드브르젼 네트워크에서 고장 노드를 포함하지 않는 최단 경로 라우팅

Nguyen Chi Ngoc*, Vo Dinh Minh Nhat*, 정 연 일*, 정회원 이 승 룡*

Fault Free Shortest Path routing on the de Bruijn networks

Nguyen Chi Ngoc*, Vo Dinh Minh Nhat*, Yonil Zhung*, Sungyoung Lee* *Regular Members*

요 약

드브르젼 그래프(dBG: de Bruijn graph)는 병렬 계산을 위한 구조나 인터커넥션 네트워크 설계에 사용 되고 있다. dBG 지향 라우팅 알고리즘은 고장포용(fault tolerance) 라우팅과 최단 경로 라우팅에 포함되어 연구되고 있지만, 아직까지 dBG에서 고장 노드를 포함하지 않는 최단 경로(FFSP) 프로토콜에 대한 연구는 없는 실정이다. 네트워크는 계속하여 그 크기가 커지기 때문에 현실적으로 네트워크 장애는 피할 수 없는 일이 생기게 된다. 더욱이, 그러한 네트워크 장애에 대비하여 보통의 라우팅 알고리즘은 긴 지체 시간과 낮은 처리량 그리고 높은 트래픽을 발생 시키게 된다. 본 논문은 양방향 드브르젼 그래프(BdBG)에 기반을 두고 네트워크 장애가 존재하는 상태에서의 두 가지 라우팅 알고리즘에 대하여 제안한다. 첫 번째는 알고리즘은 네트워크에 연결된 상태로 네트워크의 결함 노드가 존재할 경우에도 항상 최단거리 경로로 도달하게 하는 알고리즘이다. 두 번째 알고리즘은 첫 번째 알고리즘에 비해 최단 거리 경로를 찾아내는 성능을 높인 알고리즘이다. 두 알고리즘의 성능 평가 항목으로 경로 길이 측정과 이산 집합(Discrete Set: DS)의 크기를 정의하여 다른 알고리즘과 성능 평가를 비교 하였으며, 성능 평가 결과 제안한 알고리즘들은 dBG 기반을 둔 실제 네트워크를 위한 라우팅에 적합하다는 결론을 얻었다.

Key Words : de Bruijn graph; shortest path; fault tolerant routing; fault free shortest path.

ABSTRACT

It is shown that the de Bruijn graph (dBG) can be used as an architecture for interconnection networks and a suitable structure for parallel computation. Recent works have classified dBG based routing algorithms into shortest path routing and fault tolerant routing but investigation into fault free shortest path (FFSP) on dBG has been non-existent. In addition, as the size of the network increase, more faults are to be expected and therefore shortest path dBG algorithms in fault free mode may not be suitable routing algorithms for real interconnection networks, which contain several failures. Furthermore, long fault free path may lead to high traffic, high delay time and low throughput. In this paper we investigate routing algorithms in the condition of existing failure, based on the Bidirectional de Bruijn graph (BdBG). Two FFSP routing algorithms are proposed. Then, the performances of the two algorithms are analyzed in terms of mean path lengths and discrete set mean sizes. Our study shows that the proposed algorithms can be one of the candidates for routing in real interconnection networks based on dBG.

* 경희대학교 컴퓨터공학과 실시간멀티미디어 연구실(sylee@oslab.khu.ac.kr)
논문번호 : KICS2004-06-071, 접수일자 : 2004년 6월 30일
※ 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었습니다.

Ⅰ. 서 론

규칙적인 양방향 그래프는 고속 패킷 스위칭 네트워크에 있는 양방향 노드에 적용시키기에 좋은 방법이다. 네트워크를 위한 그래프는 다음과 같은 특징을 갖는 것이 좋다. 첫째, 네트워크의 저비용을 위한 적은 입출력 개수(degree). 둘째, 빠른 패킷 전달을 위한 간단한 라우팅. 세 번째, 짧은 메시지 딜레이나 높은 전송량을 위한 작은 지름(diameter, 노드들 간의 최단 거리들 중에서 가장 긴 거리). 네 번째, 높은 고장 포용(fault tolerant) 능력. 이러한 규칙적인 망구조의 대표적인 것들이 셔플(Shuffle), 맨하탄 스트리트 네트워크(Manhattan street network), 하이터규브(Hypercub), 카우츠 그래프(Kautz graph), 드브르젼 그래프(dBG)가 있다.

그 중에서도 dBG는 가장 광범위하게 연구 되고 있다[1]~[4],[6]~[16]. dBG는 NASA의 갈릴레오 프로젝트를 위하여 8096개의 노드로 구성 된 네트워크를 위해 선택되었으며, 애플리케이션이 실행되는 클러스터 사이에서 최단 경로 거리를 제공하기 위해 사용되었다. 또한, 광파 네트워크에서 파장 라우팅을 위한 논리적 망구조와 물리적 망구조에 쓰이고 있다[8][9][10]. 양방향 드브르젼 네트워크와 비교하여 단방향 드브르젼망의 역사는 논문 [7]에서 찾을 수 있다. BdBG는 양방향 링크를 이용하기 때문에 전체 네트워크의 성능을 향상시키는 이익이 있다. 따라서 본 논문은 BdBG에 초점을 맞추었다.

Samantham, Pradhan, Nadig, Sivarajan은 드브르젼 그래프(dBG)에서 하나의 노드 A와 노드 B의 좌/우의 공통된 부분을 이용하여 왼쪽이나(L) 오른쪽(R)으로 이동(shift) 시켜 노드 A로부터 노드 B로 가는 라우팅 알고리즘을 제안했다[1][4][8][9][14]. Z. Feng과Yang은 SCP(Shortest Consistent Path)[15]와 비교하여 가장 일반적인 dBG에서 세 가지 라우팅 알고리즘(RFR, NSC, PMC) 을 제안하였다[10][13]. 하지만, 이러한 방식은 특별한 경우의 BdBG에서만 최단 거리 경로를 찾을 수 있으며 고장 포용도 지원하지 못하였다.

Z. Liu 와 T.Y. Sung은 BdBG에서 여덟 가지 경우의 최단 거리 경로를 제안하였다[2]. 그렇지만 Z. Liu의 알고리즘도 마찬가지로 고장 포용은 지원하지 못하였다. J.W. Mao[14] 또한 BdBG의 일반적인 경우에서 사용되는 최단 거리 알고리즘을 제안하였다. dBG(2,k)인 그래프라면 고장 포용 이슈는 최단 거리 경로에 비하여, 다른 node-disjoint 경로의 길이가 k + log2k + 4 에 근접해야 하는 것이다. 하지만, 이러한 알고리즘들은 드브르젼 네트워크에서 오직 하나의 노드만 고장이 발생하는 것까지 포용하였고, 만약 경로 위에 다른 고장 노드가 발생 할 경우 최단 경로로 도달 하지 못하였다. Nadig와 Iyengar는 드브르젼 네트워크에서 하나의 결함을 피하기 위해서는 네 가지 홉(hops)을 추가해야 한다고 언급했다[4].

위에 언급한 내용들은 고장 노드를 피해 새로운 경로를 찾는 개념의 고장 포용 라우팅과 문자열 대조 방식으로 최단 거리 경로를 찾는 라우팅으로 분류 할 수 있다. 하지만 두 가지 알고리즘 방식은 몇 개의 고장 노드가 발생하는 실제 네트워크에서 고장 노드를 포함 하지 않는 최단 경로 라우팅을 지원하지 못하고 있다. 고장 노드를 포함 하지 않는 최단 경로(FFSP) 라우팅 알고리즘은 아직 제안된 적이 없기 때문에 본 논문에서 처음으로 고장 노드를 포함 하지 않는 최단 경로 라우팅 알고리즘을 제안한다.

본 논문에서는 두 가지의 FFSP 라우팅 알고리즘에 대하여 제안한다. 두 알고리즘은 멀티 레벨 이산 집합(Discrete Set: DS)이라는 새로운 개념을 바탕으로 만들어졌다. 첫 번째 FFSP1 알고리즘은 앞서 말한 멀티 레벨 DS를 사용하여 특정한 출발-목적쌍에서 여러 가지 서로 다른 경로를 준비하도록 하였다. 이러한 준비된 서로 다른 경로들을 중에서 최단 경로를 찾아내는 것이다. 제안하는 두 번째 FFSP2 알고리즘은 다른 최단 거리 경로를 찾아내는 성능을 높인 것으로 주 요소(dominant element)라는 새로운 개념을 사용한다. 주 요소를 이용하여 최단 경로를 찾아내는 시간을 줄임으로써 알고리즘의 성능을 높일 수가 있다. 최악의 경우, dBG(d,k)에서 k=2h 일 때 시간의 복잡성은 FFSP1의 $0((2d)^{\frac{k}{2}+1})$에 비교 하여 FFSP2의 시간의 복잡성은 $0(2^{\frac{k}{2}+1}d)$이다. 이것은 FFSP1 알고리즘 보다 최단 경로를 찾은 성능이 뛰어남을 보여준다.

본 논문은 다음과 같이 구성되어 있다. 2절에서는 dBG에 대하여 설명 하며, 3절에서는 제안하는 FFSP 라우팅 알고리즘에 대하여 설명한다. 4절에서는 SCP[13], RFR, NSC, PMC[9]과 같은 다른 라우팅 알고리즘과 비교한 시뮬레이션 결과를 보여주고, 마지막으로 5절에서 결론을 내릴 것이다.

Ⅱ. 드브르젼 그래프(The de Bruijn graph)

BdBG는 BdBG(d, k)로 표시 하며 $N=d^k$의 노드를 가지며 지름은 k, 입출력 개수는 2d 가 된다[1]. d만큼의 값을 갖는 k 길이의 상태 그래프에서 한쪽으로 상태 데이터를 이동시키고 그 반대쪽에서 하나의 수치가 나오게 하여 상태를 변화 시키게 된다. 만약 $d_j \in \{0,1,\ldots(d-1)\}, 0 \le j \le (k-1)$에서, 하나의 노드를 $d_{k-1}d_{k-2}\ldots d_1 d_0$ 라고 한다면, $p=0,1,\ldots,(d-1)$ 일 때 왼쪽으로 이동된 왼쪽 이웃 노드 L은 $d_{k-2}d_{k-3}\ldots d_0 p$ 라고 표현 하고, 오른쪽으로 이동된 오른쪽 이웃 노드 R은 $pd_{k-1}d_{k-2}\ldots d_2 d_1$ 라 표현한다.

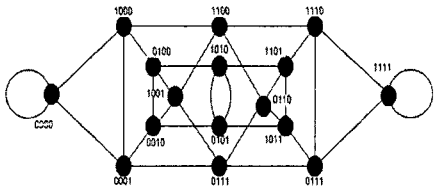


그림 1. The BdBG(2,4)

a로부터 b까지의 경로를 (a,b)-path 라고 표시하며, 오른쪽으로 이동 된 경로를 R-path, 왼쪽으로 이동된 경로를 L-path 라고 각기 따로 표시하여 사용한다. 만약 $P=R_1L_1R_2L_2$ 라고 표시된 경로 P는 R-path, L-path, R-path, L-path 등으로 이루어졌으며 아래 첨자는 다른 하부 경로와 구별하기 위해 사용된다. 아래 첨자는 $P=R_1LR_2$ 나 $P=RL$과 같이 다른 경로와 구분이 가능하면 생략할 수 있다. 또한 P 경로의 길이는 |P| 라고 표시 한다. 그림 1은 BdBG(2,4) 을 나타낸다.

Ⅲ. 고장 노드를 포함 하지 않는 최단 거리 라우팅 알고리즘

이번 절에서는 고장이 존재하는 조건에서의 최단 거리를 찾기 위한 두 개의 알고리즘에 대하여 논의 한다. 고장 노드를 발견하는 프로토콜은 제외를 하며 다른 노드들 주기적으로 이를 인지하고 있다고 가정한다.

1. FFSP(Fault Free Shortest Path) 1

Definition 1: *m 단계 Discrete Set (DS_m)은 m-1 단계 DS에 포함된 모든 요소의 이웃 요소를 포함한다. m-1 단계의 DS에서 포함 될 DS_m의 요소는 결함이 있는 요소는 제외를 하고, k 단계($k \le m$)와 중복이 되는 요소들은 제외 하게 된다.*

예제 1: BdBG(2,3)에서 DS_1이 {001, 100} 이고, 고장 노드가 {010} 라고 할 때, DS_2는 {100, 000, 010, 011, 001, 010, 110, 000} 가 된다. 여기에서 100, 001, 010, 000은 중복이 된 것을 알 수 있다. 그러므로 DS_2는 {000, 011, 110} 이 된다.

Lemma 1: *DS_m 은 고장 요소를 포함 하지 않는다.*

Lemma 2: *DS_k 에 속한 노드의 모든 이웃 노드들은 결함이 있는 노드를 제외하고 DS_{k-1}, DS_k DS_{k+1}에 포함 되어있다.*

Lemma 3: *$\forall h \le k-2$ 인 DS_h의 모든 요소들은 DS_k의 어떤 주변 노드하고 중복 되어 존재 하지 않는다.*

Corollary 1: *DS_k의 다음 단계에서 중복을 점검하기 위해서 $\forall m \le k-2$ 인 DS_m의 어떤 요소도 검사할 필요가 없다.*

그리고 출발 노드 S를 DS_1이라 하고, 상위 레벨로 전개하면, 다음과 같은 법칙을 얻을 수 있다.

Theorem 1: *dBG(d,k)에서 만일 경로가 존재 한다면, 노드 $S \in DS_1$에서 노드 $A_x \in DS_x$ ($\forall x \le k$)로 가는 FFSP는 항상 존재 한다.*

Corollary 2: *$S \in DS_1$ 에서 $A_x \in DS_x$ 로 가는 경로의 길이는 x-1이다.*

FFSP1 알고리즘은 법칙 1의 결과에 따라서 제안한다. 고장 노드가 네트워크에 연결이 되어있어도 모든 경우에서 고장 노드를 포함하지 않는 최단 경로를 찾을 수 있다. 그리고 두 개의 방향(출발 노드 S와 목적 노드 D)에서 멀티 단계로 DS를 확장하게

되면 제안하는 시스템의 성능은 보다 더 개선된다.

그림 2에서 $\forall a_{ip} \in A[i], \forall b_{jk} \in B[j]$일 때, 라인 5와 라인 8의 조건으로 DS의 배열 A와 배열 B의 주변 노드에 어떠한 노드가 존재하는지 알 수 있게 된다. 라인 14의 Expand 함수는 DM M의 다음 레벨을 찾는다. 라인 17에서 duplicate_check(N) 함수는 N의 어떤 요소들이 다른 DS 상위 레벨과 중복되는지를 검사한다. 중복성 검사를 위해서 corollary 1의 결과를 사용한다. FFSP는 각각 a_{ip}에서 s쪽으로, b_{jk} 에서 d 쪽으로 가면서 FFSP를 찾아낸다.

```
1  DECLARE two array A[n] and B[n] of DS type

2  SET A[0] to s
3  SET B[0] to d
4  SET i and j to 0
5  WHILE (aip is not neighbor of bjk)
6   CALL Expand function with A[i] and return value to A[i+1]
7   i = i+1
8   IF (aip is neighbor of bjk) THEN
9   Exit while loop
10  ENDIF
11  CALL Expand function with B[j] and return value to B[j+1]
12  j=j+1
13  ENDWHILE

14 DEFINE Expand function of a DS M
15  DECLARE array N of DS type

16  SET N to DS of all neighbors of each element in M
17  CALL duplicate_check(N)
18  RETURN(N)
19 ENDDEFINE
```

그림 2. FFSP1 알고리즘

Breadth First Search(BFS) 알고리즘[17]과 FFSP1 알고리즘은 비교하면 비슷하게 보일 수 있지만 몇 가지 다른 점이 있다. 첫째, FFSP1은 BFS 알고리즘의 트리 방식이 아닌 새로운 DS를 기반으로 만들어진다. 둘째, FFSP1은 corollary 1을 적용하여 두 방향에서 DS를 확장하기 때문에 한 방향으로 확장하는 Breadth First Search 알고리즘에 보다 더 좋은 성능을 나타낸다.

예제 **2:** dBG(3,5)에서 출발지는 {10000}, 목적지는 {01021}, 고장 노드는 {00102}일 때 FFSP를 찾는 예제는 FFSP1을 적용하면 다음과 같은 결과 결과를 얻을 수 있다.

A_1 = {10000}
B_1 = {01021}
A_2 = {00000, 00001, 00002, 01000, 11000, 21000}
B_2 = {10210, 10211, 10212, 00102, 10102, 20102}.

여기서 {00102}는 고장 노드이다. 그래서 B_2는 {10210, 10211, 10212, 10102, 20102}가 된다.
A_3 = {20000, 00010, 00011, 00012, 00020, 00021, 00022, 10001, 10002, 00100, 10100, 20100, 01100, 11100, 21100, 02100, 12100, 22100}

여기서 A_3과 B_2에 있는 02100 과 10210은 첫 번째 이웃이 된다. FFSP는 02100 에서 10000 쪽으로 그리고 10210 에서 01021 쪽으로 반대쪽으로 가면서 찾게 된다. 따라서 FFSP는 10000 → 21000 → 02100 → 10210 → 01021이 된다.

예제 2에서 FFSP1은 고장 노드가 없을 경우에는 두 개의 최단 경로를 갖게 된다. 두 개의 최단 경로는 10000 → 21000 → 02100 → 10210 → 01021 과 10000 → 00001 → 00010 → 00102 → 01021 이다. 여기서 고장 노드가 00102 이기 때문에 최단 경로는 10000 → 21000 → 02100 → 10210 → 01021이 된다. 네트워크가 커질 경우 FFSP1을 이용하여 더 많은 최단 경로를 준비시킬 수 있으며 FFSP는 그 경로들 가운데서 선택 할 수 있다.

2. FFSP(Fault Free Shortest Path) 2

예제 2에서 보면 A_2에 속하는 00000, 00002, 01000, 11000과 같은 요소들은 FFSP에서 사용되지 않는다는 것도 알 수 있다. 그러한 불필요한 요소들을 A_3에서 제거한다면 A_3의 크기가 줄어들게 되어 다음 단계로 확장하는 처리를 줄일 수 있게 되므로 제안한 알고리즘의 성능을 높일 수가 있다. 이것이 FFSP2 알고리즘의 핵심이다. 두 번째 알고리즘을 설명하기 위하여 필요한 몇 가지 정의와 표기 방식은 다음과 같다.

Definition 2: *출발지로부터 지정된 목적지로 가는 경로 중에서 어떠한 요소를 통과하여 짧은 경로가 형성이 된다면 그 요소는 "주 요소" 이다.*

예제 **3**: 위의 예제 2로부터 고장 노드가 없을 경우 10000 → 21000 → 02100 → 10210 → 01021

와 10000 → 00001 → 00010 → 00102 → 01021 의 두 가지 최단 경로를 얻을 수 있었다. 따라서 00001과 21000이 A_2의 "주 요소"가 된다. 왜냐하면 그 두 요소가 출발지에서 목적지로 가는 최단 경로를 구성하기 때문이다.

여기서 DS_k에서 어떻게 주 요소를 가려낼 수 있는지에 대한 문제가 생기게 된다. 논문 [2]에서 보면, case R_1LR_2 (type 1), case L_1RL_2 (type 1), case RL1 (type 2 RL, $l \geq r$), case RL2 (type 2 RL, $r \geq l$), case LR1 (type 2 LR, $r \geq l$), case LR2 (type 2 LR, $l \geq r$), case R (type 3), case L (type 3) 의 여덟 가지 최단 경로가 존재한다.

만약 주 요소를 찾기 위해서 DS의 모든 요소에 여덟 가지 경우를 모두 적용하기에는 몇 가지 결점이 있게 된다. 첫째, 계산하기에 너무 많은 비용이 든다. 둘째, DS의 크기가 커질수록 보다 많은 다른 경로를 준비할 수 있지만 비용 때문에 DS 의 크기를 줄이게 되면 고장 포용 능력도 줄어들게 된다. 따라서 다음과 같은 법칙을 제안한다.

Theorem 2: *만약 맨 좌측이나 우측의 1비트 주소가 다른 요소가 있다면, 맨 왼쪽 비트가 다른 경우 RL2와 R, 맨 오른쪽 비트가 다른 경우 LR2와 L 의 경우에서 목적지로 향하기 위한 주 요소는 단거리 경로를 가지는 요소이다. (이 경우 몇 개의 요소가 같은 경로 길이를 갖게 되면 랜덤 방식을 적용해서 그 요소들 가운데서 주 요소를 찾게 된다.)*

예제 **4**: 예제 2에서 목적지 D 01021로 가는 노드 A 01000, B 11000, C 21000의 주 특징에 대하여 조사를 하면, 노드 A, B, C는 맨 왼쪽 비트가 다르다. 그렇기 때문에, 최단 경로는 type RL2 와 R를 적용했다.

- R 타입을 적용 하면: 최대 같은 문자열은 A 01000 와 D 01021 사이에는 0개 이다. B 11000 와 D 01021 사이는 1개 이다. 그리고 C 21000 와 D 01021 사이는 2개이다. 따라서 노드 C의 경우에서 최소 경로 길이는 3이다.
- RL2 타입을 적용하면: 최대 같은 문자열은 A 01000 와 D 01021 사이에는 1이고 경로 길이는 6이 된다. B 11000 와 D 01021 사이에는 1이고 경로 길이는 7이 된다. C 21000와 D 01021 사이에는 2가 되고 경로 길이는 R 타입과 같다. 따라서 노드 C의 경우 최소 경로 길이는 3이다.

그러므로 최소 경로 길이가 3이고 주 요소는 C 가 된다.

Corollary 3: *주 요소를 결정하기 위해서 법칙 2를 적용할 때 DS_{m+1} 의 최대 요소는 2p 이다. 여기서 p는 DS_m 의 총 요소이다.*

FFSP1의 결과와 법칙 2에 의한 FFSP2 알고리즘은 그림 3과 같다. $\forall a_{ip} \in A[i], \forall b_{jk} \in B[j]$ 일 때 라인 5와 라인 8의 조건은, DS 의 A 집합과 DS의 B 집합의 이웃 노드가 존재하는지 아닌지 알 수 있다. 라인 14의 SPD(M) 함수는 앞서서 설명한 법칙 2에 의해서 M의 주 요소가 아닌 것을 제거하고, DS M의 다음 단계를 찾는다. 라인 19와 라인23의

```
1   DECLARE two array A[n] and B[n] of DS type
2   SET A[0] to s
3   SET B[0]  to d
4   SET i and j to 0
5   WHILE (aip is not neighbor of bjk)
6     CALL SPD function with A[i] and return value to A[i+1]
7     i = i+1
8     IF (aip is neighbor of bjk) THEN
9       EXIT while loop
10    ENDIF
11    CALL SPD function with B[j] and return value to B[j+1]
12    j=j+1
13  ENDWHILE

14  DEFINE  SPD function of a DS M
15   DECLARE array N of DS type
16   SET N to DS of all neighbors of each element in M
17   WHILE
        (there exist set of elements T which differ in 1 bit in N)
18     IF (T is leftmostbit difference) THEN
19       CALL Pathlength function type RL2 and R
20       CALL
         Eliminate function to eliminate non-dominant elements
21     ENDIF
22     IF (T is rightmostbit difference)
23       CALL Pathlength function type LR2 and L
24       CALL
         Eliminate function to eliminate non-dominant elements
25     ENDIF
26   ENDWHILE
27   CALL duplicate_check(N)
28   RETURN(N)
29  ENDDEFINE
```

그림 3. FFSP2 알고리즘

경로 길이 함수는 목적지로 향하는 각각의 요소의 경로 길이를 점검한다. 라인 20과 라인 24의 제거 함수는 다른 것에 비하여 긴 경로 길이를 가지는 요소를 제거 한다. 라인 27의 duplicate_check(N) 함수는 다른 상위 DS의 어떤 요소가 중복이 되는지 점검한다.

중복 점검을 위해서 추론 1의 결과를 사용한다. 그리고 a_{ip} 에서 s, b_{jk} 에서 d 쪽으로 각각 뒤쪽으로 가면서 FFSP를 찾아낸다.

예제 5: 예제 2에서 FFSP2 알고리즘을 적용하여 FFSP를 찾으면 다음과 같은 결과가 나온다.

A_1 = {10000}

B_1 = {01021}

A_2 = {00001, 21000}

B_2 = {10210, 00102}. 여기서, 00102 는 고장 노드 따라서 B_2 는 {10210}이 된다.

A_3 = {00010, 10000, 10001, 02100}. 여기서, 노드 10000은 A_1에 있는 10000과 일치하게 된다. 따라서 A_3 는 {00010, 10001, 02100}이 된다.

그러면 A_3 와 B_2 의 02100 와 10210과 서로 첫 번째 이웃 노드가 된다. FFSP는 각각 02100 에서 10000 쪽으로 그리고 10210 에서 01021쪽으로 가면서 찾게 된다. 따라서 FFSP는 10000 → 21000 → 02100 → 10210 → 01021이 된다.

Ⅳ. 성능 평가

제안한 알고리즘을 다른 알고리즘과 비교하고 분석하는 방법에는 여러 가지가 있다. 그 중 하나는 경로의 길이는 평가하는 방법이다. 왜냐 하면 알고리즘에 따라서 라우팅 프로토콜이 긴 경로를 따라서 더 많은 내부 트래픽(traffic)을 발생시키는 원인이 되며, 그것 때문에 경로의 길이를 관리하는데 더 많은 것을 집중하게 된다. 또 다른 하나의 평가 방법은 제안한 두 개의 알고리즘 실행을 비교하는 쓰인 DS 크기를 비교하는 것이다. 이러한 평가 방법을 일정한 네트워크에서 시뮬레이션 했으며 실험을 위해 노드의 고장을 랜덤으로 발생시켰다. 또한 네트워크 안에서 모든 노드들이 고장을 일어날 확률은 같다고 가정했다.

1. 경로의 길이(홉(hop)의 평균치)

UdBG 네트워크의 경로의 길이는 논문 [8]에서 처음으로 최대 바운드(upper bound)와 최소 바운드(lower bound)를 이끌어내서 계산했다. 최근에 Z. Feng과 Yang[9]이 원래의 공식[15]을 바탕으로 BdBG 네트워크에서 라우팅 성능을 계산해 내었다.

$$Mean\ path\ length = \frac{Total\ internal\ traffic}{Total\ external\ traffic} \quad (1)$$

하지만 FFSP 라우팅에 관한 논문은 아직까지 없었기 때문에 고장이 존재하는 BdBG 망에서의 경로의 길이를 계산하는 것은 하지 못했다. 그러나 위의 식 (1)을 이용하여 고장이 존재하는 곳에서의 경로의 길이를 계산하였다. 시뮬레이션 프로그램을 이용하여 네트워크의 모든 링크에서의 총 내부 트래픽과 출발지에서 발생하는 총 외부 트래픽에 대하여 계산 하여 경로의 길이를 측정하여 경로 길이 값을 얻었다.

표 1. dBG(d,k)에서 각 알고리즘들의 경로길이 비교

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 (1node fail)	FFSP2 (1node fail)	FFSP1 (2node fail)	FFSP2 (2node 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	246	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**

표 1은 SCP, RFR, NSC, PMC, FFSP1, FFSP2 알고리즘 등 6가지의 알고리즘을 이용하여 경로의 길이를 측정한 결과이다. SCP, RFR, NSC, PMC 알고리즘은 작은 네트워크와 적은 입출력 개수(degree)의 dBG에서는 단거리 경로로 이동이 가능했다. 그 중 작은 네트워크 특징에서는 SCP 알고리즘이 가장 좋은 결과가 나왔고, 큰 네트워크에서는 NSC가 다른 네 가지 알고리즘 보다 가장 좋은 결과가 나왔다. 하지만 이 경우 경로의 길이가 짧았기 때문에 가능 했다. 제안한 두 개의 알고리즘은 다른 네 개의 알고리즘과 비교하여 현저히 좋은 결과가

나왔다. 다른 알고리즘에 비교하여 작은 경로의 길이가 항상 나왔다. 하지만 중요한 점은 고장이 없을 때뿐 아니라 고장이 존재하는 네트워크에서도 같은 경로의 길이로 도착 할 수 있었다. FFSP2는 크기가 작은 네트워크와 적은 입출력 개수 dBG에서는 FFSP1과 같은 결과를 가져왔으나 크기가 큰 네트워크와 입출력 개수가 많은 dBG 에서는 FFSP1보다 약간 긴 경로 길이의 결과가 나왔다.

2. DS 크기

출발 노드와 목적 노드의 총 DS 크기는 출발지와 목적지 노드까지 모든 DS 요소의 총수를 나타낸다. 네트워크에서 모든 출발지와 목적지의 모든 DS 크기들의 평균값을 "DS 크기"라고 정의 한다. 수식적으로 다음과 같이 주어진다.

$$\overline{S} = \frac{\sum_{i=1}^{M} S_i}{M} \tag{2}$$

여기에서, S_i 는 특정한 출발지와 목적지 DS의 총 크기 이다(경로 i 의 모든 DS에 있는 총 노드의 수). M은 네트워크에서 고장이 포함 되지 않은 출발-목적 쌍의 총수 이다.

표 2. 고장 노드가 있는 경우에서의 FFSP1과 FFSP2의 DS 크기 비교

d,k	No. of nodes	DS mean sizes					
		FFSP1 (no node failure)	FFSP2 (no node failure)	Reducing percentage	FFSP1 (1 node failure)	FFSP2 (1 node failure)	Reducing percentage
2,2	4	2.333	2.333	0.00%	2	2	0.00%
2,3	8	3.893	3.286	15.59%	3.571	3.238	9.33%
3,2	9	3.917	2.833	27.67%	3.536	2.857	19.20%
2,4	16	6.133	4.308	29.76%	6.133	4.448	27.47%
2,5	32	10.065	5.95	40.88%	10.045	6.056	39.71%
4,2	16	5.7	3.1	45.61%	5.448	3.105	43.01%
3,3	27	7.915	4.154	47.52%	7.695	4.191	45.54%
2,6	64	16.713	8.326	50.18%	16.574	8.393	49.36%
5,2	25	7.567	3.267	56.83%	7.268	3.268	55.04%
4,3	64	12.128	4.623	61.88%	12.018	4.658	61.24%
6,2	36	9.476	3.381	64.32%	9.299	3.382	63.63%
3,4	81	17.096	6.072	64.48%	16.946	6.093	64.04%
5,3	125	16.341	4.942	69.76%	16.236	4.947	69.53%
6,3	216	20.523	5.14	74.95%	20.454	5.142	74.86%
3,5	243	36.461	9.043	75.20%	36.279	9.049	75.06%
4,4	256	36.409	7.121	80.44%	36.439	7.137	80.41%
3,6	729	76.055	13.04	82.85%	75.968	13.043	82.83%
4,5	1024	81.288	10.688	86.85%	81.215	10.694	86.83%
5,4	625	64.504	7.779	87.94%	64.38	7.781	87.91%

또한 FFSP를 찾기 위해 더 많은 노드를 검사하는 것은 계산 비용을 그 만큼 더 들어감을 의미한다. 표 2는 고장 노드가 없을 조건이나, 한 개 혹은 두 개의 노드가 결함이 있을 경우의 FFSP1과 FFSP2의 크기를 나타낸다. 크기가 큰 네트워크와 높은 입출력 개수(degree)를 가지고 있을 경우 FFSP1보다 FFSP2를 적용할 때 DS의 크기가 현저히 줄어드는 것을 확인할 수 있었다. FFSP2 알고리즘은 dBG(3,6)의 예를 들어 고장 노드가 없을 경우에 FFSP1 알고리즘에 비하여 DS 크기가 82.85% 감소되는 것을 확인 하였다. 그것은 FFSP2가 다음 단계의 DS로 확장될 때 FFSP1 보다 적은 비용으로도 중복 검사가 가능하다는 것을 의미한다. 주 점검 비용은 중복 점검 비용과 확장하는 것 보다 적다. 이것은 FFSP1의 경우 큰 DS 크기를 가지고 비교하는 만면 FFSP2의 경우 주 점검은 적은 DS 크기로 실행이 되기 때문에 적은 비용이 들게 된다.

마지막으로 제안한 알고리즘의 시간 복잡성에 대해서 설명하고자 한다. A. Sengupta[16]는 자신의 논문에서 dBG(d,k)가 d-1접속을 갖는 것을 보여줬다. 이것은 결함 노드가 최대 d-1 이라도 여전히 네트워크에 연결되어 있다는 것을 의미한다. 따라서 제안하는 논문에서 시간 복잡성은 결함의 수가 최대 d-1이라고 가정을 기반으로 한다. 앞서서 설명한 네트워크의 최대 지름은 k이다. dBG(d,k)에서 2d 주변 노드와 최대 d-1 결함 노드를 갖는다. 따라서 최악의 경우 출발지에서 도착지까지의 길이가 k 인 하나의 경로만 존재할 수 있게 된다. d-1의 고장 노드가 있는 dBG(d,k)에서 최대 경로 길이는 K 이다. 다음과 같은 경우를 갖는다.

- FFSP1에서, 두 번째 단계의 DS 복잡도는 $O(2d)$, 세 번째 단계 복잡도는 $O(2d(2d-1)) \approx O(4d^2)$, 네 번째는 $O(2d(2d-1)^2) \approx O(8d^3)$ 이 된다. 그러므로 FFSP1의 시간 복잡성의 복잡도는 $O((2d)^n)$ 이다. 최악의 경우 FFSP1의 시간 복잡성은 $O((2d)^{\frac{k}{2}+1})$ (k=2h), 혹은 $O((2d)^{\frac{k}{2}+1})$ (k=2h+1) 이고 출발지에서 목적지까지의 경로 길이는 k 이다.
- FFSP2의 시간 계산은 두 가지 부분으로 나눌 수가 있다. 하나는 다음 레벨로 확장을 위해서 중복 검사와 예를 들어 DS A[m]과 B[q] 사이의 이웃 노드를 검사를 계산하는 시간 계산이다. 이 부분은 FFSP1과 같다. 또 다른 하나는 주 요소를 찾아내는 계산을 하는 시간이다. 그

러므로 두 번째 단계의 복잡도는 $0(2+2d) \approx 0(2d)$, 세 번째 단계 복잡도는 $0(4+4d) \approx 0(4d)$, 네 번째 단계는 $0(8+8d) \approx 0(8d)$ 가 된다. 따라서 FFSP2의 시간의 복잡도는 $0(2^n d)$가 된다. 최악의 경우 FFSP2의 시간 복잡성은 $O(2^{\frac{k}{2}+1} d)$ (k=2h), 혹은 $O(2^{\frac{k}{2}+1} d)$ (k=2h+1) 이고 출발지에서 목적지까지의 경로 길이는 k 이다.

V. 결론 및 향후 연구

본 논문에선 BdBG에서의 고장 노드를 포함 하지 않는 최단 경로 라우팅 알고리즘과 새로운 개념을 제안했다. 제안한 알고리즘은 dBG를 기반으로 하고 있는 실제 네트워크를 위하여 최단 거리 경로 특징과 고장 포용(fault tolerant) 특징을 가지고 있다. 그리고 경로 길이와 DS 크기를 정의하여 다른 알고리즘과의 비교를 통하여 제안한 알고리즘의 성능을 분석하였다. 시뮬레이션 결과 FFSP1 알고리즘은 적은 입출력 개수와 노드의 수가 비교적 적은 네트워크에서는 고장 노드의 유무에 상관없이 최단 경로를 찾을 수 있었고, FFSP2 알고리즘의 경우 높은 입출력 개수와 노드의 수가 많은 실제 네트워크에서 사용할 수 있는 적합한 결과를 얻었다.

그 뿐만 아니라 제안한 멀티 레벨 DS 개념은 그래프의 특징이 아닌 이웃된 노드 개념으로 제안했기 때문에 제안한 FFSP 알고리즘은 uBG 뿐만 아니라 Kautz, Hypercube, Toroid 등 현재 상호 연결 네트워크의 구조를 사용하고 있는 클러스터 기반의 다른 구조의 네트워크에서도 적용 및 포함이 가능하다. 또한 멀티 레벨 DS는 하나의 DS 레벨의 노드들을 이용함으로 FFSP 유니캐스팅(FFSP unicasting), FFSP 멀티캐스팅(FFSP multicasting), 프로드캐스팅(broadcasting) 등 모든 네트워크로 확장이 가능하다.

따라서 향후 연구로는 멀티 레벨 DS와 주 요소 개념을 다른 네트워크 구조에 적용하여 평가할 예정이다. 또한 하나의 매카니즘으로 서로 다른 종류의 네트워크 구조에서 사용이 가능하며, 유니캐스팅/멀티캐스팅 및 브로드캐스팅이 가능한 FFSP 알고리즘에 대하여 연구할 예정이다.

부록: Lemmas와 Theorems 증명

Lemma 1 증명: 정의 1에 의하여 DS_m 은 결함이 있는 어떠한 노드도 제외가 되므로 고장 노드는 포함이 될 수 없다.

Lemma 2 증명: DS_1과 DS_2 에는 고장 노드를 제외하고 DS_1 의 이웃 노드를 포함하는 것은 확실하다. DS_1, DS_2, DS_3 에는 고장 노드를 제외하고 DS_2의 모든 이웃노드 들이 포함이 되어있다. 따라서 Lemma 2는 k=1,2 일 때 증명이 된다. 만약 Lemma 2가 k = m 일 때까지 증명이 되었다고 가정하면, m+1 일 때 옳음을 증명하면 된다. 만약 m+1 일 때 Lemma 2가 틀렸다고 가정하면, DS_{m+1} 에 속하는 요소 B의 한 이웃 노드 A 가 존재하며, 노드 A는 i<m 일 때 DS_i 에 속한다는 것을 의미한다. 왜냐하면 k=m 일 때 까지 옳다고 증명이 되었기 때문이다. 그렇다면 노드 A의 이웃 노드들은 고장 노드를 제외하고 DS_{i-1}, DS_i , DS_{i+1} 에 있게 된다. 그러므로 DS_{i-1}, DS_i, DS_{i+1} 에 속하는 요소 B' 가 존재 하고 B'=B 가 된다. 이것은 정의 1과 모순이 된다. 따라서 Lemma 2는 m+1 일 때도 성립된다. 이런 귀납적인 방법에 의하여 Lemma 2는 증명된다.

Lemma 3 증명: DS_k의 속하는 요소 B의 주변 노드A 하고 DS_m (m = k-2)의 요소 A' 와 중복이 된다고 가정하면, Lemma 2에 의하여, A'의 모든 주변 노드는 DS_{m-1}, DS_m, DS_{m+1}에 존재하게 된다. 따라서 m-1 이나 m 혹은 m+1 레벨에 있는 A'의 주변 노드 B'가 존재하게 되고, B'=B 가 된다. 이것은 정의 1과 모순 된다. 그러므로 Lemma 3이 증명된다.

Theorem 1 증명 : 귀납적인 방법을 이용하여 이 법칙을 증명하면 다음과 같다. x=1,2 일 때, 법칙 1은 옳다. 법칙1 이 m (m = k)일 때까지 옳다고 추측할 수 있다. 여기서 m+1까지 옳다고 증명한다. S로부터 A_{m+1}까지의 경로가 FFSP가 아니라고 가정한다. 그러면 다음과 같은 경우가 생긴다.

- $A_p=A_{m+1}$, p<m+1, DS_p 에 속하는 A_p가 존재한다. 이것은 정의 1과 맞지 않는다.
- FFSP 가 존재한다면 $S \rightarrow B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_k \rightarrow \dots \rightarrow B_2 \rightarrow \dots \rightarrow A_{m+1}$ 과 B_k, B_{K+1}, ... B_z 는

어떤 DS_i ($\forall i$ = m+1)에도 속하지 않는다. 왜냐하면 $B_{k-1} \in DS_j$ (j = m+1)이기 때문이다. Lemma 2에 따라서 B_{k-1} 의 모든 주변 노드는 결함 노드를 제외하고 DS_{j-1} , DS_j, DS_{j+1},에 있다. 그렇기 때문에 B_k 는 결함 노드가 되어 버린다.

따라서, Theorem 1 은 m+1일 때 옳게 된다. 이런 귀납적인 방법에 의하여 Theorem 1은 증명된다.

Corollary 2 증명: S → A2 ($A_2 \in DS_2$, A_2 는 S의 이웃 노드), A_2 → A_3 ($A_3 \in DS_3$, A_3 는 A_2의 이웃 노드)이런 식으로 해서 S로부터 A_x 까지 FFSP를 얻을 수 있다. 그러므로 경로길이는 x-1 이다.

FFSP 1 증명: 경로 $s \rightarrow \ldots \rightarrow a_{ip} \rightarrow b_{jk} \rightarrow \ldots \rightarrow d$ 가 FFSP 가 아니라고 가정하면, 다음과 같은 경우가 생기게 된다.

- FFSP $s \rightarrow \ldots \rightarrow a_{ip'} \rightarrow b_{jk'} \rightarrow \ldots \rightarrow d$ (i' = i, j' = j)가 존재한다. a_{ip} 와 b_{jk} 는 A와 B의 DS 사이의 첫 번째 주변 노드이기 때문에 이것은 위쪽의 가정과 모순이 되어버린다.
- FFSP $s \rightarrow \ldots \rightarrow a_{ip'} \rightarrow c_1 \rightarrow \ldots \rightarrow c_m \rightarrow b_{jk'} \rightarrow \ldots \rightarrow d$ (i' < i, j' < j)가 존재하고, c_1, c_2, ... c_m은 A_p 혹은 B_q (p ≤i, q≤j)의 어떤 DS에도 속하지 않는다. $a_{i'p'} \in A_{i'}$ 와 lemma 2에 의하여 모든 $a_{i'p'}$의 주변 노드는 결함 노드를 제외하고 $A_{i'-1}$, $A_{i'}$,d $A_{i'+1}$ 에 있다. 그러므로 c_1은 결함 노드가 된다.

따라서 FFSP1이 증명 된다.

theorem 2 증명: 논문 [2]에서 보듯이, 최단 거리를 위해서는 여덟 가지 경우만 있다. 그 중 오직 RL2, R, LR2, L의 네 가지 경우만 출발지가 맨 우측이나 맨 좌측 한 개의 비트가 다를 경우만 여러 다른 경로를 만들 수 있다.

참 고 문 헌

[1] Samantham, R. Maheswara , and D.K. Pradhan, "The De Bruijn Multi-processor Network: A Versatile Parallel Processing and Sorting Network for VLSI," *IEEE Trans. on Comp.*, Vol.38, NO.4, 1989.

[2] Zhen Liu, Ting-Yi Sung, "Routing and Transmitting Problem in de Bruijn Networks" *IEEE Trans. on Comp.,45*,9,1996, pp 1056 1062.

[3] M.A. Sridhar, C.S. Raghavendra, "Fault Tolerant networks Based on the de Bruijn Graph", *IEEE Trans. on Comp.,40, Issue 10,1991* pp 1167 1174.

[4] D. Nadig, S.S. Iyengar, D.N. Jayasimha, "A New Architecture for Distributed Sensor Integration", *IEEE Proceeding of Southeastcon* '93, 4-7 April 1993 pp 8 p.

[5] Alfred V. Aho, Margaret J. Corasick, "Efficient String Matching: An Aid to Bibliographic Search", *Comm. of the ACM,. 18 Issue 6*, June 1975.

[6] R.A. Rowley, B. Bose, "Fault-tolerant ring embedding in de Bruijn networks", *IEEE Trans. on Comp,42*,12,1993 pp 1480 1486.

[7] C. Hyatt, D.P. Agrawal, "Bidirectional versus unidirectional networks: cost/performance trade-offs", *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1995. MASCOTS '95.*, Proceedings of the Third International Workshop on ,Jan. 1995, pp 123 133.

[8] K.N. Sivarajan, R. Ramaswami, "Lightwave networks based on de Bruijn graphs", *IEEE/ACM Trans. on Networking,2,1*,1994 pp 70 79.

[9] Yang, Z. Feng, "DBG MANs and their routing performance", *Comm., IEEE Proc.,147, Issue 1,2000* pp 32 40.

[10] A.H. Esfahanian, G. Zimmerman, "A distributed broadcast algorithm for binaryDe Bruijn networks", *1988. ConferenceProceedings, Seventh Annual International Phoenix Conference on Comp. and Comm.*,March 1988.

[11] R.A. Rowley, B. Bose, "Distributed ring embedding in faulty De Bruijn networks", *IEEE Trans. on Comp.,46,2*,1997 pp 187 190.

[12] D.K. Pradhan and S.M. Reddy, "A fault-tolerant communication architecture for

distributed systems", *IEEE Trans. Comp. 31* (1982), 863.870.

[13] A.H. Esfahanian and S.L. Hakimi, "Fault-tolerant routing in de Bruijn communication networks", *IEEE Trans. Comp. C-34* (1985), 777.788.

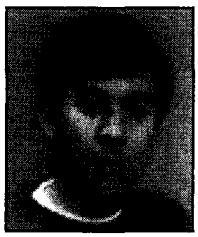
[14] Jyh-Wen Mao and Chang-Biau Yang, "Shortest path routing and fault tolerant routing on de Bruijn networks", *Networks*, Vol. 35, Issue 3, Pages 207-215 2000.

[15] L.Kleinrock, *Communication Nets; Stochastic Message Flow and Delay, McGraw-Hill (New York)*, 1964. Reprinted by Dover Publication, 1972.

[16] A.Sengupta, A.Sen, and S.Bandyopadhyay, "Fault tolerant distributed system design", *IEEE Trans. Circuit Syst.*, Vol. CAS-35, pp. 168-172, Feb. 1988.

[17] S.K. Das, C.C.Y. Chen, "A new parallel algorithm for breadth-first search on interval graphs", *Parallel Processing Symposium, 1992. Proceedings.*, Sixth International ,March 1992

Nguyen Chi Ngoc 준회원

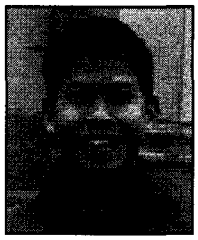


2002년 5월 : Bachelor of Electronics and Telecommunications Engineering University: HoChiMinh City University of Technology, VietNam

2003년 3월 ~ 현재: 경희대학교 컴퓨터공학과 석사과정

<관심분야> routing and fault tolerance on parallel and distributed computing, parallel and distributed architecture, sensor network

Vo Dinh Minh Nhat 준회원



2003년 1월 : Bachelor of Electronics and Telecommunications Engineering University: HoChiMinh City University of Technology, VietNam

2003년 9월 ~ 현재: 경희대학교 컴퓨터공학과 석사과정

<관심분야> Routing and fault tolerance on parallel and distributed computing, parallel and distributed architecture. Speech recognition, face recognition

정 연 일(Yonil Zhung) 준회원

2000년 8월: 경희대학교 컴퓨터 공학과 석사

2003년 2월: 경희대학교 컴퓨터 공학과 박사수료

2003년 3월~ 현재: 경희대학교 컴퓨터공학과 박사과정

<관심분야> 분산 시스템, 미들웨어, 시스템 및 네트워크 보안

이 승 룡(Sungyoung Lee) 정회원

1978년: 고려대학교 재료공학과 졸업

1986년: Illinois Institute of Technology 전산학과 석사

1991년: Illinois Institute of Technology 전산학과 박사

1991년~1993년 Governors State University 조교수

1993년~현재 경희대학교 컴퓨터공학과 교수

<관심분야> 실시간 컴퓨팅, 실시간 미들웨어, 멀티미디어 시스템, 시스템 보안