level, are enough to find FFSP. The following theorem 2 is for determining dominant elements and corollary 3 answer the question, how many dominant elements are enough.

Theorem 2: *If there are some elements different in 1 bit address at leftmost or rightmost, the dominant element among them is an element which has shorter* *path length toward destination for cases RL2, R (shown in fig. 1b) for leftmost bit difference and LR2, L for rightmost bit difference.*

Proof: as showing in fig. 1b, there are eight cases for shortest path. Only four cases RL2, R, LR2 and L make different paths when sources are different in leftmost bit or rightmost bit.

Example 3: following example 1, we check the dominant characteristic of three nodes A 01000, B 11000 and C 21000 (in A_2) to destination D 01021. Three nodes A, B and C are leftmost bit difference. So, type RL2, R are applied.

• Apply type R: the maximum match string between A 01000 and D 01021 is 0, between B **1**1000 and D 0102**1** is 1, and between C **21**000 and D 010**21** is 2 $\rightarrow$ min path length is 3, in case of node C.

• Apply type RL2: the maximum match string [5] between A **0**1000 and D 01**0**21 is 1 (path length: 6), between B **1**1000 and D 0**1**021 is 1 (path length: 7), and between C **21**000 and D 010**21** is 2 (same as case R) $\rightarrow$ min is 3, node C.

Therefore, minimum path length is 3 and dominant element is C.

```
DECLARE two array A[n] and B[n] of DS
        type
SET A[1] to s
SET B[1] to d
SET i and j to 1
WHILE (a_ip is not neighbor of b_jk)
    CALL Expand function of A[i] and return
        value to A[i+1]
    i = i+1
    IF (a_ip is neighbor of b_jk) THEN
        Exit while loop
    ENDIF
    CALL Expand function of B[j] and return
        value to B[j+1]
    j=j+1
ENDWHILE

DEFINE Expand function of a DS M
    DECLARE array N of DS type
    SET N to DS of all neighbors of each
        element in M
    CALL duplicate_check(N)
    RETURN(N)
ENDDEFINE
```

a)

```
DECLARE two array A[n] and B[n] of DS type
SET A[1] to s
SET B[1]  to d
SET i and j to 1
WHILE (a_ip is not neighbor of b_jk)
    CALL SPD function with A[i] and return value to A[i+1]
    i = i+1
    IF (a_ip is neighbor of b_jk) THEN
      EXIT while loop
    ENDIF
    CALL SPD function with B[j] and return value to B[j+1]
    j=j+1
ENDWHILE

DEFINE   SPD function of a DS M
    DECLARE array N of DS type
    SET N to DS of all neighbors of each element in M
    WHILE (there exist set of elements T which differ in 1 bit
            at leftmost or rightmost in N)
      IF (T is leftmost bit difference) THEN
         CALL Pathlength function type RL2 and R
         CALL Eliminate function to eliminate non-dominant
              elements
      ENDIF
      IF (T is rightmost bit difference) THEN
         CALL Pathlength function type LR2 and L
         CALL Eliminate function to eliminate non-dominant
              elements
      ENDIF
    ENDWHILE
    CALL duplicate_check(N)
    RETURN(N)
ENDDEFINE
```

b)

Fig. 2. a)Fault Free Shortest Path Algorithm 1 (FFSP1); b)Fault Free Shortest Path Algorithm 2 (FFSP2).

Corollary 3: *when we apply theorem 2 to determine dominant elements, the maximum elements of DS_{m+1} are 2p(p is the total elements of DS_m).*

Proof: the maximum elements of DS_{m+1} by definition 1 are 2pd (dBG(d,k)). We see that in 2pd there are 2p series of d elements which are different in 1 bit at leftmost or rightmost. By applying theorem 2 to DS_{m+1}, we obtain 1 dominant element in d elements differed in 1 bit at leftmost or rightmost.

Fault Free Shortest Path Algorithm 2 (FFSP2) is proposed in fig. 2b.

The condition in line 5 and line 8 (fig. 2a, 2b) let us know whether there exists a neighbor of array A and B of discrete set, $\forall a_{ip} \in$A[i],$\forall b_{jk} \in$B[j] . The SPD(M) function, line 14 fig. 2b, finds the next level of DS M (DS N) and eliminates non-dominant elements in N followed theorem 2. Expand(M) function, line 14 fig. 2a, finds the next level of DS M. Pathlength type p function, line 19,23 fig. 2b, checks path length followed type p of each element in T toward destination. Eliminate function, line 20, 24, eliminates element in T, which has longer path length than the other. The duplicate_check(N) function, line 17 fig. 2a and line 27 fig. 2b, check if there is a duplicate of any element in N with other higher level DS of N. For duplication checking, we use the result from corollary 1. Then, we get FFSP by going back from a_{ip} to s and b_{jk} to d.

Example 4: we try to find FFSP as in example 1. By applying FFSP2, we have, A_1 = (10000) B_1 = (01021) A_2 = (00001, 21000) B_2 = (10210, 00102). However, 00102 is a failure node. So B_2 becomes (10210).

A_3 = (00010, 10000, 10001, 02100). However, node 10000 coincides with 10000 of A_1. So A_3 becomes (00010, 10001, 02100). Then we find that 02100 and 10210 in A_3 and B_2 are the first neighbors. FFSP is found by tracking back from 02100 to 10000 and 10210 to 01021. We have FFSP 10000 → 21000 → 02100 → 10210 → 01021.

4 Performance analysis for FFSP1 and FFSP2

Mean path length is the significant to analyze and compare our algorithm to others. Z. Feng and Yang [2] have calculated it based on the original formula, Mean path length = $\frac{Totalinternaltraffic}{Totalexternaltraffic}$ for their routing performance. We can use the above equation to get the mean path length in the case of failure. We assume that failure is random, and our network is uniform. That means the probability to get failure is equal at every node in the network.

Table 1 shows the results in the simulation of mean path length using six algorithms, SCP[3], RFR, NSC, PMC[2], FFSP1 and FFSP2. Our two algorithms show to be outstanding in comparison with the four algorithms. They always achieve shorter mean path length than the other algorithms.

This section is completed with study in time complexity of our algorithms. As A. Sengupta [9] has shown that dBG(d,k) has connectivity of d-1. Hence, our time complexity study is based on assumption that the number of failures is at most d-1 and our study is focused on large network with high degree (d>>1). Therefore, diameter of our network in this case is k. We have the following cases,

• For FFSP1, the second level DS lies in the complexity class 0(2d) , the third level DS lies in the complexity class 0(2d(2d-1))≈ 0($4d^2$), the fourth lies in

Table 1. Mean path length of FFSP1, FFSP2 in comparison with others.

d,k	no. of nodes	Mean path lengths								
		SCP (fault free mode)	RFR (fault free mode)	NSC (fault free mode)	PMC (fault free mode)	**FFSP1/2 (fault free mode)**	**FFSP1 (1 node fail)**	**FFSP2 (1 node fail)**	**FFSP1 (2 nodes fail)**	**FFSP2 (2 nodes fail)**
2,2	4	1.167	1.167	1.167	1.167	**1.167**	**1.333**	**1.333**	**1**	**1**
2,3	8	1.643	1.643	1.643	1.643	**1.643**	**1.762**	**1.762**	**1.733**	**1.733**
2,4	16	2.258	2.188	2.146	2.142	**2.142**	**2.1**	**2.21**	**2.286**	**2.286**
2,5	32	2.984	2.796	2.794	2.766	**2.754**	**2.787**	**2.794**	**2.285**	**2.832**
2,6	64	3.801	3.653	3.551	3.495	**3.453**	**3.467**	**3.483**	**3.487**	**3.506**
3,2	9	1.417	1.471	1.417	1.417	**1.417**	**1.429**	**1.429**	**1.476**	**1.476**
3,3	27	2.128	2.105	2.007	2.077	**2.077**	**2.095**	**2.095**	**2.117**	**2.117**
3,4	81	2.978	2.911	2.865	2.849	**2.833**	**2.84**	**2.842**	**2.846**	**2.848**
3,5	243	3.907	3.800	3.755	3.736	**3.674**	**3.676**	**3.696**	**3.678**	**3.698**
3,6	729	4.875	4.775	4.704	4.722	**4.572**	**4.573**	**4.626**	**4.573**	**4.627**
4,2	16	1.550	1.550	1.550	1.550	**1.550**	**1.552**	**1.552**	**1.56**	**1.56**
4,3	64	2.369	2.343	2.321	2.321	**2.321**	**2.327**	**2.327**	**2.335**	**2.335**
4,4	256	3.298	3.251	3.214	3.218	**3.178**	**3.18**	**3.185**	**3.184**	**3.189**
4,5	1024	4.273	4.207	4.172	4.209	**4.1**	**4.101**	**4.13**	**4.101**	**4.13**
5,2	25	1.633	1.633	1.633	1.633	**1.633**	**1.634**	**1.634**	**1.636**	**1.636**
5,3	125	2.510	2.491	2.471	2.471	**2.471**	**2.473**	**2.473**	**2.476**	**2.476**
5,4	625	3.471	3.438	3.41	3.437	**3.378**	**3.379**	**3.385**	**3.379**	**3.385**
6,2	36	1.690	1.690	1.690	1.690	**1.690**	**1.691**	**1.691**	**1.693**	**1.693**
6,3	216	2.601	2.585	2.570	2.570	**2.570**	**2.571**	**2.571**	**2.573**	**2.573**

$0(2d(2d-1)^2) \approx 0(8d^3)$, etc... Hence, time complexity of FFSP1 lies in the complexity class $0((2d)^n)$, the value of n equals to the maximum level DS provided by FFSP1. In the worst case, time complexity of FFSP1 lies in $0((2d)^{\frac{k}{2}+1})$ (k=2h), or $0((2d)^{\frac{k+1}{2}})$ (k=2h+1), k is maximum path length from source to destination (the diameter).

• The computation time of FFSP2 can be divided into 2 parts. One is performing computation on expanding to next level, checking for duplicate and neighboring checking between DS A[m] and B[q]. This part is like FFSP1, the difference is that each DS here grows following a geometric progression with common quotient 2 and initial term 1 (as shown in corollary 3). The other part is performing computation on finding dominant elements. Hence, the second level DS lies in the complexity class 0(2+2d)≈0(2d), the third level DS lies in the complexity class 0(4+4d)≈0(4d), the fourth lies in 0(8+8d)≈0(8d), etc... Hence time complexity of FFSP2 lies in the complexity class $0(2^n d)$, the value of n equals to the maximum level DS provided by FFSP2. FFSP2 would cost us $0(2^{\frac{k}{2}+1}d)$ (k=2h), or $0(2^{\frac{k+1}{2}})$ (k=2h+1) time in the worst cases, k is maximum path length from source to destination (the diameter).

5 Broadcasting algorithm in dBG(d,k)

By applying FB, we can easily obtain k as the maximum number of steps to finish broadcasting. However, message overhead is very high in FB. Thus, how to reduce message overhead (or letting each informed vertices call its uninformed neighbors only) in FB states the motivation for our algorithm. We assume that each packet sent to the other node must contain originator address, sender's level, a shift string of receiver and all calls take the same amount of time.

There are two cases of message overhead when an informed node A wants to inform node X. Case 1, node X has been informed already. Thus, X must have lower or equal level to A. Case 2, uninformed node X can be informed by nodes B,C,D, which have the same level as A, at the same time. For case 1, we need to compare the shortest-path length between X and A to originator. And X is informed by A if X level is higher than A's level and case 2 not happen. For case 2, we have to define some conditions, based on these conditions only A or B or C or... inform X. The following theorems are proposed for calculating path length.

Theorem 3: *given p is shortest-path length between node a and b, the minimum length of matched strings between a and b is k-p (dBG(d,k)).*

Proof: as shown in fig. 1b, there are 3 types for determining shortest path (R,L; RL,LR; R_1LR_2, L_1RL_2). The minimum matched string[5] can be obtained in type R,L among them. And length for this minimum matched string is k-p.

Theorem 4: *path length between node s and d is* $min(2s_j + s_i + d_i, 2s_i + s_j + d_j)$*, where* s_i *and* d_i *are the left indices, and* s_j *and* d_j *are the right indices of matched string in s and d respectively.*

Proof: path length $2s_j + s_i + d_i$, $2s_i + s_j + d_j$ are for case $R_{s_j}L_{s_j+s_i}R_{d_i}$ and $L_{s_i}R_{s_i+s_j}L_{d_j}$ respectively . These cases are the general cases for 3 types presented in fig. 1b(ex. if $s_i, s_j, d_i, d_j \neq 0$, they become type R_1LR_2 and L_1RL_2).

To solve the above two cases of message overhead, a Boolean valued function SPL is proposed. SPL has inputs: originator S, current node P, neighboring node X, current level n (level of P), shift string Q ($q_0q_1q_2...q_{z-1}$, length z≤k) (from S to X through P). Fig. 3a shows SPL algorithm. Step 1,2,3 solve message overhead of case 1. Step 1 is a result of theorem 3. Step 4,5,6 solve case 2 message overhead. In case 2, we have several shortest paths from S to X. One shortest path must be chosen based on the following conditions:

• Shortest path corresponds with shortest matched string of S and X(step5).

• In the case, there exist 2 shortest path from the first condition. Then, shortest path which begin with shifting right is chosen. (step 6)

Step 7 compares shift string Q to the condition gotten from step 5 and 6 to determine whether X should be informed or not.

Example 5: in dBG(3,10), given input S: 0012111001, P: 0111012110, n=7, X: 1110121100, Q=01111100. By applying SPL, we have

Step 1: find all matched strings[5] which have length higher or equal 10-7-1=2. These strings are 11, 111, 1110, 01, 012, 0121, 01211, 110, 1100.

Step 2: path lengths for strings in step 1 are 12, 10, 8, 14, 12, 10, 8, 13, 11.

Step 3: shortest path length is 8.

Step 4: matched string, which make shortest path length 8, are 1110, 01211.

Step 5: minimum size string from step 4 is 1110, b=false.

Step 6:Typeiden(input $s_i = 0, s_j = 6, d_i = 4, d_j = 2$)→returned value: 1,a=1.

```
Step 1    Find all matched strings between X and S which have
          length higher or equal k-n-1
Step 2    For each matched string from step 1, calculate path
          length follow theorem 2
Step 3    Find shortest path length from step 2
          IF (shortest path length ≤ n) THEN
             RETURN false
             goto End
          ENDIF
Step 4    Find all matched strings (from step 1) which make
          shortest path length (from step 3)
          IF (only one matched string is found) THEN
             RETURN true
             goto End
          ENDIF
Step 5    Find minimum size matched strings from step 4
          IF (2 matched strings are found) THEN
             b=true
             ELSE b=false
          ENDIF
Step 6    CALL Typeiden for matched string in step 5 and
          save value to a
Step 7    IF (b=true) THEN
             FOR i=0 TO z-1 DO
                IF q_i ≠ q_i+1 THEN a=a-1
                ENDIF
             ENDFOR
             IF (a=0) and (q_0=1) THEN RETURN true
                ELSE RETURN false
             ENDIF
          ELSE
             FOR i=0 TO z-1 DO
                IF q_i ≠ q_i+1 THEN a=a-1
                ENDIF
             ENDFOR
             IF (a=0) THEN RETURN true
                ELSE RETURN false
             ENDIF
          ENDIF
End.
```

a)

```
Input s_i, s_j, d_i, d_j: left index, right index of matched
string in s and d respectively

IF (s_i,s_j,d_i,d_j ≠ 0) THEN   RETURN 2
IF (s_i=0) and (s_j,d_i,d_j ≠ 0) THEN   RETURN 1
IF (s_j=0) and (s_i,d_i,d_j ≠ 0) THEN   RETURN 1
IF (d_i=0) and (s_i,s_j,d_j ≠ 0) THEN   RETURN 1
IF (d_j=0) and (s_i,s_j,d_i ≠ 0) THEN   RETURN 1
IF (s_i=d_j=0) and (s_j,d_i ≠0) THEN   RETURN 0
IF (s_j=d_i=0) and (s_i,d_j ≠0) THEN   RETURN 0
```

b)

```
Input S: originator, P: current node, n: previous level, Q:
current shift string
n=n+1
i=0
WHILE (i<2d)
   M ← Q
   i=i+1
   X ← neighbor of P
   IF (X is right neighbor of P) THEN
     Append 1 to M
     IF (M is one of cases of theorem 3, 4) THEN
        CONTINUE WHILE Loop
     ENDIF
     IF (SPL(S,P,X,n,M)=true) THEN
        Send message to X
     ENDIF
   ELSE
     Append 0 to M
     IF (M is one of cases of theorem 3, 4) THEN
        CONTINUE WHILE Loop
     ENDIF
     IF (SPL(S,P,X,n,M)=true) THEN
        Send message to X
     ENDIF
   ENDIF
ENDWHILE
```

c)

Fig. 3. a)SPL function algorithm; b)Typeiden function algorithm; c)Broadcasting algorithm for dBG(d,k).

Step 7: there are 2 places in Q in which two adjacent bits are different → a=-1 ≠0. Consequently, X is an uninformed node (step 3,8>n), but it isn't informed by P (message overhead case 2) due to our priority given in step 5 and 6.

If we apply SPL for all 2d neighbors of one node, then it cost 0(2d) for running our algorithm. The following theorems reduce from 0(2d) to 0(1.5d). Following are some notations used, where T is the previous shifting string.

R↔T: total number of right shift in T > total number of left shift in T

L↔T: total number of left shift in T > total number of right shift in T

Theorem 5: *by shifting RLR/LRL, results are duplicate with shifting R/L.*

Proof: given a node $a_0a_1...a_{n-1}$. By shifting RLR in dBG(d,k), we have $a_0a_1...a_{n-1} \rightarrow \alpha a_0a_1...a_{n-2} \rightarrow a_0a_1...a_{n-2}\beta \rightarrow \gamma a_0a_1...a_{n-2}$, $0 \leq \alpha, \beta, \gamma <$d.

Substitute α for γ → $\gamma a_0a_1...a_{n-2} \equiv \alpha a_0a_1...a_{n-2}$.

By proving similarly for case LRL, theorem 5 is proved.

Theorem 6: *if R↔T/L↔T, results provided by next shift LR/RL are duplicate.*

Proof: assume the beginning node is $a_0a_1...a_{n-1}$. For case R↔T, we have the following cases:

•$T = R_uL_vR_w, T = L_uR_vRw, T = L_uR_v$. By shifting LR, we have shift string $R_1L_1R_2L_2$ or $L_1R_1L_2R_2$, which are not existed for shortest path (as shown in Lemma 1 of [1]).

•$T = R_uL_v$ (u>v). By shifting R u times and L v times respectively, we have

$a_0a_1...a_{n-1} \to \beta_{u-1}...\beta_1\beta_0a_0a_1...a_{n-u-1} \to \beta_{u-v-1}...\beta_1\beta_0a_0a_1...a_{n-u-1}\delta_0\delta_1...\delta_{v-1}$
where $0 \leq \beta_i, \delta_j <$ d, 0≤i<u, 0≤j<v. By shifting LR we have,

$\beta_{u-v-1}...\beta_0a_0...a_{n-u-1}\delta_0...\delta v - 1 \to \beta_{u-v-2}...\beta_0a_0...a_{n-u-1}\delta_0...\delta v \to$
$\gamma\beta_{u-v-2}...\beta_0a_0...a_{n-u-1}\delta_0...\delta v - 1$ (K)

Substitute $\gamma(0 \leq \gamma <$d) for $\beta_{u-v-1} \to$ K is duplicate.

• R=R_u. Shift string R_uLR makes duplicate as shown in theorem 3.

By proving similarly to case L↔T, we prove theorem 6.

As a result, broadcasting algorithm is proposed as shown in fig. 3c.

Theorem 7: *in the worst case, time complexity for our broadcasting algorithm is 0(1.5d).*

Proof: probability for theorem 5 happening is 25%, and for theorem 6 is less than 25%. Therefore, the probability for CONTINUE command (line 10, 18 fig. 3c) happening is 25%. So, theorem 7 is proved.

6 Conclusion

We have proposed new concepts, routing algorithms and distributed broadcasting algorithm in dBG(d,k). Our routing algorithms can provide shortest path in the case of failure existence. Our simulation result shows that FFSP2 is an appropriate candidate for the real networks with high degree and large number of nodes, while FFSP1 is a good choice for high fault tolerant network with low degree and small/medium number of nodes. In broadcasting, our algorithm requires maximum k steps to finish broadcasting process in dBG(d,k). And there is no message overhead during broadcasting. Time complexity at each node is $0(\frac{3}{2}d)$. Therefore, the algorithms can be considered feasible for routing and broadcasting in real interconnection networks.

References

1. Zhen Liu, Ting-Yi Sung, "Routing and Transmitting Problem in de Bruijn Networks" IEEE Trans. on Comp., Vol. 45, Issue 9, Sept. 1996, pp 1056 - 1062.
2. O.W.W. Yang, Z. Feng, "DBG MANs and their routing performance", Comm., IEEE Proc., Vol. 147, Issue 1, Feb. 2000 pp 32 - 40.
3. A.H. Esfahanian and S.L. Hakimi, "Fault-tolerant routing in de Bruijn communication networks", IEEE Trans. Comp. C-34 (1985), 777.788.
4. Jyh-Wen Mao and Chang-Biau Yang, "Shortest path routing and fault tolerant routing on de Bruijn networks", Networks, Vol. 35, Issue 3, Pages 207-215 2000.
5. Alfred V. Aho, Margaret J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search", Comm. of the ACM, Vol. 18 Issue 6, June 1975.
6. A.H.Esfahanian, G. Zimmerman, "A Distributed Broadcast Algorithm for Binary De Bruijn networks", IEEE Conf. on Comp. and Comm., March 1988.
7. E.Ganesan, D.K.Pradhan, "Optimal Broadcasting in Binary de Bruijn Networks and Hyper-deBruijn Networks", IEEE Symposium on Parallel Processing, April 1993.
8. S.R.Ohring, D.H.Hondel, "Optimal Fault-Tolerant Communication Algorithms on Product Networks using Spanning Trees", IEEE Symp. on Parallel Processing, 1994.
9. A. Sengupta, A.Sen, and S.Bandyopadhyay, "Fault tolerant distributed system design", IEEE Trans. Circuit Syst., Vol. CAS-35, pp. 168-172, Feb. 1988