SECTION 2.3.3.9
IEN 25

Gateway Dynamic Routing

Virginia M. Strazisar

Bolt Beranek and Newman Inc.

January 25, 1978

PRTN #241

PSPWN #98

I. Introduction

This paper presents a method for routing packets among gateways. It explains a routing algorithm and the protocol for exchanging routing information, proposes a method for handling gateway and network failure and recovery, and discusses the use of status messages and flow control to improve internet performance.

The routing algorithm is dynamic, as routing decisions are made at each gateway based on periodic calculations of delay through the networks. The algorithm minimizes the average delay of all packets in the internet and guarantees loop free routing. An optimal dynamic routing scheme, such as the one presented here, potentially provides the best service, in terms of minimizing delay, in the internet. However, the cost of such a routing procedure in terms of the complexity of the gateway and the gateway processing time should be carefully considered in comparison to other, perhaps suboptimal, yet simpler routing schemes.

The gateways currently use a static routing procedure based on routing tables assembled into each gateway. In the near future, we plan to implement a simple gateway routing scheme, which will improve internet performance by providing the capability to route around failed gateways and networks. At the same time, we will continue to study the problems of gateway dynamic routing, both to determine the desirability of implementing a more complex routing scheme and to define a dynamic routing algorithm.

II. Assumptions

The following definitions and assumptions have been made and the routing procedure presented should be considered under these constraints.

The internet is a collection of networks connected by gateways. The gateways between networks must be capable of readdressing packets among the networks to which they are attached. This is a minimal requirement for the gateways; to the extent that they implement the design proposed below, they will participate in and benefit from globally optimized routing. It is assumed that where reliable communication is needed between end devices, an end-to-end protocol such as the Transmission Control Protocol will be implemented to guarantee this reliability. The collection of gateways and networks that composes the internet is not responsible for perfectly reliable delivery.

We further assume that it is not desirable for the gateways to have any memory of end-to-end communications carried on through them. This assumption explicitly excludes the virtual call method of communication, where it is necessary for all participating nodes to set up a connection and remember parameters with respect to that connection for the life of the end-to-end communication. By requiring that gateways have no specific information about the traffic through them, it is possible to dynamically route packets around failed gateways or networks and to split traffic addressed to a particular destination among different gateways to achieve optimal performance without disrupting the end-to-end communications.

The following additional assumptions are made:

1. Each entity to be addressed has a single unique (over the entire internet) address. This specifically avoids identifying a destination by some route to that destination.

2. There is sufficient addressing information in the packet to identify its destination network, to construct a local header for each network in the internet through which the packet may pass, and to address the end processes involved in communicating across the internet.

The procedures which have been under consideration for internet routing can be categorized in three ways. There are static routing schemes in which the routes for all destinations (or set of destinations, e.g. all hosts on one network) are assembled into the gateway code. When a new net or gateway appears, or an existing net or gateway is taken away, all the tables in all the gateways must be reassembled. This type of routing is not responsive to network or gateway failures and makes no attempt to optimize routing of packets dynamically. A second category of routing which could be considered is one in which primary and alternate routes are assembled into the gateways or derived from the exchange of connectivity information between the gateways. When a network or gateway failure causes the primary route to fail, the alternate route is used. The third category of routing is optimal dynamic routing and is the one considered in this paper. In this scheme, gateways build up routing tables by exchanging routing information between the gateways. An algorithm is defined to optimize some characteristic of internet data transfer, i.e., bandwidth, delay, marginal delay, etc. This scheme is dynamic: routing updates are done periodically as well as in response to failures in the internet or to additions of new routes. On receiving a packet, each gateway decides the next destination for the packet based solely on the packet's destination address and on that gateway's routing information. The packet's route is not stored in the packet before it is sent. This routing system potentially provides the best service throughout the internet.

III. Routing Algorithm

The remainder of the paper presents a detailed description of an internet routing mechanism. It is divided into two sections; in the first, we give the algorithm for deciding a packet's route, and in the second, we describe a protocol for passing routing information between the gateways.

The routing algorithm is based on the work of Robert Gallager as reported in his paper, "A Minimum Delay Routing Algorithm Using Distributed Computation" (IEEE Transactions on Communication, Jan. 1977). This routing algorithm minimizes the average delay for all packets in a network and ensures that the route to each destination in the network is loop free. The necessary condition for minimum average delay is that the marginal delays from a node to a destination on each active link from that node be equal and that there is no inactive link for which the marginal delay is less than that of an active link. The marginal delay on a link is the derivative of the delay on the link with respect to the traffic load on the link. In the routing algorithm, the marginal delay on a link is assumed to be solely a function of the amount of traffic flowing on that link (see Gallager for a discussion of this assumption).

The algorithm is explained in terms of the initial set of information needed in each gateway, the information which must be exchanged by the gateways, the calculations to be done to update the routing information, and the handling of new gateways and failed gateways and networks. Examples are also given. In the explanation, each gateway is referred to as a node of a network. In this analogy, the networks between gateways are the links connecting the nodes.

A. Terminology

The following terms are used.

D'_{ik} is the LINK MARGINAL DELAY on the link from node i to node k (link i,k). This is the derivative of delay with respect to traffic.

$T_{ik}(j)$, the ROUTING VARIABLE, is the non-negative fraction of the traffic from node i to node j which traverses link i,k.

$D'_i(j)$ is the traffic-weighted NODE MARGINAL DELAY from node i to node j, i.e., the marginal delay seen at node i for traffic to node j.

$B_i(j)$, the set of BLOCKED NODES, is the set of nodes, k, such that $T_{ik}(j)$ may not be increased from zero; it includes the nodes k for which link i,k does not exist.

$t_i(j)$ is the total traffic through i to j.

NEIGHBORS of a node i are those nodes k that are physically connected to i (i.e., for which link i,k exists).

A DOWNSTREAM NEIGHBOR of i with respect to j is any node k, such that $T_{ik}(j)$ is non-zero.

An UPSTREAM NEIGHBOR of node i with respect to j is any node k such that $T_{ki}(j)$ is non-zero.

A ROUTING MESSAGE from node i with respect to node j is a message containing the value $D'_i(j)$ and node i's blocked status with respect to j.

The STATE of the network is the set of D'_{ik}, $T_{ik}(j)$, $D'_i(j)$ and $B_i(j)$.

B. Initialization

To initialize the network, each node, i, must be given:

its set of neighbors, k,

the set of all destinations,

a set of $T_{ik}(j)$ for all neighbors, k, and destinations, j,

D'_{ik} for every neighbor node, k,

$D'_i(j)$ for every destination, j, and

$B_i(j)$ for every destination, j.

An initial set of $T_{ik}(j)$ may be defined as:

$T_{ik}(j) = 1$ if i,k,j is the shortest path in number of hops from i to j;

otherwise, $T_{ik}(j) = \emptyset$

To add a new gateway to an existing set, the new gateway must be given a list of its neighbors. In addition, each neighbor of the new gateway must have that gateway added to its list of neighbors. The other gateways in the internet may establish routes to the new gateway, and the new gateway may establish routes to other gateways in the internet as explained in the section below on recovery of a crashed node.

C. Routing Updates

A routing update for node j is explained in terms of the series of steps taken by node j and by each node, i, to compute and propagate their marginal delays with respect to j and to compute new routing variables for traffic to j. The traffic variables computed by each node in the routing update give the fraction of traffic for node j that each node should send to each of its neighbors. The calculation of the routing variables, node marginal delay and blocked status are explained below in the section on Calculations.

1. Each node, i, calculates a set of $T_{ik}(j)$ for each neighbor, k, using the state information obtained from the last routing update. If this is the first routing update for j, the nodes use the information with which they were initialized (see Section B above).

2. Each node, i, measures D'_{ik} for each neighbor, k (see Section VII for a discussion of how nodes measure this value).

3. Node j starts a routing update for itself, by sending a routing message containing $D'_j(j) = 0$ to all its neighbors.

4. After receiving $D'_k(j)$ from all downstream neighbors, k, node i calculates $D'_i(j)$ and determines whether it is blocked with respect to j. Node i then sends a routing message to all its

neighbors. The routing message contains the value of $D_i'(j)$ and the status of node i (blocked or unblocked) with respect to j.

5. After receiving routing messages from all neighbors, go to step 1. Note that it is necessary to wait until routing messages have been received from all neighbors before updating the values of T. To compute new values of T, each node needs a complete set (from both its downstream and non-downstream neighbors) of $D_k'(j)$, and the blocked nodes, k, for each destination, j.

D. Calculations

The routing algorithm outlined above requires that the node marginal delay $D_i'(j)$, the blocked status of node i with respect to j, and the routing variables $T_{ik}(j)$ be calculated. These calculations are done as follows:

1. Node marginal delay for traffic from node i to node j:

$$D_i'(j) = \sum_k T_{ik}(j) * (D_{ik}' + D_k'(j))$$

where k is the set of downstream neighbor nodes such that k is not blocked with respect to j.

2. Blocked status: The set $B_i(j)$ is the set of blocked nodes k for which $T_{ik}(j) = \emptyset$ and for which $T_{ik}(j)$ cannot be increased from $\emptyset$ in order to prevent loops. The set $B_i(j)$ consists of nodes k such that link i,k does not exist or such that $T_{ik}(j) = \emptyset$ and k is blocked with respect to j. Node k is blocked relative to j if k has a routing path to j containing some link m,n such that $D_m'(j) \leq D_n'(j)$. In the routing algorithm outlined above, node k determines whether it is blocked relative to j as follows:

a. For each neighbor, m, such that $T_{km}(j) > \emptyset$, node k compares $D_k'(j)$ to $D_m'(j)$. If $D_k'(j) \leq D_m'(j)$, then k is blocked relative to j.

b. For each neighbor, m, if $T_{km}(j) > \emptyset$ and m has reported in a routing update for j that m is blocked relative to j, then k is blocked relative to j.

No node, i, is allowed to increase the value of its routing variable, $T_{ik}(j)$ from $\emptyset$ if node k is blocked with respect to j. By preventing a node from sending new data through a blocked node, the loop free property of the algorithm is maintained.

3. Node i's routing variables for j, $T_{ik}(j)$:

a. For each k such that $T_{ik}(j) = \emptyset$, and k is blocked with respect to j, the new value of T (noted as "new T") is

$$\text{new } T_{ik}(j) = \emptyset$$

b. For all nodes k, which are either not blocked or for which $T_{ik}(j) > \emptyset$, the routing variables are calculated as follows. (Note according to the above definition of blocked nodes, the set of all non-blocked nodes, k, excludes node i, and nodes l, for which link i,l does not exist.)

First, compute $M_i = \min_k [D'_{ik} + D'_k(j)]$.

M_i is the marginal delay on the best route to j from i.

Next, compute $A_{ik}(j) = [D'_{ik} + D'_k(j)] - M_i$

$A_{ik}(j)$ is the difference in marginal delays between each route to j from i and the best route to j from i. Note that k may be equal to j, which is simply the case in which i is sending to j directly through no intermediate nodes.

Compute

$$\Delta_{ik}(j) = \min [T_{ik}(j), E * A_{ik}(j)/t_i(j)]$$

$\Delta_{ik}(j)$ is the amount by which to decrease $T_{ik}(j)$ for any route which is not the best route from i to j. $T_{ik}(j)$ for the best route from i to j is increased by the sum of the deltas over all

other routes. Each node, i, measures $t_i(j)$, the total traffic through i to j. E is a parameter chosen experimentally and used by all nodes in the net. A value of Ø corresponds to the case in which the routing variables, T, will not change in response to a change in the marginal delays. For $E/t_i(j)$ close to Ø, the values of T will converge slowly; as $E/t_i(j)$ increases, the values of T will change more quickly, but may not converge. (The choice of this scale factor, E, is discussed more thoroughly in a paper by Robert Gallager entitled "Scale Factors for Distributed Routing Algorithms.") The new routing variables are:

$$\text{new } T_{ik}(j) = T_{ik}(j) - \Delta_{ik}(j) \text{ for k not on the best path to j}$$

$$\text{new } T_{ik}(j) = T_{ik}(j) + \sum_{k} \Delta_{ik}(j) \text{ for k on the best path to j.}$$

IV. Exchanging Routing Information

This section describes the gateway-gateway protocol that will be used in passing routing updates between the gateways. As the internet message formats and, in particular, internet addressing, are being reviewed, a specific format is not proposed here. It is assumed that there will be a method for distinguishing gateway-gateway packets from packets using some other protocol (such as TCP). This may be done by using the current message format field or by using the proposed new internet address, which is extended to the level of the process implementing a protocol within a machine.

This protocol provides for reliable delivery of routing messages. As mentioned earlier, one of the important features of the proposed routing algorithm is that it guarantees that no packet will loop through the gateways. It can be proved that loop free routing is maintained if gateways never increase their routing variables from zero through a blocked node. If the routing

updates are not delivered reliably, then some gateways may not receive information that a node is blocked, and these gateways could violate this restriction on their routing variables. Thus, the consequences of unreliable delivery are not simply that the gateways will do slightly suboptimal routing with out-of-date information, but more importantly, that loops can form in the routing paths. Note that the algorithm does not provide a method for breaking loops in routing paths; it simply provides a method for maintaining loop freedom given an initial set of loop free routing paths.

A routing update for node j is done in several steps.

1. Each routing message, $D_i^!(j)$ and i's blocked status with respect to j, has a sequence number. Sequence numbers are kept on a per destination basis, i.e., each node, i, has an expected sequence number for the next routing update for each node, j, in the network.

2. When node i receives a routing message from its neighbor, it compares the received sequence number to the expected sequence number (defined below). If the received sequence number is less than the expected sequence number, then the packet must either be a duplicate, or it is the result of a link failure (see Section V), and is therefore acknowledged but ignored. If the received sequence number is greater than the expected sequence number, then it must be from a routing update in which node i cannot yet participate, and is therefore not acknowledged and ignored. (A node cannot participate in the nth routing update until it has successfully transmitted its routing message for the (n-1)th update to all of its neighbors.) If the received sequence number matches the expected sequence number, then an acknowledgement is returned, and the information is retained.

3. If node i does not receive an acknowledgement from a neighbor, it retransmits for some time. (The retransmit scheme and timeout interval are network specific functions and are not discussed here.) If the neighbor was upstream, then the neighbor could be engaged in a previous routing update cycle and thus may not be able to respond to the routing update from i. In this case, node i cannot interpret lack of an acknowledgement as a link failure. Node i must wait to receive routing updates with sequence number, n, from all its neighbors before accepting any routing updates with sequence n+1.

4. When node i has received an acknowledgement and a routing message with sequence number n from all its neighbors, k (such that neither k or link i,k is dead), then node i sets the expected sequence number, n, to n+1.

V. Node and Link Failure and Recovery

No node has enough information to declare another node down. Even in the case where explicit destination dead messages are received from the subnet, they may be only an indication that the interface to that network failed, and the node reported dead may still be able to communicate on other networks. Thus, in the discussions on node and link failure and recovery, all failures are viewed as link failures. A node failure will appear as an inability to communicate with the node on all links.

Link failure may be detected by status messages from the subnet, status messages from network interface modules in the gateway node, or the failure of the gateway module in a node to communicate with a neighboring node. Note that a node should not assume a link has failed if it does not receive an acknowledgement for a routing update (see Section IV.3 above). However, as each node should be sending routing updates for itself periodically, if a node does not receive a routing update

for one of its neighbors for a period of time much greater than the normal routing update interval, it should assume that its link to that neighbor is down.

When node i has detected that link i,k is down, it uses this information and propagates it to other nodes as follows. Note that node i detecting link i,k is down is equivalent to node i assuming that it received a routing message from k for itself. Thus, if link i,k recovers, and node i receives a routing message from node k, while node i is executing the steps outlined below, it should be treated as a duplicate routing message. Node i should acknowledge the message, but ignore it. The information that link i,k has recovered will be propagated and used in the next routing update.

1. Node i sets D'_{ik} to infinity. For all nodes, j, such that $T_{ik}(j)$ was not zero, node i sets $T_{ik}(j)$ to zero. Non-zero routing variables for neighbor nodes, m (m not equal to k), can be increased proportionally so that the sum of the new routing variables is one. Thus, other links in use for traffic to j can take over the traffic load from link i,k. However, to maintain loop freedom, links not previously in use cannot be used until new routing information, in particular the blocked status of neighbor nodes, is propagated throughout the net.

2. For all nodes, j, to which i was sending traffic and to which i can no longer send traffic on any route, node i should set $D'_i(j)$ to infinity and should say that it is blocked with respect to j.

3. For each j such that the new value of $D'_i(j)$ is infinity, node i sends a routing message to each of its neighbors. The routing message contains the value $D'_i(j)$, which is infinity, and that i is blocked with respect to j. This message uses the sequence number that node i expects to be in the next routing update for j.

4. Each node, m, receiving a message as in step 3 treats it as a normal routing update, except that it sets $T_{mi}(j)$ to Ø and increases its non-zero routing variables proportionally so that the sum of the new routing variables is one. If the node can no longer send traffic to j on any route, it sets its node marginal delay to infinity and says that it is blocked with respect to j. These values are then passed to its neighbors in its routing update for j. If a node receives a message giving $D_i^!(j)$ as infinity and that node is still upstream from j, then it sends a special message to j to prompt j to immediately start two routing updates for itself. In the first routing update, the node that initially detected the failed link will receive routing information giving alternate routes to j (if alternate routes exist). In the second routing update, the information on alternate routes will be propagated to all nodes upstream of the node that detected the failed link (see below). By forcing the two routing updates to occur, new routing parameters can be quickly established. This is essential as some nodes may be disconnected from other nodes because of the failed link.

Recovery of link i,k can be detected in several ways. In the simplest case, the failure was detected by an explicit status message from the subnet, and this message gave a time at which the link would be back up. An example of this is the "IMP going down" message in the ARPANET. In other cases, the network specific modules in the gateway may have detected the link failure and may continually attempt to use the link until it succeeds. At this point the link is declared "up" and the gateway can be notified. An example of this is the Reliable Transmission Protocol module, which runs the VDH interface. When the VDH line is detected down, this code will continually send control messages until the line is detected up. In the most general case, the link failure was detected by the gateway module in nodes i and k when a routing message was not acknowledged. In this case, the gateways can attempt to send packets addressed to

themselves on the failed link. When the packets are received and echoed back to their source, then i and k can begin to measure their marginal delay to each node, j, over this link and can update their routing parameters accordingly. These new parameters will be passed in the next routing update for each node, j, as explained below.

When the node recovers or an alternate route is established, knowledge of the new route should be propagated as quickly as possible. A node that has set all its routing variables for another node, j, to 0 has no downstream neighbors with respect to j. Because it has no downstream neighbors, it is free to transmit its node delay ($D_i'(j)$=infinity) at any time during a routing update for j. However, if node i sends its node delay of infinity during the next routing update for j, then on each routing update only the next level of nodes upstream from the recovered link or alternate route will adjust their routing variables. In order to propagate the new node delays more quickly, the following change is made to the routing procedure explained in Section III above. If node i has set $T_{ik}(j)$ to zero for all k, then on receipt of a routing update in which $D_k'(j)$ is not equal to infinity and k is not blocked with respect to j, node i sets $T_{ik}(j)$ to 1 and says that it is not blocked with respect to j. Node i then recalculates its $D_i'(j)$ and transmits this value in its routing update for j. In this manner, all nodes calculate new values of their routing variables for j in one update after the link has recovered or a new route has been established.

The following is the procedure to be followed when a node, i, recovers from a crash. Node i is assumed to have only a table of its neighbors, k.

1. Node i requests the expected sequence number for routing updates for itself from all its neighbors, k. It then uses the

lowest sequence number and starts a normal routing update (see Section III) for itself by sending $D_i^i(i)=\emptyset$ to all its neighbors.

2. Each node, j, on receiving a routing update for i when the sum over k of $T_{jk}(i) = \emptyset$, sets $T_{jk}(i) = 1$ for the k from which the routing update was received. Note that this makes k the sole downstream neighbor from j with respect to i; thus, j can now send its routing information to all its neighbors, k. Note that this action is identical to that taken during recovery from a link failure.

Steps 1 and 2 have explained how all the nodes in the net discover that a particular node has recovered; step 3 describes how the recovered node sets its routing variables for all other nodes in the net.

3. The recovered node i awaits routing updates from its neighbors, k. On receiving a routing update for node j from node k, it sets $T_{ik}(j)$ to 1, if k is not blocked with respect to j. When $T_{ik}(j)$ has been set to 1, i broadcasts its routing information on j to all its neighbors. (Note that by setting $T_{ik}(j)$ to 1, k becomes the only downstream neighbor of i with respect to j, and i is therefore free to transmit its values for the routing update.) Node i also sets its expected sequence number for routing updates for j to x+1, where x is the sequence number in the first packet received from an unblocked node, k.

VI. Handling Gateways without Dynamic Routing Capabilities

It may not be possible for all gateways to implement the routing algorithm presented above. In a collection of networks in which some gateways implement this routing algorithm and other gateways use a different routing strategy, the average packet delay cannot be minimized nor can paths to the destination be guaranteed loop free. In this case, the proposed routing algorithm attempts to insure that:

1) gateways that do implement the routing algorithm provide optimal service (in terms of minimizing average delay) within the networks served by gateways that implement the routing algorithm, and

2) all gateways are able to route traffic between all points on the networks between which a physical path exists (although such routes may not be optimal).

To achieve these goals, gateways that do not implement the dynamic routing algorithm will not be used on any route to a destination unless these gateways provide the only route to that destination. As the marginal delay through these gateways to any destination is unknown, it is assumed that it is larger than any known marginal delay. Such an assumption prevents gateways from taking advantage of possible optimal routes through non-dynamically routing gateways, but avoids sending packets on routes that may be very suboptimal. Also, unless status information concerning these gateways and their connecting networks is made available to all gateways, then these gateways and networks must always be assumed to be up. If gateways that do not implement the routing scheme presented here do implement a static routing scheme, then given the static routing tables, the gateways implementing dynamic routing can avoid looping traffic through any of the gateways. If gateways that do not implement this routing algorithm implement an alternate routing scheme (to avoid failed components) or use a different dynamic routing scheme, then the internet cannot be guaranteed to deliver traffic. Such a use of conflicting routing algorithms within the internet is unacceptable because there is no practical way known to prevent indefinite looping and congestion between gateways that implement different routing algorithms.

VII. Measuring Marginal Delay

As stated above, this dynamic routing algorithm attempts to minimize average packet delay by sending packets on links with the lowest marginal delay. The description of the algorithm

assumed that there was a method for each gateway to calculate the marginal delay to each of its neighbors. In some networks, this information may be made available to the gateways by the communications subnet. If the marginal delay is not available from the subnet, the gateways can either attempt to measure the marginal delay directly, or can use some model of the traffic loads and delay which allows them to calculate marginal delay from other known quantities. The problem of measuring or calculating marginal delay will be considered in a separate paper.

VIII. Status Messages

In the description of node and link failure, the use of explicit status messages was mentioned. The handling of status messages is explained in terms of how each gateway obtains information local to itself, how the gateway can use this information, what information should be transmitted to other gateways, and how this transmission can be done.

Status messages can come from several different sources. Networks may send status messages to hosts on the network. For example, the IMPs in the ARPANET send IMP/destination dead messages to hosts on the net. Processes within the gateway machine may have status information; for example, the Reliable Transmission Protocol process knows the up/down status of the VDH line. Finally, the gateway can make use of statistics collected on traffic through itself; for example, the gateway can use its information on buffer allocations to detect congestion before packets must be dropped because of lack of gateway resources. The types of information specifically considered here are: link status (up or down), host status (up or down), and throughput information (i.e., the gateway is using all its resources to transmit packets to a particular network). As additional information becomes available from networks, the gateways can

make use of it. This paper is simply concerned with explaining mechanisms for gateways to collect and use whatever status information may be available.

Gateways can use status information to maintain tables on the state of their links to other gateways and on the state of local hosts (i.e., hosts attached to one of the same networks as the gateway). This information can also be exchanged with other gateways in the internet. Further, status information on local congestion (such as resource allocation information) can be used by a gateway in calculating its marginal delay to other gateways. To avoid the need to recognize in each gateway the many types of status messages available from the networks in the internet, the status messages can be grouped into classes. For example, different networks will use different status messages to indicate a dead destination, and these can all be translated into one type of gateway-gateway message. To transmit a status message to other gateways, the gateway must compose an internet message (which follows the format for gateway-gateway messages). These messages can be transmitted in the same manner as are routing updates (see the retransmit and acknowledgement scheme outlined above); or in the cases of failed nodes and links, the gateway can take action to avoid the failed components (see Section V).

IX. Flow Control and Routing

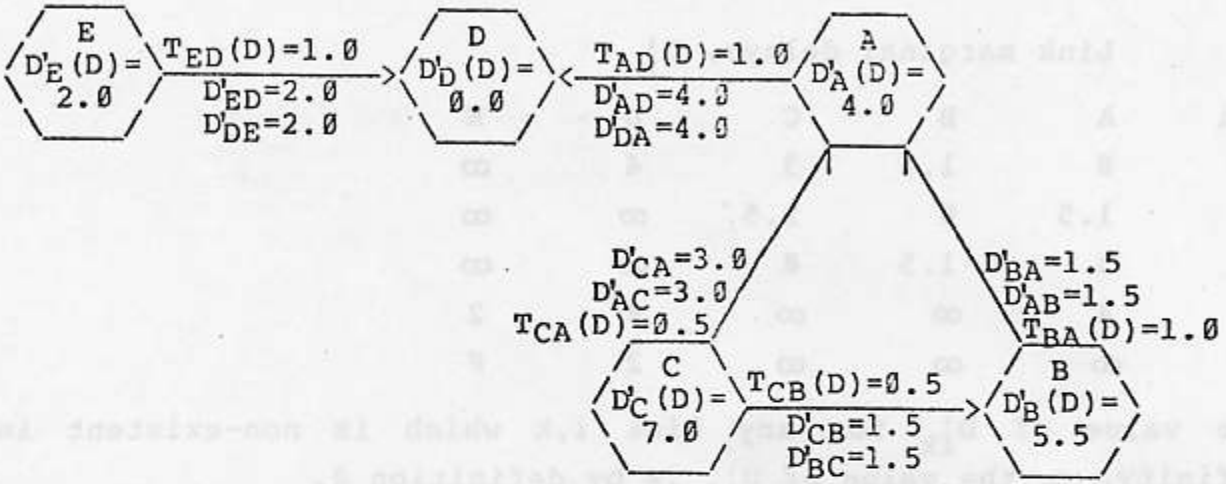
This dynamic routing scheme should alleviate congestion in the gateways as well as in the networks between gateways. The marginal delay measured or approximated should include processing and queueing delay in the gateways. Thus, the marginal delay will increase for a heavily congested gateway and the routing algorithm will attempt to adjust routing parameters to avoid the congested gateway. To avoid network congestion, any flow control procedure must ultimately quench flow into the network from the packet sources. One possible approach to the flow control

problem is to make gateways that are heavily congested notify the internet host sources of the packets travelling through the gateway. In response to this notification, the packet sources would decrease their traffic flow into the internet. As flow control procedures can be considered somewhat independently of routing algorithms, solutions to flow control problems will be discussed in a separate paper.

X. Examples

Example 1

This example gives the routing updates for node D in the network pictured below:



The example is explained by following steps 1 through 5 in the routing algorithm explained in Section III.C. Start with the initial state given below.

Routing variables, $T_{ik}(D)$

k\i	A	B	C	D	E
A	-	1	.5	0	-
B	0	-	.5	-	-
C	0	0	-	-	-
D	1	-	-	-	1
E	-	-	-	0	-

Note that - implies that node k is in the set $B_i(D)$.

From the above table, make up the table of downstream neighbors. A downstream neighbor of i with respect to node j is any neighbor, k, of i such that $T_{ik}(j)$ is non-zero. As this example is concerned with routing updates for node D, the following are the downstream neighbors with respect to node D for each node in the network:

Node	Downstream neighbors with respect to D
A	D
B	A
C	A,B
D	none
E	D

Link marginal delays, D'_{ik}

k\i	A	B	C	D	E
A	Ø	1.5	3	4	∞
B	1.5	Ø	1.5	∞	∞
C	3	1.5	Ø	∞	∞
D	4	∞	∞	Ø	2
E	∞	∞	∞	2	Ø

The value of D'_{ik} for any link i,k which is non-existent is infinity, ∞, the value of D'_{ii} is by definition Ø.

The node marginal delay can be calculated as:

$$D'_i(j) = \sum_k T_{ik}(j) * (D'_{ik} + D'_k(j))$$

As this example is concerned with routing updates for node D, substitute D for j in the above equation. The equation must be solved for i = A,B,C,D, and E. The routing variables, $T_{ik}(j)$, and the link marginal delays, D'_{ik}, can be obtained from the above two tables. In addition, it can be seen from the above equation that in order to calculate $D'_i(j)$, the value of $D'_k(j)$ must be known for each $T_{ik}(j)$ that is non-zero. (Note that k is defined as a downstream neighbor of i with respect to j if $T_{ik}(j)$ is non-zero. Thus, to calculate the node marginal delay for any node, i, the node marginal delay of all i's downstream neighbors must be known). Referring to the table of downstream neighbors, note that $D'_D(D)$ may be calculated first as D has no downstream neighbors. Next, $D'_A(D)$ and $D'_E(D)$ can be calculated, as D is the only downstream neighbor of A and E. $D'_B(D)$ can then be calculated as its downstream neighbor is A. Finally, $D'_C(D)$ can

be calculated as its downstream neighbors are A and B. By definition, $D'_D(D)$ is zero. The calculation of $D'_A(D)$, the node marginal delay to D for node A, is as follows. In the above equation, substitute D for j, A for i, and nodes B,C, and D for k (k is the set of all non-blocked nodes of i with respect to j). Values of $T_{Ak}(D)$ and $D'_{Ak}(D)$ are obtained from the tables above.

$$\begin{aligned} D'_A(D) &= T_{AB}(D) * [D'_{AB} + D'_B(D)] \\ &\quad + T_{AC}(D) * [D'_{AC} + D'_C(D)] \\ &\quad + T_{AD}(D) * [D'_{AD} + D'_D(D)] \\ &= \emptyset + \emptyset + 1*(4 + \emptyset) \end{aligned}$$

Calculating node marginal delays for all nodes gives the following results:

$D'_i(D)$	A	B	C	D	E
	4	5.5	7	Ø	2

Blocked status of nodes with respect to D, $B_i(D)$

	$B_A(D)$	$B_B(D)$	$B_C(D)$	$B_D(D)$	$B_E(D)$
A	X	-	-		X
B	-	X	-		X
C	-	-	X		X
D	-	X	X		-
E	X	X	X		X

An X in row E, column $B_C(D)$ implies that E is blocked with respect to traffic from C to D. It is not necessary to specify the set $B_D(D)$ as D will never send traffic to itself into the net. Recall that for each node, i, the set of blocked nodes, $B_i(j)$, consists of nodes k such that link i,k does not exist or such that $T_{ik}(j) = \emptyset$ and k is blocked with respect to j.

The initial state in this example was chosen so that the set of blocked nodes for each node i consists only of node i and nodes k such that link i,k does not exist.

Following the steps in the routing algorithm explained in Section III, new routing variables can be computed. As the values in the initial state in this example were chosen to be steady state values, the routing variables will not change in the first routing update.

Routing Update 1

Step 1. Each node, i, calculates a set of $T_{ik}(j)$ for each neighbor, k, using the state information obtained from the last routing update. If this is the first routing update for j, the nodes use the information with which they were initialized.

Note that these calculations are not carried out for blocked nodes; in the following tables for A and delta, blocked nodes are represented by -. Also, the calculations are done only for nodes that are sending to node D; thus, they are not done for node D. Recall that for non-blocked neighbor nodes, k, node i calculates routing variables for its traffic to node j as follows. First, compute

$$M_i = \min_k [D'_{ik} + D'_k(j)].$$

M_i is the marginal delay on the best route to j from i.

Next, compute

$$A_{ik}(j) = [D'_{ik} + D'_k(j)] - M_i.$$

$A_{ik}(j)$ is the difference in marginal delays between each route to j from i and the best route to j from i.

Compute

$$\Delta_{ik}(j) = \min [T_{ik}, E * A_{ik}(j)/t_i(j)]$$

$\Delta_{ik}(j)$ is the amount by which to decrease $T_{ik}(j)$ for any route that is not the best route from i to j. The best route from i to j is increased by the sum of the deltas over all other routes. Each node, i, measures $t_i(j)$, the total traffic through i to j. E is a parameter chosen experimentally and used by all nodes in the net. In this example, the value of E is assumed to be .2 and

the value of $t_i(j)$ is assumed to be 5. Finally, the new routing variables are computed as:

new $T_{ik}(j) = T_{ik}(j) - \Delta_{ik}(j)$ for k not on the best path to j

new $T_{ik}(j) = T_{ik}(j) + \sum_k \Delta_{ik}(j)$ for k on the best path to j.

The calculations for node A are as follows (substituting A for i, D for j, and B,C, and D for k in the above equations).

$$M_A = \min [D'_{AB} + D'_B(D)], [D'_{AC} + D'_C(D)], [D'_{AD} + D'_D(D)]$$

Substituting values from the tables above:

$$M_A = \min [1.5 + 5.5], [3 + 7], [4 + \emptyset] = 4$$

$$A_{AB}(D) = D'_{AB} + D'_B(D) - M_A$$
$$= [1.5 + 5.5] - 4$$
$$= 3$$

$$A_{AC}(D) = D'_{AC} + D'_C(D) - M_A$$
$$= [3 + 7] - 4$$
$$= 6$$

$$A_{AD}(D) = D'_{AD} + D'_C(D) - M_A$$
$$= [4 + \emptyset] - 4$$
$$= \emptyset$$

$$\Delta_{AB}(D) = \min [T_{AB}, (E * A_{AB}(D))/t_A(D)]$$
$$= \min [\emptyset, (.2 * 3)/5]$$
$$= \emptyset$$

$$\Delta_{AC}(D) = \min [T_{AC}, (E * A_{AC}(D))/t_A(D)]$$
$$= \min [\emptyset, (.2 * 6)/5]$$
$$= \emptyset$$

$$\Delta_{AD}(D) = \min [T_{AD}, (E * A_{AD}(D))/t_A(D)]$$
$$= \min [1, (.2 * \emptyset)/5]$$
$$= \emptyset$$

For B and C, which are not on the best path to D, the routing variables are:

$$\text{new } T_{AB}(D) = T_{AB}(D) - \Delta_{AB}(D)$$
$$= \emptyset - \emptyset = \emptyset$$

$$\text{new } T_{AC}(D) = T_{AC}(D) - \Delta_{AC}(D)$$
$$= \emptyset - \emptyset = \emptyset$$

For D, which is on the best path to D, the new routing variable is:

$$\text{new } T_{AD}(D) = T_{AD}(D) + \sum_{D} \Delta_{Ak}(D)$$
$$= 1 + \emptyset + \emptyset = 1$$

Note in the summation that k takes on the values of all non-blocked nodes not on the best path to D, i.e., nodes B and C. The following tables give the values of A, delta and T for all nodes in the net.

$A_{ik}(D)$, difference between a route from i through k to D and the best route from i to D

k\i	A	B	C	D	E
A	-	$\emptyset$	$\emptyset$		-
B	3	-	$\emptyset$		-
C	6	3	-		-
D	$\emptyset$	-	-		$\emptyset$
E	-	-	-		-

Δ_{ik}, the amount by which to change routes to D

k\i	A	B	C	D	E
A	-	$\emptyset$	$\emptyset$		-
B	$\emptyset$	-	$\emptyset$		-
C	$\emptyset$	$\emptyset$	-		-
D	$\emptyset$	-	-		$\emptyset$
E	-	-	-		-

New routing variables, $T_{ik}(D)$

k\i	A	B	C	D	E
A	-	1	.5		-
B	Ø	-	.5		-
C	Ø	Ø	-		-
D	1	-	-		1
E	-	-	-		-

Step 2. Each node, i, measures D'_{ik} for each neighbor, k.

In practice, D'_{ik} is a continually changing value sampled by node i. Determining new values of D'_{ik} is shown as a distinct step to point out that the calculations in step 1 are done with a D'_{ik} calculated in one time period T_1, and the calculations done in step 4 are done with a D'_{ik} calculated in the next time period, T_2. Assume that the marginal delay, D'_{ik}, on links A,B and B,A changes from 1.5 to 6 and that all other marginal delays remain the same. This gives the following values of D'_{ik}.

Link marginal delays, D'_{ik}

k\i	A	B	C	D	E
A	Ø	6	3	4	∞
B	6	Ø	1.5	∞	∞
C	3	1.5	Ø	∞	∞
D	4	∞	∞	Ø	2
E	∞	∞	∞	2	Ø

Step 3. Node j starts a routing update for itself, by sending a routing message containing $D'_j(j) = Ø$ to all its neighbors.

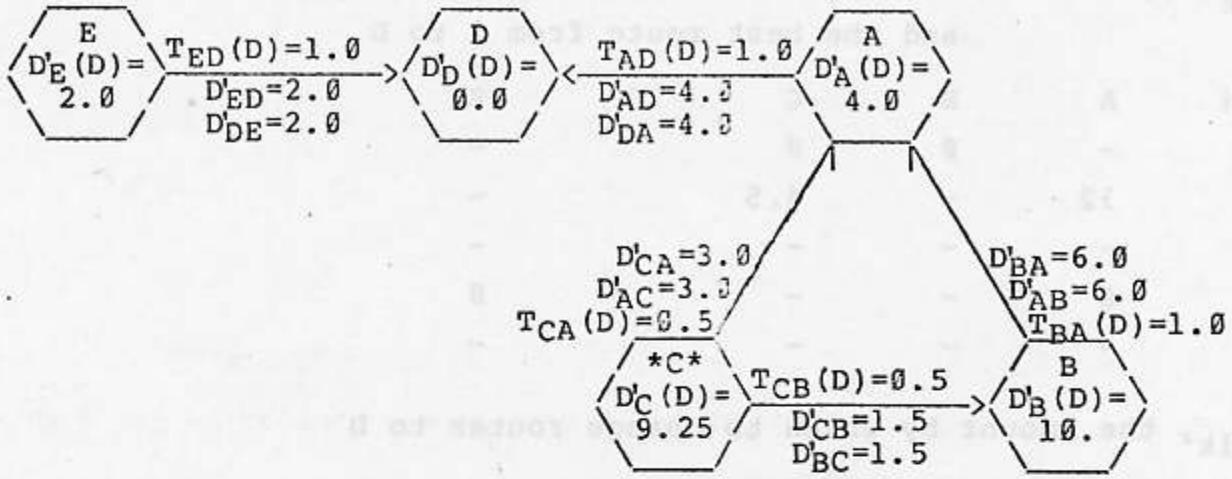
This example is concerned only with the routing update for node D; this is started by D sending $D'_D(D) = Ø$ to its neighbors, A and E.

Step 4. After receiving $D_k'(j)$ from all downstream neighbors, k, node i calculates $D_i'(j)$ and determines whether it is blocked with respect to j. Node i then sends a routing message to all its neighbors. The routing message contains the value of $D_i'(j)$ and the status of node i (blocked or unblocked) with respect to j.

The following table indicates what values each of the routing messages contains, to which nodes each message is sent, and in what order the messages may be sent. The new node marginal delays are calculated with the link marginal delays from Step 2 and the routing variables from Step 1. As each node must wait to receive routing messages from its downstream neighbors, it can be seen from the downstream neighbor table above that D can send its routing message first (it has no downstream neighbors), followed by A and E (their only downstream neighbor is D), followed by B, then C. The far right column lists the nodes that have received routing messages from all their downstream neighbors and have not yet sent their routing messages.

Node sending routing message	$D_i'(D)$	blocked status	sent to	nodes that have rcvd all downstream routing msgs
D	0	not blocked	A,E	A,E
A	4	not blocked	B,C,D	B,E
E	2	not blocked	D	
B	10	not blocked	A,C	C
C	9.25	blocked	A,B	

After exchanging routing messages, the network state is as pictured. Note the new values of link marginal delays from Step 2 (the delay on link A,B changed from 1.5 to 6) and the new node marginal delays. Node C is blocked (indicated by * in the diagram) with respect to traffic to D, because node C is sending traffic for D to node B and node B's marginal delay to D is greater than C's marginal delay to D. Routing variables have not yet been changed.



Step 5. After receiving routing messages from all neighbors, go to step 1. Note that it is necessary to wait until the routing information has been received from all neighbors before updating the values of T. To compute new values of T, each node needs a complete set (from both its downstream and non-downstream neighbors) of $D'_k(j)$, and the blocked nodes, k, for each destination, j.

Routing Update 2

Step 1. Each node, i, calculates a set of $T_{ik}(j)$ for each neighbor, k, using the state information obtained from the last routing update.

Using the link marginal delays and node marginal delays from the previous routing update (see the diagram above or the tables in Steps 2 and 4 above), the new routing variables are calculated. Values of these routing variables are given in the table below.

$A_{ik}(D)$, difference between a route from i through k to D and the best route from i to D

k\i	A	B	C	D	E
A	-	Ø	Ø		-
B	12	-	4.5		-
C	-	-	-		-
D	Ø	-	-		Ø
E	-	-	-		-

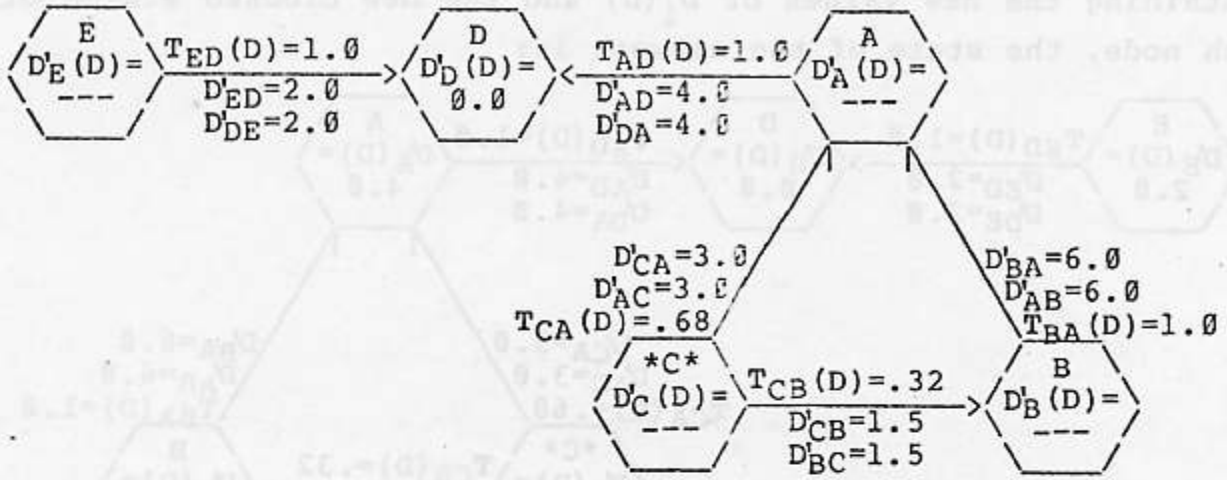
Δ_{ik}, the amount by which to change routes to D

k\i	A	B	C	D	E
A	-	Ø	Ø		-
B	Ø	-	.18		-
C	-	-	-		-
D	Ø	-	-		Ø
E	-	-	-		-

New routing variables, $T_{ik}(D)$

k\i	A	B	C	D	E
A	-	1	.68		-
B	Ø	-	.32		-
C	-	-	-		-
D	1	-	-		1
E	-	-	-		-

The new routing variables are shown in the diagram below. Note that node C is now sending more of its traffic for node D through node A. Node B is still sending all its traffic through A to D, because in the last routing message (see Step 4 above), C said that it was blocked with respect to D. (Recall that when a node is blocked, no node can increase its routing variables for that node.)



Step 2. Each node, i, measures D'_{ik} for each neighbor, k.

It is assumed that the link marginal delays do not change at this point.

Step 3. Node j starts a routing update for itself, by sending a routing message containing $D'_j(j) = 0$ to all its neighbors.

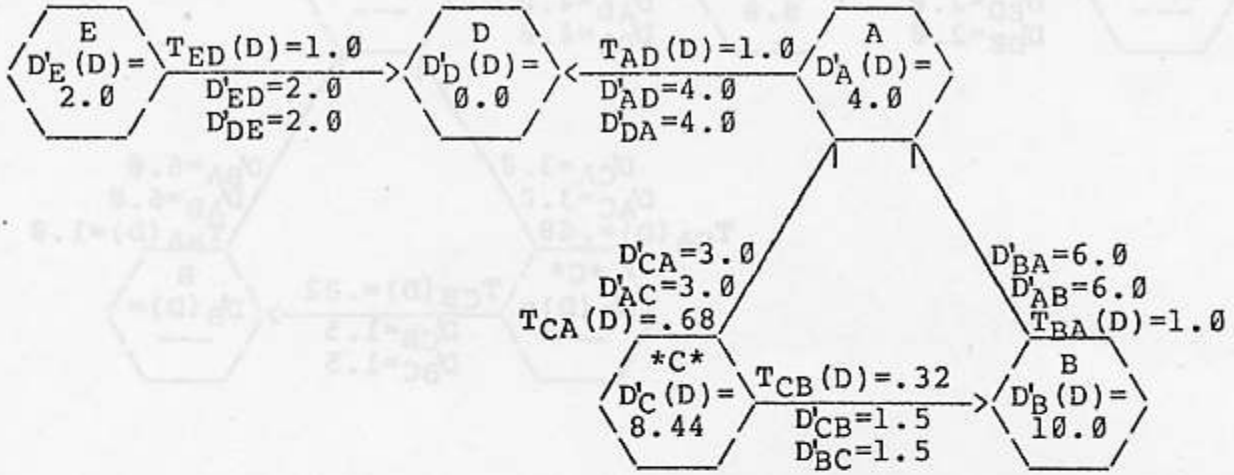
Node D starts a routing update for itself by sending $D'_D(D)$ to A and E.

Step 4. After receiving $D'_k(j)$ from all downstream neighbors, k, node i calculates $D'_i(j)$ and determines whether it is blocked with respect to j. Node i then sends a routing message to all its neighbors. The routing message contains the value of $D'_i(j)$ and the status of node i (blocked or unblocked) with respect to j. The new node marginal delays calculated with the routing variables from Step 1 above and the link marginal delays from Step 2 are:

$D'_i(D)$	A	B	C	D	E
	4	1Ø	8.44	Ø	2

Note that node C is still blocked with respect to D, as it is sending traffic through a node that has a greater marginal delay to node D than itself. After propagating routing messages

containing the new values of $D_i'(D)$ and the new blocked status of each node, the state of the network is:



Step 5. After receiving routing messages from all neighbors, go to step 1.

Routing Update N

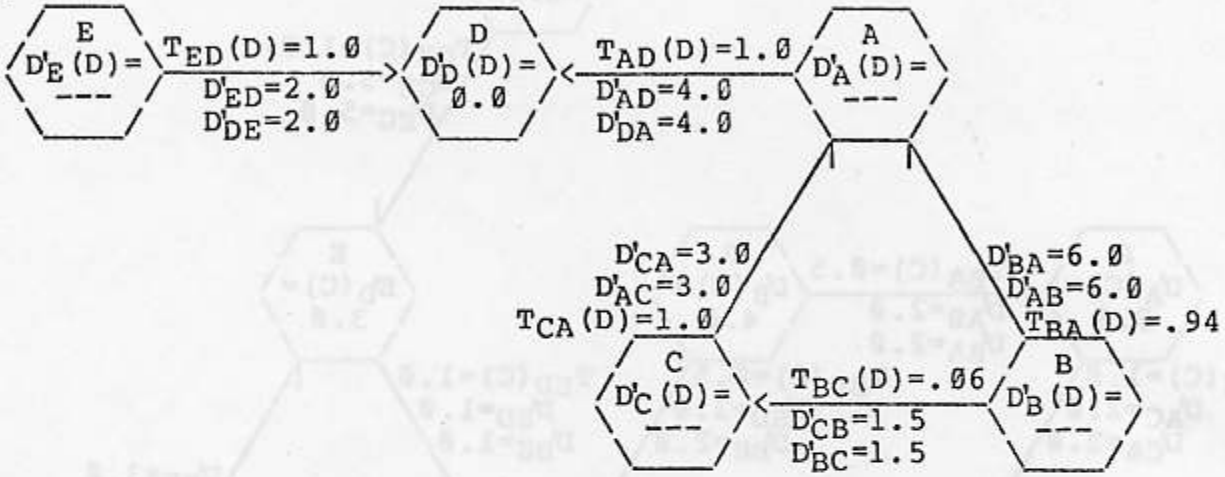
After several routing updates, assuming that the link marginal delays do not change, node C is no longer sending traffic for node D through node B. At this stage, node C becomes unblocked, and node B is free to start sending its traffic for D through node C.

Step 1. Each node, i, calculates a set of $T_{ik}(j)$ for each neighbor, k, using the state information obtained from the last routing update.

New routing variables, $T_{ik}(D)$

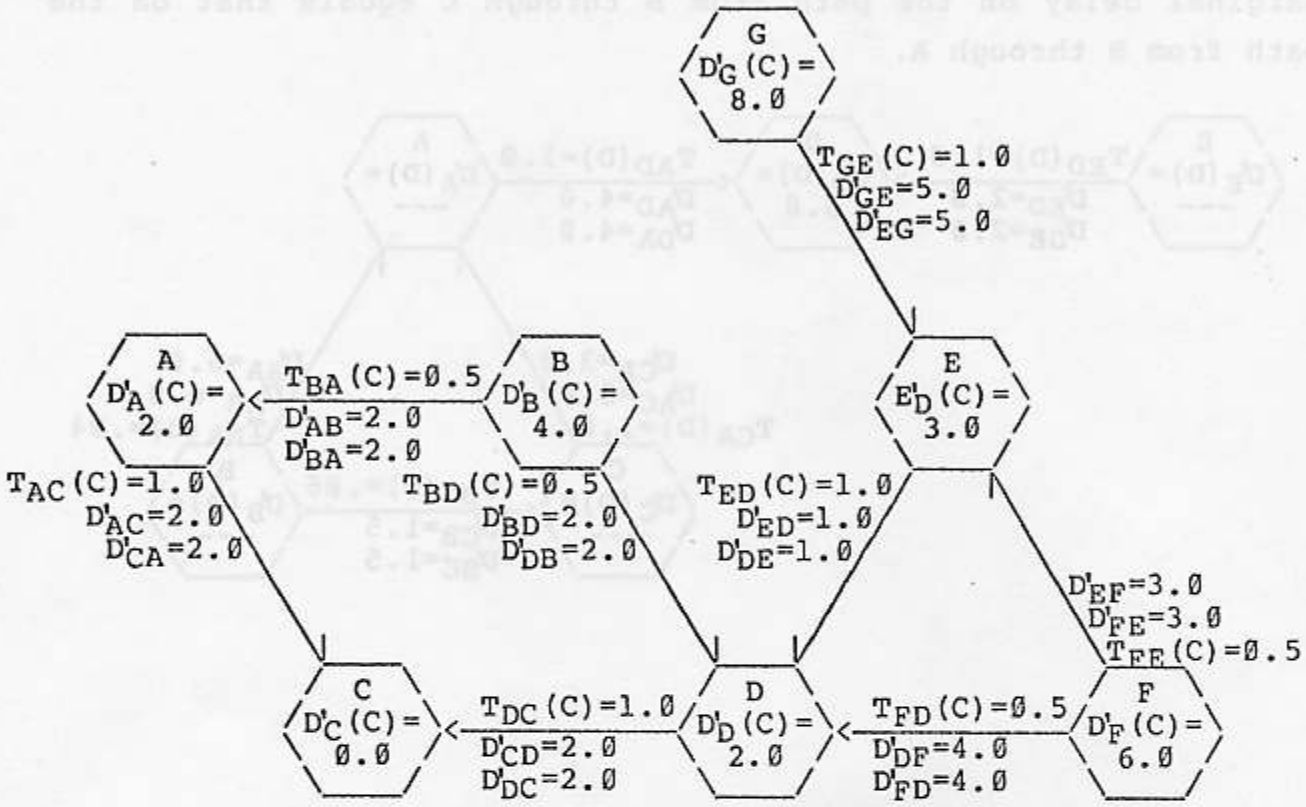
k\i	A	B	C	D	E
A	-	.94	1		-
B	Ø	-	Ø		-
C	-	.06	-		-
D	1	-	-		1
E	-	-	-		-

After calculating the new routing variables, the state of the network is as pictured. Note that a small amount of traffic is now being sent from node B through node C to node D. Succeeding routing updates will increase this amount of traffic until the marginal delay on the path from B through C equals that on the path from B through A.



Example 2

This example shows how routing variables for node C adjust when link C,D in the network pictured below fails. This example is explained in terms of the procedure in Section V.



After the first routing update for C, it is assumed that the network is in this state.

Routing variables, $T_{ik}(C)$

k\i	A	B	C	D	E	F	G
A	–	.5	∅	–	–	–	–
B	∅	–	–	∅	–	–	–
C	1	–	–	1	–	–	–
D	–	.5	∅	–	1	.5	–
E	–	–	–	∅	–	.5	1
F	–	–	–	∅	∅	–	–
G	–	–	–	–	∅	–	–

Link marginal delays, D'_{ik}

k\i	A	B	C	D	E	F	G
A	∅	2	2	∞	∞	∞	∞
B	2	∅	∞	2	∞	∞	∞
C	2	∞	∅	2	∞	∞	∞
D	∞	2	2	∅	1	4	∞
E	∞	∞	∞	1	∅	3	5
F	∞	∞	∞	4	3	∅	∞
G	∞	∞	∞	∞	5	∞	∅
$D'_i(C)$	2	4	∅	2	3	6	8

	$B_A(C)$	$B_B(C)$	$B_C(C)$	$B_D(C)$	$B_E(C)$	$B_F(C)$	$B_G(C)$
A	X	–		X	X	X	X
B	–	X		–	X	X	X
C	–	X		–	X	X	X
D	X	–		X	–	–	X
E	X	X		–	X	–	–
F	X	X		–	–	X	X
G	X	X		X	–	X	X

Failure of link D,C is detected by node D.

Step 1. Node i sets D'_{ik} to infinity. For all nodes, j, such that $T_{ik}(j)$ was not zero, node i sets $T_{ik}(j)$ to zero. Non-zero routing variables for neighbor nodes, m (m not equal to k), can be increased proportionally so that the sum of the new routing variables is one. Thus, other links in use for traffic to j can take over the traffic load from link i,k. However, to maintain loop freedom, links not previously in use cannot be used until new routing information, in particular the blocked status of neighbor nodes, is propagated throughout the net.

Node D sets D'_{DC} to ∞ and $T_{DC}(C)$ to $\emptyset$.

Step 2. For all nodes, j, to which i was sending traffic and to which i can no longer send traffic on any route, node i should set $D'_i(j)$ to infinity and should say that it is blocked with respect to j.

Node D sets $D'_D(C)$ to $\infty$and node D is blocked with respect to C.

Step 3. For each j such that the new value of $D'_i(j)$ is infinity, node i sends a routing message to each of its neighbors. The routing message contains the value $D'_i(j)$, which is infinity, and that i is blocked with respect to j. This message uses the sequence number which node i expects to be in the next routing update for j.

Node D sends routing update 2 to nodes B,E, and F.

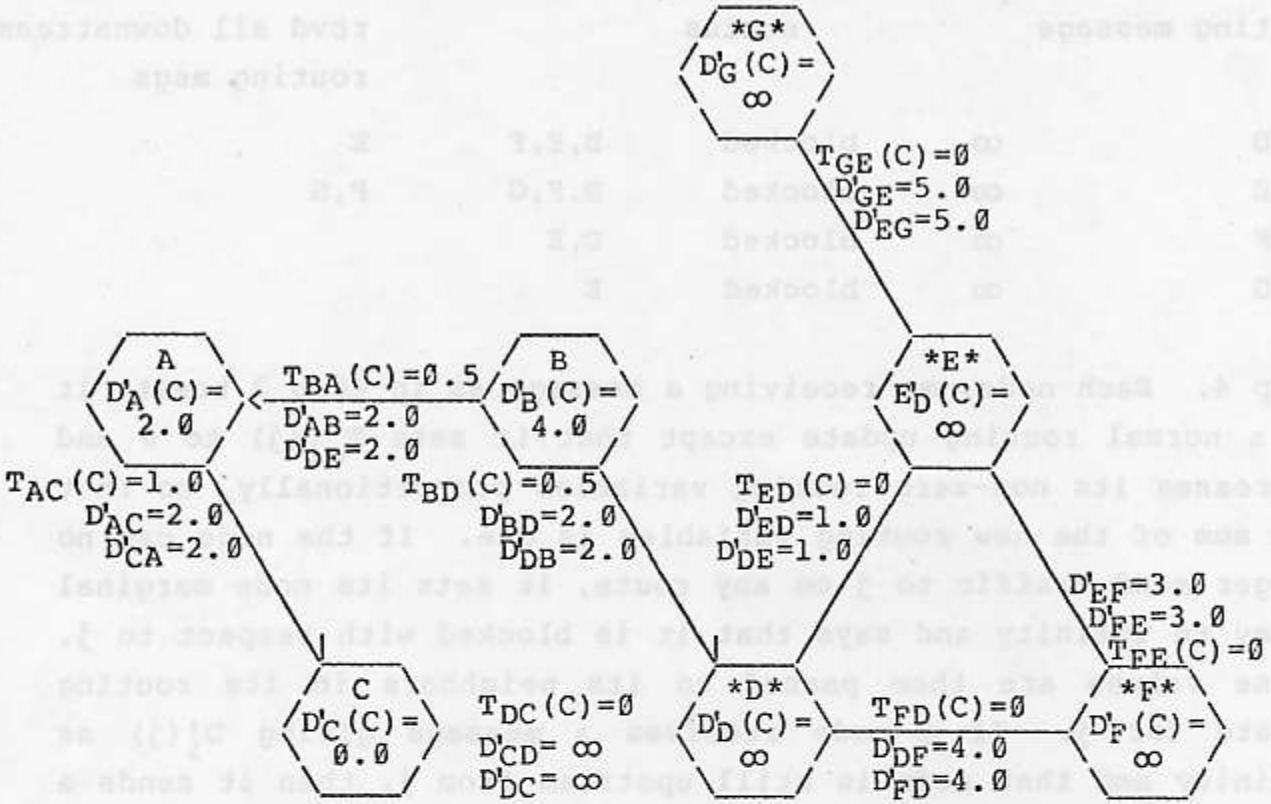
Node sending routing message	$D_i^!(C)$	blocked status	sent to	nodes that have rcvd all downstream routing msgs
D	∞	blocked	B,E,F	E
E	∞	blocked	D,F,G	F,G
F	∞	blocked	D,E	
G	∞	blocked	E	

Step 4. Each node, m, receiving a message as in step 3 treats it as a normal routing update except that it sets $T_{mi}(j)$ to $\emptyset$ and increases its non-zero routing variables proportionally, so that the sum of the new routing variables is one. If the node can no longer send traffic to j on any route, it sets its node marginal delay to infinity and says that it is blocked with respect to j. These values are then passed to its neighbors in its routing update for j. If a node receives a message giving $D_i^!(j)$ as infinity and that node is still upstream from j, then it sends a special message to j to prompt j to immediately start two routing updates for itself.

Since they have received a routing update, nodes E,F, and G calculate new routing variables. Their new routing variables for C, $T_{ik}(C)$ are all zero. Also, their node marginal delays to C are infinity, and they are blocked with respect to C.

Node B is still waiting for a routing update from its downstream neighbor, A, so it does not recalculate its routing variables for C, nor does it propagate the routing update for C which was started by D. However, B sends a special message to C to prompt C to start two routing updates for itself.

At this point, as can be seen in the diagram below, nodes D, E, F, and G are unable to send traffic to node C.



In response to the message from B, node C starts routing update 2 for itself. The node marginal delays are calculated from the routing variables and link marginal delays shown above.

Node sending routing message	$D'_i(C)$	blocked status	sent to	nodes that have rcvd all downstream routing msgs
C	0	not blocked	A	A
A	2	not blocked	B	B
B	4	not blocked	D	

Nodes A and B now calculate new routing variables for C. The values of node and link marginal delays needed to calculate new routing variables are taken from the diagram above; the values of E and t are assumed to be .2 and 5, respectively.

New routing variables, $T_{ik}(C)$

k\i	A	B
A	-	1
B	-	-
C	1	-
D	-	Ø
E	-	-
F	-	-
G	-	-

As nodes E, F, and G have already received routing messages for routing update 2 from all their downstream neighbors (see Step 3 above), they do not receive the new marginal delays that would enable them to make use of the alternate route through B at this time. Similarly, node D, in detecting that link C,D was down, essentially assumed receipt of the second routing update from C. Node D ignores receipt of the second routing update from C, as this is considered a duplicate.

Node C starts routing update 3, which propagates to all nodes in the net. All nodes set new routing variables for their traffic to C reflecting the use of link D,B in place of link D,C. The values of routing variables and link marginal delays needed to calculate the new node marginal delays are taken from the diagram above.

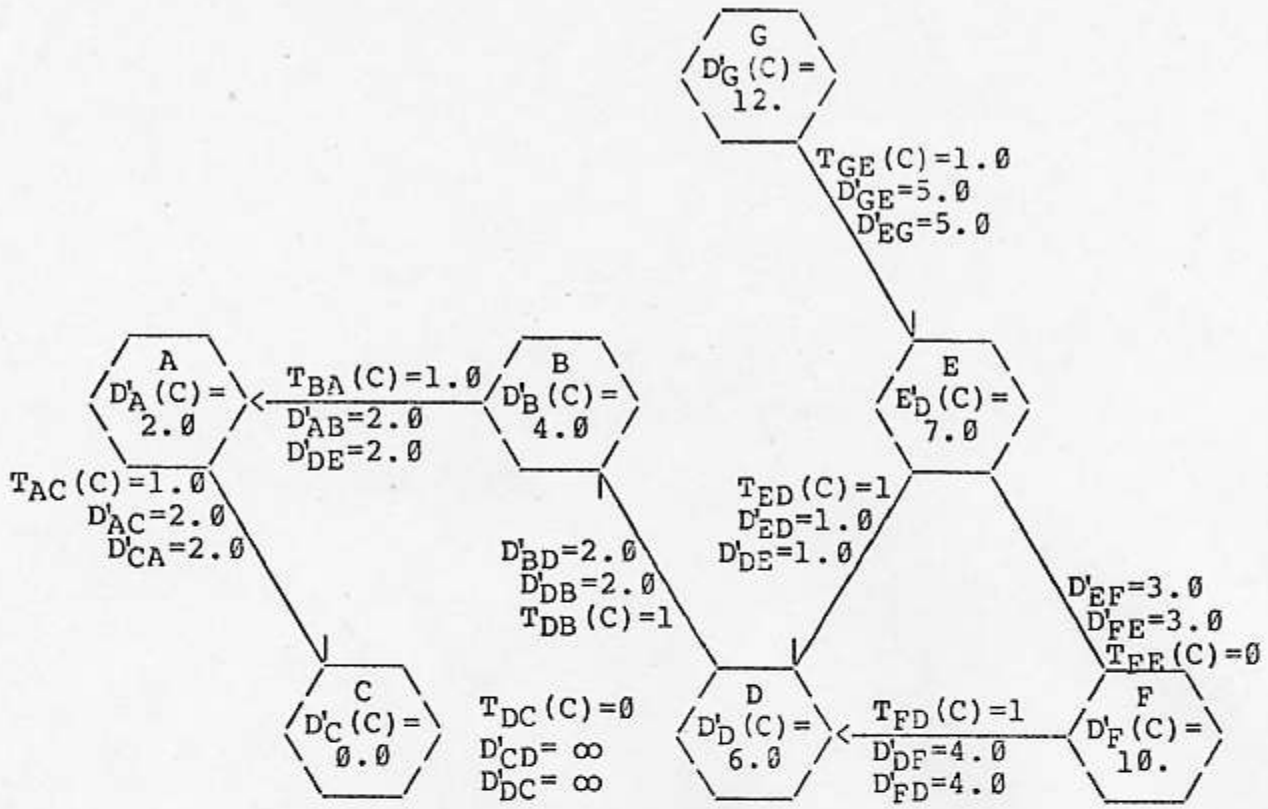
Node sending routing message	$D_i^!(C)$	blocked status	sent to	nodes that have rcvd all downstream routing msgs
C	Ø	not blocked	A	A
A	2	not blocked	B,C	B
B	4	not blocked	A,D	D
D	6	not blocked	B,E,F	E,F
E	7	not blocked	D,F,G	F,G
F	1Ø	not blocked	D,E	
G	12	not blocked	E	

The new values of the routing variables (calculated from the node marginal delays in the table above and the link marginal delays in the previous diagram) are as follows. Note that any node that had set $T_{ik}(C)$ to Ø for all k, now sets $T_{ik}(C)$ to 1 on receipt of the first $D'_k(C)$ not equal to infinity and for which k is not blocked with respect to j.

New routing variables, $T_{ik}(C)$

k\i	A	B	C	D	E	F	G
A	-	1	Ø	-	-	-	-
B	Ø	-	-	1	-	-	-
C	1	-	-	Ø	-	-	-
D	-	Ø	Ø	-	1	1	-
E	-	-	-	Ø	-	Ø	1
F	-	-	-	Ø	Ø	-	-
G	-	-	-	-	Ø	-	-

The state of the network is shown in the diagram below. Note that all nodes can now send traffic to node C. Nodes D, E, F, and G are routing traffic for C over link D,B rather than the failed link, D,C.


G
$D'_G(C)$ = 12.
$T_{GE}(C)$=1.0
D'_{GE}=5.0
D'_{EG}=5.0
E
$E'_D(C)$ = 7.0
A
$D'_A(C)$ = 2.0
$T_{BA}(C)$=1.0
D'_{AB}=2.0
D'_{DE}=2.0
B
$D'_B(C)$ = 4.0
$T_{AC}(C)$=1.0
D'_{AC}=2.0
D'_{CA}=2.0
$T_{ED}(C)$=1
D'_{ED}=1.0
D'_{DE}=1.0
D'_{BD}=2.0
D'_{DB}=2.0
$T_{DB}(C)$=1
D'_{EF}=3.0
D'_{FE}=3.0
$T_{FF}(C)$=0
C
$D'_C(C)$ = 0.0
$T_{DC}(C)$=0
D'_{CD}= ∞
D'_{DC}= ∞
D
$D'_D(C)$ = 6.0
$T_{FD}(C)$=1
D'_{DF}=4.0
D'_{FD}=4.0
F
$D'_F(C)$ = 10.