論文 94-19-7-17

평균 지연 시간의 제약 조건을 갖는 로컬 액세스 컴퓨터 네트워크에서의 링 토폴로지 설계

正會員 李 湧 振* 正會員 金 泰 潤*

Design of Ring Topology for Local Access Computer Networks with mean delay time constraint

Yong-Jin Lee*, Tai-Yun Kim* *Regular Members*

要 約

본 논문은 로컬 액세스 컴퓨터 네트워크를 설계할 때 발생되는 문제의 하나로 네트워크의 평균 지연 시간을 고려한 최소 비용 루프 설계 문제(DMCLP-Delay constrained Minimum Cost Loop Problem)를 다룬다. 이 문제는 종단 사용자의 트래픽 요구량을 만족시키는 링의 집합을 구하는 것으로 목적 함수는 전체 라인 비용을 최소화하는 것이다. 본 논문에서는 하나의 링이 서비스할 수 있는 노드의 수가 제한되어 있으며 동시에 네트워크의 평균 지연 시간이 원하는 시간이내이어야 한다는 제약 조건하에서 이 문제에 대한 2단계-휴리스틱 알고리즘을 제안한다. 이 알고리즘은 기존의 최소 비용 루프 설계(MCLP) 알고리즘에 의한 클러스터와 본 논문에서 제안한 trade-off criterion를 이용하여 유도된다. 실제 시뮬레이션의 결과, 본 논문에서 제안한 알고리즘은 수정된 기존의 MCLP 알고리즘보다 우수한 해를 제공하며 아울러 비교적 짧은 실행 시간을 갖는다.

ABSTRACT

This study deals with the DMCLP(Delay constrained Minimum Cost Loop Problem)-one of problems arising in the design of local access computer networks. The problem consists of finding a set of rings to satisfy the traffic requirements of end user terminals. In the problem, the objective is to minimize the total link cost. This paper presents heuristic algorithm which consists of two phases for this problem, under the constraints that the number of nodes served by a single ring is limited and network mean delay is dropped within the desired time. The algorithm is derived using

*高麗大學校 電算科學科
Dept. of Computer Science, Korea Univ.

論文番號 : 94123
接受日字 : 1994年 5月 4日

the clusters obtained by the existing MCLP(Minimum Cost Loop Problem) algorithms and a trade-off criterion explained in the paper. Actually, simulation results in that the proposed algorithm in this paper produces better solution than the existing MCLP algorithm modified. In addition, the algorithm has the relatively short running time.

I. 서 론

로컬 액세스 컴퓨터 네트워크(local access computer networks)는 종단 사용자와 기간망(backbone) 사이의 트래픽을 전송하는 네트워크로 그 토폴로지(topology)를 구하는 문제는 대개 최소 신장 트리로 구성되는 CMSTP(Capacitated Minimum Spanning Tree Problem)와 링(ring) 또는 루프(loop)로 구성되는 MCLP(Minimum Cost Loop Problem)로 나눌 수 있다[1,4].

본 연구는 이 중에서 네트워크 지연 시간이 고려된 MCLP를 다루고자 하며 이 문제를 DMCLP(Delay constrained MCLP)라고 한다. 기존의 MCLP는 NCP(Network Control Processor) 또는 집중기(concentrator)에 있는 포트에 각 종단 사용자를 링을 사용하여 연결하는 문제로 하나의 링에 둘 수 있는 최대 노드의 수가 제한된다. 따라서, MCLP의 목적은 모든 종단 사용자를 최소 비용으로 연결하는 링의 집합을 구하는 것이다. 이와 유사한 문제로는 수송 분야에서 이용되는 VRP(Vehicle Routing Problem)[11]가 있다.

그러나 기존의 MCLP 또는 VRP를 해결하는 알고리즘[2,3,8]등은 네트워크의 지연 시간을 고려하지 않고, 또한 라인에 대한 용량을 제한하지 않기 때문에 네트워크의 평균 지연 시간을 주어진 값 이내에 해결하면서 최소 비용의 위상 설계를 요구하는 DMCLP에는 그대로 적용될 수 없다.

또한 일반적인 기간망(backbone networks)의 위상 설계에 이용되는 알고리즘들[5,6,7,12]은 하나의 링에 둘 수 있는 노드 수를 제한하지 않기 때문에 기존의 MCLP를 해결하기가 어려우며 따라서 네트워크 지연 시간이 제약 조건으로 추가되는 DMCLP도 해결하기가 어렵다.

기존의 MCLP는 하나의 집중기 또는 NCP와 n개의 종단 사용자로 구성되므로 이들은 그래프 G(V, E)상에 노드로 표현할 수 있다. 이제, 각 노드 i(i=1, ..., n)에서 발생하는 트래픽을 q_i(i=1, ..., n), 센터 노드를 i=0라고 하고, 노드 쌍(i, j)를 연결하는 거리를 d_{ij}, 센터 노드에서 하나의 링에 둘 수 있는 최대 노드수를 MAX라고 하면 이 문제는 최대 MAX개의 종단 사용자 또는 터미날을 하나의 링으로 구성하는 제약 조건하에서 전체 비용을 최소화 하는 링의 집합을 구하는 문제가 된다. 여기에 평균 지연 시간에 대한 제약 조건을 추가하는 문제가 바로 본 연구에서 해결하려고 하는 DMCLP이다.

기존의 MCLP는 NP-hard 문제[9]로 최대 종단 사용자의 수가 25개 이하인 경우의 최적해를 제공하는 알고리즘[3]등 이외에는 대부분의 알고리즘이 휴리스틱에 의존하고 있다. 따라서 네트워크 지연 시간이 추가적인 제약 조건으로 부가되는 DMCLP 역시 NP-hard문제가 된다.

본 연구에서는 계산 시간을 절감하기 위해 기존의 MCLP 알고리즘[2,8]에 의해 생성되는 클러스터의 노드 집합을 이용하고 본 연구에서 제시하는 trade-off 기준에 의해 클러스터 사이의 노드를 교환 또는 이동하여 토폴로지를 변화시켜 나가면서 요구되는 평균 지연 시간이내에서 라인용량을 최적으로 할당함으로써 최소 비용의 링 토폴로지의 집합을 구할 수 있는 2-단계 휴리스틱을 제시한다. 전체 4장으로 구성되는 본 연구는 II장에서 알고리즘의 모델링 및 해법을, III장에서 성능평가 및 분석을 그리고 IV장에서는 결론을 제시한다.

II. 링 토폴로지 설계 알고리즘의 모델링 및 해법

1. 용어, 기호의 정의 및 가정

본 연구에서 제시하는 알고리즘을 DMCLP 알고리즘이라고 명명하고 사용되는 용어및 기호를 아래와 같이 정의한다.

n : 센터 노드를 제외한 전체 노드 수
m : 임의의 토폴로지에 대한 전체 라인 수
lcnt : 해에 포함되는 클러스터의 갯수
kcnt(p) : 클러스터 p에 포함되는 노드의 갯수

(p=1, ..., lcnt)

cost(p) : 클러스터 p에 대응되는 라인 비용 (p=1, ..., lcnt)

q_i : 노드 i에서의 트래픽 요구량 (i=1, ..., n)

node(p) : 클러스터 p에 대응되는 노드 집합

path(p) : node(p)에 대응되는 TSP tour상의 라인 집합

d_{ij} : 노드 쌍(i, j) 사이의 거리 (i=0, 1, ..., n, j=0, 1, ..., n)

D : 거리 매트릭스

d : 라인 용량의 단위 비용

d_k : 임의의 토폴로지에서 라인 k의 순회 거리 (k=1, ..., m)

MAX : 하나의 링에 둘 수 있는 최대 노드 수

sub1 : 임의의 노드 i가 포함되어 있는 클러스터의 색인

sub2 : 임의의 노드 j가 포함되어 있는 클러스터의 색인

ks_{ij} : 노드 교환 휴리스틱에서의 노드 쌍(i, j)의 trade-off (i=1, ..., n ; j=1, ..., n)

ks'_{ij} : 노드 이동 휴리스틱에서의 노드 쌍(i, j)의 trade-off (i=1, ..., n ; j=1, ..., n)

T : 네트워크 평균 지연 시간(계산 값 : 초)

T_k : 임의의 토폴로지에 포함된 라인 k의 평균 지연 시간(초)

γ : 전체 트래픽 비율

$1/\mu$: 평균 메시지 길이(bits/message)

C_k : 임의의 토폴로지에 포함된 라인 k의 용량 (bits/초)

λ_k : 임의의 토폴로지에 포함된 라인 k의 평균 도착율(messages/초)

$link_k$: 임의의 토폴로지에 포함된 라인 k의 부하

Delay : 허용되는 최대 지연시간(초)

D_{cost} : DMCLP 알고리즘의 각 단계에서 얻는 네트워크의 전체 비용

NEW_{cost} : 임의의 토폴로지에 대한 전체 라인 비용

다음으로 본 연구에서 가정한 사항은 아래와 같다.

① 메시지의 도착은 poission 분포를 따른다.

② 메시지의 길이는 지수 분포를 따른다.

③ 각 라인은 무결성(error-free)이다.

④ 경로배정(routing)은 fixed routing을 사용한다.

2. 기존 MCLP 알고리즘

먼저 알고리즘[8]은 아래와 같다.

단계 1 : TSP(Traveling Salesman Problem) tour상의 노드에 대해 센터 노드에서 부터 시계방향으로 1에서 n까지 번호를 부여한다. i=0, opt=∞로 놓고 에지(1, n)으로 에지(0, 1)과 (0, n)을 대치한다.

단계 2 : set i=i+1 ; 노드 i에서 부터 시작하여 TSP tour를 ⌈n/MAX⌉개의 클러스터로 분할한다. 이 때, 각 클러스터는 최대 MAX개의 노드를 갖는다. 각 클러스터의 종점(endpoints)을 센터 노드에 연결하고, 이 비용을 g_i(MAX)라고 놓는다. set opt=Min{opt, g_i(MAX)}

단계 3 : if i=n, stop ;opt는 생성된 n개의 해 중에서 최소값이다.

다음으로 알고리즘[2]는 아래와 같다.

단계 1 : TSP tour상의 노드에 대해 센터 노드에서 부터 시계 방향으로 1에서 n까지 번호를 부여한다. i=0, opt=∞로 놓는다.

단계 2 : set i=i+1 ; TSP tour를 i+1에서 부터 ⌊(n−i)/MAX⌋MAX+i까지 각각 MAX개씩 되도록 클러스터로 분할한다. 이 때, 2개의 추가적인 클러스터가 생성될 수 있는 데, 첫번째 클러스터는 노드 1에서 노드 i까지 이고 노드의 갯수에 따라 마지막 클러스터는 ⌊(n−i)/MAX⌋MAX+i+1에서 n까지이다. 두 번째 클러스터에서 부터 ⌊(n−i)/MAX⌋ 클러스터까지의 종점을 센터 노드에 연결하고, 노드 i와 노드 ⌊(n−i)/MAX⌋MAX+i+1을 센터 노드에 연결한다. 이 비용을 F_i(MAX)라고 놓는다.(노드 1과 n은 이미 TSP에 의해 센터 노드에 연결되어 있다)
set opt=Min{opt, F_i(MAX)} ;

단계 3 : if i=MAX, stop ; opt는 생성된 MAX개의 해 중에서 최소값이다.

기존의 MCLP 알고리즘[2,8]은 평균 지연 시간의 제약 조건을 고려하지 않으며 라인 용량을 할당하지 않는다. 따라서 위 알고리즘들은 단지 클러스터에 대한 링의 집합을 제공한다. 본 논문에서는 알고리즘[2]를 QIOTP 알고리즘으로 알고리즘[8]을 IOTP 알고리즘으로 표현하기로 한다.

3. DMCLP의 모델링

일반적으로 네트워크 평균 지연 시간(T)은 식 (1)과 같이 주어진다[13].

$$T = 1/\gamma \sum_{k=1}^{m} \lambda_k T_k \quad (1)$$

가정에 의해 임의의 라인 k는 독립적인 M/M/1 queue로 간주될 수 있으므로 queueing theory로 부터 라인 k의 평균 지연 시간(T_k)은 식 (2)로 유도된다.

$$T_k = 1/(\mu C_k - \lambda_k) \quad (2)$$

따라서 식 (2)를 식 (1)에 대입하면 식 (3)을 얻을 수 있다.

$$T = 1/\gamma \sum_{k=1}^{m} \lambda_k [1/(\mu C_k - \lambda_k)] \quad (3)$$

다음으로 네트워크의 전체 비용은 선형 함수를 가정하여 식 (4)로 정의한다.

$$D_{cost} = \sum_{k=1}^{m} dC_k d_k \quad (4)$$

이로 부터 로컬 액세스 컴퓨터 네트워크에서 평균 지연 시간을 고려한 DMCLP를 해결하기 위한 모델링은 기존의 기간망 정식화[7]를 확장하여 그림 1과 같이 표현될 수 있다.

문제는 하나의 루프에 둘 수 있는 노드의 갯수가 MAX 이하(식 7)이어야 하고, 네트워크의 평균 지연 시간(T)이 원하는 지연 시간(Delay)보다 작아야 하며(식 6), 특정 라인에 흐르는 평균 flow(λ_k/μ)가 그 라인의 용량보다 작다는 조건(식 5)하에서 전체 비용을 최소로 하는 링 토폴로지의 집합을 구하는 것이다.

4. DMCLP 알고리즘

DMCLP 알고리즘은 기존의 MCLP 알고리즘[2,8]에 의해 얻어진 클러스터의 노드 집합을 기초로 하여, 2단계 trade-off criterion에 의해 해를 개선해 나가는 알고리즘으로 노드 교환 휴리스틱(NEH : Node Exchange Heuristic)과 노드 이동 휴리스틱(NSH : Node Shift Heuristic)의 2-단계 휴리스틱으로 구성되며 2개의 휴리스틱은 평균 지연 시간의 제약 조건을 만족하기 위한 라인 용량 할당 루틴(Mean_Delay( ))을 공통으로 이용한다.

4.1 노드 교환 휴리스틱

이 휴리스틱은 기존의 MCLP 알고리즘에 의해 얻어진 클러스터의 노드 집합을 기초로 하여, trade-off criterion에 의해 서로 다른 클러스터에 있는 노드를 교환하여 해를 개선해 나가는 단계로 노드 i와 노드 j가 동일한 클러스터에 있는 경우에는 노드를 교환해도 동일한 전체 비용을 얻으므로 해에서 배제하기 위해 trade-off(ks_{ij})의 값을 $-\infty$로 놓는다.

노드 교환을 위한 trade-off criterion의 기본 개념은 다음과 같이 설명될 수 있다. 임의의 토폴로지에서 $i \in node(sub1)$, $j \in node(sub2)$, $sub1 \neq sub2$이고 node(sub1)을 TSP tour 상의 순서대로 {..., i−1, i, i+1, ...}, 마찬가지로 node(sub2)를 {..., j−1, j, j+1, ...}이라고 하자.

Given	:	traffic requirement at node $i(q_i)$, distance matrix, D
Objective	:	Minimize $D(A,C) = D_{cost} = \sum_{k \in A} dC_k d_k$
		where the set of arcs A specifies the ring topology
Design Var.	:	Ring topology
Constraints	:	$\lambda_k / \mu \le C_k$ .....(5)
		$T = 1/\gamma \sum_{k=1}^{m} \lambda_k [1/(\mu C_k - \lambda_k)] \le Delay$ .....(6)
		$number(R_j) \le MAX$ .....(7)
		where R_j specifies single ring

그림 1. DMCLP의 모델링

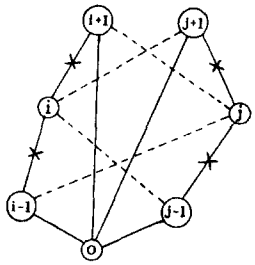

그림 2. 노드 교환의 예

그림 2에서 보듯이, 노드 i와 노드 j를 교환하는 경우 절단되는 라인은 $d_{i,i-1}$, $d_{i,i+1}$, $d_{j-1,j}$, $d_{j,j+1}$이고 추가되는 라인은 $d_{i-1,j}$, $d_{j,i+1}$, $d_{i,j+1}$, $d_{i,j-1}$이다. 따라서 이 경우 ks_{ij} = 절단비용 − 추가비용 = $(d_{i,i-1}+d_{i,i+1}+d_{j,j-1}+d_{j,j+1})-(d_{i-1,j}+d_{j,i+1}+d_{i,j+1}+d_{i,j-1})$이 된다. 여기서 $ks_{ij}>0$의 의미는 노드 i와 j를 교환하면 해가 개선됨을 나타낸다. 물론 노드 i와 노드 j가 속한 노드 집합내의 위치에 따라 이 기준은 달라지게 된다. 식 (8)~식 (20)은 발생 가능한 trade-off criterion으로 사용된 비용 개념은 센터 노드를 기준으로 한 순회 거리(traversal distance)이다.

4.1.1 클러스터 sub1이 노드 i로만 구성된 경우

① 클러스터 sub2가 노드 j로만 구성된 경우

$$ks_{ij}=(2d_{0i}+2d_{0j})-(2d_{0j}+2d_{0i})=0 \quad (8)$$

② 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 첫번째 노드인 경우

$$ks_{ij}=(2d_{0i}+d_{0j}+d_{j,j+1})-(2d_{0j}+d_{0i}+d_{i,j+1}) \quad (9)$$

③ 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 마지막 노드인 경우

$$ks_{ij}=(2d_{0i}+d_{0j}+d_{j,j-1})-(2d_{0j}+d_{0i}+d_{i,j-1}) \quad (10)$$

4.1.2 클러스터 sub1의 노드 갯수가 2 이상이고 노드 i가 첫번째 노드인 경우

① 클러스터 sub2가 노드 j로만 구성된 경우

$$ks_{ij}=(d_{0i}+d_{i,i+1}+2d_{0j})-(d_{0j}+d_{j,i+1}+2d_{0i}) \quad (11)$$

② 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 첫번째 노드인 경우

$$ks_{ij}=(d_{0i}+d_{i,i+1}+d_{0j}+d_{j,j+1})-(d_{0j}+d_{j,i+1}+d_{0i}+d_{i,j+1}) \quad (12)$$

③ 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 마지막 노드인 경우

$$ks_{ij}=(d_{0i}+d_{i,i+1}+d_{0j}+d_{j-1,j})-(d_{0j}+d_{j,i+1}+d_{0i}+d_{i,j-1}) \quad (13)$$

4.1.3 클러스터 sub1의 노드 갯수가 2 이상이고 노드 i가 마지막 노드인 경우

① 클러스터 sub2가 노드 j로만 구성된 경우

$$ks_{ij}=(d_{0i}+d_{i,i-1}+2d_{0j})-(d_{0j}+d_{i-1,j}+2d_{0i}) \quad (14)$$

② 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 첫번째 노드인 경우

$$ks_{ij}=(d_{0i}+d_{i,i-1}+d_{0j}+d_{j,j+1})-(d_{0j}+d_{i-1,j}+d_{0i}+d_{i,j+1}) \quad (15)$$

③ 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 마지막 노드인 경우

$$ks_{ij}=(d_{0i}+d_{i,i-1}+d_{0j}+d_{j-1,j})-(d_{0j}+d_{i-1,j}+d_{0i}+d_{i,j-1}) \quad (16)$$

4.1.4 클러스터 sub1의 노드 갯수가 3 이상이고 노드 i가 중간 노드인 경우

① 클러스터 sub2가 노드 j로만 구성된 경우

$$ks_{ij}=(d_{i-1,i}+d_{i,i+1}+2d_{0j})-(d_{i-1,j}+d_{j,i+1}+2d_{0i}) \quad (17)$$

② 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 첫번째 노드인 경우

$$ks_{ij}=(d_{i-1,i}+d_{i,i+1}+d_{0j}+d_{j,j+1})-(d_{i-1,j}+d_{j,i+1}+d_{0i}+d_{i,j+1}) \quad (18)$$

③ 클러스터 sub2의 노드 갯수가 2 이상이고 노드 j가 마지막 노드인 경우

$$ks_{ij} = (d_{i,i-1} + d_{i,i+1} + d_{0j} + d_{j-1,j}) - (d_{i-1,j} + d_{j,i+1} + d_{0i} + d_{i,j-1}) \quad (19)$$

④클러스터 sub2의 노드 갯수가 3 이상이고 노드 j가 중간 노드인 경우

$$ks_{ij} = (d_{i,i-1} + d_{i,i+1} + d_{j-1,j} + d_{j,j+1}) - (d_{i-1,j} + d_{j,i+1} + d_{i,j+1} + d_{i,j-1}) \quad (20)$$

노드 교환 휴리스틱은 먼저 기존의 MCLP 알고리즘에서 얻은 각 클러스터의 노드 집합에 대해 TSP (Traveling Salesman Problem) 알고리즘을 적용하여 초기 토폴로지를 구하고 포함된 각 클러스터의 노드 집합에 대해 요구되는 지연시간(Delay)을 만족하도록 최적의 라인 용량 할당을 수행한 후 초기 비용(NEW_{cost})을 구한다.

본 논문에서 사용한 TSP 알고리즘[10]은 아래와 같다. 먼저, S는 모든 링크의 집합으로 센터 노드를 제외한 노드 수가 n인 경우, S의 cardinality는 n(n+1)/2이다. 다음으로 T는 tour를 형성하는 S의 n-subset이다. 또한 |x|는 특정 링크 x의 길이이다. 알고리즘의 각 단계는 다음과 같다.

step 1 : 임의의 초기해 T를 생성

step 2 : (a) set i = 1 ;

(b) i번째 step에서 링크 $x_1, \ldots, x_i$를 $y_1, \ldots, y_i$로 대치하는 경우 해를 최대로 개선할 수 있는 x_i를 $T-\{x_1, \ldots, x_{i-1}\}$에서, y_i를 $S-T-\{y_1, \ldots, y_{i-1}\}$에서 선택

(c) if $\sum_{i=1}^{k} g_i = |x_i| - |y_i| \le 0$, for all k, goto step 3 ;

else, set i = i + 1 ; goto step 2(b) ;

step 3 : for i = k, 만일 최대 $\sum_{i=1}^{k} g_i > 0$가 얻어지면 $x_1, \ldots, x_k$를 $y_1, \ldots, y_k$로 대치하고 새로운 T를 저장한 후 goto step 2 ;

if $\sum_{i=1}^{k} g_i \le 0$, goto step 4 ;

step 4 : step 1부터 반복

위 알고리즘의 시간 복잡도는 $O(n^2)$이고 노드 수가 110개 까지 최적해를 제공한다. 다음으로 서로 다른 클러스터의 노드 집합에 소속된 노드 쌍(i, j)(i < j)에 대해, 식 (8)~식 (20)을 이용하여 ks_{ij}를 구하고 (동일한 클러스터의 노드 집합에 소속된 노드쌍(i, j)에 대해서는 $ks_{ij} = -\infty$로 놓는다) 그 값이 양의 최대값을 갖는 노드 쌍(i, j)부터 시작하여 노드 i와 노드 j를 교환하면 새로운 클러스터의 노드 집합을 얻게 된다. 이 변화된 두 개의 클러스터의 노드 집합에 대해 TSP 알고리즘을 적용하여 TSP tour를 구한다. 여기서 두 개의 클러스터의 노드 집합에만 TSP 알고리즘을 적용하는 이유는 나머지 클러스터의 노드들은 이미 TSP 경로상에 존재하기 때문이다.

또한 두 개의 클러스터의 노드 집합에 대응되는 TSP tour상의 라인들에 대해 fixed routing[14]을 적용하여 각 라인상의 평균 도착율(λ_k)을 구한 후 각 라인에 대한 용량(C_k)을 λ_k/μ를 만족하는 최대값으로 일시 할당한다. 이를 기초로 식 (3)을 이용하여 네트워크의 평균 지연 시간(T)을 계산한다. 다음으로 식 (4)를 이용하여 전체 비용을 계산하고 계산된 평균 지연 시간(T)을 이용하여 라인 용량(C_k)을 최적 할당(optimal allocation)하며 그 식은 다음과 같이 주어진다[1].

$$C_k = \lambda_k/\mu \left[1 + 1/(\gamma \cdot T) \sum_{j=1}^{m} \sqrt{\lambda_j d_j} / \sqrt{\lambda_k d_k}\right] \quad (21)$$

또한 라인 k에 대한 라인 부하를 다음 식으로 정의한다.

$$link_k = \lambda_k / C_k \quad (22)$$

계산된 평균 지연 시간(T)이 원하는 지연 시간(Delay)보다 큰 경우에는 라인 부하가 가장 큰 라인의 용량을 한 단위씩 증가시켜 T의 값을 감소시키고, 작은 경우에는 라인 부하가 제일 작은 라인의 용량을 한 단위씩 감소시켜 T의 값을 증가시킨다. 물론 이 경우 라인 용량은 라인의 평균 flow(λ_k/μ)보다 감소될 수 없다. 만일 감소시키려는 라인 용량이 평균 flow보다 작아지는 경우에는 라인 부하가 그 다음으로 작은 라인 용량을 감소시킨다. 이런 식으로 계속하면 T가 Delay에 최근접할 때의 비용(D_{cost})을 얻을 수 있다.

다음으로 D_{cost}와 NEW_{cost}를 비교하여 D_{cost}가 NEW_{cost}보다 작은 경우에는 $D_{cost} \leftarrow NEW_{cost}$로 대치하고 해당되는 $ks_{ij} \leftarrow -\infty$로 놓고 ks_{ij} 매트릭스를 다시 계산한 후 위의 전 과정을 반복하고, 그렇지 않은 경우에는 해당되는 $ks_{ij} \leftarrow -\infty$로 놓은 후 이전의 NEW_{cost}에 대응되는 ks_{ij}중에서 양의 최대 값을 갖는 노드 쌍(i, j)에 대해 위의 전 과정을 반복한다. 즉, 전체 비용이 개선된 경우에는 새로운 토폴로지에 대해 ks_{ij} 매트릭스를 다시 계산하고 그렇지 않은 경우에는 ks_{ij}

```
휴리스틱: Node Exchange Heuristic(NEH)
input:    기존 MCLP 알고리즘에서 얻은 lcnt,kcnt(p),node(p)
output:   NEH에서 얻는 NEW_cost,lcnt,kcnt(p),node(p),cost(p)
variable: flag  /* flag = 1:(초기), flag = 0:(초기가 아닐 때) unit:단위 라인용량  */
          temp: TSP tour상의 라인 집합(path(p))을 임시 저장하기 위한 vector
process:
   step 1:  /* 초기화 */
        ①  flag = 1;
           for node(p) (p=1,2,...,lcnt),
             TSP 알고리즘을 적용하여 path(p) 구함;
        ②  path(p)(p=1,2,...,lcnt)를 이용하여
              Mean_delay();     /* 라인용량 할당 및 비용계산 루틴으로 제어 이전 */
        ③  NEW_cost  ← D_cost;       /* 초기 전체 비용 */

   step 2: /* trade-off criterion(ks_ij) 계산 */
        ①  노드 쌍(i,j)(i<j,∀(i,j))에 대해,
             if(i∈node(p)) sub1 ← p;  if(j∈node(p)) sub2 ← p;
        ②  if(sub1 != sub2)
                    식(8)~식(20)을 이용하여 ks_ij 계산;
           else     ks_ij ← -∞;
        ③  if (ks_ij < 0) ∀(i,j),    알고리즘 종료

   step 3: /* 새로운 클러스터의 생성 (노드 교환) */
           while( ks_ij >  0){
             ①  가장 큰 ks_ij 값을 갖는 노드 쌍(i,j)에 대해,          /* 노드 교환 */
                   if(i∈node(p)) sub1 ← p;  if(j∈node(p)) sub2 ← p;
                     node(sub1)←node(sub1)∪j; node(sub1)←node(sub1)-i;
                     node(sub2)←node(sub2)∪i; node(sub2)←node(sub2)-j;
             ②  for node(sub1),node(sub2),
                    TSP 알고리즘을 적용하여 path(sub1),path(sub2) 구함;
             ③  path(sub1)과 path(sub2)를 이용하여
                  flag = 0;
                  Mean_Delay();          /* 라인용량 할당 및 비용계산 루틴으로 */
             ④  /* 전체 비용 비교 */
               if (D_cost ≥ NEW_cost){        /* 노드 환원 */
                   node(sub1)←node(sub1)∪i; node(sub1)←node(sub1)-j;
                   node(sub2)←node(sub2)∪j; node(sub2)←node(sub2)-i;
                   ks_ij ← -∞;   repeat step 3;}
               else {
                       NEW_cost ← D_cost;   /* 절감된 비용을 새로운 전체 비용으로 놓음 */
                       ks_ij  ← -∞;    go to step 2; }
           };

/* 라인 용량 할당,평균 지연 시간,전체 비용 계산 루틴 */
function: Mean_Delay()
{
   step 1:  /* 트래픽에 따른 최대 라인 용량 임시 할당(초기화) */
               if(flag==1) temp ← ∀k ∈ path(p),(p=1,2..,lcnt)
               else        temp ← ∀k ∈ path(p),(p=sub1,sub2)

               for temp,{
               C_k ← ∑q_i[for i∈node(p) corresponding to temp];
               fixed routing에 의해 λ_k 구함;}
```

```
step 2:  /* 평균 지연 시간(T) 계산 */
             for temp,
             T ← 1/γ Σk λk[1 /(μCk - λk)];

step 3:      for temp,{
             Ck ← λk /μ [1+ 1/(γ·T)Σj=1^n λjdj /λkdk]; /* 최적 라인 용량 할당 */
             linkk ← λk / Ck; }                        /*  라인 부하 계산 */

             if(flag == 1){                            /*  전체 비용 계산 */
                     for(p=1;p<=lcnt;p++)
                         cost(p) ← Σk dCkdk (∀k ∈ path(p));
                         Dcost ← Σp=1^lcnt cost(p); }
              else{
                     for(p=sub1,p=sub2)
                     cost(p) ← Σk dCkdk (k ∈ path(p));
                     Dcost ← Σp=1,p≠sub1,sub2^lcnt cost(p)+cost(sub1)+cost(sub2); }

step 4:  if (T ≃ Delay )  return;        /* NEH(또는 NSH)로 제어 이전 */
         else if (T > Delay){
                       for temp,{
                            imsi ← Maximum{linkk}; index1 ← k; }
                            Cindex1 ← Cindex1 + unit;   go to  step 2; }

         else{
                  ①  for temp,{
                            imsi ← Min{linkk}; index2 ← k; }
                            Cindex2 ← Cindex2 - unit;
                                     if( Cindex2 ≥ λindex2/μ)  go to step 2;
                                     else { linkindex2 = ∞;  go to ①; }
             }
}
```

그림 3. 노드 교환 휴리스틱(NEH)

매트릭스를 다시 계산하지 않는다. 노드 교환 휴리스틱은 그림 3과 같다.

4.2 노드 이동 휴리스틱

이 휴리스틱은 노드 교환 단계에서 얻어진 클러스터의 정보를 이용하여 특정 클러스터의 노드 집합에 포함되는 노드를 이동하는 휴리스틱으로, trade-off criterion(ks'_{ij})이 양수의 최대값을 갖는 노드 쌍에 대해 해당 노드를 다른 클러스터의 노드 집합으로 이동시켜 해를 개선해 나간다. 즉, 서로 다른 클러스터에 있는 노드 쌍(i, j)의 trade-off criterion을 계산한 결과, 노드 j가 이동 가능하다고 하면 node(sub2)에 소속된 노드 j를 노드 i가 소속된 node(sub1)으로 이동하고, node(sub2)에서 노드 j를 삭제하여 새로운 클러스터의 노드 집합을 구한다. 이 때, 노드 i와 j는 각 노드 집합의 양단에 위치한 노드들만을 고려한다. 그 이유는 노드를 이동하는 경우 각 노드 집합의 양단 노드들이 이동될 때, 해의 개선 가능성이 있기 때문이다. 이 경우, 모든 노드 집합의 노드 수가 MAX인 경우와 ks'_{ij}가 전부 음수인 경우에는 휴리스틱은 수행되지 않는다. 그렇지 않은 경우에는 양수의 최대값을 갖는 ks'_{ij}부터 차례로 알고리즘을 적용한다.

이 휴리스틱에서 사용하는 trade-off criterion(ks'_{ij})은 다음 조건하에서 정의한다. 즉, 노드 i가 있는 클

러스터 sub1과 노드 j가 있는 클러스터 sub2가 같지 않은 조건하에 node(sub1)의 노드 갯수에 1을 더한 값이 MAX보다 큰 경우에는 노드를 이동시킬 수 없으므로 ks'_{ij}의 값을 $-\infty$로 놓고, 그렇지 않은 경우에는 식 (23)~식 (25)에 의해 ks'_{ij}를 계산한다.

노드 이동을 위한 trade-off criterion의 기본 개념은 다음과 같이 설명될 수 있다. $i \in node(sub1)$, $j \in node(sub2)$, $(sub1 \neq sub2)$이고 $node(sub1) = \{i-2, i-1, i\}$이고 $node(sub2) = \{j, j+1, j+2\}$라고 하자.

그림 4에서 노드 j를 이동하는 경우 절단되는 라인은 $d_{0,i}$, $d_{j,j+1}$이고 추가되는 라인은 $d_{i,j}$, $d_{0,j+1}$이다. 따라서 이 경우 ks'_{ij} = 절단비용 − 추가비용 = $(d_{0,i} + d_{j,j+1}) - (d_{i,j} + d_{0,j+1})$이 된다. 이상의 경우를 고려한 ks'_{ij}는 식 (23)~식 (25)와 같으며 비용 개념은 노드 이동 휴리스틱에서와 마찬가지로 순회 거리(traversal distance)이다.

4.2.1 클러스터 sub2의 노드 갯수가 1인 경우

$$ks'_{ij} = d_{0i} + d_{0j} - d_{ij} \tag{23}$$

4.2.2 클러스터, sub2의 노드 갯수가 2 이상이고 j가 첫번째 노드인 경우

$$ks'_{ij} = (d_{0i} + d_{j,j+1}) - (d_{ij} + d_{0,j+1}) \tag{24}$$

4.2.3 클러스터 sub2의 노드 갯수가 2 이상이고 j가 마지막 노드인 경우

$$ks'_{ij} = (d_{0i} + d_{j,j-1}) - (d_{ij} + d_{0,j-1}) \tag{25}$$

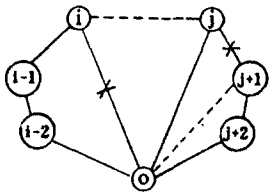

그림 4. 노드 이동의 예(MAX = 4)

휴리스틱: Node Shift Heuristic(NSH)

```
input:    NEH에서 얻은 NEW_cost, lcnt, kcnt(p), cost(p), node(p), path(p)
output:   NSH에서 얻는 NEW_cost, ring topology
variable: flag, unit, temp
process:
   step 1: /* 노드 이동의 가능 여부 조사 */
           for p, (p=1, 2..., lcnt),
                 if (kcnt(p) == MAX)  알고리즘 종료;
                 else flag = 0;

   step 2: /* trade-off criterion(ks'_ij) 계산 */
       ① 노드 쌍(i, j)(∀(i, j))에 대해,
           if(i∈node(p)) sub1 ← p;  if(j∈node(p)) sub2 ← p;

       ② if(sub1 != sub2 && Σ_{m∈node(sub1)} q_m + q_j ≤ MAX)
                 식(25)~식(28)을 이용하여 ks'_ij 계산;
           else
                 ks'_ij ← -∞;
       ③ if (ks'_ij < 0) ∀(i, j),   알고리즘 종료

   step 3: /* 노드 이동을 통한 새로운 토폴로지 생성 */
           while( ks'_ij > 0){
             ① 가장 큰 ks'_ij값을 갖는 노드 쌍(i, j)에 대해        /* 노드 이동 */
                 if(i∈node(p)) sub1 ← p;  if(j∈node(p)) sub2 ← p;
```

```
        kcnt(sub1) ← kcnt(sub1)+1; node(sub1) ← node(sub1)∪j;
        kcnt(sub2) ← kcnt(sub2)-1; node(sub2) ← node(sub2)-j;
② NEH의 step 3의 ②와 동일
③ NEH의 step 3의 ③과 동일
④ /* 전체 비용 비교 */
        if (D_cost ≥ NEW_cost){          /* 노드 환원 */
              kcnt(sub1) ← kcnt(sub1)-1; node(sub1) ← node(sub1)-j;
              kcnt(sub2) ← kcnt(sub2)+1; node(sub2) ← node(sub2)∪j;
              ks_ij ← -∞;      repeat step 3;}
        else {
                  NEW_cost ← D_cost;    /* 절감된 비용을 전체 비용으로 놓음 */
                  ks'_ij  ← -∞;     go to step 2; }
};
```

그림 5. 노드 이동 휴리스틱(NSH)

노드 이동 휴리스틱은 그림 5와 같다.

이제, DMCLP 알고리즘의 시간 복잡도(time complexity)를 계산하기 위해 먼저, 기존 MCLP 알고리즘의 시간 복잡도를 고려하면 IOTP 알고리즘[8]인 경우 $O(n^2)$, QIOTP 알고리즘[2]인 경우 $O(MAX \cdot n)$이다. 이 경우에는 초기해로 입력되는 전체 TSP tour에 대한 계산 시간은 포함되어 있지 않다. 본 논문에서 사용한 TSP 알고리즘[10]의 시간 복잡도는 노드 수가 n일때 $O(n^2)$이므로 IOTP 알고리즘의 전체 시간 복잡도는 $O(n^2)$, QIOTP 알고리즘의 전체 시간 복잡도 역시 $O(n^2)$이다. 이제 IOTP 또는 QIOTP 알고리즘의 클러스터를 이용하여 직접 평균 지연 시간을 고려한 해를 구하기 위해서는 DMCLP 알고리즘의 Mean_Delay( )루틴만을 적용하면 된다. 알고리즘[2,8]에서 얻는 클러스터의 최대 갯수는 $\lceil n/MAX \rceil$, 하나의 클러스터에 존재 가능한 최대 노드 수는 MAX이고, MAX개의 노드를 센터 노드에 연결할 수 있는 tour 상의 라인 수는 MAX+1이므로 Mean_Delay( )루틴에서 계산해야 하는 라인 수는 $\lceil n/MAX \rceil(MAX+1)$이다. Mean_Delay( ) 루틴에서 각 수식의 계산을 위한 시간 복잡도는 $O(1)$이고, 평균 지연 시간(T)이 원하는 시간(Delay)에 근접하기 위한 최대 반복 횟수는 $MAX\lceil n/MAX \rceil(MAX+1)$이며 라인 부하의 최대값을 얻기 위한 복잡도는 $O[\lceil n/MAX \rceil(MAX+1)]=O(n)$이므로 전체 실행 시간=IOTP(또는 QIOTP) 알고리즘의 실행 시간+Mean_Delay( ) 루틴의 실행 시간=$Maximum[O(n^2), MAX\lceil n/MAX \rceil(MAX+1)Maximum\{O(1), O(n)\}]=O(MAXn^2)$이다. 다음으로 DMCLP 알고리즘의 NEH를 고려해 보자. 먼저, step 1의 초기해를 구하기 위해서 기존 MCLP 알고리즘에 의한 클러스터의 수만큼 최대 MAX개 노드에 TSP 알고리즘[10]을 적용하므로 이를 위한 시간 복잡도는 $\lceil n/MAX \rceil O(MAX^2)=O(MAX \cdot n)$이다. 이 때의 전체 라인 수는 $\lceil n/MAX \rceil(MAX+1)$이고 위에서와 동일한 방법으로 구하면 시간 복잡도는 $O(MAXn^2)$이다. step 2에서 ks_{ij}는 $\forall(i, j)(i \in node(sub1), j \in node(sub2) : sub1 \neq sub2)$에 대해 계산되고 그 최대 수는 $\lceil n/MAX \rceil MAX$이므로 step 2의 시간 복잡도는 $O(n)$이다. step 3은 최악의 경우, $ks_{ij} > 0$인 경우의 수만큼 Mean_Delay( )루틴, step 2를 반복 수행해야 한다. 즉 ks_{ij}의 최대 반복 횟수는 $\lceil n/MAX \rceil MAX$이다. 먼저, ①의 계산 시간은 $O(\lceil n/MAX \rceil MAX)=O(n)$이다. ②에서 2개의 변화된 클러스터의 노드 집합에만 TSP 알고리즘을 적용하므로 TSP tour를 얻기 위한 시간 복잡도는 $O(MAX^2)$이다. ③에서 Mean_Delay( ) 루틴의 step 1~3의 시간 복잡도는 $2(MAX+1)O(1)=O(MAX)$이다. Mean_Delay( ) 루틴의 step 4는 최대 2MAX(MAX+1)개 만큼 step 2~step 4가 반복될 수 있고, 최대 또는 최소 링크 부하를 얻기 위한 시간 복잡도는 $O(MAX)$이므로 Mean_Delay( ) 루틴의 전체 시간 복잡도는 $2MAX(MAX+1)Maximum[O(MAX), O(MAX)]=O(MAX^3)$이다. 따라서 NEH의 step 3의 전체 시간 복잡도는 $MAX\lceil n/MAX \rceil Maximum[O(n), O(MAX^2), O(MAX^3)]=O(MAX^3n)$이다. 이로 부터 NEH의 전체 시간 복잡도는 $Maximum[O(MAXn^2), O(MAX^3n)]=O(MAXn^2)$이다($\because MAX \ll n$). 동일한 방법으로 NSH의 시간 복잡도 역시 O

(MAX^2n)이 됨은 자명하다. 따라서 DMCLP 알고리즘의 전체 실행 시간=IOTP(또는 QIOTP) 알고리즘의 실행 시간+NEH 실행 시간+NSH 실행 시간 $=Maximum[O(n^2),\ O(MAXn^2)]=O(MAXn^2)$이다. 결국, 기존 MCLP 알고리즘에 평균 지연 시간의 제약 조건을 추가한 경우의 시간 복잡도와 DMCLP 알고리즘의 시간 복잡도는 동일하다.

5. 계산 예

DMCLP 알고리즘의 진행 과정을 보이기 위해 20개의 노드로 구성되는 문제를 고려한다. 센터 노드의 색인은 0이고 사용자 노드의 색인은 1~19이다. 사용자 노드에서의 트래픽은 $q_i=1$, $(i=1, \ldots, 19)$이고 하나의 링에 둘 수 있는 최대 노드수(MAX)는 6이다. 메시지의 평균 길이는 800 bit이고 요구되는 최대 평균 지연 시간(Delay)은 0.8초이다. 단위당 라인 비용(d)은 1이고 거리 매트릭스 D는 대칭으로 표 1과 같다.

먼저 TSP 알고리즘[10]을 적용한 tour는 그림 6과 같다. 다음으로 QIOTP 알고리즘을 적용한 결과는 그림 7과 같다.

또한, 그림 7의 토폴로지에 Mean_Delay( ) 루틴을 적용한 결과는 표 2와 같다.

다음으로 DMCLP 알고리즘의 NEH를 적용하면 다음과 같다. 먼저 NEH의 입력은 그림 7의 lcnt=4,

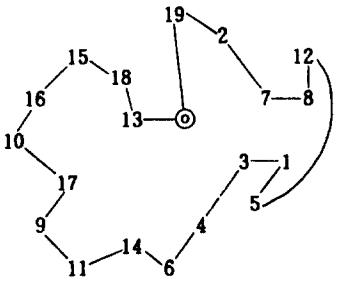

그림 6. 전체 TSP tour

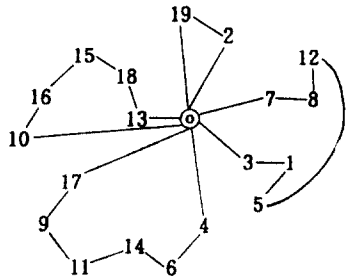

그림 7. MCLP 알고리즘의 토폴로지

표 1. 거리 매트릭스(d_{ij})

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	46	11	57	68	26	93	27	35	62	97	81	34	109	87	108	110	85	106	10
1	46	0	35	11	22	22	48	34	48	56	88	48	48	82	41	88	96	87	83	46
2	11	35	0	46	57	15	82	20	32	55	91	71	31	100	76	100	103	82	98	11
3	57	11	46	0	11	32	37	42	55	59	87	42	56	76	30	84	94	90	78	57
4	68	22	57	11	0	43	26	52	63	63	88	39	65	72	19	82	93	94	76	68
5	26	22	15	32	43	0	68	14	29	47	82	56	29	87	62	88	93	76	85	26
6	93	48	82	37	26	68	0	75	84	76	92	39	86	65	6	79	93	104	72	93
7	27	34	20	42	52	14	75	0	14	35	71	55	15	82	69	81	83	63	78	27
8	35	48	32	55	63	29	84	14	0	27	62	58	2	79	79	75	75	50	74	35
9	62	56	55	59	63	47	76	35	27	0	35	41	29	53	73	48	47	31	47	62
10	97	88	91	87	88	82	92	71	62	35	0	53	64	39	91	25	14	23	30	97
11	81	48	71	42	39	56	39	55	58	41	53	0	61	34	38	43	55	65	37	81
12	34	48	31	56	65	29	86	15	2	29	64	61	0	81	81	77	77	51	76	34
13	109	82	100	76	72	87	65	82	79	53	39	34	81	0	67	16	32	61	9	109
14	87	41	76	30	19	62	6	69	79	73	91	38	81	67	0	80	93	101	73	87
15	108	88	100	84	82	88	79	81	75	48	25	43	77	16	80	0	16	48	7	108
16	110	96	103	94	93	93	93	83	75	47	14	55	77	32	93	16	0	37	23	110
17	85	87	82	90	94	76	104	63	50	31	23	65	51	61	101	48	37	0	52	85
18	106	83	98	78	76	85	72	78	74	47	30	37	76	9	73	7	23	52	0	106
19	10	46	11	57	68	26	93	27	35	62	97	81	34	109	87	108	110	85	106	0

표 2. 기존 MCLP 알고리즘의 토폴로지에 평균 지연을 고려한 해

cluster (p)	line	dist.	lambda (msgs/s)	line cost	capacity (kbps)	waiting time(ms)	cost(p)
1	0-19	10.00	2.00	30.21	3.02	563.04	83.29
	19- 2	11.00	1.00	19.85	1.80	796.25	
	2- 0	11.00	2.00	33.23	3.02	563.04	
2	0- 7	27.00	6.00	196.05	7.26	325.07	1011.19
	7- 8	14.00	5.00	87.45	6.25	356.10	
	8-12	2.00	4.00	10.42	5.21	398.13	
	12- 5	29.00	3.00	120.07	4.14	459.72	
	5- 1	22.00	4.00	114.61	5.21	398.13	
	1- 3	11.00	5.00	68.71	6.25	356.10	
	3- 0	57.00	6.00	413.88	7.26	325.07	
3	0- 4	68.00	6.00	493.75	7.26	325.07	1869.17
	4- 6	26.00	5.00	162.41	6.25	356.10	
	6-14	6.00	4.00	31.26	5.21	398.13	
	14-11	38.00	3.00	157.33	4.14	459.72	
	11- 9	41.00	4.00	213.59	5.21	398.13	
	9-17	31.00	5.00	193.64	6.25	356.10	
	17- 0	85.00	6.00	617.19	7.26	325.07	
4	0-10	97.00	5.00	605.92	6.25	356.10	1501.83
	10-16	14.00	4.00	72.93	5.21	398.13	
	16-15	16.00	3.00	66.24	4.14	459.72	
	15-18	7.00	3.00	28.98	4.14	459.72	
	18-13	9.00	4.00	46.88	5.21	398.13	
	13- 0	109.00	5.00	680.88	6.25	356.10	
Mean Delay Time : 0.78(sec) Total line Cost : 4465.47							

kcnt(1) = 2, node(1) = {2, 19}, kcnt(2) = 6, node(2) = {1, 3, 5, 7, 8, 12}, kcnt(3) = 6, node(3) = {4, 6, 9, 11, 14, 17}, kcnt(4) = 5, node(4) = {10, 13, 15, 16, 18}이다. 다음으로 NEH의 각 단계는 아래와 같다.

step 1 : ①4개의 클러스터의 노드 집합에 TSP 알고리즘을 적용하여 초기 토폴로지를 구하면 이는 그림 8과 같다. path(1) = {0-2, 2-19, 19-0}, path(2) = {0-5, 5-1, 1-3, 3-7, 7-8, 8-12, 12-0}, path(3) = {0-4, 4-6, 6-14, 14-11, 11-9, 9-17, 17-0}, path(4) = {0-13, 13-18, 18-15, 15-16, 16-10, 10-0}이다. ②path(p)(p = 1, 2, 3, 4)를 이용하여 Mean_Delay( ) 루틴을 적용하면 표 3과 같다. ③$NEW_{cost} \leftarrow 4343.99(D_{cost})$

step 2 : ①서로 다른 크러스터의 노드 집합에 속한 노드 쌍(i, j)에 대해 ②식(8)~식(20)을 사용하여 초기 trade-off($ks_{i,j}$) 매트릭스를 계산한다. ③$ks_{ij} < 0(\forall i, j)$이므로 NEH 종료, 따라서 구하는 토폴로지는 그림 8과 동일하다.

이제, DMCLP 알고리즘의 NSH를 적용하면 다음과 같다. 먼저 NSH의 입력은 그림 8의 lcnt = 4, kcnt(1) = 2, node(1) = {2, 19}, kcnt(2) = 6, node(2) = {1, 3, 5, 7, 8, 12}, kcnt(3) = 6, node(3) = {4, 6, 9, 11, 14, 17}, kcnt(4) = 5, node(4) = {10, 13, 15, 16, 18}, cost(1) = 83.29, cost(2) = 889.69, cost(3) = 1869.17, cost(4) = 1501.83이고 NEW_{cost} = 4343.99이다. 다음으로 NSH의 각 단계는 아래와 같다.

step 1 : 모든 클러스터 p에 대해, kcnt(p) ≠ MAX이므로 goto step 2 ;

step 2 : trade-off($ks'_{i,j}$) 매트릭스를 계산하면 $ks'_{10,17} = 43$, $ks'_{13,17} = 17$, $ks'_{17,2} = 7$, $ks'_{13,2} = 10$, 이외의 $ks'_{ij} < 0$이다.

step 3 : ①$ks'_{10,17} = 43$이 가장 큰 값이므로 sub1 = 4, sub2 = 3이다. node(3)에 있는 노드 17을 node(4)로 이동하여 새로운 노드 집합을 구한다. ②node(3), node(4)에 대해, TSP 알고리즘을 적용하여 tour를 구하면 path(3) = {0-4, 4-14, 14-6, 6-11, 11-9, 9-0}, path(4) = {0-17, 17-10, 10-16, 16-15, 15-18, 18-13, 13-0}이다. ③이 두개의 path(3), path(4)에 대해

표 3. NEH의 초기해

cluster (p)	line	dist.	lambda (msgs/s)	line cost	capacity (kbps)	waiting time(ms)	cost(p)
1	0- 2	11.00	2.00	33.23	3.02	563.04	83.29
	2-19	11.00	1.00	19.85	1.80	796.25	
	19- 0	10.00	2.00	30.21	3.02	563.04	
2	0- 5	26.00	6.00	188.79	7.26	325.07	889.69
	5- 1	22.00	5.00	137.42	6.25	356.10	
	1- 3	11.00	4.00	57.30	5.21	398.13	
	3- 7	42.00	3.00	173.89	4.14	459.72	
	7- 8	14.00	4.00	72.93	5.21	398.13	
	8-12	2.00	5.00	12.49	6.25	356.10	
	12- 0	34.00	6.00	246.87	7.26	325.07	
3	0- 4	68.00	6.00	493.75	7.26	325.07	1869.17
	4- 6	26.00	5.00	162.41	6.25	356.10	
	6-14	6.00	4.00	31.26	5.21	398.13	
	14-11	38.00	3.00	157.33	4.14	459.72	
	11- 9	41.00	4.00	213.59	5.21	398.13	
	9-17	31.00	5.00	193.64	6.25	356.10	
	17- 0	85.00	6.00	617.19	7.26	325.07	
4	0-13	109.00	5.00	680.88	6.25	356.10	1501.83
	13-18	9.00	4.00	46.88	5.21	398.13	
	18-15	7.00	3.00	28.98	4.14	459.72	
	15-16	16.00	3.00	66.24	4.14	459.72	
	16-10	14.00	4.00	72.93	5.21	398.13	
	10- 0	97.00	5.00	605.92	6.25	356.10	
	Mean Delay Time : 0.78(sec) Total line Cost : 4343.99						

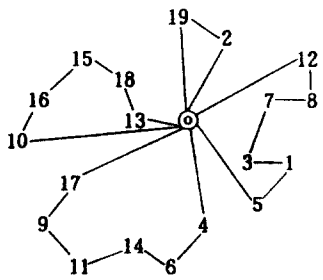

그림 8. NEH의 토폴로지

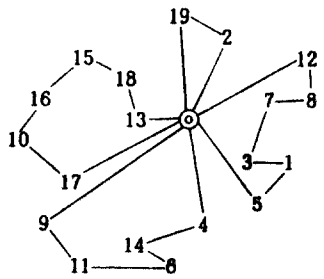

그림 9. DMCLP 알고리즘의 최종 토폴로지(NSH)

Mean_Delay( ) 루틴을 적용하면, cost(3)=1310.94, cost(4)=1784.17을 얻고 node(1), node(2)는 변화하지 않았으므로 cost(1)=83.29, cost(2)=889.69를 그대로 사용하면 전체비용(D_{cost})은 4068.09가 된다.

④ $D_{cost}=4068.09<NEW_{cost}=4343.99$이므로, $NEW_{cost} \leftarrow 4068.09$, $ks'_{10,17} \leftarrow -\infty$;

step 2 : trade-off($ks'_{i,j}$) 매트릭스를 계산하면 $ks'_{i,j} < 0(\forall i, j)$이므로 NSH가 종료된다.

따라서 구하는 최소 비용 토폴로지는 표 4및 그림 9와 같다.

표 4. DMCLP 알고리즘의 해

cluster (p)	line	dist.	lambda (msgs/s)	line cost	capacity (kbps)	waiting time(ms)	cost(p)
1	0-19	11.00	2.00	33.23	3.02	563.04	83.29
	19- 2	11.00	1.00	19.85	1.80	796.25	
	2- 0	10.00	2.00	30.21	3.02	563.04	
2	0- 5	26.00	6.00	188.79	7.26	325.07	889.69
	5- 1	22.00	5.00	137.42	6.25	356.10	
	1- 3	11.00	4.00	57.30	5.21	398.13	
	3- 7	42.00	3.00	173.89	4.14	459.72	
	7- 8	14.00	4.00	72.93	5.21	398.13	
	8-12	2.00	5.00	12.49	6.25	356.10	
	12- 0	34.00	6.00	246.87	7.26	325.07	
3	0- 4	68.00	5.00	424.77	6.25	356.10	1310.94
	4-14	19.00	4.00	98.98	5.21	398.13	
	14- 6	6.00	3.00	24.84	4.14	459.72	
	6-11	39.00	3.00	161.47	4.14	459.72	
	11- 9	41.00	4.00	213.59	5.21	398.13	
	9- 0	62.00	5.00	387.29	6.25	356.10	
4	0-17	85.00	6.00	617.19	7.26	325.07	1784.17
	17-10	23.00	5.00	143.67	6.25	356.10	
	10-16	14.00	4.00	72.93	5.21	398.13	
	16-15	16.00	3.00	66.24	4.14	459.72	
	15-18	7.00	4.00	36.47	5.21	398.13	
	18-13	9.00	5.00	56.22	6.25	356.10	
	13- 0	109.00	6.00	791.45	7.26	325.07	
	Mean Delay Time : 0.78(sec) Total line Cost : 4068.09						

Ⅲ. 성능 평가 및 분석

DMCLP 알고리즘을 평가하기 위해 노드의 좌표를 $x=[0, 100]$, $y=[0, 100]$사이에서 일양분포(uniform distribution)로 난수 발생(random number generation)하고 대응되는 유클리디안 거리는 다음 식을 사용하였다. 임의의 2개의 노드의 좌표를 (x_1, y_1), (x_2, y_2)라고 하면, 두 좌표 사이의 거리는

$$d_{ij}=\lfloor(\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}+0.5)\rfloor*100 \quad (26)$$

으로 정의하고 거리 자체를 라인 비용(d_{ij})으로 간주하며 그 값은 대칭으로 가정한다.

다음으로, 센터 노드의 위치에 따라 다음의 두 가지 형태의 네트워크를 고려한다.

TC : 센터 노드가 좌표 (0, 0)에 위치
TE : 센터 노드가 좌표 (50, 50)에 위치

각 네트워크의 형태에 대해, 노드의 수(n)는 20, 40, 60, 80으로 하고 최대 노드수(MAX)는 2에서 15까지를 사용하였다. 사용 언어는 FORTRAN-77 및 C 언어이고 실행 환경은 MS-DOS하의 IBM-486 기종이다. 다음으로 기존 MCLP 알고리즘[2,8]에 대한 DMCLP 알고리즘의 절감율을 식 (27)로 정의한다.

$$\text{Savings Rate}=((A-B)/A)*100 \quad (27)$$

그림 10은 식 (27)을 사용하여 얻은 평균 절감율이다. 그림 10에서 IOTP + DMCLP는 알고리즘[8]의 토폴로지에 Mean_Delay( ) 루틴을 적용한 해를 A로, 알고리즘[8]의 토폴로지를 초기해로 사용하여 얻은 DMCLP 알고리즘의 해를 B로 놓고 얻은 절감율이고, QIOTP + DMCLP는 알고리즘[2]의 토폴로지에 Mean_Delay( ) 루틴을 적용한 해를 A로, 알고리즘[2]를 초기해로 사용하여 얻은 DMCLP 알고리즘의 해를 B로 놓고 얻은 절감율이다.

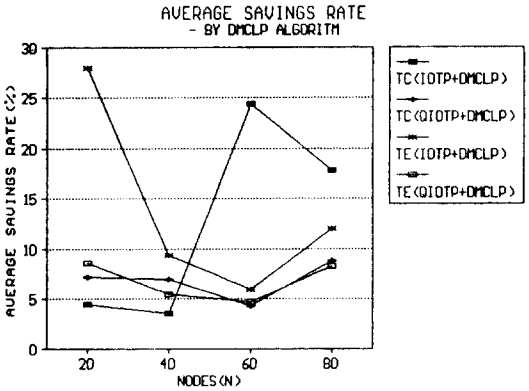

그림 10. DMCLP 해의 평균 절감율(%)

그림 10에서 TC TEST 및 TE TEST사이의 절감율에는 일정한 관계가 없으며 문제의 성격에 따라 알고리즘[8]의 해를 초기해로 사용하는 것이 절감율이 높은 경우와 알고리즘[2]의 해를 초기해로 사용하는 것이 절감율이 높은 경우가 서로 다르다. 한편 시뮬레이션 결과 최대 노드수(MAX)와 절감율 사이에도 일반적인 관계를 발견할 수 없었다.

다음으로 평균 메시지 길이($1/\mu$)와 절감율 사이의 관계를 그림 11, 12를 통해 알아 본다. 그림 11, 12에서 X축은 평균 메시지 길이이고 Y축은 절감율이며 그림에서 보듯이 평균 메세지 길이의 변화가 절감율에 그다지 영향을 주지 않고 있음을 알 수 있다.

이상의 결과로 부터 로컬 액세스 컴퓨터 네트워크에서의 최소 비용 링 설계 문제는 주어진 노드 사이의 거리, 즉 노드의 위치에 따라 해가 영향을 받음을 알 수 있다.

다음으로 평균 실행 시간(CPU time)에 대해 정리한 것이 표 5이다. 표 5에서 IOTP + DELAY는 알고리즘[8]의 토폴로지에 Mean_Delay( ) 루틴을 적용한 평균 실행 시간이고 QIOTP + DELAY는 알고리즘[2]의 토폴로지에 Mean_Delay( ) 루틴을 적용한 평균 실행 시간이며 DMCLP는 본 연구에서 제안한 알고리즘의 평균 실행 시간이다.

표 5에서 보듯이 실행 시간과 TEST 종류 사이에는 일정한 관계가 없다. 평균 실행 시간은 QIOTP + DELAY < DMCLP < IOTP + DELAY의 순이다. 표 5와 그림 10을 비교하면 DMCLP 알고리즘은 PC상에서 평균 CPU 시간을 6초 정도 더 추가하여 평균 10%의 절감율을 얻음을 알 수 있다. DMCLP의 경우

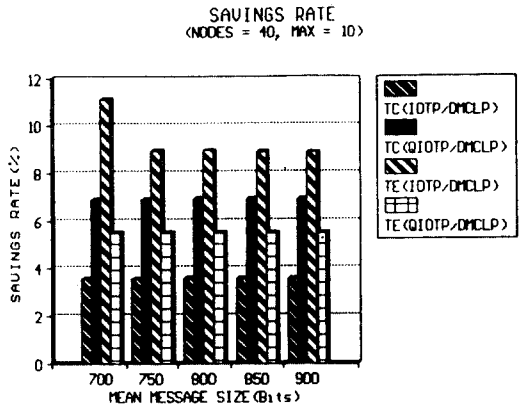

그림 11. 평균 메시지 길이와 절감율(노드 = 40)

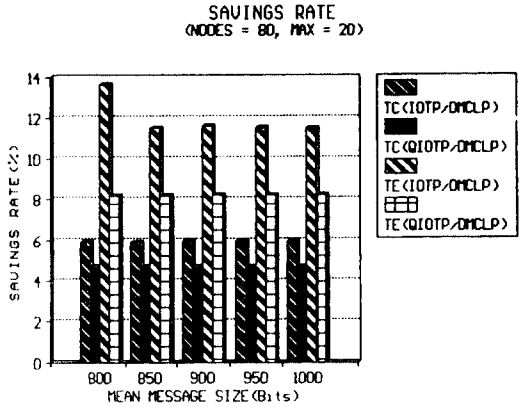

그림 12. 평균 메시지 길이와 절감율(노드 = 80)

표 5. 평균 실행 시간

(단위 : 초, IBM-486 PC)

문 제 \ 노드 수(N)		20	40	60	80
TC TEST	IOTP + DELAY	4.78	6.90	9.51	12.46
	QIOTP + DELAY	1.00	2.92	5.58	8.71
	DMCLP	1.46	3.74	8.51	10.66
TE TEST	IOTP + DELAY	3.93	9.70	14.09	10.34
	QIOTP + DELAY	0.97	2.61	5.50	6.07
	DMCLP	1.43	3.82	10.13	8.52

노드 수가 80인 경우의 CPU 시간(IBM-PC 486기종)은 약 10초 정도로 토폴로지 설계 문제가 실시간 처리(real time processing)를 요구하는 문제가 아님을 고려할 때 비교적 빠른 해를 제공한다고 할 수 있다.

Ⅳ. 결 론

본 논문에서는 로컬 액세스 컴퓨터 네트워크를 설계할 때 발생되는 문제의 하나로 단일 링에 둘 수 있는 최대 노드 수가 제한되고 네트워크의 평균 지연 시간이 임의의 주어진 시간 이내이어야 하는 DMCLP에 대해, 기존의 MCLP 알고리즘에 의해 생성되는 클러스터의 정보를 입력으로하여 해를 개선해 나가는 2단계 휴리스틱을 제시하였다.

이 휴리스틱은 본 연구에서 제안한 trade-off criterion에 기초하여 클러스터 사이에서 노드들을 교환 또는 이동하는 방법을 사용하고 큐잉 이론에 기초하여 주어진 평균 지연 시간을 만족하도록 라인의 용량을 최적으로 할당하는 휴리스틱이다.

성능 평가 결과, 기존의 MCLP 알고리즘에 의한 토폴로지에 평균 지연 시간의 제약 조건을 추가한 해를 평균 10% 이상 절감시킬 수 있었으며 비교적 짧은 계산 시간 이내에 우수한 해를 얻을 수 있었다. 끝으로, 앞으로의 연구에서는 절감율을 더욱 높이면서 실행 시간을 감소시킬 수 있는 효율적인 알고리즘이 기대된다.

참 고 문 헌

1. V. Ahuja, Design and Analysis of Computer Communication Networks, pp. 117-145, McGraw-Hill, 1985.
2. K. Altinkemer and B. Gavish, "Heuristics for Delivery Problems with Constant Error Gurantees," Tran. Sci., Vol.24, No.4, pp.294-297, 1990.
3. N. Christofides, A. Mingozzi and P. Toth, "Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations," Mathematical Programming, Vol. 20, pp.255-282, 1981.
4. B. Gavish, "Topological Design of Telecommunication Networks-Local Access Design Methods," Annals of Operations Research 33, pp. 17-71, 1991.
5. B. Gavish, "A General Model for the Topological Design of Computer Networks," Proceedings of GLOBECOM 86, pp.1584-1588, 1986.
6. B. Gavish and I. Neuman, "A system for Routing and Capacity Assignment in Computer Communication Networks," IEEE Trans. on comm., Vol.37, No.4, pp.360-366, 1989.
7. M. Gerla and L. Kleinrock, "On the Topological Design of Distributed Computer Networks," IEEE Trans on Comm., Vol. COM-25, No.1, pp. 48-60, 1977.
8. M. Haimovich and A. Rinnooy Kan, "Bounds and Heuristics for Capacitated Routing Problems," Mathematics of Operations Research, Vol.10, No.4, pp.527-542, 1985.
9. J. Lenster and A. H. G. Rinnooy Kan, "Complexity of Vehicle Routing and Scheduling Problems," Networks, vol.11, pp.221-227, 1981.
10. S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for Traveling Salesman Problem," Opn. Res., vol.21, pp.498-516, 1973.
11. T. Magnanti, R. K. Ahuja and J. B. Orlin, Network Flows-Theory, Algorithms, and Applications, pp.623-627, Prentice-Hall, Inc., 1993.
12. K. T. Newport, "Incorporating Survivability Considerations Directly into the Network Design Process," Proceedings IEEE-INFOCOM '90, pp.215-220, 1990.
13. M. Schwartz, Telecommunication Networks : Protocols, Modeling and Analysis, Prentice-Hall, pp.262-263, 1987.
14. M. Schwartz and T. E. Stern, "Routing Techniques Used in Computer Communication Networks," IEEE Trans. on comm., Vol. COM-28, No.4, pp.539-552, 1980.

李 湧 振(Yong-Jin Lee) **正會員**
1958년 10월 26일생
1983년 : 고려대학교 산업공학과(학사, 석사)
1993년 : 고려대학교 전산과학과 박사과정 수료
1984년 ~ 1986년 : 한국과학기술원 시스템공학센터 연구원
1986년 ~ 현재 : 중경공업전문대학 전자계산과 부교수
※주관심분야 : 컴퓨터 네트워크, EDI, ISDN, 성능평가, 분산 시스템

金 泰 潤(Tai-Yun Kim) **正會員**
1956년 7월 13일생
1981년 : 고려대학교 산업공학과(학사)
1983년 : Wayne State University 전산과학과(석사)
1987년 : Auburn University 전산과학과(박사)
1988년 ~ 현재 : 고려대학교 전산과학과 부교수
※주관심분야 : 컴퓨터 네트워크, EDI, ISDN, 이동통신, 컴퓨터 그래픽스