ORSA Journal on Computing
Vol. 4, No. 3, Summer 1992

0899-1499/92/0403-0250 $01.25


# A New Multiplier Adjustment Procedure for the Distributed Computation of Routing Assignments in Virtual Circuit Data Networks

FRANK Y. S. LIN / *Bellcore, 3 Corporate Place, PYA2J318, Piscataway, NJ 08854, lin1@moscow.cc.bellcore.com*

JAMES R. YEE / *Department of Electrical Engineering, University of Hawaii at Manoa, Honolulu, HI 96822, jyee@wiliki.eng.hawaii.edu*

(Received: November 1990; final revision received: April 1991; accepted: October 1991)

**In this paper, the routing problem in virtual circuit networks is considered. In virtual circuit networks, all of the packets in a session are transmitted over exactly one path established between the origin and the destination. We consider the problem of choosing a path for each origin-destination pair so as to minimize the average number of packets in the network. We consider the formulation of this problem as a nonlinear multicommodity flow problem with integer decision variables.The emphasis of this work is to develop a distributed algorithm to solve this optimization problem. The basic approach is Lagrangean Relaxation. We introduce a new multiplier update rule which facilitates the solution of the nonlinear integer programming problem using distributed computation. In computational experiments, our proposed distributed algorithm determines solutions that are within 1% of an optimal solution in less than 2 minutes of CPU time for networks with 26 to 81 nodes. In addition, the proposed multiplier adjustment procedure provides better bounds and is less sensitive to the algorithm parameters than the subgradient method. An analysis of the communication delays of the control messages needed to support a distributed implementation is given.**

Computer communications networks play an important role in satisfying our communication and computational needs. The applications of computer communications networks include electronic mail, telephone networks, cellular phone systems, airline reservation systems, automated teller machines, tactical military $C^3$ systems, etc.. In order for a computer network to operate efficiently and reliably, it is essential that the routing algorithm be carefully designed. For this reason, the routing problem has been studied intensively.[27, 28]

An overwhelming majority of the research literature on routing assumes datagram service. Two primary reasons for this are (i) the ARPANET,[22] which is a datagram network, inspired a large amount of the research and (ii) the routing problem in datagram networks can be solved by utilizing many standard convex programming techniques.[1, 5] In datagram networks, the packets sent for a particular user pair may be routed over different paths so that the decision variables are continuous variables. In contrast, all of the packets for a particular user pair in virtual circuit networks are transmitted over exactly one path. Consequently, the routing problem in virtual circuit networks is a combinatorial optimization problem which is a more difficult problem. Many networks of today (e.g., SNA,[13, 18] TYMNET,[26, 35] TELENET[14] and TRANSPAC[3]) are virtual circuit networks. Furthermore, networks of the future such as ISDN will provide virtual circuit service.[11] A major advantage of virtual circuit service over datagram service is that all packets in a particular session arrive in the same order in which they are sent. In datagram networks, packets for a user pair usually arrive out of order. For these reasons, we focus on virtual circuit networks in this paper.

A centralized routing algorithm is one where all of the computations are performed at one central node. A distributed routing algorithm is one where all of the nodes in a coordinated fashion collectively solve a common optimization problem. For determining routing assignments during the operation of a network, a distributed algorithm has many advantages over a centralized algorithm. First, a distributed algorithm is more reliable. The failure of the central node when using a centralized algorithm causes the entire network to become dysfunctional. The failure of a node when using a distributed algorithm only causes the failure of a small portion of the network. Second, the use of a centralized algorithm requires the transmission of more control messages. This is due to the transmission of messages describing the network status of each node to the central node and the transmission of the routing assignments from the central node to each of the other nodes. In a distributed algorithm, control messages (which implicitly describe the network status many hops away) are transmitted between neighbors. There are other advantages, but the first one is sufficient for many researchers to argue that a distributed algorithm is essential for use in an operational network. In this paper, we will consider only distributed algorithms.

Although virtual circuit networks are common, the number of papers on routing in virtual circuit networks is small relative to the number of papers on routing in datagram networks. Segall[29] formulated the routing problem in virtual circuit networks as a convex programming problem and extended Gallager's algorithm[6] to develop a dis-

*Subject classifications:* Communications: virtual circuit data networks, routing, distributed computation. Networks/graphs: multicommodity flows.

tributed routing algorithm. Courtois and Semal[2] modified Fratta, Gerla and Kleinrock's Flow Deviation method[5] to develop a heuristic routing algorithm for virtual circuit networks. The Flow Deviation method is the Frank-Wolfe method specially tailored to solve an uncapacitated multicommodity flow problem with a convex objective function. They obtained solutions that are within 3% (on the average) of the optimal solutions for lightly loaded networks. Gavish and Hantler[8] first formulated the problem of optimal route selection in virtual circuit networks to minimize average delay as a nonlinear multicommodity flow problem with 0–1 decision variables. They applied Lagrangean Relaxation and the subgradient optimization method to develop a centralized algorithm. Their computational results showed that this algorithm is effective in finding good feasible solutions and determining tight lower bounds on the minimal expected delay. Narasimhan, Pirkul and De[24] reformulated the problem considered in [8]. Their formulation facilitated a new Lagrangean relaxation which resulted in tighter bounds.

In solving the dual problem, Gavish and Hantler[8] used the standard update rule[17] for the dual variables. A distributed implementation of this multiplier update rule would result in more overhead messages than desired. The main contribution of this paper is to introduce a new multiplier adjustment procedure which requires far fewer overhead messages in a distributed implementation. In addition, the distributed routing algorithm with our multiplier update rule is less sensitive to the algorithm parameters and finds slightly better lower bounds on the optimal solution than the usual multiplier update rule for most of the computational experiments.

The remainder of this paper is organized as follows. A formulation of the routing problem as a nonlinear combinatorial optimization problem is presented in Section 1. In Section 2, a Lagrangean Relaxation approach to the problem is presented. In Section 3, we introduce a distributed multiplier adjustment procedure. Computational results are reported in Section 4. In Section 5, we analyze the performance of the distributed routing algorithm and discuss several implementation issues.

## 1. Problem Formulation

Basically the formulation presented in [8] is adopted in this paper. A brief review of the formulation is given below. A virtual circuit communications network is modeled as a graph where the processors are represented by nodes and the communication links are represented by arcs. Let $V = \{1, 2, \ldots, N\}$ be the set of nodes in the graph and let $L$ denote the set of directed arcs in the network. Let $W$ be the set of origin-destination (O-D) pairs (commodities) in the network. For each O-D pair $w \in W$, the arrival of new traffic, which may consist of multiple sessions, is modeled as a Poisson process with rate $\gamma_w$ (packets/sec). Then the arrival of new traffic to the network is a Poisson process with rate $\Gamma = \Sigma_w \gamma_w$. For O-D pair $w$, the overall traffic is transmitted over one path in the set $P_w$, a given set of simple directed paths from the origin to the destination of O-D pair $w$. Let $P$ be the set of all simple directed paths in the network. For each link $l \in L$, the capacity is $C_l$ packets/sec.

For each O-D pair $w \in W$, let $x_p$ be 1 when path $p \in P_w$ is used to transmit the packets for O-D pair $w$ and 0 otherwise. In a virtual circuit network, all of the packets in a session are transmitted over one path from the origin to the destination. Thus $\Sigma_{p \in P_w} x_p = 1$. For each path $p$ and link $l \in L$, let $\delta_{pl}$ denote an indicator function which is one if link $l$ is on path $p$ and zero otherwise. Then, the aggregate flow of packets over link $l$ is given by the left hand side of (1).

In the network, there is a buffer for each outbound link. Using Kleinrock's independence assumption,[19] the arrival of packets to each buffer is a Poisson process where the rate is the aggregate flow over the outbound link. It is assumed that the transmission time for each packet is exponentially distributed with mean $C_l^{-1}$. Thus, each buffer is modeled as an $M/M/1$ queue.

The problem of determining a path for each O-D pair to minimize the average number of packets in the network is formulated as the following nonlinear combinatorial optimization problem.

$$Z_{\overline{IP}} = \min \sum_{l \in L} \frac{\Sigma_{w \in W} \Sigma_{p \in P_w} x_p \gamma_w \delta_{pl}}{C_l - \Sigma_{w \in W} \Sigma_{p \in P_w} x_p \gamma_w \delta_{pl}} \qquad (\overline{IP})$$

subject to

$$\sum_{w \in W} \sum_{p \in P_w} x_p \gamma_w \delta_{pl} \leqslant C_l \qquad \forall l \in L \qquad (1)$$

$$\sum_{p \in P_w} x_p = 1 \qquad \forall w \in W \qquad (2)$$

$$x_p = 0 \text{ or } 1 \qquad \forall p \in P_{w, w \in W}. \qquad (3)$$

The objective function represents the average number of packets in the network. We can also use the average packet delay as our objective function by multiplying $1/\Gamma$ to the objective function of $\overline{IP}$. However, by including the term $1/\Gamma$, an extra number of overhead messages will be needed to estimate $\Gamma$. Constraint (1) requires that the aggregate flow not exceed the capacity for each link. Constraints (2) and (3) require that all of the traffic for one O-D pair be transmitted over exactly one path.

An equivalent formulation of the above problem is given by $(IP)$ below. We redefine $x_p$ to be the rate at which packets for O-D pair $w \in W$ are transmitted over path $p \in P_w$. $(IP)$ is better suited for the development of a distributed algorithm and the application of the Lagrangean Relaxation method.

$$Z_{IP} = \min \sum_{l \in L} \frac{f_l}{C_l - f_l} \qquad (IP)$$

subject to

$$\sum_{w \in W} \sum_{p \in P_w} x_p \delta_{pl} \leqslant f_l \qquad \forall l \in L \qquad (4)$$

$$0 \leqslant f_l \leqslant C_l \qquad \forall l \in L \tag{5}$$

$$\sum_{p \in P_w} x_p = \gamma_w \qquad \forall w \in W \tag{6}$$

$$x_p = 0 \text{ or } \gamma_w \qquad \forall p \in P_w, w \in W. \tag{7}$$

For each link $l$, a variable $f_l$ is introduced. We interpret these variables to be "estimates" of the aggregate flows, whereas in [8] it is interpreted as a link utilization. We prefer to be consistent with the routing literature where routing variables are either flows or probabilities. Since the objective function is strictly increasing with $f_l$ and $(IP)$ is a minimization problem, each $f_l$ will equal the aggregate flow in an optimal solution. As the reader will see in the next section, the introduction of $\{f_l\}$ decouples the problem into two subproblems in the Lagrangean Relaxation.

## 2. Lagrangean Relaxation and Dual Problem

Lagrangean relaxation has been applied to obtain excellent heuristic solutions and tight lower bounds for the traveling salesman problem,[15] the concentrator location problem,[23] a topological design problem in centralized computer networks[7] and many other NP-complete problems. Gavish and Hantler[8] have successfully applied this technique to develop a centralized routing algorithm for virtual circuit networks. As in [8], we dualize constraint (4) to obtain the following relaxation

$$Z_D(u) = \min \left\{ \sum_{l \in L} \frac{f_l}{C_l - f_l} + \sum_{l \in L} u_l \left\{ \sum_{w \in W} \sum_{p \in P_w} x_p \delta_{pl} - f_l \right\} \right\} \qquad (LR)$$

$$\text{subject to} \quad 0 \leqslant f_l \leqslant C_l \qquad \forall l \in L \tag{8}$$

$$\sum_{p \in P_w} x_p = \gamma_w \qquad \forall w \in W \tag{9}$$

$$x_p = 0 \text{ or } \gamma_w \qquad \forall p \in P_w, w \in W. \tag{10}$$

The solution to $(LR)$ is given below. For every $l \in L$,

$$f_l = \begin{cases} C_l \left( 1 - \sqrt{\dfrac{1}{u_l C_l}} \right) & \text{if } u_l > \dfrac{1}{C_l} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

To find the solution $\{x_p\}$, for each O-D pair $w \in W$, a shortest path $p_w^1 \in P_w$ is found, where $u_l$ is the arc weight for link $l$. Then,

$$x_p = \begin{cases} \gamma_w & \text{if } p = p_w^1 \\ 0 & \text{if } p \neq p_w^1. \end{cases} \tag{12}$$

It is clear that $\{f_l\}$ can be computed using a distributed algorithm. If link $l$ corresponds to directed arc $(i, k)$, then we will refer to node $i$ as the tail and node $k$ as the head of directed link $l$, i.e. $i = \text{tail}(l)$ and $k = \text{head}(l)$. In the distributed routing protocol (to be described), tail$(l)$ will compute a multiplier $u_l$ associated with directed link $l$. From this and knowing $C_l$, tail$(l)$ can compute $f_l$ from equation (11). To calculate $\{x_p\}$, a number of distributed shortest path algorithms can be used.[22, 30, 31]

At this point, it would be useful to point out the reason for specifying the feasible values of $x_p$, $p \in P_w$, as $\{0, \gamma_w\}$ instead of $\{0, 1\}$ as in [8]. Note in finding $\{x_p\}$ above all commodities solve a shortest path problem with respect to one weighted graph where the arc weights are $\{u_l\}$. By dualizing constraint (1), the Lagrangean Relaxation "appears" to include $|W|$ different shortest path problems (for each $w \in W$, there is a weighted graph with arc weights $\{u_l \gamma_w\}$). In reality, one would make an adjustment in the implementation so there is just one weighted graph. But, the formulation $(IP)$ is cleaner.

For any $u \geqslant 0$, by using the weak Lagrangean duality theorem, the optimal objective function value of $(LR)$, $Z_D(u)$, is a lower bound on $Z_{IP}$.[10] Naturally, one wants to determine the greatest lower bound by

$$Z_D = \max_{u \geqslant 0} Z_D(u). \qquad (D)$$

There are several methods for solving the dual problem $(D)$[4]. The most popular method is the subgradient method.[4, 10, 17] Let an $|L|$ vector $y$ be a subgradient of $Z_D(u)$. In iteration $k$ of the subgradient optimization procedure, the multiplier for each link $l \in L$ is updated by

$$u_l^{k+1} = u_l^k + t^k y_l^k.$$

The step size $t^k$ is determined by

$$t^k = \delta \frac{Z_{IP}^h - Z_D(u^k)}{\|y^k\|^2} \tag{13}$$

where $Z_{IP}^h$ is an objective function value for a heuristic solution (upper bound on $Z_{IP}$) and $\delta$ is a constant, $0 < \delta \leqslant 2$.

Both $Z_D(u^k)$ and $\|y^k\|^2$ can be expressed in the form $\sum_{l \in L} \beta_l$. In [36], an efficient distributed protocol (based upon using an arborescence) is given. Thus, the subgradient method can be distributed. However, in the next section, we present a distributed algorithm to solve $(D)$ based upon a multiplier adjustment procedure which requires fewer overhead messages.

## 3. A Distributed Multiplier Adjustment Method

In this section, we introduce a distributed multiplier adjustment method. The standard multiplier update rule[17] given in (13) was used by Gavish and Hantler[8] to develop a centralized algorithm for the routing problem in virtual circuit networks. During the operation of a network, it is essential for network reliability that the routing assignments be computed by a distributed algorithm. Then the operation of the system does not rely on a central node which might fail. If the distributed routing algorithm is well designed, the portion of the network that fails can be isolated and the remaining portion of the network can still deliver messages. The multiplier update rule presented below (i) has a lower communication complexity than (13) and (ii) results in a distributed algorithm that is more

stable to the input parameters and finds slightly tighter lower bounds on the minimum objective function value than an algorithm using (13) for most cases in the computational experiments.

For iteration $k$ and link $l$, let $f_l^k$ be the solution given by (11) and let $g_l^k$ be the aggregate link flow determined from $\{x_p\}$. Our approach is to develop a multiplier adjustment scheme so that $Z_D(u)$ is very likely to increase from iteration to iteration. To illustrate the idea, the structure of $Z_D(u)$ is given in Figure 1. Let $x$ be the vector of routing decision variables $x_p$'s. Since $X = \{x | \Sigma_{p \in P_w} x_p = \gamma_w, w \in W;\ x_p = 0 \text{ or } \gamma_w, \forall p \in P_w, w \in W\}$ contains a finite number of elements, $X$ can be represented by $\{x^t | t = 1, \ldots, T\}$ where each $x^t$ is a feasible routing decision vector with respect to constraints (9) and (10). Then $(LR)$ can be rewritten as

$$Z_D(u) = \min \left\{ \sum_{l \in L} \frac{f_l}{C_l - f_l} + \sum_{l \in L} u_l \left\{ \sum_{w \in W} \sum_{p \in P_w} x_p^t \delta_{pl} - f_l \right\} \right\} \qquad (LR')$$

$$\text{subject to} \quad 0 \leqslant f_l \leqslant C_l \qquad \forall l \in L \qquad (14)$$

$$x^t \in X. \qquad (15)$$

Minimizing the objective function with respect to $\{f_l\}$ first and using (11), we obtain the following equivalent form for $(LR)$

$$Z_D(u) = \sum_{l \in L} h_l(u_l) + \min_{x^t \in X} \sum_{l \in L} u_l \left\{ \sum_{w \in W} \sum_{p \in P_w} x_p^t \delta_{pl} \right\} \qquad (16)$$

where

$$h_l(u_l) = \begin{cases} -\left(1 - \sqrt{u_l C_l}\right)^2 & \text{if } u_l > \dfrac{1}{C_l} \\ 0 & \text{otherwise.} \end{cases} \qquad (17)$$

If all of the multipliers except $u_l$ in (16) are fixed, then $Z_D(u)$ is reduced to a one-dimensional function which is denoted by $z_l(u_l)$. Figure 1 shows a typical graph of $z_l(u_l)$. Each smooth dotted curve in Figure 1 is obtained by fixing the routing vector to be an $x^t$ on the right hand side of (16) where all of the multipliers except $u_l$ are fixed. The curve corresponding to $x^t$ is referred to as curve $t$. Since for every value of $u_l$, $z_l(u_l)$ is the minimum objective function value of (16) over all $x^t \in X$, $z_l(u_l)$ is the lower envelope (solid curve) of those dotted curves.

For each $u_l$, there is a curve $t$ that coincides with $z_l(u_l)$. We refer to the corresponding $x^t$ as a shortest path flow at $u_l$. For example, in Figure 1 the shortest path flow corresponding to $u_l^k$ is $x^1$. Note that for each $u_l$, the shortest path flow is not necessarily unique. For example, at the break point $\tilde{u}_l$ in Figure 1, both $x^1$ and $x^2$ are shortest path flows.

Let $g_l(u_l)$ be the aggregate flow on link $l$ (determined from $\{x_p\}$) given that all the multipliers except $u_l$ are fixed. There are several properties of the functions $z_l(u_l)$, $f_l(u_l)$, and $g_l(u_l)$. First, $z_l(u_l)$ is concave since $Z_D(u)$ is concave. Second, $(g_l(u_l) - f_l(u_l))$ is a subgradient of $z_l(u_l)$ at $u_l$. Third, $z_l(u_l)$ is nondifferentiable which results from changes in the shortest path flow when $u_l$ changes at break points. Fourth, from equation (11), $f_l$ is a nondecreasing function of $u_l$. The aggregate flow $g_l(u_l)$ is a nonincreasing function of $u_l$. This follows from the fact that if $u_l$ is increased, then the number of shortest paths using link $l$ will not increase. Consequently, $g_l(u_l)$ does not increase.

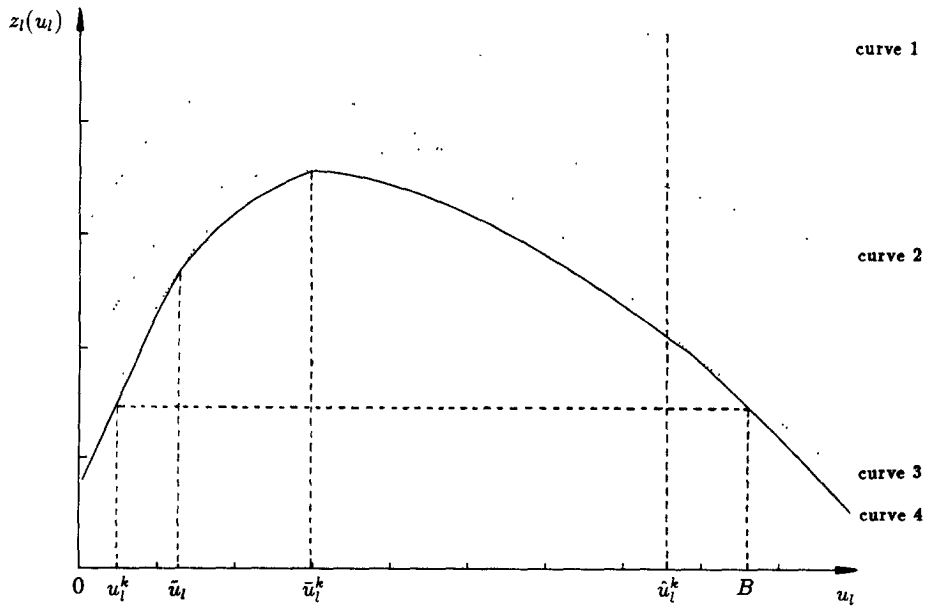


**Figure 1.** A typical graph of $z_l(u_l)$.

Let $\bar{u}_l$ be the value that maximizes $z_l(u_l)$. Let $\hat{u}_l^k = C_l(C_l - g_l(u_l^k))^{-2}$ which is the point where the slope of the curve that coincides with $z_l(u_l^k)$ (i.e. curve 1 in Figure 1) is zero ($f_l(\hat{u}_l^k) = g_l(u_l^k)$).

We developed a Cyclic Coordinate Multiplier Adjustment Procedure (CCMAP) which finds $\bar{u}_l^k$.[20] Since $z_l(u_l)$ is maximized in each iteration, the dual objective function increases in every iteration. However, this procedure requires too many overhead messages to be passed in each iteration. Furthermore, the convergence rate is slow which is a characteristic of the cyclic coordinate ascent method.[21]

To reduce the communication complexity of the CCMAP, we develop a simpler scheme which uses only local information ($f_l^k$, $g_l^k$, and $u_l^k$) to update $u_l$ in such a way that $Z_D(u_l)$ is very likely to increase. We find a range which includes $\bar{u}_l^k$. Then we design a procedure that determines $u_l^{k+1}$ in this range. The following lemma specifies a range that includes $\bar{u}_l^k$.

**Lemma 1.** *$\bar{u}_l^k$ is between $u_l^k$ and $\hat{u}_l^k = C_l(C_l - g_l^k)^{-2}$.*

The proof of Lemma 1 is given below. If $g_l^k(u_l^k) - f_l^k(u_l^k) = 0$, then $u_l^k = \bar{u}_l^k = \hat{u}_l^k$ by the optimality condition to a nondifferentiable function. If $g_l^k(u_l^k) - f_l^k(u_l^k) > 0$, then there are two cases to consider. The first case is that the directional derivative along the positive $u_l$ axis direction is less than or equal to 0. Then $u_l^k = \bar{u}_l^k$. The second case is that the directional derivative along the positive $u_l$ axis direction is greater than 0, which is shown in Figure 1. Obviously, the positive $u_l$ axis direction is an ascent direction. We then increase $u_l$ up to $\hat{u}_l^k$ where $g_l^k(u_l^k) - f_l^k(\hat{u}_l^k) = 0$. At $\hat{u}_l^k$, the corresponding $x^t$ (and $g_l(u_l)$) may change. Since $g_l(u_l)$ is a nonincreasing function of $u_l$, we find at $\hat{u}_l^k$ a subgradient $g_l(\hat{u}_l^k) - f_l(\hat{u}_l^k) = g_l(\hat{u}_l^k) - g_l^k(u_l^k) \leqslant 0$. Therefore, the directional derivative at $\hat{u}_l^k$ along the positive $u_l$ axis direction is either 0 or negative. Because $z_l(u_l)$ is concave, $\bar{u}_l^k \in [u_l^k, \hat{u}_l^k]$. In a similar way, it can be shown that if $g_l^k(u_l^k) - f_l^k(u_l^k) > 0$, then $\bar{u}_l^k \in [\hat{u}_l^k, u_l^k]$. ■

Lemma 1 provides a range that includes $\bar{u}_l^k$. To find $\bar{u}_l^k$, one can limit the search for $\bar{u}_l^k$ in the range specified by Lemma 1 instead of the real line. The step size in each iteration is then bounded above by $|\bar{u}_l^k - u_l^k|$. If the stepsize is carefully controlled (to be small), then $Z_D(u)$ will very likely increase in each iteration. For example, in Figure 1 starting from $u_l^k$, if the step size is less than $|B - u_l^k|$, where $z_l(u_l^k) = z_l(B)$, then $z_l(u_l)$ (and $Z_D(u)$) will increase. A number of schemes could be devised to find a point in this range to use as the multiplier in the next iteration ($u_l^{k+1}$). For example, we might choose $u_l^{k+1} = ((m_k - 1)u_l^k + \hat{u}_l^k)/m_k$ where $m_k \geqslant 1$ is a parameter used to control the stepsize. The following scheme, which determines $u_l^{k+1}$ indirectly (by a mapping from the flowspace), results in better computational results. Consider the flow values $f_l^k$ and $g_l^k$ which correspond to $u_l^k$ and $\hat{u}_l^k$, respectively. We then choose $f_l^{k+1}$ to be the following convex combination of $f_l^k$ and $g_l^k$

$$f_l^{k+1} = \frac{(m_k - 1)f_l^k + g_l^k}{m_k}$$

$$= f_l^k + \frac{1}{m_k}(g_l^k - f_l^k). \tag{18}$$

By (11), $u_l^{k+1}$ and this flow value are related by

$$f_l^{k+1} = C_l\left(1 - \sqrt{\frac{1}{u_l^{k+1}C_l}}\right). \tag{19}$$

By (18) and (19), the new multiplier that yields $f_l^{k+1}$ is

$$u_l^{k+1} = C_l^{-1}\left[1 - \frac{(m_k - 1)f_l^k + g_l^k}{m_k C_l}\right]^{-2}. \tag{20}$$

The idea is illustrated in Figure 2. In Figure 2, the curve shows the relationship between $f_l$ and $u_l$ given by (11). The flows corresponding to the two boundary points $u_l^k$ and $\hat{u}_l^k$ (Lemma 1) are $f_l^k$ and $g_l^k$, respectively. Suppose $m_k = 2$. Then $f_l^{k+1}$ is the midpoint between $f_l^k$ and $g_l^k$. Then $u_l^{k+1}$ is determined by mapping from $f_l^{k+1}$.

It is possible that $f_l^{k+1} > C_l$ if $g_l^k$ is large. In this case, $u_l^{k+1}$ cannot be determined from (18). This was handled by setting $g_l^k$ to $C_l$ which guarantees that $f_l^{k+1} < C_l$.

In order to assure the existence of the inverse of $f_l(u_l)$, we assume that $u_l^k \geqslant C_l^{-1}$, $\forall l \in L$, $k \geqslant 1$. This is a valid restriction since $\{u | u_l \geqslant C_l^{-1}, \forall l \in L\}$ contains an optimal solution to ($D$). Assume the contrary that there exists some $u_l < C_l^{-1}$ in an optimal solution. From (16), (17) and Figure 1, $z_l(u_l)$ is a piecewise linear and nondecreasing function from 0 to $C_l^{-1}$. Then $u_l$ can be increased to $C_l^{-1}$ without decreasing the dual objective function value.

Equation (20) can also be expressed in the following iterative form

$$u_l^{k+1} = u_l^k + t_l^k(g_l^k - f_l^k)$$

where

$$t_l^k = \frac{C_l[2m_kC_l - (2m_k - 1)f_l^k - g_l^k]}{(C_l - f_l^k)^2[m_kC_l - (m_k - 1)f_l^k - g_l^k]^2}. \tag{21}$$

From (21), it is clear that the parameter $m_k$ affects the stepsizes. Note that in (21) there is a stepsize for each link. This is analogous to the distributed routing algorithm for datagram networks proposed in [36] where there is one stepsize for each node. In the subgradient method, there is one stepsize given by (13).

There are a number of advantages of the above multiplier adjustment method over the subgradient method. First, it is simpler and the calculation of stepsizes requires only local information [cf. (20)]. Second, from (20), $u_l^k \geqslant 0$, $\forall l \in L$, $k \geqslant 1$. This eliminates any need for a feasibility check (and a projection) for the multipliers. Third, it can be shown that if the objective function is $z_l(u_l)$ and the sequence $\{m_k\}$ is carefully chosen, $\{u_l^k\}$ will converge to an optimal solution to the dual problem. A proof of this is given in the Appendix. Note that under the same assumptions, the sequence $\{u_l^k\}$ determined by the subgradient method is not guaranteed to converge to an optimal solu-

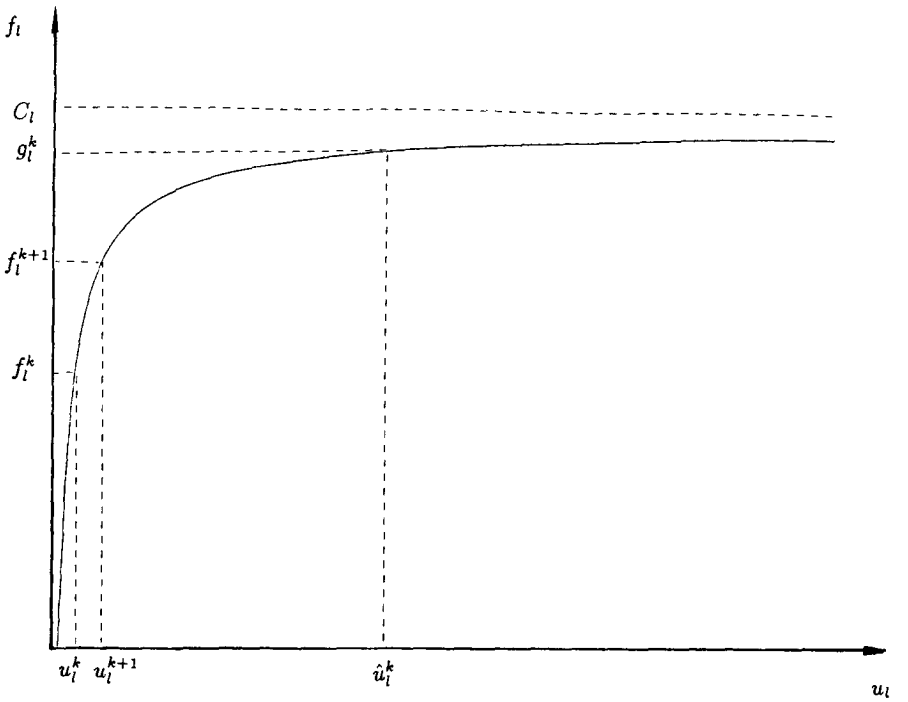


**Figure 2.** A geometrical illustration of the multiplier adjustment procedure.

tion if the optimal objective function value is not known a priori.

The above multiplier adjustment method can be implemented either in a one-at-a-time manner, i.e., only one multiplier is adjusted in each iteration, or in an all-at-once manner, i.e., all multipliers are adjusted in each iteration. To make $\{u^k\}$ converge to a limit, we use a monotone increasing sequence $\{m_k | k = 1, 2, \ldots\}$ where $m_k$ tends to infinity as $k$ approaches infinity. Then from equation (18), $f_l^{k+1}$ will be equal to $f_l^k$ as $k$ (and $m_k$) approaches infinity. From (20), the convergence of $\{f_l^k\}$ implies that $\{u_l^k\}$ will converge to a limit.

The overall algorithm is

0. **Initialization**
   - 0.a Generate a candidate route set for each O-D pair.
   - 0.b Assign a nonnegative value to each multiplier.
   - 0.c Set the iteration counter $k$ to zero.
1. **Test stopping criteria**
   - If the number of iterations has reached the prespecified limit, stop.
   - Otherwise, go to step 2.
2. **Solve the Lagrangean Relaxation**
   - 2.a For each link $l$, each tail($l$) calculates $f_l^k$ by using (11).
   - 2.b A distributed shortest path algorithm is used to solve subproblem 2 for each O-D pair.
   - 2.c For each link $l$, each tail($l$) estimates $g_l^k$.
3. **Adjust the multipliers**
   - 3.a For each link $l$, each tail($l$) uses (20) to calculate $u_l^{k+1}$.
   - 3.b $k \leftarrow k + 1$.
   - 3.c Go to step 1.

Note that Steps 2 and 3 can be performed using distributed computation. Furthermore, Steps 2 and 3 possess a high degree of parallelism so that the implementation of the algorithm is computationally efficient on a parallel system.

## 4. Computational Results

The distributed routing algorithm for virtual circuit networks described in Section 3 was coded in FORTRAN 77 and run on a SUN 4/60 workstation. In the multiplier adjustment procedure, the all-at-once method (all multipliers are adjusted in each iteration) was implemented. Recall that the algorithm parameter in the multiplier adjustment procedure that effects the stepsize is $m_k$. We chose $m_k = k + 1$. The maximum number of iterations allowed was 200 iterations. The choice of the initial values of the multipliers was $\{4C_l^{-1}\}$ (however we found in our computational experiments that the initial values had little effect on the results).

The algorithm was tested on three networks-ARPA, RING and OCT with 61, 32 and 26 nodes, respectively. Their topologies are shown in Figures 3, 4 and 5. Each undirected arc in the networks represents two directed arcs oriented in opposite directions. For each of the three networks, it was assumed that for each O-D pair the total traffic rate at which packets are generated is 1 packet/sec. Other characteristics of the test problems are given in Table I. In the third column, the maximum number of

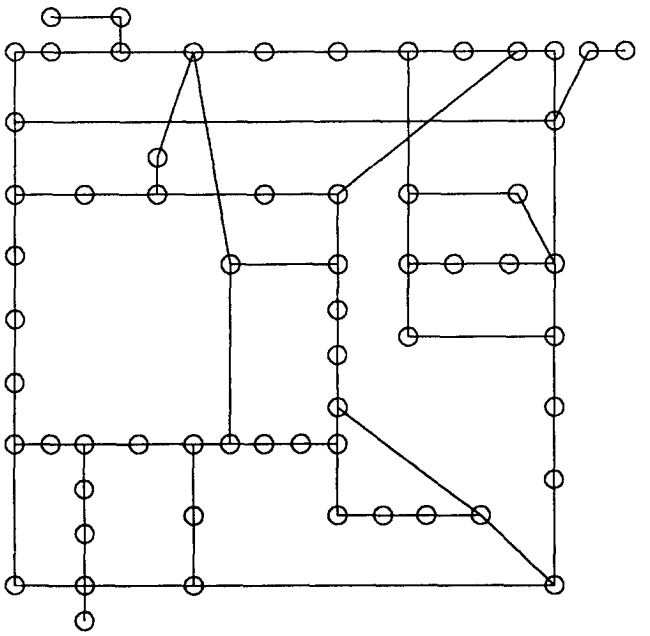

**Figure 3.** The 61-node 148-link ARPA1 net.

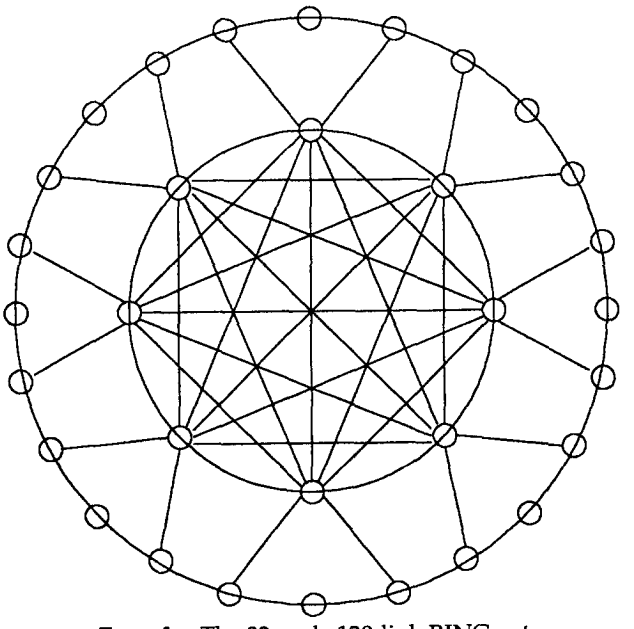

**Figure 4.** The 32-node 120-link RING net.

candidate paths for each O-D pair is given. The fourth column specifies the total number of candidate paths (the number of integer decision variables) in the network. The fifth column specifies the capacity of each link in each network. For each O-D pair, distinct shortest paths were found with respect to several sets of randomly generated arc weights. Another possible scheme to generate candidate paths is the $k$ shortest path algorithm[32, 34] or the route generation algorithms proposed in [9].

Table I summarizes the results of our computational

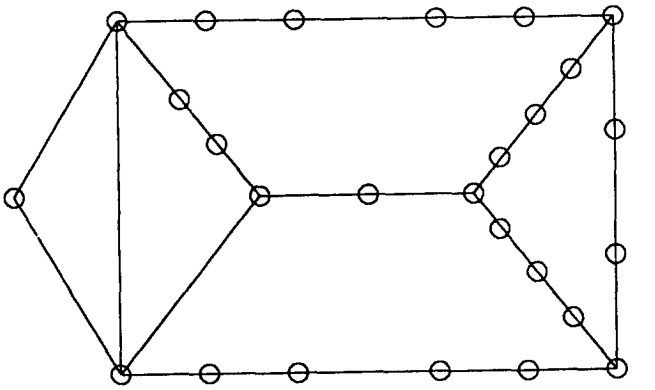

**Figure 5.** The 26-node 60-link OCT net.

experiments. The sixth column is the largest lower bound on the optimal objective function value found in 200 iterations. Recall that this is the best objective function value of the dual problem. The seventh column gives the best objective function value for ($IP$) in 200 iterations. The percentage difference [(upper-bound—lower-bound) × 100/lower-bound] is an upper bound on how far the best feasible solution found is from an optimal solution. The ninth column provides the CPU times which includes the time used to input the problem parameters. These reported CPU times measure the total CPU time used when every operation in the algorithm is performed sequentially. When implementing the algorithm in an actual virtual circuit network, the computations will be performed on $N$ separate computers and most of the computations will be done in parallel. An analysis of the computation and communication times of a distributed implementation of the algorithm is given in the next section.

From an inspection of Table I, it is clear that the distributed routing algorithm is efficient and very effective in finding near-optimal solutions. For every test problem (networks with up to 61 nodes), the distributed algorithm determines a solution that is within 1% of an optimal solution in less than 2 minutes of CPU time on a SUN 4/60 workstation.

We also implemented the subgradient method to compare with our multiplier adjustment procedure. A subset of the test problems was used for the comparison. In our implementation, $Z_{IP}^h$ was initially chosen as 1 and updated to the best upper bound found from iteration to iteration. In (13), $\delta$ was initially set to 2 and halved whenever the objective function value did not improve in a certain number of iterations which is referred to as the improvement counter limit ($ICL$). For each test problem, three different values (5, 10, and 15) were chosen as the improvement counter limit. The maximum number of iterations allowed was 200 iterations for each choice of the improvement counter limit. The choice of the initial values of the multipliers was also $\{4C_l^{-1}\}$. Taking advantage of our previous result that $\{u_l | u_l \geq C_l^{-1}, \forall l \in L\}$ contains an optimal solution to ($D$), each multiplier $u_l$ was projected to $\max\{C_l^{-1}, u_l\}$. The results are reported in Table II.

From Table II, observe that the performance of the sub-

**Table I. Summary of Computational Results when $|P_w| \leqslant 3, \forall w \in W$ and the Multiplier Adjustment Procedure Was Adopted**

| Case No. | Network ID | Routes /Pair | Total Routes Generated | Link Capacities | Lower Bounds (msec) | Upper Bounds (msec) | Percentage Difference (%) | CPU Time (sec) | No. of Iter. |
|---|---|---|---|---|---|---|---|---|---|
| 1 | ARPA | 3 | 6178 | 500.0 | 17.4840 | 17.4855 | 0.009 | 68.8 | 200 |
| 2 | " | 3 | 6178 | 375.0 | 29.8638 | 29.8671 | 0.011 | 64.5 | 200 |
| 3 | " | 3 | 6178 | 333.3 | 39.7388 | 39.7526 | 0.035 | 64.6 | 200 |
| 4 | " | 3 | 6178 | 300.0 | 55.4848 | 55.5512 | 0.120 | 64.8 | 200 |
| 5 | " | 2 | 5290 | 500.0 | 17.5499 | 17.5514 | 0.008 | 52.7 | 200 |
| 6 | " | 2 | 5290 | 375.0 | 30.1057 | 30.1099 | 0.014 | 52.5 | 200 |
| 7 | " | 2 | 5290 | 333.3 | 40.2991 | 40.3069 | 0.019 | 54.7 | 200 |
| 8 | " | 2 | 5290 | 300.0 | 57.2823 | 57.3000 | 0.031 | 52.8 | 200 |
| 9 | OCT | 3 | 999 | 83.33 | 133.969 | 134.179 | 0.157 | 7.6 | 200 |
| 10 | " | 3 | 999 | 76.92 | 170.861 | 171.277 | 0.243 | 7.6 | 200 |
| 11 | " | 3 | 999 | 71.43 | 225.927 | 226.435 | 0.225 | 7.6 | 200 |
| 12 | " | 3 | 999 | 66.67 | 321.001 | 321.882 | 0.274 | 7.6 | 200 |
| 13 | " | 3 | 999 | 62.50 | 551.968 | 553.538 | 0.284 | 7.6 | 200 |
| 14 | " | 2 | 794 | 83.33 | 138.764 | 138.816 | 0.037 | 6.5 | 200 |
| 15 | " | 2 | 794 | 76.92 | 180.730 | 180.863 | 0.074 | 6.3 | 200 |
| 16 | " | 2 | 794 | 71.43 | 249.540 | 249.975 | 0.174 | 6.6 | 200 |
| 17 | " | 2 | 794 | 66.67 | 405.505 | 409.330 | 0.943 | 6.2 | 200 |
| 18 | RING | 3 | 2422 | 100.0 | 40.4421 | 40.4916 | 0.122 | 14.0 | 200 |
| 19 | " | 3 | 2422 | 75.00 | 61.6944 | 61.8227 | 0.208 | 14.2 | 200 |
| 20 | " | 3 | 2422 | 60.00 | 90.6356 | 91.0166 | 0.420 | 14.3 | 200 |
| 21 | " | 3 | 2422 | 50.00 | 133.643 | 134.798 | 0.864 | 15.0 | 200 |
| 22 | " | 2 | 1646 | 100.0 | 41.0024 | 41.0335 | 0.076 | 10.8 | 200 |
| 23 | " | 2 | 1646 | 75.00 | 63.0833 | 63.1527 | 0.110 | 10.8 | 200 |
| 24 | " | 2 | 1646 | 60.00 | 93.8908 | 94.1015 | 0.224 | 10.8 | 200 |
| 25 | " | 2 | 1646 | 50.00 | 141.529 | 142.589 | 0.749 | 10.9 | 200 |

gradient method is sensitive to the choice of *ICL*, e.g. cases 2, 3. There does not seem to be a way of choosing *ICL* which consistently gives the best results. For example, in cases 2 and 3, 15 is the best value for *ICL*, while in cases 18 and 19, 5 is the best choice. Also observe that for case 4 the subgradient method did not provide a satisfactory result. A possible explanation for this is that the performance of the subgradient method strongly depends on $Z_{IP}^h$. In case 4, the network is heavily loaded and it is harder to find a good $Z_{IP}^h$.

Another set of experiments was performed to investigate the effect of the number of candidate paths for each O-D pair. We tested the extreme case where $P_w$ was the set of all simple paths for O-D pair $w$. The multiplier adjustment procedure and the subgradient method were tested and the results are reported in Tables III and IV, respectively. In this set of experiments for the RING network, the simple scheme we used to obtain heuristic solutions to (*IP*) did not perform well. We developed a similar (and simpler) randomization routing scheme to the one mentioned in [8] to obtain better heuristic solutions. Our randomization procedure recorded the routing assignments determined by solving the $K$ (*LR*)'s for the past $K$ iterations (including the current one) starting from the $i$th iteration. For each O-D pair, the heuristic routing was then determined by randomly selecting one routing assignment from the $K$ alternatives (not necessarily distinct). The rationale for our randomized routing scheme is that the step size $t^k$ is small when $k$ is large. Consequently, when $k$ is large, the shortest path costs between two consecutive iterations for each O-D pair is small. Thus those shortest paths for different iterations can be considered as good. In the experiments, we found that the above randomization routing scheme was needed only for the RING network, where $K$ and $i$ were chosen to be 4 and 14, respectively. This scheme resulted in average percentage differences 0.7% and 1.9% for the multiplier adjustment procedure and the subgradient method, respectively. Whereas, the original simple heuristic routing scheme resulted in average percentage differences 20.0 % and 16.9% for the multiplier adjustment procedure and the subgradient method, respectively.

Comparing Table I with Table II and Table III with Table IV, observe that our multiplier adjustment procedure provides slightly better lower bounds (and slightly smaller percentage differences) for most of the cases. The advantage is more pronounced when the network was heavily loaded. Another advantage is that the subgradient method is sensitive to the value chosen for the improvement counter

**Table II. Summary of Computational Results when $|P_w| \leq 3, \forall w \in W$ and the Subgradient Method Was Adopted**

| Case No. | Network ID | Routes /Pair | Total Routes Generated | Link Capacities | *ICL* | Lower Bounds (msec) | Upper Bounds (msec) | Percentage Difference (%) | CPU Time (sec) | No. of Iter. |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ARPA | 3 | 6178 | 500.0 | 5 | 17.4840 | 17.4849 | 0.005 | 94.9 | 200 |
| 1 | " | 3 | 6178 | 500.0 | 10 | 17.4841 | 17.4855 | 0.008 | 94.9 | 200 |
| 1 | " | 3 | 6178 | 500.0 | 15 | 17.4841 | 17.4855 | 0.008 | 94.8 | 200 |
| 2 | " | 3 | 6178 | 375.0 | 5 | 18.7365 | 30.3853 | 62.17 | 95.5 | 200 |
| 2 | " | 3 | 6178 | 375.0 | 10 | 29.8621 | 29.8679 | 0.020 | 95.5 | 200 |
| 2 | " | 3 | 6178 | 375.0 | 15 | 29.8632 | 29.8673 | 0.014 | 90.9 | 200 |
| 3 | " | 3 | 6178 | 333.3 | 5 | 26.1334 | 40.4198 | 54.66 | 91.2 | 200 |
| 3 | " | 3 | 6178 | 333.3 | 10 | 38.9452 | 39.9008 | 2.454 | 91.2 | 200 |
| 3 | " | 3 | 6178 | 333.3 | 15 | 39.6230 | 39.8268 | 0.514 | 91.1 | 200 |
| 4 | " | 3 | 6178 | 300.0 | 5 | 33.5300 | 56.8388 | 69.52 | 91.4 | 200 |
| 4 | " | 3 | 6178 | 300.0 | 10 | 33.5300 | 56.3559 | 68.08 | 91.3 | 200 |
| 4 | " | 3 | 6178 | 300.0 | 15 | 35.8739 | 56.1676 | 56.57 | 91.2 | 200 |
| 9 | OCT | 3 | 999 | 83.33 | 5 | 133.845 | 134.200 | 0.265 | 12.1 | 200 |
| 9 | " | 3 | 999 | 83.33 | 10 | 133.960 | 134.179 | 0.163 | 12.1 | 200 |
| 9 | " | 3 | 999 | 83.33 | 15 | 133.969 | 134.179 | 0.157 | 11.6 | 200 |
| 10 | " | 3 | 999 | 76.92 | 5 | 170.778 | 171.275 | 0.291 | 11.6 | 200 |
| 10 | " | 3 | 999 | 76.92 | 10 | 170.856 | 171.297 | 0.258 | 11.6 | 200 |
| 10 | " | 3 | 999 | 76.92 | 15 | 170.855 | 171.296 | 0.258 | 11.7 | 200 |
| 11 | " | 3 | 999 | 71.43 | 5 | 225.495 | 226.320 | 0.366 | 12.1 | 200 |
| 11 | " | 3 | 999 | 71.43 | 10 | 225.828 | 226.751 | 0.409 | 11.6 | 200 |
| 11 | " | 3 | 999 | 71.43 | 15 | 225.847 | 226.736 | 0.394 | 11.6 | 200 |
| 12 | " | 3 | 999 | 66.67 | 5 | 318.254 | 323.671 | 1.702 | 11.7 | 200 |
| 12 | " | 3 | 999 | 66.67 | 10 | 318.449 | 322.713 | 1.339 | 11.6 | 200 |
| 12 | " | 3 | 999 | 66.67 | 15 | 318.287 | 322.843 | 1.431 | 11.6 | 200 |
| 13 | " | 3 | 999 | 62.50 | 5 | 539.612 | 562.007 | 4.150 | 11.6 | 200 |
| 13 | " | 3 | 999 | 62.50 | 10 | 546.903 | 554.272 | 1.347 | 11.6 | 200 |
| 13 | " | 3 | 999 | 62.50 | 15 | 551.003 | 555.356 | 0.790 | 11.8 | 200 |
| 18 | RING | 3 | 2422 | 100.0 | 5 | 40.4430 | 40.4665 | 0.058 | 21.0 | 200 |
| 18 | " | 3 | 2422 | 100.0 | 10 | 40.4430 | 40.4925 | 0.123 | 20.0 | 200 |
| 18 | " | 3 | 2422 | 100.0 | 15 | 40.4427 | 40.4928 | 0.124 | 20.1 | 200 |
| 19 | " | 3 | 2422 | 75.00 | 5 | 61.6945 | 61.8022 | 0.175 | 20.2 | 200 |
| 19 | " | 3 | 2422 | 75.00 | 10 | 61.6953 | 61.8322 | 0.222 | 20.2 | 200 |
| 19 | " | 3 | 2422 | 75.00 | 15 | 61.6861 | 61.8372 | 0.245 | 20.2 | 200 |
| 20 | " | 3 | 2422 | 60.00 | 5 | 90.5655 | 91.0165 | 0.498 | 20.3 | 200 |
| 20 | " | 3 | 2422 | 60.00 | 10 | 90.6340 | 91.0412 | 0.449 | 20.3 | 200 |
| 20 | " | 3 | 2422 | 60.00 | 15 | 90.6299 | 91.0833 | 0.500 | 20.4 | 200 |
| 21 | " | 3 | 2422 | 50.00 | 5 | 131.957 | 134.552 | 1.967 | 20.3 | 200 |
| 21 | " | 3 | 2422 | 50.00 | 10 | 133.241 | 134.708 | 1.101 | 20.2 | 200 |
| 21 | " | 3 | 2422 | 50.00 | 15 | 133.456 | 134.774 | 0.988 | 20.3 | 200 |

limit. Whereas, our multiplier adjustment procedure is not sensitive to the value chosen for $m_k$. For example, in test problem 29 in Table IV, when the improvement counter limit was chosen as 5, 10 and 15, the percentage differences were 124.8%, 4.1% and 19.7%, respectively. For the same problem (test problem 29 in Table III), the multiplier adjustment procedure yielded solutions where the percentage differences were all within 1% for several different values of $m_k$, e.g. $(\log_2(k+3))^2, \log_2(k+3), 2k, (k+1)/2$ and $\sqrt{k}$. The experiments also showed that the multiplier adjustment procedure consistently performed well when $m_k$ was always chosen as $(k+1)$. In other words, no fine tuning of any parameter is needed in the multiplier adjustment procedure. In addition, the subgradient method requires more CPU time for each choice of *ICL*. If both methods are implemented as distributed algorithms, the multiplier adjustment procedure requires fewer overhead messages and has a higher degree of parallelism than the subgradient method.

We next investigate the effect of $|P_w|$ on the best objective function value. A comparison of Table I with Table III shows that the improvement resulting from using all sim-

**Table III. Summary of Computational Results when $|P_w|$ Is the Set of All Possible Simple Paths for O-D Pair $w \in W$ and the Multiplier Adjustment Procedure Was Adopted**

| Case No. | Network ID | Routes /Pair | Link Capacities | Lower Bounds (msec) | Upper Bounds (msec) | Percentage Difference (%) | CPU Time (sec) | No. of Iter. |
|---|---|---|---|---|---|---|---|---|
| 26 | ARPA | All | 500.0 | 17.2419 | 17.2413 | 0.031 | 100.5 | 200 |
| 27 | " | All | 375.0 | 28.8370 | 28.8516 | 0.051 | 100.6 | 200 |
| 28 | " | All | 333.3 | 37.5638 | 37.5934 | 0.079 | 100.7 | 200 |
| 29 | " | All | 300.0 | 50.3963 | 50.5282 | 0.262 | 100.8 | 200 |
| 30 | OCT | All | 83.33 | 132.372 | 133.218 | 0.639 | 9.5 | 200 |
| 31 | " | All | 76.92 | 167.612 | 168.792 | 0.704 | 9.5 | 200 |
| 32 | " | All | 71.43 | 218.774 | 220.801 | 0.927 | 9.6 | 200 |
| 33 | " | All | 66.67 | 301.877 | 306.703 | 1.599 | 9.6 | 200 |
| 34 | RING | All | 100.0 | 39.9422 | 40.0468 | 0.262 | 21.8 | 200 |
| 35 | " | All | 75.00 | 60.3848 | 60.6205 | 0.390 | 21.9 | 200 |
| 36 | " | All | 60.00 | 87.4219 | 88.0207 | 0.685 | 21.9 | 200 |
| 37 | " | All | 50.00 | 125.395 | 127.207 | 1.445 | 21.9 | 200 |

ple paths over at most 3 candidate paths for each O-D pair ranges from 0.7% to 9.9%. The improvement is greater than 5.7% for only one case. However, the corresponding increase in the computation time varied from 25% to 46%. This suggests that if a small but carefully chosen candidate route set is used, very good routing assignments will be obtained. Similar results were reported in [8].

The next set of experiments were performed to compare our distributed multiplier adjustment procedure with another distributed subgradient-based scheme for solving $(D)$. In [12], three different subgradient-based methods were discussed. The first one is the standard subgradient method given by (13). In the second and third schemes, the multiplier updating rules are given by

$$u^{k+1} = u^k + t^k(y^k/\|y^k\|) \tag{22}$$

$$u^{k+1} = u^k + t^k y^k \tag{23}$$

where $y^k$ is a subgradient and $t^k$ is the step size. (22) and (23) were proposed by Shor[33] and Held and Karp,[16] respectively. It was stated in [12] that for a piecewise linear objective function, there was no reason to believe that either updating scheme is superior to the other. However, (23) is better suited for distributed computation since it does not involve the calculation of $\|y^k\|$. But the convergence rate by using (23) is slow.[12] We compared the relative performance of our multiplier adjustment procedure with (23).

If $t^k$ satisfies the following two conditions: (i) $\lim_{k\to\infty} t^k = 0$ and (ii) $\sum_{k=1}^{\infty} t^k \to \infty$, then $(D)$ is solved optimally.[25] A natural choice for $\{t^k\}$ is $\{(\alpha k)^{-1}\}$ where $\alpha$ is a positive scalar. The results of using (23) for the OCT network are reported in Table V.

From an inspection of Table V and test problems 12 and 13 in Table I, our multiplier adjustment procedure resulted in better percentage differences (0.274% versus 0.746% and 0.284% versus 3.379%). Note that 0.746% and 3.379% are the best results obtained from using different values of $\alpha$. In addition, the performance of (23) is sensitive to the value of $\alpha$. From Table V, the best choice of $\alpha$ was 32 and 8 for test problems 12 and 13, respectively. When 8 was used in test problem 12 and 32 is used in test problem 13, the percentage differences were 7.741% and 16.144% respectively. This suggests that there may be no good way of choosing $\alpha$ for a given network with varying loads.

## 5. Performance Analysis and Implementations

In the previous section, we principally focused on the CPU time needed to determine near-optimal routing assignments. In this section, we also consider the overhead messages needed to support the implementation of the routing algorithm in an actual network. Considering the delays of these overhead messages, we analyze the performance of the distributed virtual circuit routing algorithm. In the process of analyzing the algorithm, several implementation issues are discussed. We refer to the time to perform one iteration of the algorithm as the *cycle time*. This consists of the computation time plus the communication delays of control messages to solve the Lagrangean relaxation and to update the multipliers. The performance measure used to evaluate the algorithm is the average time needed to determine a routing assignment with an objective function value which is within 1% of the optimal objective function value. Below we analyze the average cycle time. Then the performance measure is the average cycle time times the number of iterations needed (based upon computational experience) to determine a solution that is within 1% of an optimal solution.

To simplify the analysis, the following assumptions are made.

1. Each cycle of the routing algorithm is initiated at the same time at each destination node in the network. The overhead to synchronize the system is ignored.
2. The determination of the candidate route sets is a preprocessing procedure and $|P_w| \leq 3$.

**Table IV. Summary of Computational Results when $|P_w|$ Is the Set of All Possible Simple Paths for O-D Pair $w \in W$ and the Subgradient Method Was Adopted**

| Case No. | Network ID | Routes /Pair | Link Capacities | *ICL* | Lower Bounds (msec) | Upper Bounds (msec) | Percentage Difference (%) | CPU Time (sec) | No. of Iter. |
|---|---|---|---|---|---|---|---|---|---|
| 26 | ARPA | All | 500.0 | 5 | 17.2417 | 17.2463 | 0.027 | 113.5 | 200 |
| 26 | " | All | 500.0 | 10 | 17.2421 | 17.2459 | 0.022 | 113.6 | 200 |
| 26 | " | All | 500.0 | 15 | 17.2418 | 17.2469 | 0.030 | 113.6 | 200 |
| 27 | " | All | 375.0 | 5 | 18.7366 | 29.7556 | 58.80 | 113.8 | 200 |
| 27 | " | All | 375.0 | 10 | 28.8363 | 28.8548 | 0.064 | 113.8 | 200 |
| 27 | " | All | 375.0 | 15 | 28.8368 | 28.8512 | 0.050 | 113.8 | 200 |
| 28 | " | All | 333.3 | 5 | 26.1333 | 57.0993 | 118.5 | 114.9 | 200 |
| 28 | " | All | 333.3 | 10 | 37.2166 | 37.6878 | 1.266 | 114.4 | 200 |
| 28 | " | All | 333.3 | 15 | 37.4134 | 37.6197 | 0.551 | 114.6 | 200 |
| 29 | " | All | 300.0 | 5 | 33.5300 | 75.3962 | 124.9 | 114.6 | 200 |
| 29 | " | All | 300.0 | 10 | 48.6000 | 50.5679 | 4.053 | 114.3 | 200 |
| 29 | " | All | 300.0 | 15 | 47.6061 | 56.9923 | 19.72 | 114.7 | 200 |
| 30 | OCT | All | 83.33 | 5 | 109.440 | 137.190 | 25.36 | 9.5 | 200 |
| 30 | " | All | 83.33 | 10 | 132.267 | 133.238 | 0.742 | 9.5 | 200 |
| 30 | " | All | 83.33 | 15 | 132.361 | 133.218 | 0.648 | 9.4 | 200 |
| 31 | " | All | 76.92 | 5 | 126.252 | 174.528 | 38.24 | 9.5 | 200 |
| 31 | " | All | 76.92 | 10 | 167.395 | 168.792 | 0.834 | 9.5 | 200 |
| 31 | " | All | 76.92 | 15 | 167.484 | 168.792 | 0.781 | 9.5 | 200 |
| 32 | " | All | 71.43 | 5 | 211.972 | 220.801 | 4.166 | 9.5 | 200 |
| 32 | " | All | 71.43 | 10 | 218.045 | 220.801 | 1.264 | 9.5 | 200 |
| 32 | " | All | 71.43 | 15 | 218.744 | 220.801 | 0.940 | 9.5 | 200 |
| 33 | " | All | 66.67 | 5 | 290.660 | 308.636 | 6.184 | 9.5 | 200 |
| 33 | " | All | 66.67 | 10 | 301.644 | 308.153 | 2.158 | 9.5 | 200 |
| 33 | " | All | 66.67 | 15 | 300.975 | 308.358 | 2.453 | 9.5 | 200 |
| 34 | RING | " | 100.0 | 5 | 39.9568 | 40.0546 | 0.249 | 21.8 | 200 |
| 34 | " | All | 100.0 | 10 | 39.9593 | 40.0719 | 0.282 | 21.7 | 200 |
| 34 | " | All | 100.0 | 15 | 39.9580 | 40.0526 | 0.237 | 21.8 | 200 |
| 35 | " | All | 75.00 | 5 | 60.4201 | 60.7632 | 0.568 | 21.8 | 200 |
| 35 | " | All | 75.00 | 5 | 60.4292 | 60.6761 | 0.409 | 22.0 | 200 |
| 35 | " | All | 75.00 | 15 | 60.4060 | 60.6949 | 0.478 | 21.8 | 200 |
| 36 | " | All | 60.00 | 5 | 86.2898 | 88.3229 | 2.356 | 21.9 | 200 |
| 36 | " | All | 60.00 | 10 | 87.5353 | 88.1803 | 0.737 | 21.8 | 200 |
| 36 | " | All | 60.00 | 15 | 87.5202 | 88.1960 | 0.772 | 21.6 | 200 |
| 37 | " | All | 50.00 | 5 | 116.452 | 132.689 | 13.94 | 22.0 | 200 |
| 37 | " | All | 50.00 | 10 | 125.500 | 127.443 | 1.548 | 22.2 | 200 |
| 37 | " | All | 50.00 | 15 | 125.069 | 127.268 | 1.759 | 22.0 | 200 |

3. At each node, there is a single processor for all of the computations. Whenever a control message arrives at a node, it is placed in the "computation queue". After a control message has been processed, it is then routed to the appropriate outbound link. It is assumed that the computation processor speed is large relative to the average arrival rate of control messages. Then the delay for a control message at the computation processor is approximately equal to its service (computation) time.
4. The arrivals of data and control messages to each queue are Poisson processes.
5. Control messages are given higher priority than data messages. The queue discipline is nonpreemptive.
6. Acknowledgments for the control messages are piggybacked. The window size is large enough so that the transmissions of the control messages are not delayed.
7. The channels are error-free so that no retransmissions are needed.
8. A supervisor node collects information needed to compute the primal and dual objective function values to determine the quality of the routing assignments obtained so far.
9. There are two types of control messages used in solving the shortest path problem. In the upstream phase, each destination sends a Type 1 control message for each path for which it is a destination. Each Type 1 control message contains 4 bytes (32 bits) specifying the path cost, the origin ($\lceil(\log_2 N)\rceil$ bits), the destination

**Table V. Computational Results when the Multiplier Updating Rule Specified by Equation (23) Was Implemented**

| Case No. | Net ID | Routes /Pair | Total Routes Generated | Link Capacities | $t^k$ | Lower Bounds (msec) | Upper Bounds (msec) | Percentage Difference (%) | CPU Time (sec) | No. of Iter. |
|---|---|---|---|---|---|---|---|---|---|---|
| 12 | OCT | 3 | 999 | 66.67 | $(k)^{-1}$ | 201.364 | 333.688 | 65.71 | 7.7 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(2k)^{-1}$ | 235.740 | 326.001 | 38.29 | 7.6 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(4k)^{-1}$ | 272.267 | 325.752 | 19.64 | 7.8 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(8k)^{-1}$ | 300.713 | 323.990 | 7.741 | 7.6 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(16k)^{-1}$ | 317.066 | 322.815 | 1.813 | 7.6 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(32k)^{-1}$ | 320.374 | 322.764 | 0.746 | 7.7 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(64k)^{-1}$ | 313.981 | 322.842 | 2.822 | 7.7 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(128k)^{-1}$ | 299.813 | 324.911 | 8.371 | 7.7 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(256k)^{-1}$ | 279.909 | 330.996 | 18.25 | 7.7 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(512k)^{-1}$ | 256.302 | 340.240 | 32.75 | 7.7 | 200 |
| 12 | " | 3 | 999 | 66.67 | $(1024k)^{-1}$ | 232.147 | 505.580 | 117.8 | 7.7 | 200 |
| 13 | " | 3 | 999 | 62.50 | $(k)^{-1}$ | 176.689 | 564.947 | 219.7 | 7.7 | 200 |
| 13 | " | 3 | 999 | 62.50 | $(2k)^{-1}$ | 417.939 | 564.286 | 35.01 | 7.6 | 200 |
| 13 | " | 3 | 999 | 62.50 | $(4k)^{-1}$ | 527.676 | 563.783 | 6.843 | 7.6 | 200 |
| 13 | " | 3 | 999 | 62.50 | $(8k)^{-1}$ | 545.992 | 564.441 | 3.379 | 7.7 | 200 |
| 13 | " | 3 | 999 | 62.50 | $(16k)^{-1}$ | 525.360 | 561.919 | 6.959 | 7.7 | 200 |
| 13 | " | 3 | 999 | 62.50 | $(32k)^{-1}$ | 488.368 | 567.209 | 16.14 | 7.7 | 200 |
| 13 | " | 3 | 999 | 62.50 | $(64k)^{-1}$ | 445.444 | 593.925 | 33.33 | 7.7 | 200 |

$([(\log_2 N)]$ bits) and overhead (2 bytes for CRC and ACK). The path cost at a node is the length of the path from the node to the destination. Each node updates the Type 1 message and passes it upstream. After each origin receives a Type 1 message for each candidate path, it determines a shortest path. Then each origin sends a Type 2 control message down the shortest path in the downstream phase. The Type 2 control message has the same format as a Type 1 message where the path cost is replaced by an estimate of the traffic requirement between the O-D pair.

10. The average data packet length is $B$ bits. The load of the network is controlled such that the utilization factor of each link due to data traffic does not exceed $\rho_{\max}$.

Assumption (2) can be relaxed so that all simple paths are candidate paths. This would certainly increase the reliability of the network since more alternative paths are available for each O-D pair. Moreover, the paths could be calculated during the operation of the network taking into account failed links or nodes. Assumption (4) is not valid since the arrivals of control or data messages are not Poisson processes at intermediate nodes. However, this is a standard assumption made for analytical tractability.

In Assumption (8), the supervisor node only evaluates the primal and dual objective function values. The supervisor does not participate in solving $(LR)$'s or updating the multipliers. Therefore, the distributed routing algorithm can still function even if the supervisor node fails. However, when the error bound is within a prespecified tolerance (e.g. 1%), the supervisor node can broadcast a control message to terminate the algorithm. In Assumption (10) we assume that a flow control mechanism limits the link utilization factor for data traffic. In the experiments we investigate how the average cycle time varies with different upper bounds on the link utilization factor due to the data traffic.

Let Phase 1 be the period when each origin determines a shortest path for each destination. Let Phase 2 be the period when each node estimates the aggregate flow for each outbound link and updates the multipliers. The following notation will be used in analyzing the average cycle time.

$D_1^l$: The total number of control packets that traverse link $l$ during the upstream phase
$D_2^l$: The total number of control packets that traverse link $l$ during the downstream phase
$K_1$: the length of a Type 1 message (bits)
$K_2$: The length of a Type 2 message (bits)
$C$: The capacity of each link (bps)
$t_1$: The time to calculate $f_l$ by equation (11)
$t_2$: The time to update a Type 1 message
$t_3$: The time to read a Type 2 message and to update the aggregate link flow
$t_4$: The time for an origin to compare the path costs for a destination to determine a shortest path
$t_5$: The time for a node (processor) to update the associated multipliers
$G$: The maximum outdegree in the network

Let $t$ be the average time for Phase 1. In order for every Type 1 control message to reach the origin and for the origin to compare the path costs and to determine a shortest path for every destination, the following inequality

must hold

$$(N-1)t_4 + \sum_{l \in h_p} \left\{ t_2 + \frac{K_1}{C} + \frac{\frac{D_1^l}{t}\left(\frac{K_1}{C}\right)^2 + 2\frac{\rho_{\max} B}{C}}{2\left(1 - \frac{D_1^l K_1}{tC}\right)} \right\} \leqslant t \quad \forall p \in P \tag{24}$$

where $h_p$ is the set of links along path $p$. The first term of the left hand side of inequality (24) is an upper bound on the time for an origin to determine a shortest path for each of its destinations. Each term in the summation is the computation time for updating a Type 1 message plus the average communication delay for the control messages. The third term in the summation is the average waiting time in the queue for the higher priority traffic in a nonpreemptive M/G/1 queue where $D_1^l/t$ and $(K_1/C)^2$ are the average arrival rate and the second moment of the service (deterministic) time of the control messages. The term $2\rho_{\max} B/C$ upper bounds the mean arrival rate times the second moment of the service time of the data messages (exponentially distributed). Inequality (24) states that the average time for Phase 1 must be at least as large as the average end-to-end delay for each path plus the time for an origin to calculate a shortest path for each of its destinations. The shortest time for Phase 1, denoted by $T_{C1}$, is given by

$$T_{C1} = \min\{t | t \text{ satisfies (24) for all } p \text{ in } P\}. \tag{25}$$

Note the left hand side of (24) is convex and monotone decreasing. The value of $t$ that satisfies (24) with equality is unique. To find $T_{C1}$, we set the difference between the left hand side and the right hand side of (24) to zero and then use Newton's method to determine the root of the equation.

The average time for Phase 2 is found by finding the smallest $t$ satisfying the inequality

$$Gt_1 + Gt_5 + \sum_{l \in h_p} \left\{ t_3 + \frac{K_2}{C} + \frac{\frac{D_1^l}{t}\left(\frac{K_2}{C}\right)^2 + 2\frac{\rho_{\max} B}{C}}{2\left(1 - \frac{D_1^l K_2}{tC}\right)} \right\} \leqslant t \quad \forall p \in P. \tag{26}$$

The first term on the left hand side of inequality (26) is an upper bound on the time for a node to determine the estimate of aggregate flow for each of its outbound links. The second term is an upper bound on the time for a node to update the multiplier for each of its outbound links. Note that $D_2^l \leqslant D_1^l$ since a Type 2 control message will be sent only over the shortest path for each O-D pair. Since it is difficult to predict the shortest paths to be determined in each iteration, we use $D_1^l$ as an upper bound on $D_2^l$. We apply Newton's method to (26) to determine $T_{C2}$ which is the shortest time for Phase 2. Then $(T_{C1} + T_{C2})$ is a conservative estimate of the average cycle time.

The above analysis is applied to a number of actual networks. The topologies of the test networks are shown in Figures 3 to 12. In the calculations, $C$, $B$ and $\rho_{\max}$ were assumed to be 50Kbps, 500 bits, and 0.6, respectively. $t_1$ through $t_5$ were estimated by computational experiments. Since each of these computation times is too small to measure, we measured the time needed to do each opera-

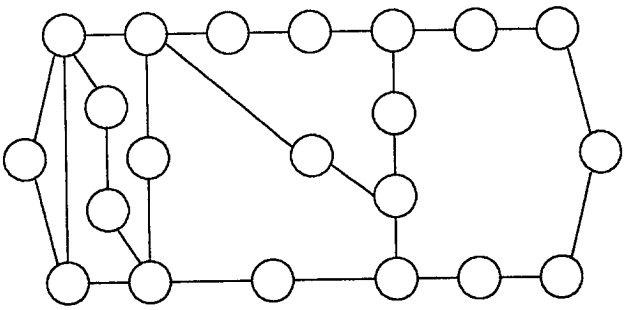

**Figure 6.** The 21-node 52-link ARPA2 net.

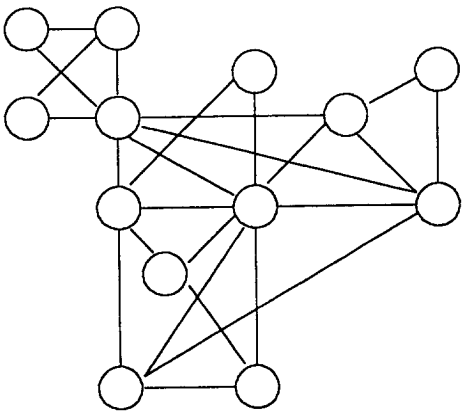

**Figure 7.** The 13-node 48-link NORDIC net.

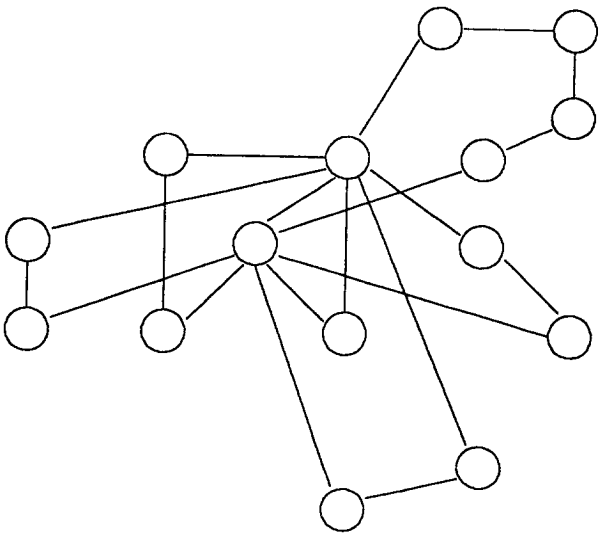

**Figure 8.** The 15-node 38-link SWIFT net.

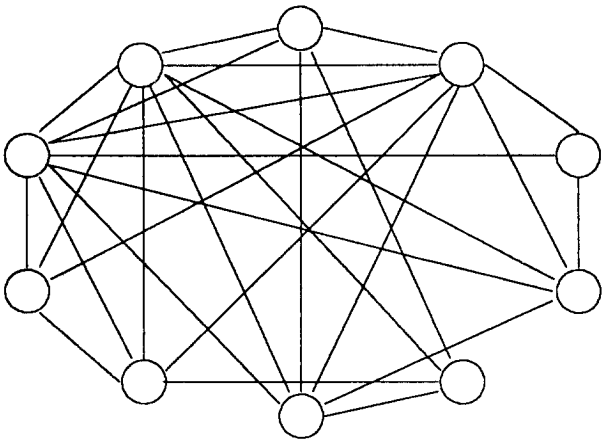

**Figure 9.** The 10-node 56-link SITA net.

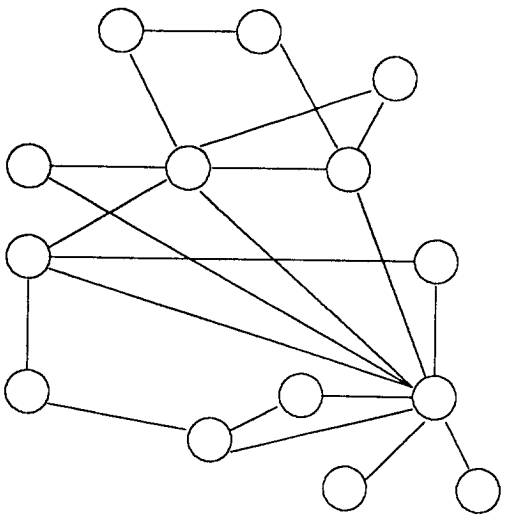

**Figure 10.** The 14-node 42-link PSS net.

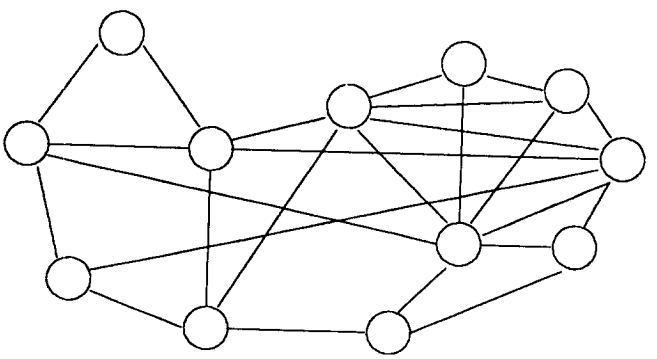

**Figure 11.** The 12-node 50-link GTE net.

tion a very large number of times and calculated the average time for each operation. From experiments on a SUN 4/60 machine, $t_1 = 1.7 \times 10^{-5}$ seconds, $t_2 = 1.6 \times 10^{-7}$ seconds, $t_3 = 1.5 \times 10^{-7}$ seconds, $t_4 = 2.0 \times 10^{-7}$ seconds and $t_5 = 1.5 \times 10^{-7}$ seconds.

The average cycle times for each of the network topologies are reported in Table VI. The second and third columns show the computation time and the communication delays in a cycle, respectively. The fourth column gives the average cycle time. From an inspection of Table VI, the computation times are negligible (no greater than 0.11%) compared with the communication delays. This shows that an overwhelming majority of the real time needed to compute a near-optimal routing assignment is due to the communication delays. Since the distributed computation of routing assignments is essential for the reliability of the network, the communication overhead is unavoidable.

Since the performance measure is the average cycle time times the number of iterations needed to determine a solution that is within 1% of an optimal solution, we did a set of experiments to determine the number of iterations needed. For all of the networks, except the SWIFT network, the number of iterations needed was less than 46 iterations. For the SWIFT network, we were only able to obtain a solution that was within 1.3% of an optimal solution in 300 iterations. From these experiments, the performance

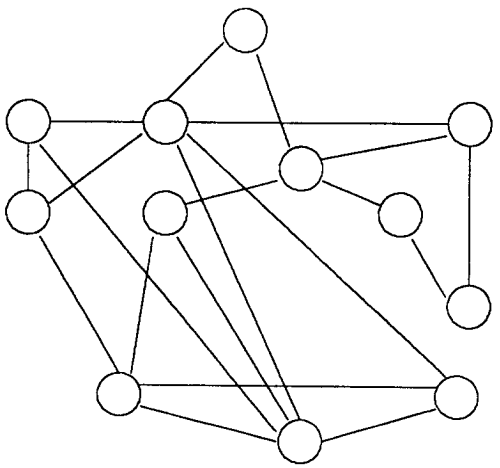

**Figure 12.** The 12-node 44-link TRANSPAC net.

**Table VI. Average Cycle Time for Different Networks**

| Network ID | Computation Time (msec) | Communication Time (msec) | Average Cycle Time (msec) |
|---|---|---|---|
| ARPA1 | 0.08556 | 1555.40 | 1555.49 |
| RING | 0.16272 | 319.955 | 320.118 |
| OCT | 0.07670 | 353.159 | 353.236 |
| ARPA2 | 0.07601 | 291.680 | 291.756 |
| NORDIC | 0.12431 | 147.647 | 147.772 |
| SWIFT | 0.12471 | 146.025 | 146.150 |
| SITA | 0.14086 | 107.039 | 107.180 |
| PSS | 0.15912 | 150.924 | 151.084 |
| GTE | 0.12411 | 115.104 | 115.228 |
| TRANSPAC | 0.10696 | 121.796 | 121.903 |

measure for each network was evaluated. It was found that the longest time (ARPA1) was about 14 seconds.

We next investigated the effect of the link capacities on the average cycle time. The GTE network was used as the test network. The relationship between the capacities and the average cycle time are reported in Table VII. Column 1 shows how the capacities were varied. Columns 3 and 4 reports the communication delays and average cycle times determined from (24) and (26). Table VII shows that the average cycle time is approximately inversely proportional to the link capacities when the link capacities are small (when the communication delays are dominant). As expected, the computation times become dominant as the link capacities are increased.

The total computation time in the distributed implementation is considerably less than the computation time in the centralized implementation reported in Table I for the ARPA1, RING, and OCT networks. The reported CPU times in Table I for the implementation on one computer includes the time to input the parameters and the time to calculate the error bounds. In order to make a fair comparison, we calculate the computation time for the centralized implementation analytically by

$$(N-1)Nt_4 + \sum_{l \in L} (D_1^l t_2 + D_2^l t_3) + L(t_1 + t_5). \quad (27)$$

The first term is the total time for each origin to compare the path costs for each destination and to determine a shortest path. The second term is the total time to calculate the path costs and to calculate the aggregate link flows. The last term is the total time to calculate the estimates of link flows and the time to update all the multipliers. (27) divided by the computation time in the distributed implementation reported in Table VI is the speedup due to the parallelism of the algorithm. Since $t_1$ dominates the other time components by about two orders of magnitude, the speedup for the GTE network is about $50 \times 1.7 \times 10^{-5}/(1.2411 \times 10^{-4}) = 6.849$ where 50 is the number of links in the GTE network. In general, the computation time in the distributed implementation is approximately equal to $t_1 G$ and the speedup is approximately $L/G$.

We also investigated the impact of the data traffic on the average cycle time. The link utilization factor due to the data traffic was parameterized for the GTE network and the corresponding cycle time was calculated. The relationship between the link utilization factor and the average cycle time are depicted in Figure 13. Observe that the curve in Figure 13 tends to be linear when the link utilization factor due to the data traffic is larger than 0.3.

The algorithm can be implemented in an asynchronous manner. We refer to the *degree of synchronization* as the level of coordination among the nodes in the network. In an asynchronous algorithm, the level of coordination is low. In this type of implementation, the initiation of the solution of the shortest path problems is not coordinated among the destination nodes. Furthermore, each node is permitted to update the multiplier of one of its outbound links whenever it receives a Type 2 control message indicating that the aggregate flow on the link will change. Each new session would be routed over the current shortest path. A drawback of an asynchronous implementation is that each node will use a different $m_k$. Also it is harder to compute a lower bound since different portions of the network will be solving different $(LR)$s.

**Table VII. Impact of Link Capacities on the Average Cycle Time**

| Link Capacities (Kbps) | Computation Time (msec) | Communication Time (msec) | Average Cycle Time (msec) |
|---|---|---|---|
| 50 | 0.12411 | 115.104 | 115.228 |
| 100 | 0.12411 | 57.5355 | 57.6596 |
| 200 | 0.12411 | 28.7515 | 28.8757 |
| 400 | 0.12411 | 14.3597 | 14.4838 |
| 800 | 0.12411 | 7.16410 | 7.28821 |
| 1600 | 0.12411 | 3.56686 | 3.69097 |
| 3200 | 0.12411 | 1.76932 | 1.89343 |
| 6400 | 0.12411 | 0.87245 | 0.99656 |
| 12800 | 0.12411 | 0.42695 | 0.55106 |
| 25600 | 0.12411 | 0.20773 | 0.33184 |
| 51200 | 0.12411 | 0.10104 | 0.22515 |
| 102400 | 0.12411 | 0.04929 | 0.17340 |
| 204800 | 0.12411 | 0.02407 | 0.14818 |

For a synchronous algorithm, the level of coordination is high so that each node is performing the same iteration of the algorithm as described in Section 4. That is, each destination initiates the solution of the shortest path problem at the same time. Also, the multiplier for each link is updated at the same time. With the synchronous implementation, extra control messages must be exchanged among nodes to determine the times to (i) initiate the solution of the shortest path problems and (ii) update the multipliers. The asynchronous implementation requires fewer overhead messages than the synchronous implementation we analyzed. However, it is harder to define the cycle time and analyze the time performance for an asynchronous algorithm.

Another issue is to control $m_k$ so that the algorithm adapts to changes in the topology or requirements. Recall that $\{m_k\}$ is a monotonically increasing sequence. For very large values of $\{m_k\}$, the step size is very small. If a severe requirement or topological change occurs, $m_k$ can be reset to its initial value to improve the adaptability of the algorithm.

## 6. Summary and Conclusions

This paper focuses on the development of a distributed routing algorithm for virtual circuit networks. We presented Gavish and Hantler's[8] formulation of this problem as a nonlinear combinatorial optimization problem. We modified their formulation in a way that is mathematically trivial but results in a distributed protocol which requires fewer status messages. As in [8], we applied Lagrangean relaxation to develop an algorithm.

We found that the standard multiplier update rule used

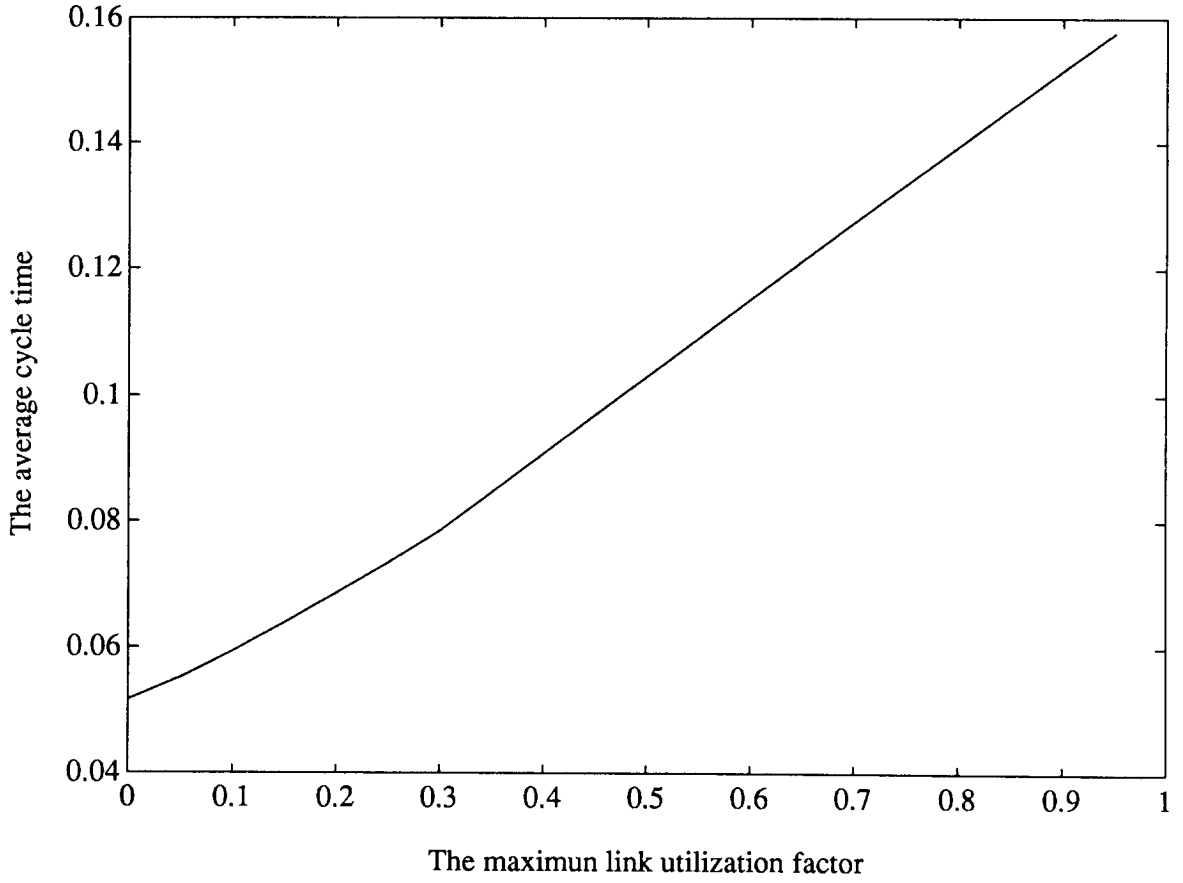


**Figure 13.** The impact of the maximum link utilization factor on the average cycle time.

in [8] required the transmission of too many overhead messages. We introduced a new way of updating the multipliers that requires much fewer overhead messages. In addition, the resulting distributed algorithm determines slightly better lower bounds and error bounds on the minimal average delay in less CPU time for most of the test problems. Furthermore, our proposed multiplier adjustment procedure is more stable than the subgradient method. That is, the performance of the multiplier adjustment procedure is less sensitive to the choice of algorithm parameters. We also found that the size of the candidate route set for each O-D pair has little effect on the final objective function value.

We compared the multiplier adjustment procedure with another distributed subgradient-based method. The computational results showed that our proposed multiplier adjustment procedure provides better lower bounds, error bounds and is more stable.

We analyzed the time to perform each iteration of the distributed algorithm. This includes the computation (CPU) and communication (queueing and transmission) delays of the overhead messages needed to support the distributed implementation. The results showed that the communication delays dominate the computation time. In addition, we showed that the speedup of the distributed algorithm with respect to computation time is approximately equal to the number of nodes in the network divided by the largest outdegree of the network.

## Appendix

In this appendix, we prove the convergence property of the multiplier adjustment procedure when $z_l(u_l)$ is maximized.

As proven in Section 3, only the region $\{u | u_l \geq 1/C_l, \forall l \in L\}$ needs to be considered when solving $(D)$. By (11), $f_l = C_l(1 - \sqrt{1/u_l C_l})$, or equivalently $u_l(f_l) = C_l/(C_l - f_l)^2$. Using Taylor's theorem to expand $u_l$ at $f_l^k$ yields

$$u_l = u_l^k + \frac{2C_l}{(C_l - f_l')^3}\Delta f_l \tag{28}$$

where $f_l'$ is between $f_l^k$ and $f_l^k + \Delta f_l$.

From (18) and (28),

$$u_l^{k+1} = u_l^k + \frac{2C_l}{(C_l - f_l')^3}\frac{g_l^k - f_l^k}{m_k}. \tag{29}$$

From (29) and the fact that $(C_l - f_l') \geq 0$, the step size $t_l^k$ expressed in (21) is lower bounded by $2/C_l^2 m_k$. By the result proven in [25], it can be shown that if $\lim_{k\to\infty} t_l^k = 0$ and $\sum_{k=1}^{\infty} t_l^k = \infty$ then $\max_{u_l \geq 0} z_l(u_l)$ can be solved optimally. It is easy to construct a sequence $\{m_k\}$ that makes $\{t_l^k\}$ satisfy the above two conditions, i.e., $\{m_k = k\}$.

## Acknowledgments

Support for this research was made in part by Contract No. N00228-87-R-4196 from the Naval Postgraduate School (to F.Y.S.L. and J.R.Y.) and in part by Contract DAAL 03-88-k0059 from the Army Research Office and National Science Foundation grant NCR-9016348 (to J.R.Y.).

## References

1. D.G. CANTOR and M. GERLA, 1974. Optimal Routing in a Packet Switched Computer Network, *IEEE Transactions on Computers C-23*, 1062–1069.
2. P.J. COURTOIS and P. SEMAL, 1981. An Algorithm for the Optimization of Nonbifurcated Flows in Computer Communication Networks, *Performance Evaluation 1*, 139–152.
3. A. DANET, R. DESPRES, A.L. REST, G. PICHON and S. RITZENTHALER, 1976. The French Public Packet Switching Service: The TRANSPAC Network, in *Proceedings Third International Computer Communication Conference*, pp. 251–260.
4. M.L. FISHER, 1981. The Lagrangian Relaxation Method for Solving Integer Programming Problems, *Management Science 27:1*, 1–18.
5. L. FRATTA, M. GERLA and L. KLEINROCK, 1973. The Flow Deviation Method: An Approach to Store-and-forward Communication Network Design, *Networks 3*, 97–133.
6. R.G. GALAGER, 1977. A Minimum Delay Routing Algorithm Using Distributed Computation, *IEEE Transactions on Communications COM-25:1*, 73–85.
7. B. GAVISH, 1982. Topological Design of Centralized Computer Networks: Formulations and Algorithms, *Networks 12*, 355–377.
8. B. GAVISH and S.L. HANTLER, 1983. An Algorithm for Optimal Route Selection in SNA Networks, *IEEE Transactions on Communications COM-31:10*, 1154–1160.
9. B. GAVISH and I. NEUMAN, 1986. Capacity and Flow Assignment in Large Computer Networks, in *Proceedings IEEE Infocom*, pp. 275–284.
10. A.M. GEOFFRION, 1974. Lagrangean Relaxation and Its Uses in Integer Programming, *Mathematical Programming Study 2*, 82–114.
11. M. GERLA, 1986. Routing and Flow Control in ISDN's, in *Proceedings 1986 ICCC*, pp. 643–647.
12. J.L. GOFFIN, 1977. On the Convergence Rates of Subgradient Optimization Methods, *Mathematical Programming 13*, 329–347.
13. J.P. GRAY and T.B. MCNEILL, 1979. SNA Multiple-system Networking, *IBM System Journal 18*, 263–297.
14. GTE Telenet Communications Corporation, 1982. *Functional Description of GTE Telenet Packet Switching Networks*, Vienna, VA (May).
15. M. HELD and R.M. KARP, 1970. The Traveling Salesman Problem and Minimum Spanning Trees: Part I, *Operations Research 18*, 1138–1162.
16. M. HELD and R.M. KARP, 1971. The Traveling Salesman Problem and Minimum Spanning Trees: Part II, *Mathematical Programming 1*, 6–25.
17. M. HELD, P. WOLFE and H.D. CROWDER, 1974. Validation of Subgradient Optimization, *Mathematical Programming 6*, 62–88.
18. V.L. HOBERECHT, 1980. SNA Function Management, *IEEE Transactions on Communications COM-28*, 594–603.
19. L. KLEINROCK, 1975 and 1976. *Queueing Systems*, Volumes 1 and 2. Wiley-Interscience, New York.
20. Y.S. LIN and J.R. YEE, 1988. A Distributed Method for the Routing Problem in Virtual Circuit Data Networks, Technical Report, Communication Science Institute, University of Southern California, Los Angeles, CA (August).
21. D.G. LUENBERGER, 1984. *Linear and Nonlinear Programming*, Addison Wesley, Reading, MA.
22. J. MCQUILLAN, 1974. Adaptive Routing Algorithms for Distributed Computer Networks, Ph.D. Thesis, Harvard University (May).
23. A. MIRZAIAN, 1985. Lagrangean Relaxation for the Star-star Concentrator Location Problem: Approximation Algorithm and Bounds, *Networks 15*, 1–20.
24. S. NARASIMHAN, H. PIRKUL and P. DE, 1988. Route Selection in Backbone Data Communication Networks, *Computer Networks and ISDN Systems 15*, 121–133.
25. B.T. POLYAK, 1967. A General Method for Solving Extremal Problems, *Soviet Mathematics Doklady 8*, 593–597.
26. A. RAJARAMAN, 1978. Routing in TYMNET, in *Proceedings European Computation Conference*.
27. H. RUDIN, 1976. On Routing and "Delta Routing": A Taxonomy and Performance Comparison of Techniques for Packet-switched Networks, *IEEE Transactions on Communications COM-24:1*, 43–58.
28. M. SCHWARTZ and T.E. STERN, 1980. Routing Techniques Used in Computer Communication Networks, *IEEE Transactions on Communications COM-28*, 539–552.
29. A. SEGALL, 1979. Optimal Routing for Virtual Line Switched Data Networks, *IEEE Transactions on Communications COM-27*.
30. A. SEGAL, 1981. Advances in Verifiable Fail-safe Routing Procedures, *IEEE Transactions on Communications COM-29*, 491–497.
31. A. SEGAL, 1983. Distributed Network Protocols, *IEEE Transactions on Information Theory IT-29:1*, 23–35.
32. D.R. SHIER, 1979. On Algorithms for Finding the $k$ Shortest Paths in a Network, *Networks 9*, 195–214.
33. N.Z. SHOR, 1968. On the Rate of Convergence of the Generalized Gradient Method, *Kibernetika 4:3*.
34. D.M. TOPKIS, 1988. A $k$ Shortest Path Algorithm for Adaptive Routing in Communications Networks, *IEEE Transactions on Communications COM-36:7*, 855–859.
35. L.R. TYMES, 1981. Routing and Flow Control in TYMNET, *IEEE Transactions on Communications COM-29*, 392–398.
36. J.R. YEE, 1985. Distributed Routing and Flow Control Algorithms for Communication Networks, Ph.D. Thesis, Massachusetts Institute of Technology (December).