A Path Constrained Approach to Dynamic Network Routing: A Virtual Clustering and Flow Deviation Algorithm

S. David Wu
Hakan Golbasi
Lehigh University

Report No. 98T-005

A Path Constrained Approach to Dynamic Network Routing: A Virtual Clustering and Flow Deviation Algorithm

HAKAN GOLBASI and S. DAVID WU / *Department of Industrial and Manufacturing Systems Engineering, Lehigh University, Bethlehem, Pennsylvania 18015, Email: hag2@lehigh.edu and sdw1@lehigh.edu*

We propose a new routing algorithm for telecommunication networks which integrates static centralized routing and dynamic decentralized routing using the notion of virtual clustering. First proposed by Bartolacci and Wu [1], *virtual clustering* reconfigures the network topology periodically based on projected traffic requirements in the network. We propose a method which views *virtual clustering* as added path constraints in the network. We develop a flow deviation algorithm which solves this path-constrained routing problem. This results in an inter-cluster routing which balances and regulates global network traffic. Within each cluster, decentralized dynamic routing algorithm is applied which handles dynamic traffic fluctuations in real-time. The proposed routing scheme is designed for nonstationary traffic patterns which are often observed in real-life telecommunication networks. Intensive testing of the proposed routing scheme was conducted on randomly generated problem instances. We show that the routing scheme consistently outperform the traditional dynamic routing algorithm. Under nonstationary traffic conditions, the proposed method significantly outperform both the static and the dynamic algorithms.

Most existing network routing algorithms are dynamic and distributed by nature where routing decisions are made locally and often myopically. Static route optimization algorithms are typically used in the design stage, which assume steady state traffic behavior while ignoring short term traffic fluctuations. Consequently, most static algorithms assume constant traffic arrival rates which prohibit their use for short-term routing problems where traffic rates may change significantly over the course of a day. In this paper we propose a routing scheme which attempts to integrate both static and dynamic routing principles. This routing scheme is based on the notion of *virtual clustering* proposed by Bartolacci and Wu [1]. The *virtual clustering* method logically reconfigures the network in a periodic basis according to its traffic patterns. The logical topology imposes restrictions on the potential paths between certain groups of source-destination node pairs in the network, thus regulates and balances the global network traffic while maintaining sufficient local flexibility for real time routing. Virtual clustering transforms the network routing problem into an *inter-cluster* routing problem across the clusters and several *intra-cluster* routing problems. In this paper we propose a new approach to the *inter-cluster* routing problem where static optimal path routing is applied.

Our approach generalizes the concept of *virtual clustering* in that we view clustering as a means of reducing the variance of network traffic used by the routing algorithm. This can be easily explained by the fact that the traffic consolidated from a specific cluster (of nodes) to another cluster tends to exhibit a much smaller variance compared to the node to node traffic. Since the local traffic fluctuations can be handled sufficiently within each cluster using an existing dynamic routing method, we concentrate on optimizing the "global," or inter-cluster traffic. This reformulation of the routing problem allows the use of analytic routing results on short-term inter-cluster routing problems. Since analytic models typically assumes a steady-state traffic behavior based on mean-values [4],[5],[6].

Efficient clustering is important for virtual clustering to work well. It has to be done *a priori* on a period basis before any routing decisions are made. In the paper, we first propose a mathematical model which establishes an analytic framework for the routing scheme, we then describe the routing algorithm developed under the scheme, and finally demonstrate the performance of this method and its ability to handle short-term traffic fluctuations.

Relationship to Past Work in Network Routing

The optimal routing problem is a special case of the multicommodity flow problem. Bertsekas

1. PROBLEM STATEMENT

A telecommunication network can be represented as a directed graph consisting of links and nodes. Each node generates data, receives data, and transmits data to the other nodes whereas links are bidirectional transmission media between nodes. Before transmitting to the other nodes, at the source node, data is separated into smaller packets and each packet is delivered to the destination node by visiting intermediate nodes on its way. The routing problem is to determine the sequence of nodes (or links) that each packet visits. During a visit to an intermediate node if a packet finds that there are other packets at the buffer of an outgoing link , it joins the queue and waits until its turn comes for transmission on that link. Most mathematical models assume infinite buffer capacity for the nodes although there is a buffer size limit for each link.

Consider a directed graph *D(N,A)*, where N is the set of nodes and A is the set of all links, assumes bidirectional links (i.e. for every link $(i,j) \in A$, there is a link $(j,i) \in A$). For *D(N,A)* there are *R* pairs of source-destination pairs. (If there is a communication between each source-destination pair then there are $R=n(n-1)$ source-destination pairs where n is the number of the nodes) The source destination pairs are represented by (s^r,d^r), where $r=1,2,....,R$. If we consider the traffic carried for each source destination pair as a commodity, the routing optimization problem is a multicommodity flow problem with the following nonlinear objective function:

$$\text{Minimize} \quad \sum_{(i,j)} d_{ij}(x_{ij})$$

$$\text{where} \quad d_{ij}(x_{ij}) = \frac{x_{ij}}{c_{ij}-x_{ij}} + \eta_{ij}x_{ij}$$

and, η_{ij}: propagation delay of a link *(i,j)*

x_{ij} : sum of all flows over link *(i,j)* in packets/sec

c_{ij} : total capacity of a link *(i,j)* in packets/sec

The delay function formula $d_{ij}(.)$ is based on the assumption that each queue behaves as an M/M/1 queue of packets- a consequence of the Kleinrock independence approximation and Jackson's and Gallager [2] describe the formulation where the objective is to minimize packet delay subject to flow constraints in the network. Analytic model typically adopt the assumptions that (1) Kleinrock's independence assumption [3] is satisfied, (2) traffic arrival process to the nodes of the network is homogenous Poisson, and (3) packet lengths are exponentially distributed. Optimal solution to the problem generates paths and traffic flows assigned to the paths, sometimes bifurcated, between source-destination pairs. There has been an extensive literature on various algorithms to solve the optimal routing problem both optimally and heuristically. Among the exact algorithms, Fratta, Garla and Kleinrock [4] applied the Frank-Wolfe decomposition, Bertsekas and Gallager [5] used the gradient projection method, Cantor and Gerla proposed extremal flow technique [6]. Mahey, Ouorou, LeBlanc, Chifflet [7] developed an algorithm which has less bifurcation. In this paper, we implemented the algorithm of Bertsekas and Gallager.[5] as a static routing benchmark and compare the solution with our approach.

Dynamic routing algorithms, which update routing tables according to the changing traffic information tries to minimize average delay throughout the network. The dynamic shortest path routing algorithm [8] proposed, implemented and improved for ARPANET has been practically in use for years. In this research we coded the ARPANET dynamic routing algorithm for the purpose of Intra-cluster routing.

The concept of organizing the network as a hierarchy of clusters has been considered by quite a few authors. They approach are typically used in the context hierarchical routing which uses clustering as a means of dealing with routing complexity. Kleinrock and Kamoun [9] showed that as the number of nodes increases the clustering of network nodes become crucial to routing performance. Anthony, Huang and Tsai [10] developed a routing algorithm for Balanced Hierarchically Clustered (BHC) networks. In their algorithm, routing within clusters is performed in a distributed fashion on a cluster by cluster basis similar to our approach. Muralidhar and Sundareshan [11] and Boorstyn and Livne [12] both proposed two- level hierarchical routing models that seek to integrate the routing of the clustered network nodes. Bartolacci and Wu [1] propose the *virtual clustering* method which uses logical clustering as a means of regulating network traffic. This paper will be a direct extension and generalization of this approach.

Theorem. While this assumption is typically violated in practice, the expression for $d_{ij}(.)$ represents a useful measure of performance. In our model, we drop the propagation delay term $\eta_{ij}x_{ij}$ since it is a function of network design rather than routing efficiency and do not significantly effect the results.

The above models assume a stationary, homogenous Poisson traffic arrival process. A traffic requirement matrix, F, defines the required traffic flows for each pair of source-destination node in packets/sec(or bits/sec). However; in practice the traffic requirement matrix change over time, and the network topology, D(N,A) may also change due to node or link failures.

1.1 A SUMMARY OF THE VIRTUAL CLUSTERING SCHEME

The virtual clustering scheme [1] cluster the network at the beginning of a planning period T. This clustering process can be repeated during the planning period if major changes occur which renders a different clustering more attractive. Given the clustering structure, we categorize the links that connect the nodes within the cluster as intra-cluster links and the links which connect the end nodes of different clusters as inter-cluster links. The nodes attached to the inter-cluster links are gate nodes (see Figures 1 and 2). Routing within each cluster is performed by intra-cluster routing. For traffic section of any particular source-destination pair, inter-cluster routing decides all the entry and exit gate nodes for each of the clusters (see Figure 3).

The virtual clustering model seeks to regulate the overall for the interest of the given objective function(e.g. total delay). Among the large number of clusters, the model is set out to find the one which gives the best result for the planning period, T. This Virtual Clustering Problem can be formulated as follows with the corresponding notation:

X: set of all virtual clusters

Γ: number of clusters

N_c : set of nodes in cluster c

x_{ic} : Cluster membership variable of a node i, $(0,1)$ variable

α_c : size of cluster c

l_{ij} : indicator variable that equals 1 if link (i,j) connects two clusters in a clustering solution

Ψ: $\{1,2,\ldots\ldots, c,\ldots\ldots,\Gamma\}$ a set specifying a particular virtual clusters solution

k_1 and k_2: edge-connectivity requirement for intra- and inter- cluster network, respectively

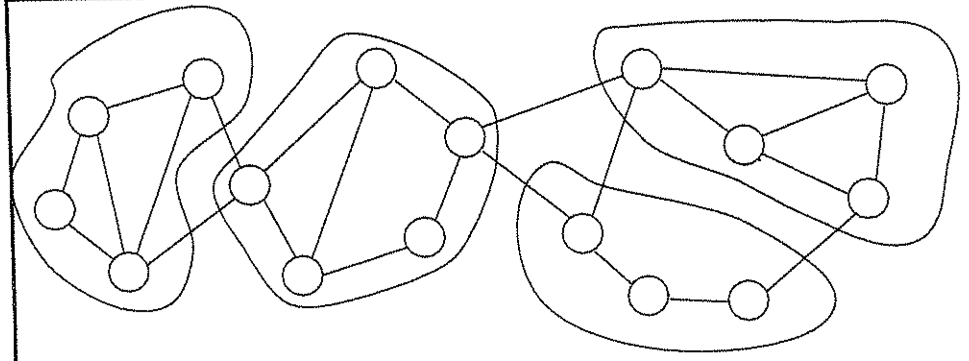


Figure 1 Clustered Structure

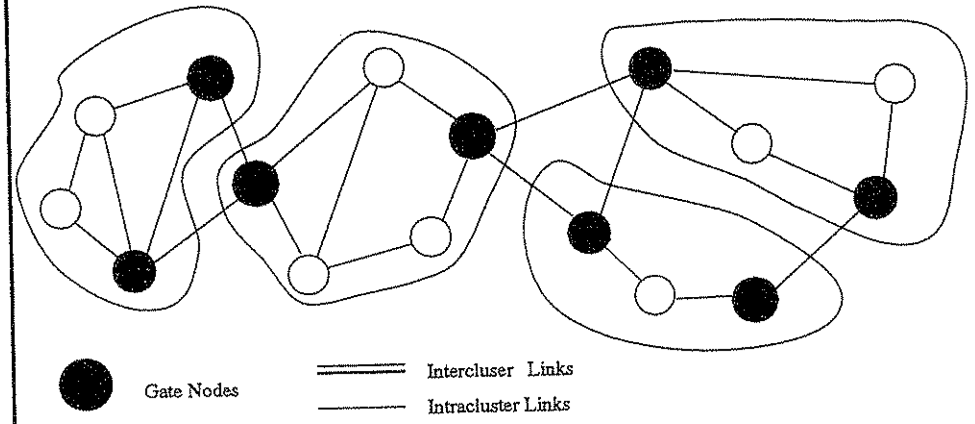
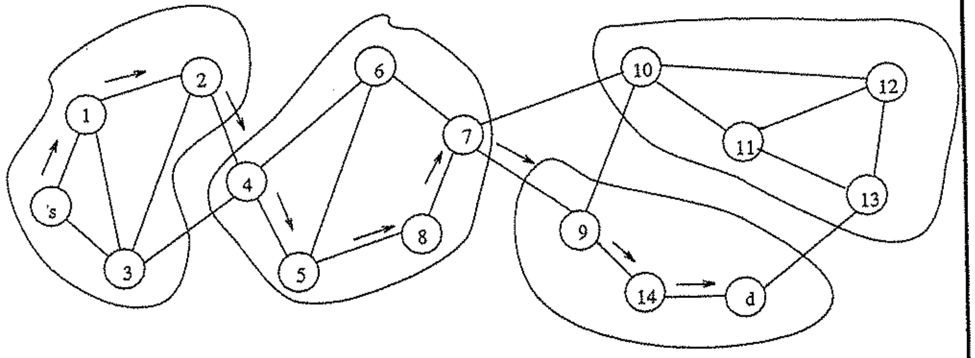


Figure 2 Gate Nodes and Intracluster Links



For a route s-1-2-4-5-8-7-9-14-d;
s-1-2, 4-5-8-7, 9-14-d : Intracluster routing
2-4, 7-9: Intercluster routing

Figure 3: Intercluster & Intracluster routing

J: the set of intercluster links

n : number of nodes in the original network $D(N,A)$

$Minimize_{\epsilon \epsilon} z1(x,g)$
where g solves the intercluster routing subproblem(1.6)-(1.10)

$$\sum_{i=1}^{r} x_{ic} = 1, \quad \forall i \in N \tag{1.1}$$

$$\sum_{i=1}^{n} x_{ic} \leq \alpha_c \ , \quad \forall c \in \Psi \tag{1.2}$$

$$\sum_{i \in N} x_{ic} x_{jc} \geq k_1 \ , \quad \forall c \in \Psi \wedge (i,j) \in A \tag{1.3}$$

$$\sum_{i \in N_c} l_{ij} \geq k_2 \ , \forall c \in \Psi \wedge (i,j) \in J \tag{1.4}$$

$$\text{where set } J \equiv A - \{(u,v) \mid \sum_{c \in \Psi} x_{uc} x_{vp} = 1, u \neq v\}$$

$$x_{ic} \in \{0,1\} \tag{1.5}$$

Constraint set (1.1) are cluster membership constraints which ensures each node to be a member of only one cluster. Constraint set (1.2) limits the size of each cluster. Constraints (1.3) and (1.4) are connectivity constraints within (k_1) and among the clusters (k_2). Intra-cluster connectivity constraints are useful to isolate localized traffic whereas intercluster connectivity constraints provide a degree of reliability.

The solution of the virtual clustering problem based on α_c, k_1, k_2 forms a framework for inter-cluster and intra-cluster routing. In their study ,Bertolacci and Wu [1] proposed a particular network structure as follows:

An inter-cluster network, generated from a particular virtual clustering, is a network constructed by connecting all nodes within the same cluster to a "virtual node", thus forms a star structure. These "star networks" are then connected together using existing links in the original network topology. For a specific source-node destination pair, the inter-cluster routing specifies the following: (1) Via which gate node the originated traffic will use to leave the current cluster, (2) the entering and leaving gate nodes this traffic will use for a " passing through" cluster. and (3) the entering gate node this traffic will use for the "destination cluster." The actual routing within each cluster will be later handled by a dynamic routing algorithm. Solution to inter-cluster routing forms a basis for intra-cluster routing as it sets up the traffic requirement for source-destination pairs within the cluster.

1.2 PROPOSED SIMPLIFICATIONS OF THE VIRTUAL CLUSTERING SCHEME

In this paper we propose a streamlined approach to virtual clustering where we solve the inter-cluster routing problem as a restricted static routing problem. More specifically, we add path constrains to the optimal routing problem based on the results of virtual clustering such that after visiting a specific cluster a packet is *prohibited from visiting the same cluster again.* This in effect reduces localized congestions and balance the overall network traffic. We later demonstrate that this routing scheme outperforms the conventional static and dynamic schemes. In the following, we first reformulate the combined inter-cluster and intra-cluster routing problem using the path formulation of multicommodity flow.

P : set of all paths connecting all source-destination pairs.

P_r : set of all directed paths connecting the source-destination pair (s^r, d^r)

x_p : flow of path p∈P in packets/sec

$P_{(i,j)}$: set of all paths containing (i,j)

$$Z_1(x,g) = \quad Minimize_{g\in Y} \sum_{(i,j)} d_{ij}(g_{ij}) \tag{1.6}$$

$$s.t. \sum_{p\in P^c_{(i,j)}} x_p = g_{ij} \tag{1.7}$$

$$\sum_{p\in P_r} x_p = f^r \ for\ all\ r\in R \tag{1.8}$$

$$\sum_{p\in P^c_{(i,j)}} x_p \le c_{ij} \qquad for\ all\ (i,j) \in A \tag{1.9}$$

$$x_p \ge 0 \qquad for\ all\ p \in P \tag{1.10}$$

Note that $P_{(i,j)}{}^c$ is the set of paths that pass through link (i,j), and which satisfies the stated virtual clustering condition: for each path $p\in P_{(i,j)}{}^c$, the entry into each cluster can be no more than once, i.e., on the way to its destination, a packet can not visit the same cluster again once leaving the cluster. Whenever a new virtual clustering is generated, the set of paths $P_{(i,j)}^c$ changes accordingly. The key is to cluster the network in such a way that the traffic is better balanced via the use of the cluster-constrained paths.

2. SOLUTION METHODOLOGY

In this section we describe the methodology we use to solve the formulated routing problem. We first propose a clustered routing algorithm (CRA), then, for the purpose of performance comparison we describe a static optimal routing algorithm (SRA), and a dynamic shortest path routing algorithm (DRA). All these algorithms are implemented and will be explained in detail.

2.1 CRA (Clustered Routing Algorithm)

We first summarize the main steps of the clustered routing algorithm (CRA) in the following:

Phase 1: Clustering and Optimal Path Generation

For a given communication network D(N,A),

1.1. Find the shortest hop distances between each source-destination pair

1.2. Form the distance matrix M using the shortest hop distances

1.3. Using a hierarchical clustering algorithm, form clusters in the network based on the distance matrix M and a prescribed number of clusters Γ

1.4. Check the connectivity k_1 and k_2 of the clusters generated, repeat Step 3 if necessary

1.5. Based on the cluster structure, obtain *cluster-constrained optimal paths* using a static optimal routing algorithm (SRA)

Phase 2: Dynamic Routing

2.1. When a packet is generated at a specific source node *s* for destination *d* use the generated optimal path from *Phase 1* to identify the entry, and the exit gate nodes for each clusters it is going to visit.

2.2 Route the packet from its source to the exit gate node of the current cluster using a dynamic routing algorithm (DRA)

2.3 When the packet arrives at the exit gate node, use the intercluster link identified in the *cluster-constrained optimal path* to send the packet to next cluster's entry gate node, this node become the new source node

2.4 Repeat Steps *2.2-2.3* until the packet reaches the entry node of its destination cluster.

2.5 In the destination cluster, route the packet from the entry node to its destination node using DRA

In the following, we describe major components of CRA in detail.

The Clustering Algorithm

CRA bases its routing on the virtually clustered network in any planning period T. In essence, clustering put physically nearby nodes that have the most traffic going between them together so as to localize high intensity traffic within the cluster, and stabilize inter-cluster traffic. On the other hand, to ensure feasible connectivity within and across clusters, nodes close to each other in terms of hop distances should be members of the same cluster. Existing clustering algorithms, such as the ones available from the IMSL library, use a symmetric distance matrix as input. Based on the distances given between each pair of set elements, the algorithm partitions the set into mutually exclusive and exhaustive clusters. Our approach is to form a *logical distance matrix M* which makes use of the information in network topology $D(N,A)$ and the traffic requirement matrix F. Specifically, for each *pair* (s',d') in M we compute a logical distance as the convex combination of the normalized physical shortest hop distances and the optimal link flows generated from F using an static optimal routing algorithm (SRA). Note that the use of optimal link flows for all network links distinguishes our *logical distance* calculation from that of Bartolacci and Wu [1] where they use only the traffic requirement between source-destination pairs, i.e., no effort is made in estimating the impact of current traffic requirements given the network topology. Consider the situation where a few nodes in the network generate most of the traffic, or networks that are sparse. It is would be critical to know both the traffic requirements, and how the traffic propagates though the network. To further explain the *logical distance*, we first adopt the following notation:

$m_{ij} = m_{ji}$: logical distance between node i and node j

f_{ijp}: output from a SRA solution given traffic requirement matrix F; amount of flow on path p that passes through node i then node j divided by the number of links between i and j on that path

f_{ij} : summation of all f_{ijp} over p, or more specifically

$$f_{ij} = \sum_{p \in (P_r^o \cap P_{(i,j)})} f_{ijp}$$

$h_{ij} = h_{ji}$: shortest hop distance between node i and node j

h_{ij}': normalized h_{ij}

$t_{ij} = t_{ji} = f_{ij} + f_{ji}$ (f_{ij}, f_{ji} may be different)

t_{ij}': normalized t_{ij}

t_{max} : max of t values among all links

$$h_{ij}' = \frac{(d_{ij}-1)}{\sum_{l,m \in N} (d_{lm}-1)} \tag{2.1}$$

$$t_{ij}' = \frac{(t_{max} - t_{ij})}{\sum_{f,g \in N} (t_{max} - t_{fg})} \tag{2.2}$$

$$m_{ij} = \mu\, f_{ij}' + (1-\mu)\, h_{ij}' \qquad \mu \in [0,1] \tag{2.3}$$

The definition of logical distance m_{ij} ensures that the greater the total path flows passing through node i and j, and the smaller the shortest hop distance between nodes *i* and *j*, the smaller is m_{ij}. μ is a parameter which adjusts the weights between hop distances and adjusted total optimal flow term. It needs to be chosen at a level which ensures connectivity requirements among and within the cluster are satisfied. In our implementation we set the inter and intra-cluster connectivity (k_1 and k_2) toone. We compute the logical distance matrix, then use the *complete linkage hierarchical clustering* from IMSL to perform the clustering. After each clustering, we check if all the clusters are connected. In case any of the clusters aren't connected , the algorithm backtrack and seek for a different clustering solution.

Inter Cluster Routing using an SRA

After virtual clusters are formed, we develop a static optimal routing algorithm (SRA) to solve for the path-constrained routing problem defined by (1.6)-(1.10). The purpose here is to obtain delay minimizing paths which satisfy the new path constraint imposed by virtual clustering. The algorithm we developed is modified from Bertsekas-Gallager's [2] minimal-delay flow deviation algorithm (BG Algorithm). We propose a modified BG algorithm which generates *cluster constrained paths* when optimize routing. This *Cluster-Constrained Shortest Path (CCSP)* Algorithm finds cluster-constrained paths from a specific node, s to all destinations nodes.

BG algorithm is optimal under Kleinrock's independence assumption, Poisson packet arrivals, and exponential packet length. Nonetheless, it has been shown that violation of these assumptions to some degree do not significantly affect accuracy of the performance measure. Similar to many other optimality seeking algorithms [4],[5],[6],[7] BG algorithm is designed based on the notion of flow-deviation. Input to the algorithm includes a traffic requirement matrix F network topology $D(N,A)$ in the form of an $n \times n$ adjacency matrix, and an $n \times n$ capacity matrix, C (n is the number of nodes). The output of the algorithms is a path set $P_r^o \equiv \{P_r^{o1}, P_r^{o2}, \ldots\ldots. P_r^{oK}\}$ for each source-destination pair (s^r, d^r) and path flows associated with each path.

Flow deviation algorithms start with calculating feasible path flows and then move traffic from congested paths to less congested ones while maintaining feasibility. The idea behind BG algorithm is to move a small amount of flow (at a time) from a path with larger incremental delay to a path with smaller incremental delay then the average. The incremental delay is the sum of all incremental delays of the links that form the path. If the delay function of a link (i,j) is given by $d_{ij}(x_{ij})$ (x_{ij}: total flow on link (i,j)) the incremental delay on the link is:

$$I_{ij} = \lim_{\epsilon\to 0} \frac{d_{ij}(x_{ij}) - d_{ij}(x_{ij}-\epsilon)}{\epsilon} \qquad (2.4)$$

$d_{ij}(x_{ij})$ is total delay on the link observed by packets. If $d_{ij}(x_{ij})$ is a convex function then I_{ij} is well-defined and it is in fact the derivative of the delay with respect to the flow. Thus, I_{ij} is the change in the contribution of link (i,j) to the total delay. In order to move the flow from links that have larger I_{ij}'s to the ones that has smaller I_{ij}'s then total delay, BG Algorithm seeks to find a path where the total incremental delay of links that constitutes that path is less than the total incremental delays of current paths. This can be accomplished by finding a shortest path where I_{ij}'s are used as arc distances. Then, a small amount of flow is moved from nonshortest paths to the shortest path identified. The idea is that if flow is moved from a path with larger incremental delay to a path with smaller incremental delay then the total (hence the average) delay can be decreased. In our implementation of the BG algorithm, we assume all the queues in the network are M/M/1, thus we define I_{ij} as follows as suggested by Kershenbaum [13]:

$$I_{ij} = \frac{c_{ij}}{(c_{ij} - x_{ij})^2} \tag{2.5}$$

c_{ij}, x_{ij} is in terms of packets/sec. Given the distance matrix defined by I_{ij}'s, we then find the shortest paths from a specific source node to all destination nodes. For each (s^r,d^r) pair, we move a certain amount of flow δ from nonshortest paths to the shortest path. δ is given by the following expression:

$$\delta = \frac{\alpha(L_p - L_{p^*})}{H_p} \tag{2.6}$$

where p is a nonshortest path, p^* is the shortest path, L_p is the length of the path p, L_p^* is the length of the shortest path. α is a stepsize function, and H_p is the second derivative of the path length given as follows:

$$H_p = \sum_{(i,j)\in E} \frac{2c_{ij}}{(c_{ij} - x_{ij})^3} \tag{2.7}$$

where E is the set of links that are in p and p^* but not in both. It is recommended by Kershenbaum[13] that $1/N \leq \alpha \leq L/N^2$. L is the number of directed links. N is the number of nodes in the network. It is also recommended that α is set to a higher value for the first iterations of the algorithm, and it is to be decreased as the algorithm proceeds. In some cases, a shortest path can't increase its flow by δ due to its link capacity, and a nonshortest path can not decrease its flow by δ since its path flow is less than δ . In these cases, δ is adjusted heuristically.

After path flows are adjusted, the distances of the all links (l_{ij}'s) are recalculated. Based on these new link distances, shortest paths from another source to destination nodes are generated and the above process is repeated. For each source node the above computation is done sequentially. Until all the source nodes are completed, the algorithm then starts over with the first source node. The iterations continue, moving flows from nonshortest paths to shortest paths, until no more "significant" progress can be made. In our implementation, progress is measured based on the value of total delay (hence average delay). If the improvement on the delay between two subsequent iteration is less than a prespecified value, Δ, we terminate the algorithm.

BG algorithm generates an analytic solution for the routing problem. However the minimum average delay found by the algorithm is a theoretic approximation, not the actual delay of the network. To get the exact delay figure, we need to simulate the routing algorithm using traffic data. In Section 4 we describe a simulation implementation of the routing algorithm. In this paper, we use the static routing algorithm, or SRA, described above as the inter-cluster routing algorithm for CRA, and we use the algorithm as a benchmark for comparison. To use SRA for inter-cluster routing, we need to modify the shortest path algorithm to recognize the virtual clustering identified by the clustering algorithm. We now describe this specialized shortest path algorithm.

For a given network *D(N,A)*, l_{ij} is the length of a link *(i,j)*. *d(i)* vector shows the distance of a node from a source node. *cl[i][j]* is a matrix which denotes the forbidden clusters for node *i*. *cl[i][j]* =1 means that successor node of node i can be from cluster j. If it equals to 0, successor node of node i on the path, can not belong to cluster j . *clus(i)* denotes the cluster number of node *i*. *A(i)* denotes the set of outgoing links from node *i*. *pred(i)* is predecessor node of node *i*. *CL* is the set of clusters of network *D(N,A)* represented by numbers. We summarize CCSP as follows:

```
algorithm CCSP;
begin
        $S := \varnothing$;  $S^- := N$
        $d(i) := \infty$ for each node $i \in N$
        $d(s) := 0$ and $pred(s) := 0$;
      cl [i][j] = 1 for all $i \in N$ , and for all $j \in CL$
        while $|S| < N$ do
        begin
                let $i \in S^-$ be a node for which $d(i) = \min \{d(j): j \in S^-\}$;
                cl[i][clus(i)] :=0    // it makes the cluster forbidden for i
```

```
            $S := S \cup \{i\}$;
            $S^- := S^- - \{i\}$
            for each $(i,j) \in A(i)$ do
        begin
                if $\{d(j) > d(i) + l_{ij}$ and $clus(i) = clus(j)\}$ do
                begin
                    $d(j) := d(i) + l_{ij}$ ;
                    $pred(j) := i$;
                    cl[j][k]=cl[i][k] for all $k \in CL$
                                        // forbidden for i is forbidden for j
                end
                else if $\{d(j) > d(i) + l_{ij}$ and cl[i][clus(j)] = 1 } do
                                        // if cluster j is not forbidden for i

              begin
                    $d(j) := d(i) + l_{ij}$ ;
                    $pred(j) := i$;
                    cl[j][k]=cl[i][k] for all $k \in CL$
                    cl [j][clus(j)] := 0;

                end;
            end;
        end;
    end;
```

The algorithm starts from the source node, s. The cluster s belongs to clus(s) is declared "forbidden". For an adjacent node, $j \in A(i)$ of node s there are two conditions:

(1) j belongs to different cluster, or

(2) j belongs to the same cluster.

If j belongs to the same cluster and $d(j) > d(s) + I_{ij}$, j becomes successor of node s, $d(j)$ is set to $d(s) + I_{ij}$, and all forbidden clusters for s becomes forbidden for j also. On the other hand, if j belongs to a different cluster it is checked if j belongs to a forbidden cluster for s. From a non-labeled set S^- the node which has minimum delay is selected and procedure for s is repeated for the chosen node. The algorithm terminates when there are no unlabeled node left in the set, S^-.

CCSP uses a similar construct as the well-known Dijkstra's algorithm. When the algorithm terminate, the predecessor array defines a tree structure induced from the network. A main variation in CCSP is that even though an adjacent node j of a node i can hold the inequality $d(i) + I_{ij} \geq d(j)$ it may not be admissible on the path because it may belong to a forbidden cluster for node i. Note that CCSP is not an exact algorithm for the cluster-constrained shortest path problem. On the other hand, an exact solution to the problem does not necessarily have a tree structure. Computationally CCSP is proven to be a very effective heuristic although optimality can not be guaranteed. Occasionally infeasibilities may occur when virtual clustering constraints prohibits the access of some essential path, but this occurs rarely in the problems we tested. Since all the paths generated by the CCSP algorithm enforce the constraints imposed by virtual clustering, it generates a set of desirable paths given the current virtual clustering configuration.

Intra-Cluster Routing using DRA

After the (cluster-constrained) inter-cluster routing paths are found, the sequence of gate nodes for each packet to follow is completely defined. When a packet enters a specific cluster c, the entering and the exiting gate nodes become the source (s_c) and the destination nodes (d_c) for this packet in cluster c. The collection of all packets passing through cluster c form the traffic for the intra-cluster routing problem for c. To generate intra-cluster routing any existing dynamic routing algorithm (DRA) can be used. We implemented a dynamic shortest path algorithm (DRA) [8] which handles dynamically all the intra-cluster routing problems. In other words, when a data packet enters the gate node of a specific cluster, to reach the exit gate node of the same cluster, it follows the paths found by DRA at that point in time instead of following the static paths determined *a priori* by SRA. Thus, static optimal routing finds a priori the gate nodes sequence for each source-destination pair, while the dynamic routing algorithm finds the exact paths between entry gate node and exit gate nodes based on current traffic situations. The entry gate nodes and exit gate nodes of the clusters may be different for each path, and each source-destination pair.

Clearly SRA and DRA are both integral parts of CRA. On the other hand, SRA and DRA by themselves can be used to represent stand-alone static and dynamic routing algorithms, respectively. In Section 5, we compare CRA against "pure" SRA and "pure" DRA, where routing are generated for the entire network by *a priori* static routing, or by dynamic routing without using virtual clustering.

4. SIMULATING THE PERFORMANCE OF THE ROUTING ALGORITHMS

Routing procedures such as the BG algorithm uses a theoretical estimate of packet delay as its objective function. In order to compare different routing algorithms in a more realistic setting, we simulate and record the actual end-to-end delay of each and every packet generated for the network. In this section, we describe simulation procedures developed for SRA, DRA and CRA.

4.1 Simulating the Performance of SRA

Recall that for SRA the static optimal paths P_r^o, for each source destination pair (s^r, d^r) has to be known. Besides, we have to know network topolgy $D(N,A)$, capacities of the links matrix, C, and traffic requirement matrix, F. Traffic requirement denoted by f^r for (s^r, d^r), is the average number of packets per unit time(sec) that is generated to be transmitted from a source node s^r to a destination node d^r. Packet generation is a random process which we assume that it follows a specific distribution. f^r is the mean of that process.

At the beginning of the simulation we set the current time $t=0$. We create an event list which has a linked list structure. Every event represents a generation of a packet at a source node or a transmission of a packet on a link. Implemented as C++ objects, we define a class type "event" that represents these events. The data fields of these class objects (event) are as follows:

source: shows the source node that a packet is going to be generated

destination: shows the destination node that a packet is going to be sent

time: shows the number of time units from now the event is going to take place (in sec.)

aors: identify the event as arrival or service (“arrival” represents a packet generation event at a source node, while “service” represents the event of transmitting from one node to another)

node: if the event is a service event this field holds the node number that the packet is currently at, if the event is an arrival event this field is set to 0.

nextnode: if a service event this field shows the nextnode that packet at a node is going to be transmitted to.

next: this field holds the pointer to the next event.

At time 0 we generate only arrival events and we generate arrivals for all source-destination pairs and corresponding interarrival times. Then according to the time fields we sort all the events on the linked list (see Figure 4). Important variables used in the simulation are as follows:

Queuefirst(i)(j): a two dimensional array of pointers that point to the first packet at the queue of a link (i,j)

Queuelast(i)(j): a two dimensional array of pointers that point to the last packet at the queue of a link (i,j)

evpo: current event pointer

Totaldelay: total delay of the packets that are generated and arrived to their destinations. It is set to 0 at the beginning.

Averagedelay: total delay divided by the number of services. It is set to 0.

number of services: number of packet generated but also arrived to their destinations.

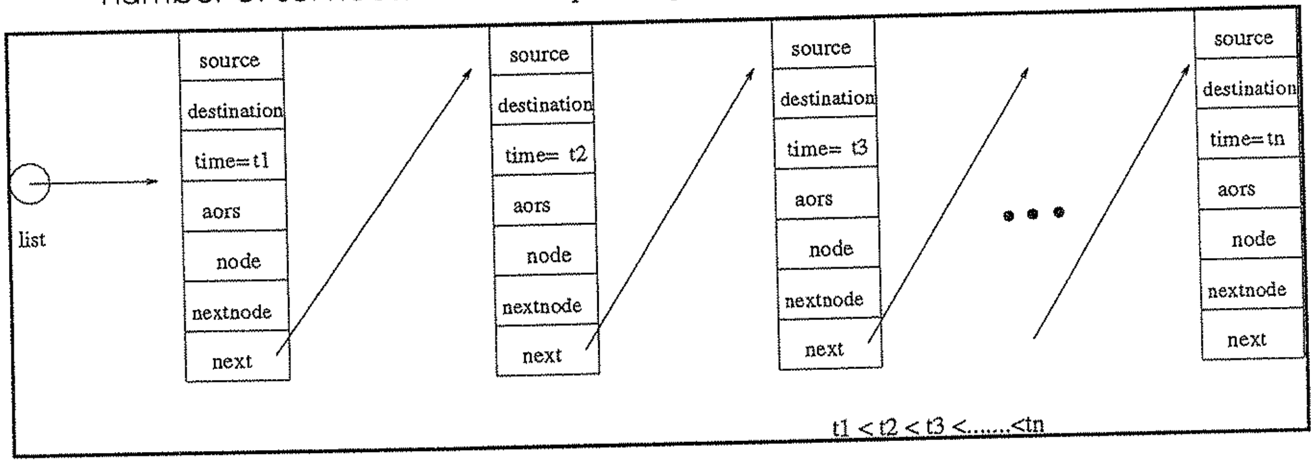


Figure 4. Event Lists in the Simulation

number of packets: number of packets generated

Currenttime: clock time of the simulation

In the initial event list there is no service event, but as the simulation time proceeds service events are generated and inserted to the event list according to their "time" fields. The following steps take place for each event on the event list based on its type (arrival or service).

The Arrival Events

If aors ='arrival'

1. Currenttime is increased by the amount of event's time.
2. Event's time is subtracted from all other events' times on the event list.

3. A packet is generated which has the following data fields and data fields of this packet are filled with appropriate values.

number of packets is increased by one.

ARRTime: generation time of the packet. It is set to the Currenttime:

Length: randomly generated which has a mean of 1. From random number stream by inverse transform technique this field is set to a random number that follows some distribution.

number: the number of the packet generated it is set to the number of packets

currentnode: this represents the number of the currentnode that packet is waiting at one of its outgoing link's queues.

destination: the packet's destination node . This is copied to the current event's destination.

currentlink: a pointer to the current link in the path structure. Whenever a packet moves to another node, currentlink advances to the other link on the path. First, path of the packet should be found.

nextpacket: pointer to the next packet. This is set to nil pointer. If there are no packets then this field is set to nil.

4. Appropriate paths are found for the packet generated. There may be only one path or more than one path (bifurcation). If there is more than one path, one path is selected as follows: suppose the total requirement is $f = p_1 + p_2 + p_3 + \ldots p_k$ where p_i is the flow of the ith path and K is the degree of bifurcation. Generate a uniform random number, u, and

if $u<p_1$ then path1 is selected

if $p_1 < u<p_1 +p_2$ then path2 is selected

.

.

if $p_1+ p_2 +.. +p_{K-1} <u< p_1+ p_2 +.. +p_K$ then path K is selected

5. From the path structure, firstlink of the path becomes currentlink

6. The packet generated is put to the end of the queue of the outgoing link (i.e., Queuelast(currentnode)(nextnode) points to the packet) if there is a queue (Figure 5). Its time is assigned to the packet length (length) divided by currentlink's capacity. Its node field is assigned to currentnode and nextnode field is assigned to nextnode. The event is then inserted to an appropriate position on the linked list based on its time.

7. Create the next arrival event for the same source-destination pair based on the traffic arrival process (i.e., based on the interarrival time). After initializing all the datafields, insert this event to the event list.

8. After processing the current event, delete the event from the event list and move on to the next event on the list. Update evpo.

The Service Events

if aors='service' then the following steps take place:

1. Advance Currenttime by the amount of the time of the event and this amount is subtracted from each event's time on the list.

The packet under consideration is to be transmitted from node to nextnode.

2. If nextnodeof the event is also the destination of the packet, then

2a) increase number of services by one.

2b) update Currenttime

2c) Delete the packet from the event list.

Otherwise, advance the packet's currentlink to the nextlink on the path. At this point, the packet is assumed to have reached the nextnode.

3. Delete the processed event from the list and advnace the list pointer evpo to the next event.

Events on the event list are processed and deleted while new events are generated and inserted. Up-to-minute record on packet delays (DD(n)) and number of services (number of services) are collected and updated. The simulation stop when Currenttime reaches to a predefined time, T. Also there is a transient stage at the beginning of the simulation when the system goes from empty to the point of steady-state. This transient period is typically excluded for performance evaluation purposes.

We test the above simulation procedure by comparing its results to the analytical values obtained by the BG algorithm. We complete the test using standard and generated test problems using different random number seeds, the interarrival distribution and packet length distribution are both exponential. As can be observed in Table 1 the delay calculated from the simulation is quite consistent with the analytic results. For low traffic levels where the network is lightly congested simulation gives higher delay measures, but for heavily congested networks like OCT network with the links having 71 packets/sec capacities, simulation results are slightly lower. This is parallel to the results obtained

TABLE 1. Comparison Between Analytic Delay Estimation and Simulated Results (Delays in milisec).

Type of Network	Analytic Delay	Simulated Results (for 10sec-30 sec period)				
		Seed No:1	Seed No:2	Seed No:3	Seed No:4	Average
RING	9.74141	10.081	9.55989	9.76579	9.88231	9.82224
ARPA	4.85349	5.3512	4.88806	5.04416	5.2849	5.14208
OCT congest.	220.04	213.531	201.405	193.985	216.794	206.428
OCT	56.8962	61.9384	57.8645	63.1224	62.565	61.3257
USA	5.47422	5.586	5.66631	5.49132	5.5124	5.5640
M61	36.2959	40.3639	39.5464	39.8142	40.2117	39.9840
M62	33.2022	34.4908	34.6594	34.9904	37.4618	35.4006
M63	42.7931	45.3714	46.3812	44.1212	46.1349	45.502
M64	51.199	55.236	54.2918	56.4514	57.4318	55.8527
M65	25.5363	27.5261	26.9147	28.1825	27.1675	27.1977
M31	42.6219	44.1256	43.9192	44.1056	45.1127	44.31578
M32	70.7229	76.2514	73.3241	74.273	75.1094	74.7394
M33	60.512	65.1224	64.6326	64.2536	68.3527	65.59032
M34	56.6035	60.1571	59.1442	59.9138	60.1225	59.8344
M35	73.7779	76.3543	78.1413	78.1676	77.1524	77.4539

in the literature [2].

4.2 Simulating the Performance for DRA and CRA

Basic elements of simulation procedures used in DRA and CRA is nearly the same as the SRA's simulation procedure. The difference is in finding the nextnode for a packet at a specific node to arrive its destination. For DRA, we need to define additional variables, and extend the data fields for packet objects. Additional important variables for the simulation are:

Delay(i)(j): average delay of packets passing through link(i,j).

S(i)(d): It holds the nextnode information for packets that are currently at node i, to reach destination node d. This matrix is updated for each node i frequently, we will update every 10 sec.

NP(i)(j): statistics that shows the number of packets passing through link (i,j).

Additional field for a packet object are NodeARR and NodeDPR. Node ARR keeps track of the packet's arrival time to a node. NodeDPR is used to define packet's arrival time to the next node. (NodeDPR - NodeARR) is the total amount of time a packet spends at a specific node.

We also need a new type of event "update." Beginning from node 1 an update event is created for each node in the network. For a node *i*, the event's node is set to *i* and its time field is set to 10 + *(i-1)/N*. When the turn comes to an update event, another event having same node field is created. Its time is set to 10. So, for node 1, time epochs that update is done is (10, 20, 30,.......) and for node i (10+(i-1)/N, 20+(i-1)/N, 30+(i-1)/N,.........). All 'update' events are inserted to the event list.

For an arrival or a service event, the simulation steps are similar to that of SRA described in section 4.1except the following adjustments:

- When a packet is generated there is no need to set its currentlink field since nextnode that a packet at a specific node *i* is going to visit is obtained from S(i)(d) not from optimal path. So it's set to 0.

- Whenever a service event is created nextnode for a packet under consideration is defined by the current S(i)(d) (i: current node d: destination node of the packet).

-Whenever a packet is generated its NodeARR is set to the Currenttime.

-Whenever a packet is transmitted from node *i* to node *j* its NodeDPR is set to Currenttime. The amount in NP(i)(j) is increased by 1 and packet's time delay (NodeDPR-

NodeARR) is added to Delay(i)(j). Packet's NodeARR is set to the Currenttime.

For an update event the following steps take place:

1. Average delay for the outgoing link's of the event's node is recalculated. (Avdelay(i)(j)= Delay(i)(j)/NP(i)(j) for all j ∈ A(i)) where *i* is the event's node, *A(i)* is the set of nodes adjacent to node *i*

2. Shortest paths for all-source destination pairs are recalculated based on updated average delay values. (For node i they are currently updated)

3. Based on these shortest paths S(i)(j) for all i ∈ N and j ∈ N is updated.

4. For current node *i*, Delay(i)(j) and NP(i)(j) for all j ∈ A(i) is set to 0. (Avdelay(i)(j) remains the same until the next update. Meanwhile new Delay(i)(j) and NP(i)(j) will be calculated.

5. Another update event is created for the node, its time is set to 10 and it is inserted to the event list.

For the first 10 sec, since there is no Delay value statistics, S[i][j] matrix is obtained using shortest hop distances. At time t=10 all of the delay statistics obtained are used to generate corresponding shortest paths.

CRA uses cluster-constrained optimal path set instead of the set used for SRA and also clustering structure has to be known in advance. We keep track of this structure by clus(i) for i∈N. Additional variables, data fields for packet object and additional event type 'update' used for DRA is also used for CRA implementation. But;

-currentlink data field of a packet object is important because sometimes nextnode calculations are done based on it (for intercluster links) so we keep track of it and it advances using clus-constrained optimal path structure.

-Two data fields cno and cdest which shows the current cluster number and current cluster destination is added to the packet class.

-Whenever an 'update' event is processed, shortest paths only within the clusters are defined and found and S[i][j] calculations are done based on these shortest paths assuming clusters of the networks as seperate networks. (i.e. S[i][j] where clus[i]≠clus[j] is set to -1 which shows it is not defined) Also S[i][d] is found in such a way that d represents cluster's destination.

-Whenever a packet is generated a clus-constrained path is assigned to it and its cluster destination cdest is found based on this path and clustered structure. Until it reaches to that destination it uses S(i)(cdest) matrix to find the nextnode that has to be visited. (i: current node). When it reaches, its nextnode is obtained from currentlink data. (So intra-cluster routing is dynamic whereas inter-cluster routing is not). Nextnode is an element of different cluster so a new cdest has to be found also cno of the packet has to be updated. Final cluster destination is global destination for the packet.

5. COMPUTATIONAL TESTING

To compare the performance of the proposed clustered routing algorithm (CRA) with the conventional routing algorithms, (SRA) and (DRA) we implemented all the algorithms and the discrete event simulation in C++ on a IBM RISC 6000 workstation. The discrete event simulation model specifies the specific implementation of the algorithms under realistic operational environment using different test problems. In the following sections we describe the testing environment and the computational results.

5.1 TESTING ENVIRONMENT

To compare CRA with other algorithms DRA, and SRA, we designed a simulation experiment using 20 test problems. We use the 50 node random geometric networks generated by Bartolacci and Wu[1]. But instead of homogenously capacitated networks we used heterogenously capacitated networks which we randomly alternate the capacities between: 155 and 185 packets/sec, and 235 and 270 packets/sec. One of the factors effecting the result of CRA is the clustering structure and number of clusters, Γ. We perform a preliminary testing investigating the effect of the number of clusters Γ and clustered structure. We found that neither factors have a significant impact unless Γ is less than a certain number. In our experiments we set Γ to 5 . For CRA, we used the shortest hop distances as the distance matrix for the sake of simplicity and we use the complete linkage algorithm from the IMSL library as the clustering algorithm.

Furthermore, connectivity, congestion, and size of the network are among the factors that may have an effect on the result. Five of the networks generated by Bertolacci and Wu [1] has an average connectivity of 6 and the other five of them have 3 . For highly connected networks there are more alternative paths for source-destination pairs. Traffic congestion is high for low capacitated networks where oscillation and loop problems increases for dynamic algorithms and congestion is low for high-capacitated networks. As mentioned above we tested both cases. Also we set the size of the networks tested to 50. In summary, there are 20 test problems:

1. 5 Highly connected (Degree of Connectivity=6), Heavily congested (Capacities:155, 185 packets/sec), 50 node networks: M6H1, M6H2, M6H3, M6H4, M6H5
2. 5 Highly connected (Degree of Connectivity=6), Lightly congested (Capacities:235, 270 packets/sec), 50 node networks: M6L1, M6L2, M6L3, M6L4, M6L5
3. 5 Lowly connected (Degree of Connectivity=3), Heavily congested (Capacities:155, 185 packets/sec), 50 node networks: M3H1, M3H2, M3H3, M3H4, M3H5
4. 5 Lowly connected (Degree of Connectivity=3), Lightly congested (Capacities:235, 270 packets/sec), 50 node networks: M3L1, M3L2, M3L3, M3L4, M3L5

Traffic arrival process and distributions of packet lengths are also important factors. We tested three combinations of traffic arrival process and packet lengths:

Process 1: Homogenous Poisson arrivals (traffic generated at the source node) with $\lambda= 0.25$ packets/sec for each source destination process and exponentially distributed packet length

Process 2. Uniform arrivals (mean of arrivals is 0.25 packets/sec), uniformly distributed packet length

Process 3. Nonhomogenous Poisson arrivals (rate of the traffic changes accordingly to the currenttime of the simulation. Figure 5 shows rate-time graphic of the process) , exponentially distributed packet length

We believe that Process 3 is the most representative of the real world and it is also the type of traffic CRA is designed to deal with. In most network operations, traffic rates varies during the course of the day. For example after 12:00am at night there is less connection to the networks, but at noon perhaps the rate is at its maximum value. We attempt to represent this traffic characteristics using nonhomogenous Poisson arrivals.

Other simulation parameters we used in our experiments:

Simulation Length= 150 sec. (Nearly 100000 packets generation takes place during this period. We believe that it is long enough for an accurate estimate)

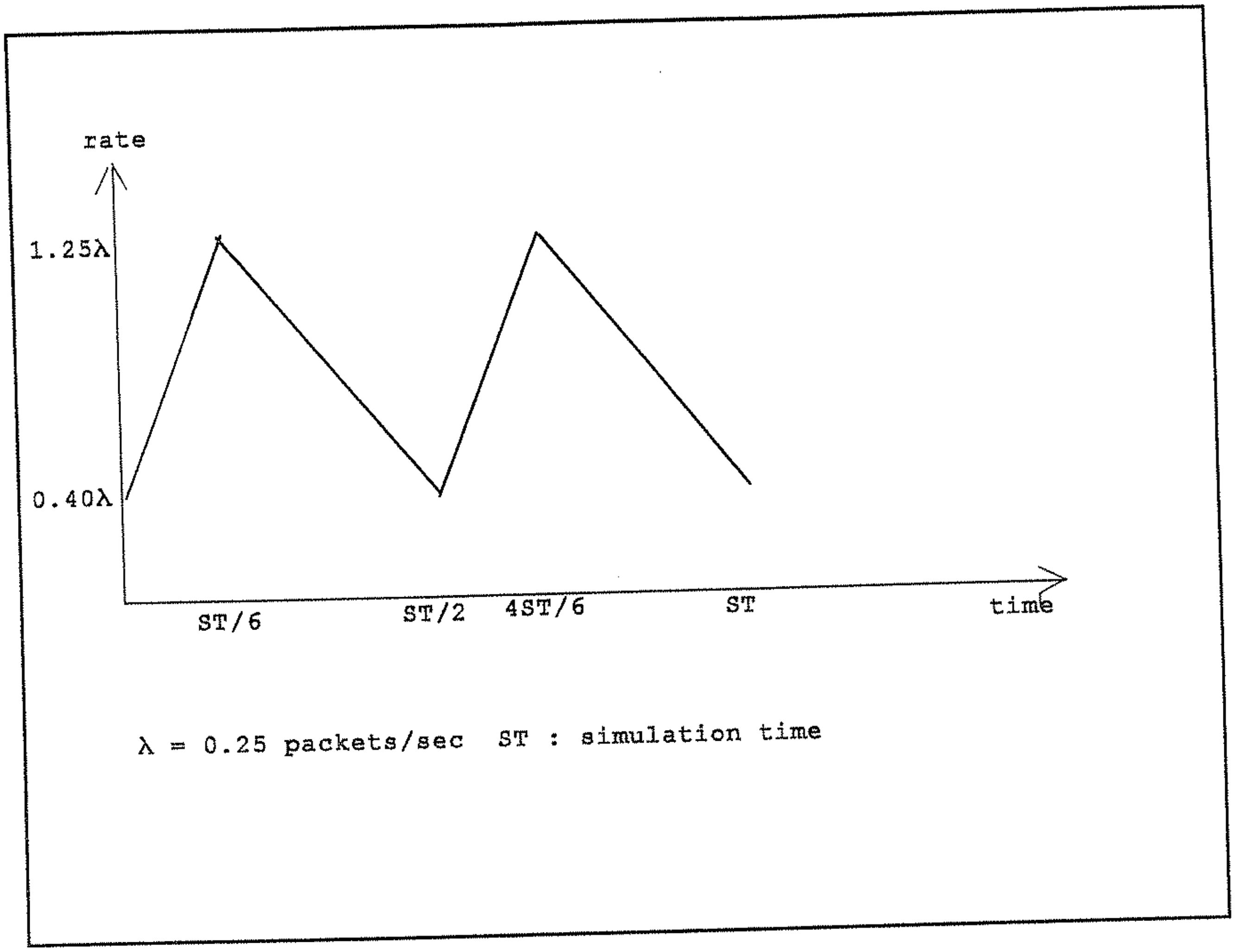


Figure 5. Nonhomogeneous Poisson Arrival

Period statistics of delay is collected= 20 sec - 150 sec. (we empirically determine that the system reaches steady state after 20 second)

# of replications: We used 5 different random number streams represented by seed numbers.

20 networks with 3 different traffic generation process were tested using 3 different routing algorithms. Each test was repeated 5 times using different random number streams. So 900 testings were done on IBM/RS 6000 computers. We tabulated the results.

5.2 Computational Results

For each test problem and random number stream we labeled the algorithm that gives the minimum delay, we then calculated the percent deviation from the minimum for each algorithms. This is calculated as follows:

$$(\%\text{ from best})_k = \left\{ \frac{z_k - Min_j\{z_j\}}{Min_j\{z_j\}} \right\} \times 100\%$$

Results are given by Tables 2-4. A few observations can be made from the tables,

1. For nonhomogeneous traffic CRA outperforms both SRA and DRA which is consistent with the expectation. However, notice when the connectivity is low and the traffic congestion is high (as is the case for problem instances M3H1, M3H2, M3H3, M3H4 andM3H5) CRA does not perform as well. This is quite reasonable since in sparsely connected network virtual clustering has less opportunity to effect the traffic. Also notice that in the combination of sparseness and high traffic intensity creates much higher variations in performance across the three methods..

2. In highly-connected networks under nonhomogeneous and heavily congested traffic (As is the case in M6H1, M6H2, M6H3, M6H4, M6H5) CRA gives the most superior results. For lightly loaded traffic (M6L1- M6L5 and M3L1- M3L5) CRA provides high performance in most instances

3. For homogenous Poisson traffic and Uniform traffic CRA is often outperformed by SRA. This is to be expected since the traffic does not vary significantly over time and the static routing algorithm provide near-optimal results. However, the difference between CRA and SRA in these cases are not significant. On the other hand, CRA is superior to DRA in nearly for all cases. Especially for

TABLE 2a) NONHOMOGENOUS TRAFFIC ARRIVALS, EXPONENTIAL PACKET LENGTHS

Test Problems	Algorithm	% from best					
		seed 1	seed 2	seed 3	seed 4	seed 5	average
M6H1	SRA	44.75	56.61	0.59	0.00	11.71	22.5
	DRA	12.02	33.72	19.96	4.93	33.25	20.4
	CRA	0.00	0.00	0.00	0.94	0.00	0.00
M6H2	SRA	9.37	9.98	10.77	9.38	9.42	9.78
	DRA	24.34	26.14	26.36	21.71	23.38	24.38
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M6H3	SRA	25.34	73.13	5.09	28.69	79.13	38.6
	DRA	31.04	81.67	68.39	24.67	75.20	52.62%
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M6H4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	333.00	639.30	236.90	368.40	263.60	367.00
	CRA	96.78	74.72	83.60	232.40	90.25	116.00
M6H5	SRA	0.85	3.67	1.09	2.94	2.41	2.19
	DRA	8.63	6.24	2.50	0.96	0.96	4.86
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M6L1	SRA	7.24	8.01	7.85	6.45	4.55	6.81
	DRA	14.79	15.74	14.90	13.19	11.72	14.0
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M6L2	SRA	2.19	1.57	3.73	3.06	2.06	2.51
	DRA	6.02	6.85	4.71	6.58	5.25	5.89
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M6L3	SRA	5.79	5.13	6.19	4.49	4.57	5.23
	DRA	16.81	16.89	18.58	15.16	15.72	16.6
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M6L4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	36.14	36.51	22.68	32.17	38.61	33.2
	CRA	6.04	5.88	4.48	2.14	2.53	4.21
M6L5	SRA	1.03	3.36	0.80	3.04	2.18	2.08
	DRA	15.39	15.51	13.36	12.86	14.43	14.30
	CRA	0.00	0.00	0.00	0.00	0.00	0.00

TABLE 2 b) NONHOMOGENOUS TRAFFIC ARRIVALS, EXPONENTIAL PACKET LENGTH

Test Problems	Algorithm	% from best					
		seed 1	seed 2	seed 3	seed 4	seed 5	average
M3H1	SRA	21.14	6.71	0.00	24.27	0.00	0.00
	DRA	0.00	0.00%	54.54	0.00	10.11	2.19
	CRA	22.38	7.60	0.86	25.55	2.32	1.18
M3H2	SRA	6.50	0.00	30.05	0.00	32.43	9.55
	DRA	0.00	0.53	0.00	15.95	0.00	0.00
	CRA	10.31	3.78	70.26	3.51	37.48	20.2
M3H3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	23.79	33.61	64.73	1.73	28.48	29.9
	CRA	70.63	8.96	20.37	64.56	34.21	39.0
M3H4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	80.80	103.6	114.0	96.41	93.62	97.
	CRA	82.53	66.58	145.1	142.3	62.93	99.7
M3H5	SRA	3.51	7.58	4.31	11.12	6.97	6.70
	DRA	2.66	25.18	20.85	9.67	7.80	13.1
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M3L1	SRA	2.31	4.52	4.31	4.02	2.44	3.51
	DRA	7.81	12.40	13.05	10.07	8.46	10.
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M3L2	SRA	0.77	1.66	1.80	0.03	1.03	1.06
	DRA	6.59	6.68	4.98	6.70	6.59	6.31
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M3L3	SRA	6.77	6.44	3.73	5.81	1.34	4.78
	DRA	28.44	26.81	24.71	22.44	20.64	24.5
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M3L4	SRA	11.35	5.11	13.11	12.15	10.18	10.3
	DRA	21.81	23.39	28.94	26.87	21.80	24.5
	CRA	0.00	0.00	0.00	0.00	0.00	0.00
M3L5	SRA	2.98	3.35	6.64	3.07	2.57	3.71
	DRA	2.52	1.11	1.79	2.07	2.05	1.91
	CRA	0.00	0.00	0.00	0.00	0.00	0.00

TABLE 3a) UNIFORM TRAFFIC ARRIVALS, UNIFORM PACKET LENGTH

Test Problems	Algorithm	% from best					
		seed 1	seed 2	seed 3	seed 4	seed 5	average
M6H1	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	4.15	2.41	4.03	7.08	3.21	4.17
	CRA	0.75	0.51	0.87	0.43	0.89	0.69
M6H2	SRA	0.00	0.00	0.06	0.00	28.21	0.00
	DRA	26.78	19.06	0.00	19.47	0.00	5.85
	CRA	9.24	12.33	4.12	8.57	38.60	8.31
M6H3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	4.61	3.46	3.65	4.61	4.00	4.06
	CRA	2.52	2.39	2.14	2.21	1.91	2.23
M6H4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	200.2	127.9	23	224.0	407.8	240.
	CRA	47.74	50.49	50.29	77.09	69.56	59.0
M6H5	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	11.86	10.53	11.95	11.42	10.59	11.2
	CRA	0.87	1.01	1.21	1.26	1.28	1.12
M6H1	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	2.33	1.49	1.49	1.77	1.02	1.62
	CRA	1.15	0.61	0.81	0.82	0.74	0.82
M6L2	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	2.74	2.10	2.65	3.02	2.83	2.67
	CRA	1.57	2.04	1.55	1.51	1.42	1.62
M6L3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	6.56	6.06	5.99	7.10	6.57	6.46
	CRA	3.74	3.73	3.09	2.91	4.21	3.54
M6L4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	15.16	15.08	16.29	17.85	16.66	16.2
	CRA	7.65	7.00	7.50	7.57	8.22	7.59
M6L5	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	3.22	3.10	4.04	3.49	4.54	3.68
	CRA	0.54	0.48	0.58	0.48	0.54	0.53

TABLE 3b) UNIFORM TRAFFIC ARRIVALS, UNIFORM PACKET LENGTH

Test Problems	Algorithm	% from best					
		seed 1	seed 2	seed 3	seed 4	seed 5	average
M3H1	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	0.10	4.28	1.36	1.21	0.53	1.50
	CRA	0.31	0.32	0.34	0.07	0.22	0.25
M3H2	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	1.46	1.12	2.68	7.93	1.74	2.98
	CRA	1.50	1.66	20.83	1.47	1.38	5.33
M3H3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	11.11	16.81	10.56	11.61	10.48	12.12
	CRA	10.97	9.79	11.60	8.09	9.67	10.01
M3H4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	50.93	51.28	46.56	49.80	61.27	51.97
	CRA	25.64	25.48	24.94	26.42	28.94	26.28
M3H5	SRA	2.25	0.10	0.93	0.63	0.68	0.92
	DRA	0.00	0.00	0.00	0.00	0.00	0.00
	CRA	2.27	0.37	1.29	0.89	0.73	1.11
M3L1	SRA	0.00	0.00	0.00	0.00	0.05	0.00
	DRA	1.52	2.42	1.49	1.79	1.56	1.75
	CRA	0.19	0.03	0.40	0.51	0.00	0.21
M3L2	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	1.98	1.36	1.54	2.81	1.47	1.83
	CRA	0.93	0.91	0.99	0.90	1.40	1.03
M3L3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	7.91	6.28	6.68	7.51	6.54	6.98
	CRA	4.54	5.64	5.18	5.88	5.12	5.27
M3L4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	3.84	3.64	3.48	3.46	2.75	3.44
	CRA	3.60	3.67	3.67	3.47	3.83	3.65
M3L5	SRA	0.00	0.61	0.02	0.18	0.00	0.06
	DRA	0.52	0.00	0.00	0.00	0.00	0.00
	CRA	0.96	0.41	0.76	0.49	0.46	0.51

TABLE 4a) HOMOGENOUS TRAFFIC ARRIVALS, EXPONENTIAL PACKET LENGTH

Test problem	Algorithm	% from best					
		seed 1	seed 2	seed 3	seed 4	seed 5	average
M6H1	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	15.10	16.45	3.60	10.74	41.32	17.5
	CRA	5.07	4.41	5.10	7.23	6.84	5.75
M6H2	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	15.35	18.90	12.88	14.18	14.98	15.26
	CRA	8.03	10.49	8.34	11.12	9.52	9.50
M6H3	SRA	0.00	0.00	6.87	11.21	0.00	0.00
	DRA	84.93	17.87	0.00	0.00	34.68	18.0
	CRA	26.58	18.55	22.92	27.14	25.03	18.9
M6H4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	478.2	647.3	432.8	486.2	509.9	510.38
	CRA	246.7	124.1	158.5	231.0	195.	191
M6H5	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	42.77	39.39	48.42	40.57	65.86	47.3
	CRA	6.40	5.84	6.64	6.18	6.98	6.41
M6L1	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	10.04	8.98	7.11	6.66	9.59	8.47
	CRA	3.20	3.19	3.53	4.11	3.90	3.59
M6L2	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	7.98	10.06	7.09	8.42	9.13	8.53
	CRA	4.62	4.50	5.54	6.71	5.90	5.45
M6L3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	23.81	22.64	21.98	23.08	23.00	22.9
	CRA	10.54	11.32	12.15	11.22	10.65	11.1
M6L4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	62.07	66.42	56.62	67.48	61.22	62.7
	CRA	31.48	29.52	26.89	34.59	28.68	30.2
M6L5	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	13.33	17.06	13.27	13.66	14.87	14.4
	CRA	4.31	1.60	2.01	2.61	2.21	2.55

TABLE 4b) HOMOGENOUS TRAFFIC ARRIVALS, EXPONENTIAL PACKET LENGTH

Test Problems	Algorithm	% from best					
		seed 1	seed 2	seed 3	seed 4	seed 5	average
M3H1	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	6.10	5.62	9.44	2.31	18.52	8.43
	CRA	6.66	6.59	4.04	4.75	3.18	5.04
M3H2	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	15.32	27.60	5.90	19.54	13.85	16.36
	CRA	5.96	5.72	6.87	6.13	4.99	5.94
M3H3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	17.47	22.91	30.57	22.94	23.58	23.43
	CRA	14.49	41.80	72.94	84.93	18.36	46.39
M3H4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	697.0	240.7	137.2	163.0	223.9	292.3%
	CRA	77.15	69.77	99.75	82.44	70.96	80.04
M3H5	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	4.07	16.99	3.98	6.10	5.91	7.39
	CRA	5.00	5.95	10.17	6.64	11.30	7.83
M3L1	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	8.05	5.89	6.92	5.74	11.85	7.68
	CRA	6.47	4.61	4.20	5.00	5.50	5.15
M3L2	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	5.90	8.79	4.71	5.27	7.47	6.43
	CRA	4.34	3.78	4.57	4.05	3.26	4.00
M3L3	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	17.67	20.52	19.19	20.33	19.04	19.35
	CRA	12.00	12.50	10.40	14.72	10.97	12.12
M3L4	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	15.63	13.74	13.29	12.27	13.77	13.74
	CRA	13.85	13.68	14.75	13.95	14.45	14.14
M3L5	SRA	0.00	0.00	0.00	0.00	0.00	0.00
	DRA	1.25	4.73	2.43	1.97	2.70	2.61
	CRA	3.25	2.67	2.84	2.53	2.82	2.82

highly-congested cases DRA performs very poorly compared to SRA and CRA. If one consider CRA as a dynamic routing algorithm which uses virtual clustering as a means of processing, this preprocessing appears to be very effective. This supports strongly the idea of using virtual clustering to balance and regulate global network traffic while using decentralized dynamic method for real-time routing.

4. In addition to mean values in delay we also collect the variance data from the test instances [15]. We observe a high correlation between the mean and the variance of delay figures. Variance of the delay is high especially for heavily congested networks. It appear that there is a point of saturation in network capacity. Before this point is reached delay increases smoothly and variance is comparatively low. But once the saturation point is reached the level of delay increases sharply. However, since there is no significant difference in variances among different algorithms for the same delay levels, we do not show the variance figures in the results for the interest of briefness.

5.The performance of CRA over all traffic patterns appear to be quite robust. This performance can be further improved in practice since in our experiments we fix the number of clusters Γ to 5. One may choose to adjust Γ over time and make CRA abitrarily closer to SRA or DRA.

5. CONCLUSIONS

We propose a new routing algorithm for telecommunication networks which reconfigures the network topology by means of *virtual clustering*. We found that this method balances and regulates the global network traffic while allowing flexibility for real-time decentralized routing. Given a virtual clustering we identify near-optimal static paths for inter-cluster traffic using a flow-deviation algorithm. A dynamic routing algorithm is used to handle fluctuations over time for intra-cluster traffic. We compare this routing scheme with pure static and dynamic routing algorithm using discrete-event simulation. Test results show that the proposed algorithm outperforms the other algorithms for nonhomogenous network traffic where traffic rates between source-destination pairs change by the course of the simulation time.

The proposed approach can be applied to other stochastic network routing problems similar to telecommunication networks. By clustering individual nodes variance in traffic flow tend to reduce which make analytic optimization models more suitable. In this paper we tested only one particular pattern of nonhomogenous traffic. Different types of traffic patterns can be experimented to further study the performance of this routing scheme. We believe that testing routing algorithms under nonhomogenous and nonstationary traffic processes has significant practical importance.

REFERENCES

1. Bartolacci M.R., Wu S.D. (1998), "A Virtual Clustering Approach for Routing Problems in Telecommunication Networks," *INFORMS Journal on Computing,* Vol.10, No.1, Winter 1998, pp.12-24.

2. Bertsekas D., Gallager R. (1987), *Data Networks,* Prentice-Hall, Englewood Cliffs, NJ

3. Kleinrock, L. (1964), *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill, New York

4. Fratta L., Gerla M., Kleinrock L.,(1973), "The Flow Deviation Mehod: An Approach to Store-and Forward Communication Network Design", *Networks*, Vol. 3, pp. 97-133

5. Gafni E., Bertsekas, D. and Gallager R.(1984), " Second derivative algorithms for minimum delay distributed routing in networks," *IEEE Trans on Communications*, Vol. 32, pp 911-914

6. Cantor D.G., Gerla M. (1974), " Optimal Routing in a Packet-Switched Computer Network" , *IEEE Trans on Computers*, Vol. c-23, pp.1062-1069

7. Mahey P., Ouorou A., LeBlanc L., Chifflet J.(1995), "A New Proximal Decomposition Algorithm for Routing in Telecommunication Networks",*SIAM Journal on Optimization* Vol. 5, pp 361-373

8. McQuillan, J.M., Richer I., Rosen E.(1980)," The New Routing Algorithm for the ARPANET", *IEEE Transactions on Communications*, Vol. COM-28, No.5, pp711-719

9. Kleinrock, L., and Kamoun F., (1977), "Hierarchical routing for large networks", *Computer Networks*, Vol.1 pp. 155-174

10. Anthonio, J., Huang G., and Tsai W.K. (1992), "A Fast Distributed Shortest Path Algorithm for a Class of Hierarchically Clustered Data Networks", *IEEE Transactions on Computers*, Vol. 41, No.6, pp. 710-723

11. Muralidhar K., Sundareshan M. (1983), " A Hierarchical Scheme for Multiobjective Adaptive

Routing in Large Communication Networks", *Proceedings of the IEEE*, Vol. 71, No. 12, pp. 1461-1463

12. Boorstyn R., and Adam Livne(1983), " A Technique for Adaptive Routing in Networks", *IEEE Transactions on Communications*, Vol. COM-20, No. 4, pp. 474-480

13. Kershenbaum A., (1993), *Telecommunications Network Design Algorithms*, McGraww Hill, New York

14. Stallings W., (1994), *Data and Computer Communications*, Macmillan Publishing Company, New York

15. Golbasi, H. and Wu, S.D. (1996), "A Path Constrained Approach to Dynamic Network Routing: Virtual Clustering and Flow Deviation Algorithms," *IMSE Technical Report 96T-009,* Lehigh University, Bethlehem Pennsylvania.