01 Dec 1984

Survey of Routing Algorithms for Computer Networks

Lu Yu

Thomas J. Sager

SURVEY OF ROUTING ALGORITHMS FOR COMPUTER NETWORKS

Lu Yu*

and Thomas J. Sager

CSc-84-16

Department of Computer Science

University of Missouri-Rolla

Rolla, MO 65401 (314) 341-4491

*This report is substantially the M.S. thesis of the first author, completed December 1984

ABSTRACT

This thesis gives a general discussion of routing for computer networks, followed by an overview of a number of typical routing algorithms used or reported in the past few years. Attention is mainly focused on distributed adaptive routing algorithms for packet switching (or message switching) networks. Algorithms for major commercial networks (or network architectures) are reviewed as well, for the convenience of comparison.

ACKNOWLEDGEMENT

I wish to express my deep gratitude to my advisor, Dr. Thomas J. Sager, who gave good lectures for the course of computer networks and provided valuable guidance and assistance throughout my work on the thesis.

Thanks are also due to Dr. John B. Prater and Dr. Farroll T. Wright for their special concerns over my work.

Last, but not least, I would like to thank Mr. Charles Spradlin of Monsanto Inc. and Mr. Luke Lien of IBM Inc., who gave me help in searching the literature.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

As computers have become smaller, cheaper, and more numerous, people have become more interested in connecting them together to form networks and distributed systems. The merging of computers and communications has had a profound influence on the way computer systems are organized. Hence, the advent of computer networks, by which we mean an interconnected collection of autonomous computers. The goal of such networks is twofold. One is to end the tyranny of geography, the other to provide high reliability by having alternative sources of supply. As a natural consequence of such goals, computer networks can provide a powerful communication medium among widely separated people. Some of the major advantages of building a large system from many small localized machines are: a favorable price/performance ratio, graceful degradation upon failure, and incremental growth.

In any network, there exists a collection of machines intended for running user programs. We call these machines hosts. They are connected by the communication subnet whose job is to carry messages from host to host. A subnet consists of two basic components: switching elements (or nodes) and transmission lines (or links or channels).

Broadly speaking, there are two general types of subnets: point-to-point and broadcast. The former type of subnet contains numerous cables or leased telephone lines, each connecting a pair of nodes. When a message is sent from one node to another via one or more intermediate nodes, the message is received at each intermediate node in its entirety, stored there until the required outgoing line is free, and then forwarded. Hence the name store-and-forward subnet.

Since the computer-to-computer traffic needs intermittent use of a high bandwidth channel, it entails packet switching or message switching rather than circuit

switching used in telephone networks for human-to-human traffic. The fundamental property of packet switching (or message switching) networks is that the bandwidth is acquired and released as it is needed, instead of being reserved in advance.

To conquer the complexity, a highly structured way is needed in designing networks. That is why most networks are organized as a series of layers (or levels), each built upon its predecessor. One of the most widely accepted models today is the 7-Layered Reference Model of OSI (Open Systems Interconnection) proposed by ISO (International Standards Organization) [94]. Figure 1 is a good illustration for this model.

When there are multiple paths (or routes) possible between source-destination pairs, at some point in the hierarchy of layers, a routing decision must be made. Such routing decisions are often a key design issue at layer 3, the network layer, or sometimes called communication subnet layer, in the ISO's OSI model. They could be based on static tables that are "wired" into the network and rarely changed. They could be determined at the start of each conversation. Finally, they could also be highly dynamic, being determined anew for each packet, to reflect the current network load.

The last class of routing techniques mentioned above are called adaptive ones, which have received considerable attention in recent years. A great number of new designs and implementations have appeared in the literature. The purpose of this thesis is to provide a survey of such adaptive routing techniques, with the emphasis on distributed algorithms, by which we mean that decisions are made by individual nodes throughout the network as opposed to the usage of central control. The scope of the review of algorithms will also be limited mainly to packet (or message) switching networks with point-to-point subnets.

Following the introduction, the problem of routing,

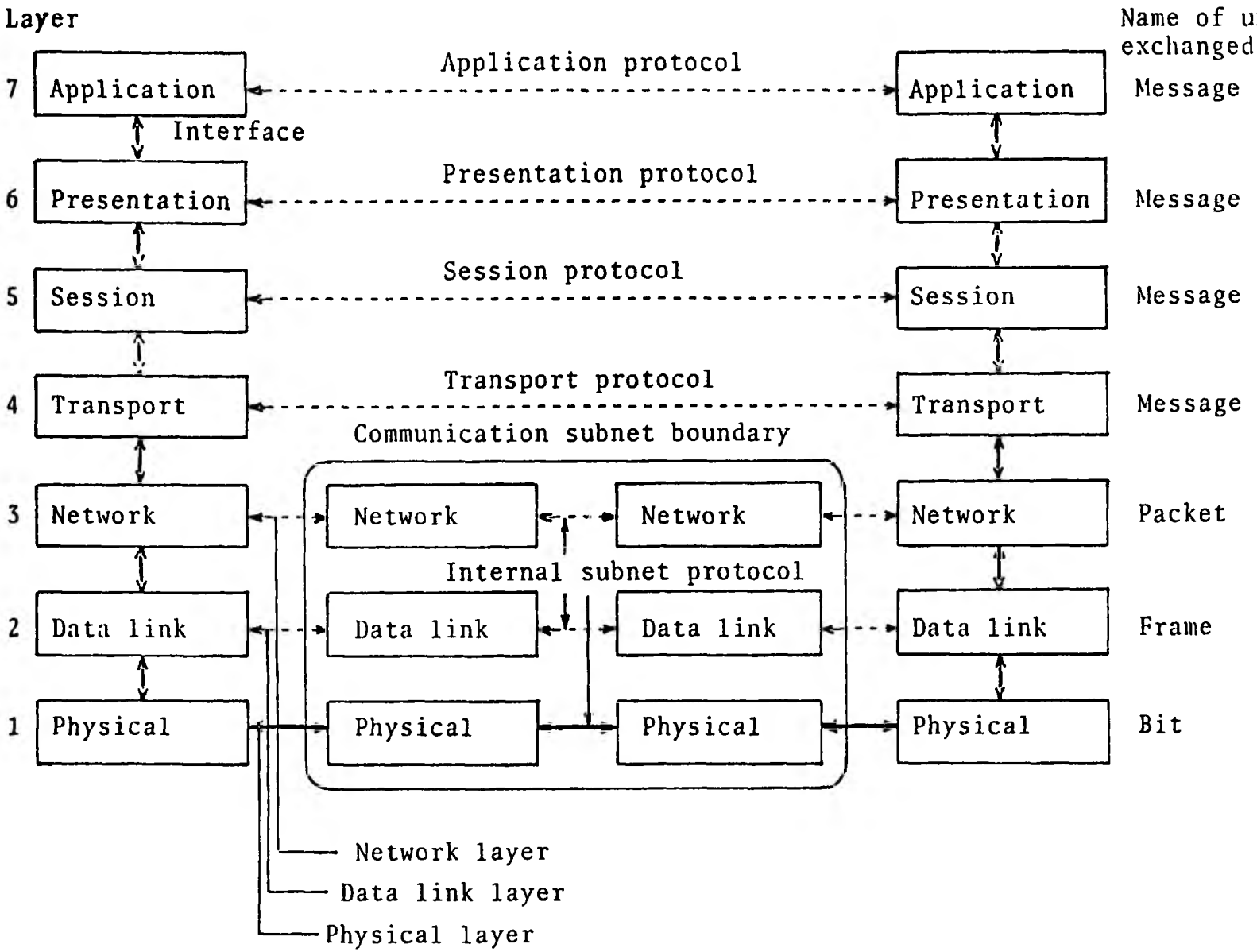


Figure 1. The network architecture based on ISO's OSI model

the classification of routing algorithms, and the advantages of adaptive and distributed routing are discussed in separate chapters. Then a comprehensive review is given of the typical algorithms developed and proposed in the past few years, which will hopefully provide a useful overview of the recent advancement of research in this area. Finally, an extensive bibliography is supplied for reference.

II. THE PROBLEM OF ROUTING

A computer network can be viewed as a network graph G = (X,A), where X is the set of network nodes and A is the set of transmission lines connecting the nodes. A path of a network joins two network nodes through a collection of connected lines. Such a directed path is a sequence of arcs (a_i, a_{i+1}, a_{i+2},..., a_n) such that the ending node of arc a_{i+k} is the same as the beginning node of arc a_{i+k+1}. These paths through the network are also called routes. A message starting from its source node follows the path to reach its destination node. Thus, the routing algorithms are the rules that determine the path(s) for each message from its source node to its destination node. Throughout this thesis, except for specifically indicated, a path is meant for a bidirectional path, i.e. duplex, in communications terminology.

Routing in networks involves sending each incoming message to its destination intelligently via a continuous path usually incorporating several lines. The implementation of the route chosen consists of setting up at each node along the path a routing table that directs messages with particular destinations to the appropriate outgoing line at that node. Since routing can be defined as the process of picking the "best" paths for traffic flow in the network, we should first discuss what the "best" means. Regardless of the variations and differences in design philosophies and implementations, there are certain properties that are desirable in a routing algorithm, i.e. correctness, simplicity, robustness, stability, fairness and optimality [86].

Correctness is quite self-explanatory. The property of simplicity assumes increasing importance as further requirements are placed on the algorithm and as complexity tends to grow. Robustness is very important, because once a network starts running, it is expected to be able to run

continuously for years without system-wide failures. This requires that the routing algorithms be able to cope with changes in topology and traffic. Such property may also imply reliability, adaptability or recoverability. Another basic requirement is that, given a static set of input data, the routing algorithm should arrive at a steady state solution rather than oscillating. Though elementary, stability should not be neglected either in the early design or in the later operation. Besides, the routing algorithm should be fair to competition for shared resources. The last, but not the least, property is optimality. Routing choices can stablize at many points in a given situation. In all but the best case, however, some network resources are being wasted and some network traffic handled inefficiently. The routing algorithm must seek to select the optimal paths, based on some combination of availability, error rate, queue lengths and estimated delays of the alternative paths. In a word, we seek to minimize the average delay for interactive traffic and maximize the total throughput for bulk traffic. Sometimes, however, strict global optimality would completely shut off traffic between some nodes, and this is unfair. So we need to find a trade-off between the two conflicting goals [50].

As Gerla analyzed in [31], the optimization of packet delay can be approached in two different ways, i.e. system optimization and user optimization. Using the former, the paths between all source-destination pairs are optimized jointly according to a common objective, the overall average delay. With the latter, on the other hand, each source-destination requirement is optimized independently until a competitive equilibrium is reached. It turns out that the routing solutions obtained using these distinct criteria are not very different, especially for large networks with uniform requirements.

It was summarized by McQuillan in [50] that evaluation

of a routing algorithm is usually in terms of performance and cost. The performance is considered in four respects: delay, throughput, cost and reliability. Five specific costs are likely to be incurred by any routing plan. They are: nodal bandwidth, nodal delay, nodal storage, line bandwidth and line delay.

The problem of designing routing algorithms has received considerable attention over the past few years. Considerable improvements have been made especially in terms of robustness and optimality, resulting in many valuable new techniques worth mentioning in the following part of this thesis.

What makes the routing problem a challenging one is that it is a problem distributed in space and in time. One must consider how to best allocate the resources available to a network to accomplish the work the network has to do at a certain time, but any global characterization of such work can be based only on the past as opposed to the current information values, which are usually used as an indication of the global state of the network. Because of the complexity of the problem, much of the existing comparison of algorithms has been carried out by simulation, the amount of analytical studies is very limited.

Most of the routing algorithms developed or implemented turn out to be variants, in one form or another, of shortest path algorithms that route packets from source to destination over a path of least cost [73]. Poisson arrivals, exponential message independence assumptions are usually made in the analysis so as to force the queueing model to be the M/M/1 type*. This is referred

---

* The notation M/M/1 is widely used for queueing models where the interarrival-time probability density and the service-time probability density are both exponential and the number of servers is 1.

to as the optimum routing rule [43]. Numerical methods such as flow deviation [27], gradient projection [3], [72] and others have been used to solve for the optimum flow distribution.

While the difference lies primarily in the choice of a line cost function used to establish the minimum cost path, the routing algorithms may also differ in the following aspects:

-- The place at which the algorithms are run.

-- How dynamic they are, i.e. how rapidly and in what manner they adapt, if at all, to changes in network traffic and/or topology information.

-- The actual implementation, e.g. the size of the routing table, the routing overhead required, etc.

-- The number of routes a packet (or message) is assigned (single-path routing or bifurcated routing).

-- The range in which optimization is attempted, system-wide optimization or user (end-to-end) optimization.

From different points of view, the routing algorithms are variously classified as in the next chapter.

III. CLASSIFICATION OF ROUTING ALGORITHMS

Research and development in the area of network routing algorithms is characterized by their increasing growth and diversity. A classification is needed before we can proceed to further talk about them.

For the purpose of classifying numerous routing algorithms, a cube was suggested by Rudin in [67] as is illustrated in Figure 2. One dimension tells where the decisions are made, either at the node in distributed fashion (D) or centrally (C). This dimension is shown in the horizontal line in Figure 2. The second dimension of the horizontal plane describes the kind of strategy to be used, on the one end is nonadaptive or invariant (I), and on the other adaptive (A). This axis can also be thought of as a measurement of the speed at which the routing algorithm can change or adapt. The vertical dimension describes the kind of information to be used in making decisions, either local (L), i.e. using only the information locally available at the nodes, or global (G) information.

As can be seen in the cube, one important way to classify the routing algorithms is according to how adaptive they are, with the ends of the scale consisting of purely static and completely dynamic strategies.

With purely static strategies, given fractions of the traffic at node i of the network for each of the other nodes $j \neq i$ are directed on each of the outgoing lines of node i. The paths for any source-destination pair are decided upon before the network starts operating. They are fixed in time, and depend only on the time and ensemble averages of the message flow requirements in the network.

At the other end of the scale are the completely dynamic strategies, which allow continuous changes of the paths. The paths can be varied not only as functions of time, but also according to topology and traffic changes in

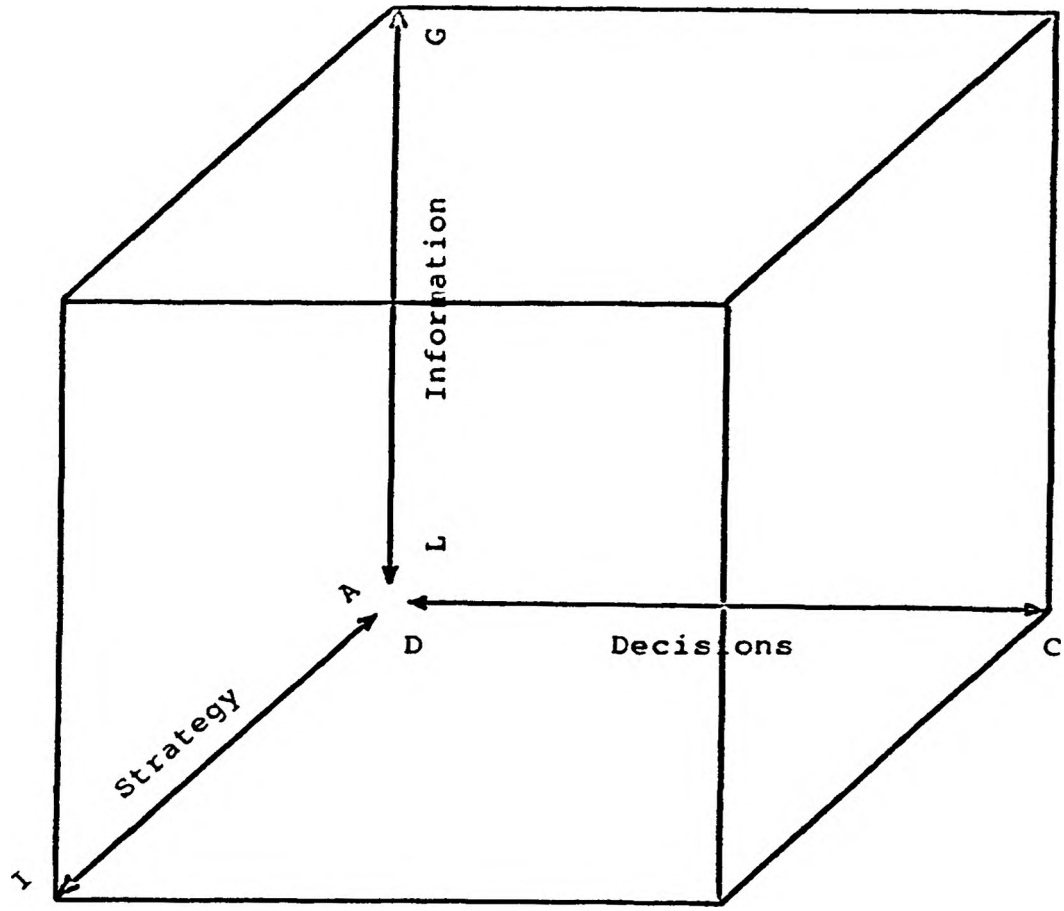


Figure 2. Routing algorithms in 3-space

various portions of the network. Dynamic routing is based upon the instantaneous state of the network.

Each of the extreme strategies has some advantages and drawbacks. The static routing is simple but unable to cope with changes in traffic and topology effectively. The completely dynamic ones are supposed to be able cope with these changes, but on the other hand, they may require a large amount of overhead. To have the desired properties of both, a strategy somewhere in between the two extremes is also often considered, according to Gallager [28]. That is quasistatic routing, where changes of paths will only be needed relatively infrequently. Reordering and individual addressing of messages are not needed, but if the topology changes or the traffic and delays build up in a particular section of the network, the paths will be changed accordingly.

In this thesis, we choose to include both the completely dynamic and the quasistatic into the category of adaptive routing.

The choice of control regime to be used in the operation of the algorithm is also a frequently used way of classification. Centralized routing means one in which routing decisions are made centrally by an NRC (Network Routing Center) and then sent to the nodes for execution. On the other hand, in decentralized routing, the decisions are made by individual nodes throughout the network.

Decentralized routing, however, can be further divided into isolated and distributed ones, depending on whether they make exclusive use of local information (isolated) or utilize the internode cooperation and exchange of information to arrive at routing decisions (distributed) [52].

As Rudin pointed out in [67], one would, ideally, like to operate at the top of the rearmost plane with a very adaptive (fast reacting) strategy based on global information. Whether this is achieved by means of

distributed or centralized decision is a question of implementation. Unfortunately, considerations of physical realizability may often prevent operation in this ideal region, the reason being that too much line capacity must be used in propagating status and routing information, leaving too little capacity for the transmission of "useful" data.

Routing algorithms have been studied, compared, and classified according to various criteria. In general, each approach seeks to optimize some set of performance criteria under a particular set of system constraints. The problem of choosing the best routing technique for a proposed new application requires careful study and considerable thought.

An excellent list of references can also be found in the paper by Schwartz et al [73]. There were many other studies on the classification of routing algorithms. Examples can be found in [43] and [23].

V. THE NEED FOR ADAPTIVE ROUTING

After an overview of a large variety of the routing algorithms available, the question arises: Which of them are better, and why?

In answering such a question, a comparison between static and adaptive routing strategies and between the philosophies behind them may be in order.

As mentioned before, the routing algorithms are designed for intelligently transmitting messages under various traffic conditions of the networks. The traffic conditions may vary due to a number of factors such as traffic input rates, transmission capacities of lines, processing capacities of nodes, topology changes, and flow control mechanisms used in the networks.

Static routing strategies, by its name, are predetermined as part of the network design, based on factors like network topology and average traffic conditions. Usually, they do not change during message transmission and network operation. Thus, an apparent merit of them is simplicity in implementation. No overhead is required for route recalculation, status information communication, etc. This may sound ideal. However, this is only good in situations where traffic requirements are predictable and without great variation. Unfortunately, much computer traffic in reality is bursty in nature. A user may ask to have a large file sent between two machines, putting a heavy load on portions of the subnet for a few minutes, and may then abstain from using the subnet for a long period of time. In such cases, the average traffic conditions, on which the static algorithms are based, can be of little value.

Adaptive algorithms, on the other hand, are capable of adapting to the network changes by changing the selected paths on which the packets are routed. Apparently, they seem more appropriate for actual computer networks.

Besides, nodes and lines are subject to failures. It is highly desirable to have networks capable of adapting to such topology changes. Another reason is that the inherent capability of the limited length of data units (packets) in packet switching networks can only be well exploited with adaptive routing. While the adaptive nature appears to be more advantageous than the static, it is not without drawbacks. The overhead caused by the routing calculations and status information exchanges is not negligible. Another sort of difficulty involves practical implementation. At this point, it becomes unclear which is more desirable after all.

Indeed, there are three different schools of thought. There are those who are in strong favor of the dynamic strategies. They usually base their conclusions on mathematical models and the simulations of these mathematical models. There are also those who prefer the static ones. Their conclusions are usually based on the experience with some operating networks. There are also some people who hypothesize that a combination of the two would probably result in a more ideal strategy. Their beliefs are usually derived from network measurements.

Before we draw our conclusion in this issue, some recent research work done by Chou, Bragg and Nilsson [15], [16], [17] is worth reviewing.

The approach in which they studied this problem is by classifying the traffic conditions into four categories. Investigations of preference for a static or an adaptive routing strategy were made with respect to the following four traffic categories:

1) balanced, emulating known and stationary traffic conditions;

2) balanced with surge, emulating a balanced traffic condition with possible unexpected sudden increase in traffic demands between some source-destination pairs;

3) unbalanced, emulating unknown or nonstationary

traffic conditions with low to moderate traffic loading;

4) chaotic, emulating unknown or nonstationary traffic conditions with heavy traffic loading.

A simulation program was used in the evaluation of the static and adaptive routing strategies under the above four different traffic conditions.

From a quantitative point of view, they characterized a routing strategy by two features:

1) The delay metric function used to determine routes and routing table. Associated with each line in the network is a metric. It is usually a function of the delay experienced by a packet queueing and transmitting through the line or a function of the number of packets queued for the line.

2) The frequency of updating routing tables. This is a compromise between the desire to propagate the changes as soon as they are detected and the amount of the overhead generated by the updates.

In their simulation, they generalized the metric function into

$$a_0 + a_1 Q + a_2 Q^2$$

where Q is the queue size at the time of routing update and a_0, a_1 and a_2 are coefficients. By appropriately choosing the coefficients, as they observed, such a metric could define a routing strategy that behaves almost statically when the traffic is reasonably balanced (queue size Q is small) and adaptively otherwise (due to the increased impacts of the second and third terms).

For each of the four traffic conditions, one static and three adaptive strategies are compared. The three adaptive strategies are:

1) metric is 1 + Q and update frequency is 10 seconds (similar to the new ARPANET strategy);

2) metric is 1 + 0.25Q and update frequency is 0.25

second (similar to the old ARPANET strategy);

3) metric is $1 + 2Q/15 + Q^2/50$ and update frequency is 0.25 second (a well-chosen strategy derived from their analysis and simulation on a hypothetical network model).

Their simulation results for average message delay as a function of network throughput for the four traffic conditions are given in Figures 3, 4, 5 and 6. From the results, we can see that among the four traffic conditions, only the balanced conditions verify the static routing stategies, and adaptive ones are definitely more desirable for unbalanced or chaotic conditions.

Although the flexibility of adaptive routing is achieved at the cost of additional software complexity, the transmission facility resources saved in providing the same grade of service as the nonadaptive ones more than offset the additional cost under unbalanced or chaotic conditions.

In the perspective of new development in the future, adaptive routing is undoubtedly a likely direction. The fast growing computer technology will, in the long term, justify the complexity of adaptive routing.

It is also reasonable for some operational networks to keep using static routing strategies for some particular traffic conditions, since the cost of changing the entire routing mechanism may not be worthwhile. Besides, static routing finds an important application in the network design process, because the analysis of adaptive routing is an extremely difficult task.

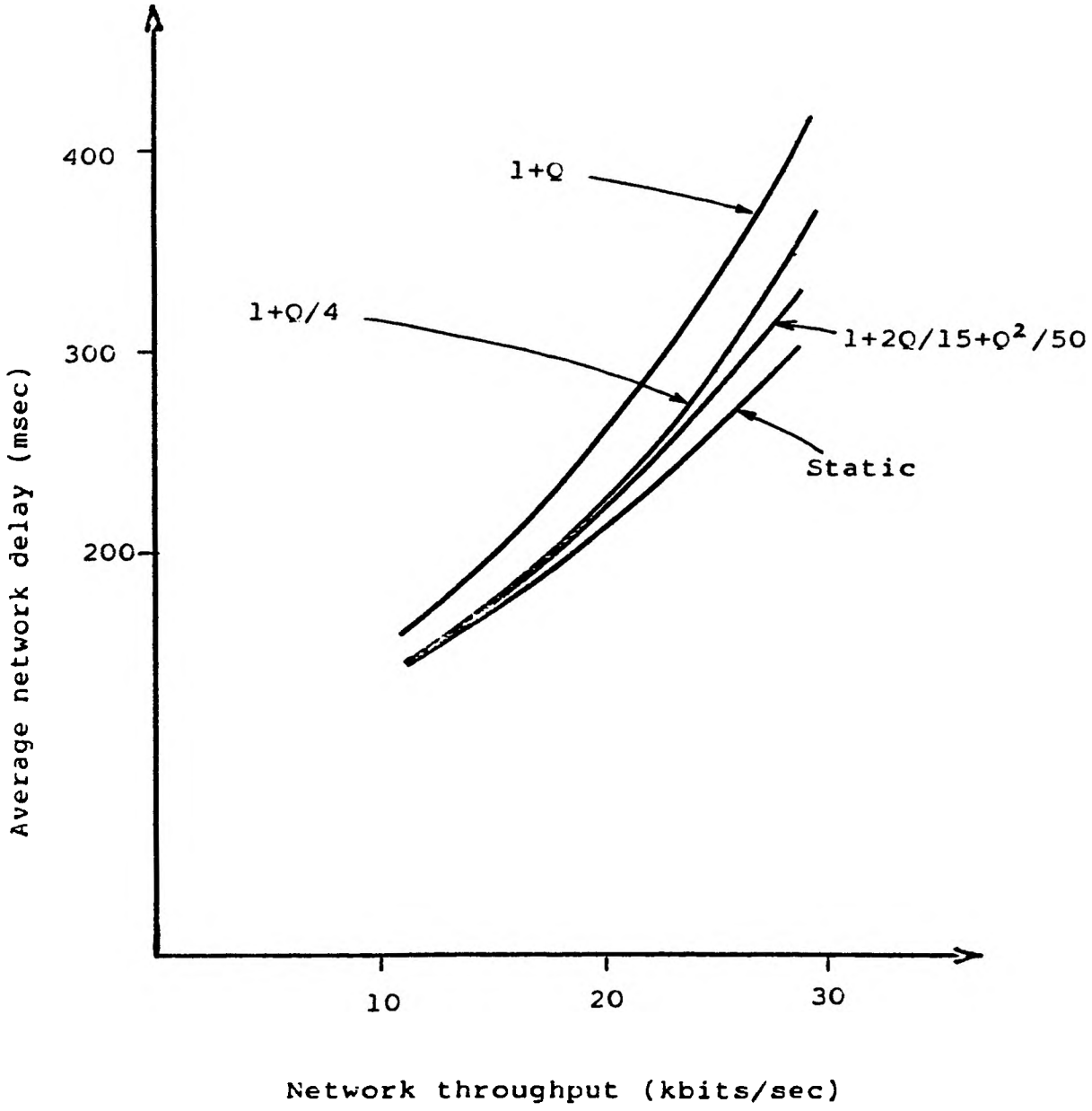


Figure 3. Performance with balanced traffic

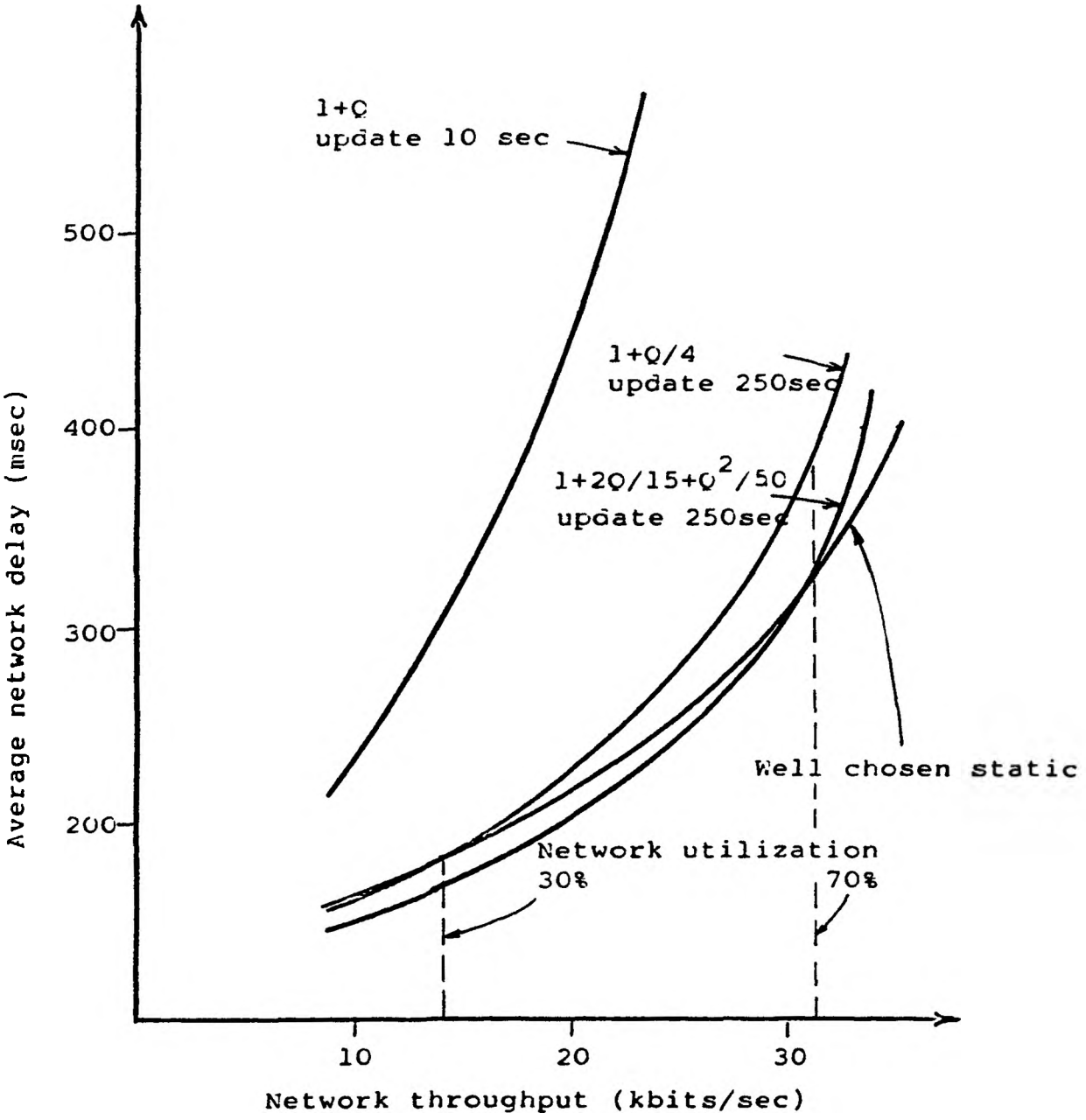


Figure 4. Performance with balanced traffic with surge

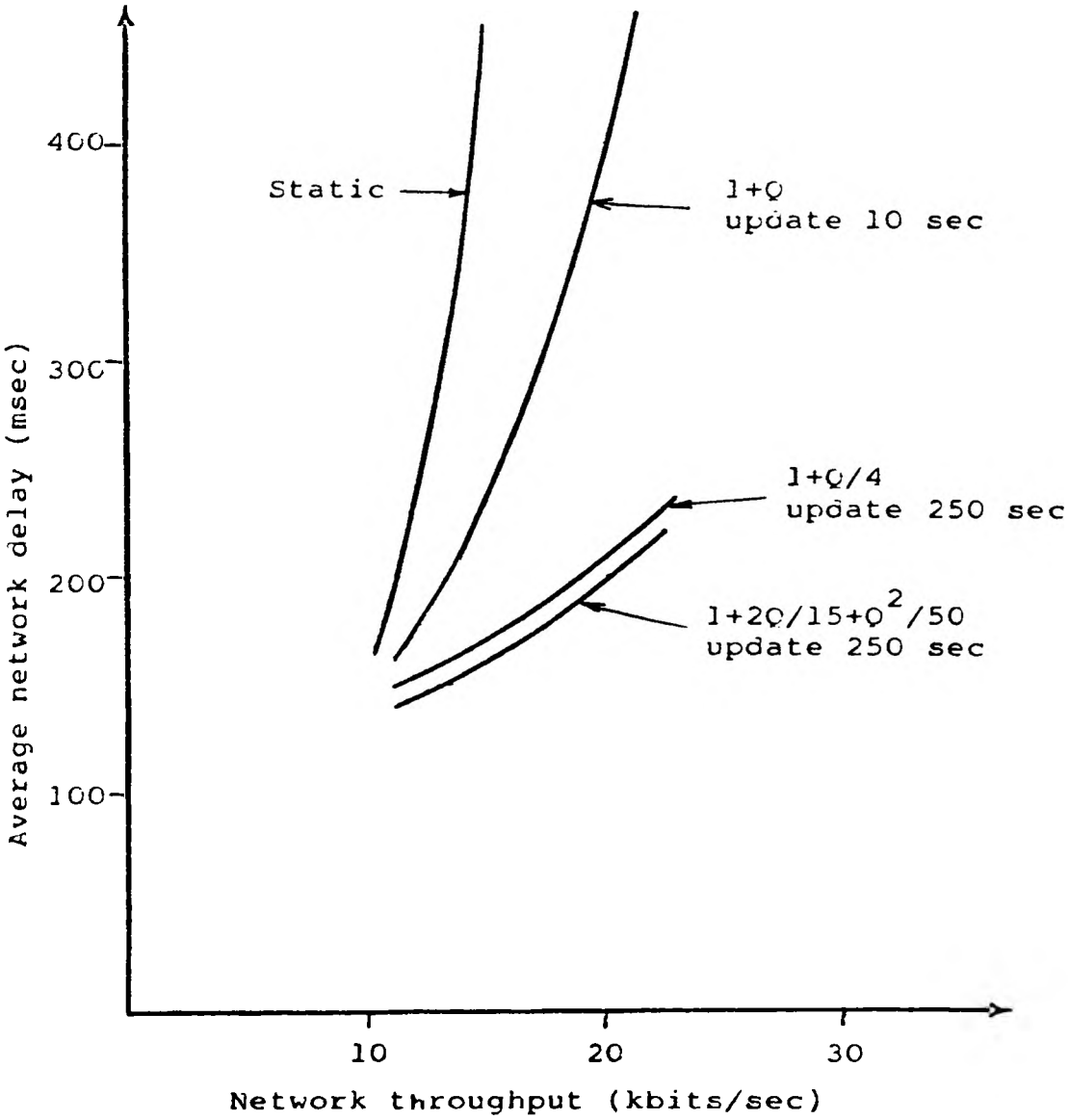


Figure 5. Performance with unbalanced traffic

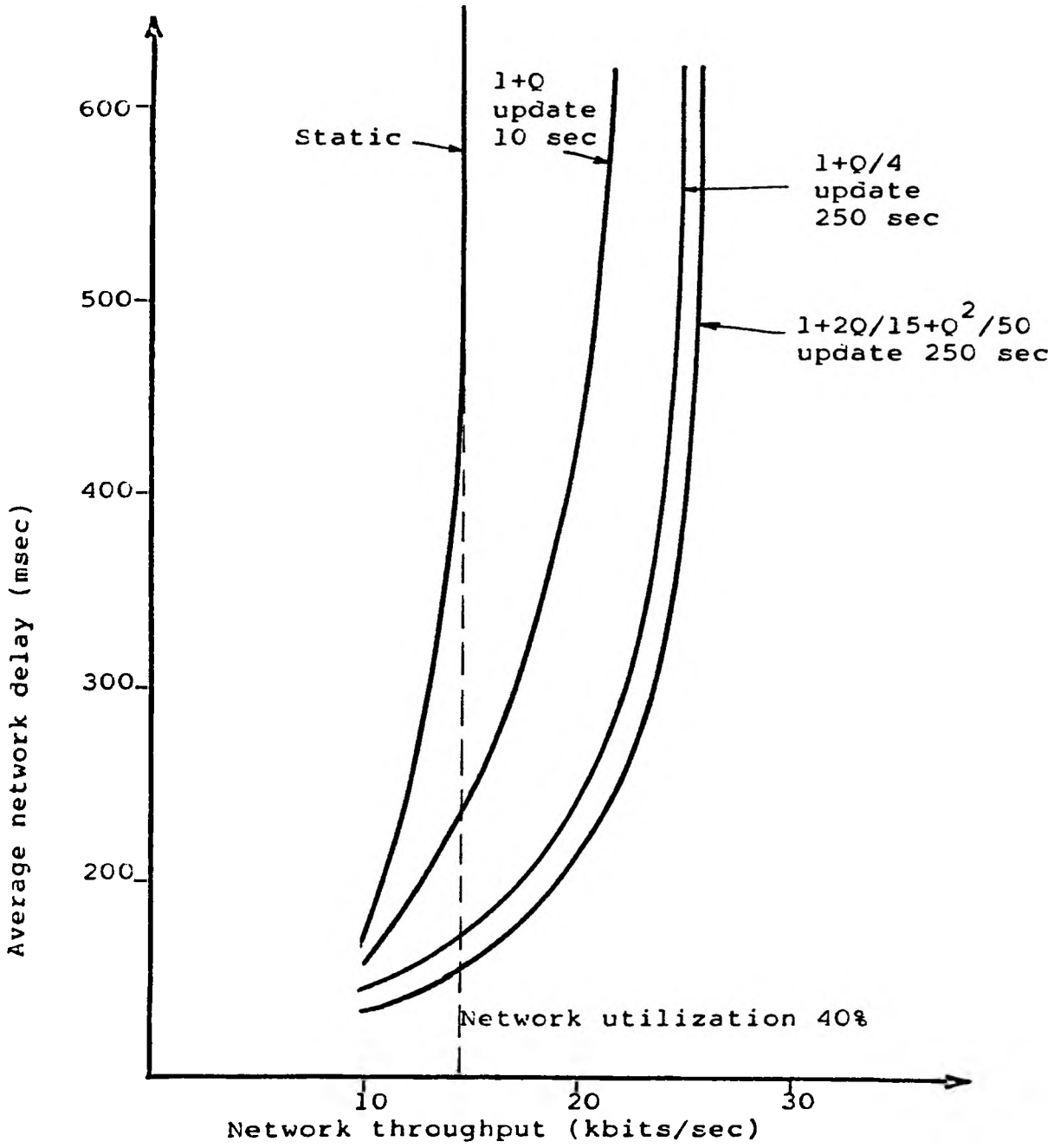


Figure 6. Performance with chaotic traffic

VI. THE NEED FOR DISTRIBUTED ROUTING

Now we turn to another question: Which is preferable, centralized routing or distributed routing, and why?

With centralized routing, somewhere within the network there is a Network Routing Center (NRC), which periodically receives status information sent from each of the nodes and uses the collected global information to compute the optimal paths for the source-destination pairs. From the results of such computation, it builds new routing tables and distributes them to all the other nodes.

Distributed algorithms, on the contrary, exercise no central control over the network routing. Each node exchanges status information with other nodes and makes routing decisions on its own.

Two aspects of the performance of routing algorithms can be used in judging the relative advantages and disadvantages of each of the two philosophies. One is long term in nature, in which one hopes that the network is operated in an efficient manner, i.e. the resources are used wisely so that one resource does not remain idle while another (equivalent) resource is overtaxed. Another is of short term, in which one wants, in addition, the network to react quickly when a traffic burst must be handled or when a resource fails.

Some experience has shown that centralized routing strategies are more efficient in the long term aspect, given stable traffic flows. This is because a single entity (NRC) with global knowledge of the network status as last reported can make consistent decisions. The decisions made distributively at each node tend to be efficient only in the environment local to that node, possibly resulting in a network not working consonantly as a whole. Looping in the old ARPANET algorithm is such an example.

On the other side of the coin, distributed strategies allow a node to respond much more rapidly to a change in

traffic or topology in its own immediate environment. In addition, distributed routing exhibits a number of other advantages in those aspects where centralized routing appear very weak, as Tanenbaum noted in [86].

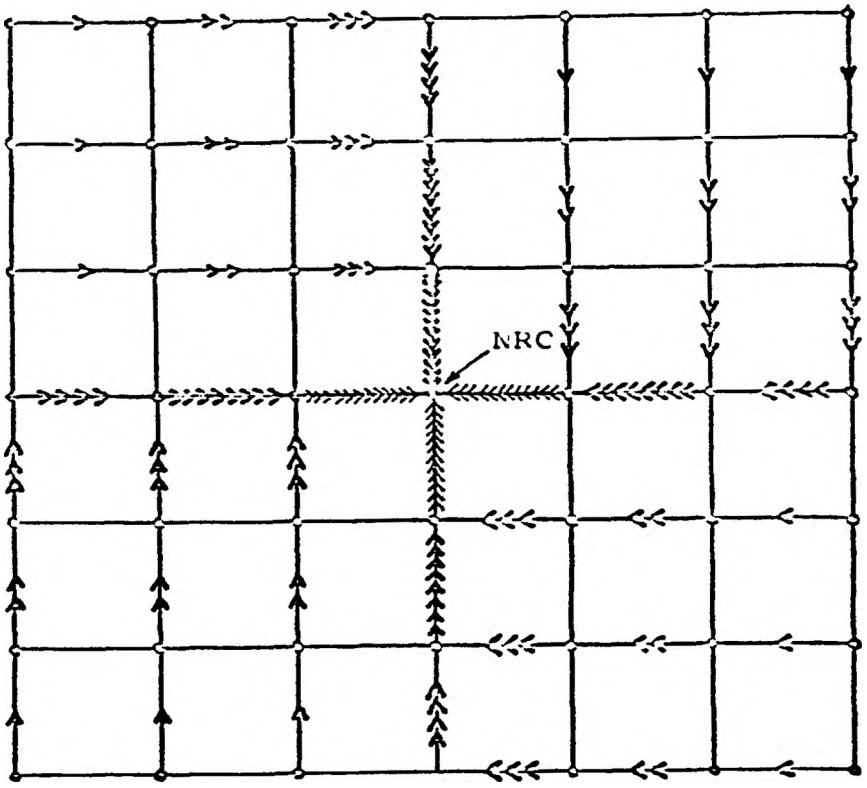
For centralized routing, if a subnet is to be able to respond to changes in traffic, the routing calculation will have to be performed very frequently. If the network is a large one, then the amount of such calculation will impose a heavy burden on the CPU.

A more serious problem of centralized routing is the vulnerability of NRC. In the situation where the NRC goes down or isolated by line failures, the subnet is suddenly put in a disaster. If a second machine is used to work as a backup to remedy the vulnerability, it will result in even more computation, and an arbitration method is also needed in case the primary NRC and backup NRC present inconsistent results.

The theoretical argument in favor of using centralized routing is, in the first place, that it can find optimal paths. However, if it does not use alternate paths for the source-destination pairs, the failure of even a single line or node will probably cut some nodes from the NRC, resulting in disastrous consequences. If alternate paths are to be used, then the advantage of centralized routing stated above will be weakened.

Since the NRC has to collect status information from all nodes throughout the network, the routing traffic will be heavily concentrated on the lines leading into the NRC. Those lines near the NRC with heavy load will consequently be very vulnerable. This situation can be illustrated as in Figure 7.

Besides the above vulnerabilities, the way in which the NRC distributes the routing information to the nodes throughout the network may lead to some other undesirable problems. For example, the nodes that are close to the NRC will receive their new routing tables early and will switch



On the shortest path from each node to the NRC, there are a number of arrows. Each arrow represents that the node is reporting to the NRC via that line. The closer to the NRC a line is, the more arrows there are on that line, consequently the more vulnerable that line is.

Figure 7. Routing traffic concentration near the NRC

over to the new paths before the distant nodes have got their tables. Inconsistency may arise and the packets, including those of the routing tables for the distant nodes, may be delayed, making the inconsistency from bad to worse.

Distributed routing strategies are supposed to be able to resolve those problems stated above for centralized ones. They get the traffic burden of transmitting routing information more evenly distributed within the network. Failures of lines or nodes will not cause so serious consequences as with centralized routing.

The nature of distributed routing allows the status and routing information to be exchanged and processed more quickly than with centralized ones. Therefore the decisions are made based on more up-to-date information and the network has better adaptability to changes in traffic and/or topology. That is to say that the essential philosophy behind adaptive routing can be better realized with distributed strategies.

Of course, distributed routing is not without weakness. For all its drawbacks, it is still a preferable direction of development for routing algorithms, in the author's viewpoint.

XI. EXAMPLES OF ROUTING ALGORITHMS

In recent years, many developments in the design and implementation of routing algorithms for computer networks have been reported in the literature. A large part of them fall into the catagories of adaptive routing and distributed routing. Some of the major commercial networks or network architectures use routing algorithms not belonging to these categories. For the convenience of comparison, however, they are reviewed as well as the adaptive and distributed ones in this chapter.

A. ARPANET ALGORITHM

The last decade has seen numerous designs, implementations and operations of distributed routing algorithms. ARPANET is one of the earliest and most important.

It was generally agreed that the first published description of a packet switching concept was contained in a 1964 study report by P. Baran of the Rand Corporation. In 1966, an experimental packet system was set up under the sponsorship of the Advanced Research Projects Agency (ARPA). The first link joined a computer at the System Development Corporation with one at M.I.T. Lincoln Laboratory. Out of this beginning grew the ARPANET, which now connects well over one hundred universities and research facilities across the United States, Hawaii, and Europe. It is a research-oriented system operated by the United States Defense Communications Agency (DCA), and is used as a test bed for many research areas including routing and flow control.

Though the original routing algorithm designed in 1969 for the ARPANET had served remarkably well considering how long ago in the history of packet switching it was conceived, many corrective modifications had been made before 1979. Then, a new algorithm was designed and installed. The new algorithm has undergone extensive tests and turned out to be an effective improvement over the old one. In this section, an overview of the new algorithm will be given after a brief introduction of the old one. Details of these algorithms can be found in [50], [51], [52], [54], [54], [55], [66].

The original ARPANET routing algorithm can be summarized as follows: Each packet is directed toward its destination along a path for which the total estimated transit time is smallest. Instead of determining this path in advance, each node, also called IMP (Interface Message

Processor) in ARPANET terminology, individually decides which line to use in transmitting a packet addressed to a destination. A simple table lookup procedure is used for this selection. For each possible destination, an entry in the routing table at each node designates the appropriate next line in the path.

Each node also maintains a network delay table giving the delay calculated for a packet to reach every possible destination over each of its outgoing lines. Every 2/3 of a second, the node calculates the minimum delay to each destination and puts them in its minimum delay table. The number of the line giving minimum delay is accordingly kept in the routing table for use in routing packets. Each node also sends its minimum delay table to each of its neighbors every 2/3 second. Therefore each node receives a minimum delay table from each of its neighbors every 2/3 second. After all the neighbors' estimates have arrived, the node adds its own contribution to the total delay to each destination. Thus the node accomplishes the computation of the total delay to each destination.

In parallel with the above computation, the nodes also compute and propagate shortest (minimum hop count) path information in a similar fashion. An upper limit of the number of the hops in the longest path in the network is used as cut-off for disconnected or nonexistent nodes. This information is only used for the "reachability test". It also travels at roughly 2/3 second per line, so that changes in topology are recognized by the whole network in only a few seconds.

The algorithm was a good design in that it was simple, inexpensive and performed well in steady state and in reacting to small changes in traffic. However, it did have some problems, some of which being fundamental that required a complete redesign. As summarized in [55], the following are the major problems to be addressed.

1) As the network grew larger, the size of routing

packets would become correspondingly larger and could adversely affect the flow of network traffic.

2) The distributed manner of route calculation could not easily ensure the consistency of the routes used by different nodes.

3) The rate of exchanging routing tables and the distributed nature of calculation made the network adapt too slowly to congestion and to important topology changes, yet too quickly (perhaps inaccurately) to minor changes.

4) Periodically the node counted the number of packets queued for transmission on its lines and added a constant to it. This delay measurement procedure was quite simple, but was inaccurate, because the queue length was only one of the many factors that might affect a packet's delay. Lines have different speeds and propagation delays, and packets queued for each line have different sizes. The waiting time for a packet to get some resources before being queued may be long. Yet none of these were reflected by the delay measurement -- queue length. And the significant realtime fluctuation in queue length at any traffic level could not be predicted by the instantaneous measurement of queue length, either.

McQuillan et al reported in [55] that the new algorithm is an improvement over the old one in that it uses fewer network resources, operates on more realistic estimates of network conditions, reacts faster to important network changes, and does not suffer from long-term loops or oscillations. This new algorithm is described here in terms of three of its basic components.

1) Routing Calculation.

The SPF (Shortest Path First) Algorithm attributed to Dijkstra [22] is employed for this purpose. A tree representing the minimum delay paths from a given root node to every other node is generated using a database that specifies which nodes are directly connected to which other nodes, and what the average delay per packet is on each

network line, both types of data being updated dynamically on the basis of realtime measurement. Starting from just the root node, the tree is augmented to contain the node that is closest (in delay) to the root and that is adjacent to a node already on the tree. The process continues by repetition of this last step. Eventually the furthest node from the root is added to the tree and the algorithm terminates. The tree constructed is used in creating the routing table, and the routing table is used in forwarding packets.

To reduce the amount of computation, an important modification has been made to the SPF algorithm. When a single line delay changes (or if a line or node is added or deleted), each node does a partial computation to reconstruct its shortest path tree. Thus it is an incremental calculation rather than a complete recalculation of all shortest paths.

2) Delay Measurement

This is a crucial aspect of the routing algorithm. Each node measures the actual delay (including processing, queueing, transmission, retransmission and propagation time) of each packet flowing over each of its outgoing lines by means of time-stamp, and calculates the average delay every 10 seconds. Only when the change in line delay since last report exceeds a certain threshold will the delay measurement be transmitted. The threshold is a decreasing function of time.

The choice of 10 seconds as the measurement period represents a significant departure from the old algorithm. Though a longer period means less adaptive routing if conditions actually change, a shorter period means less optimal routing because of inaccurate measurements. The queue lengths varied rapidly with time and the short measurement period might result in adaptivity so quick that the perceptions of shortest paths could change during the period a packet traversed the network, i.e. too frequent to

be accurate. Since the routing update generated by a particular node contains information only about the delays on its outgoing lines and is transmitted less frequently, the total communication overhead involved in delay update exchanges is quite small (less than one percent).

Another aspect is that the measurement periods are not synchronized across the network. In different nodes the measurement periods are randomly phased. This is an important property, because synchronized measurement periods could, in theory, lead to instability.

3) Updating Policy

This is also of critical importance, because it must ensure that each "update" packet is actually received at all nodes so that identical databases of routing information are maintained at all nodes. Hence the flooding method, in which each update packet is transmitted unchanged to all nodes (not just to the neighbors) on all lines. Transmitting update packets back to the adjacent node from which it was received provides an automatic acknowledgement mechanism. Duplicated update packets are dropped. While such information propagates through the network, it does not circulate infinitely. Since the update packets are handled with the highest priority, they flow very quickly (within 100 ms) through the network.

One difficult point is that some nodes may become disconnected and then join the network after some period of time. How to ensure that databases at all nodes are correctly updated? To take care of this problem, an "age" field is used in each update packet. Out-of-date delay information can be recognized and discarded when lines are reconnected and routing tables recomputed. Also helpful to this purpose is the mechanism of the "waiting" state for a node to get enough updates before it can actually come up.

Since all nodes perform the same calculation on an identical database, there are no permanent routing loops. Transient loops may still form for a few packets when a

change is being processed. This is, however, quite acceptable, since it has no significant impact on the average delay in the network.

If the new algorithm is to be compared against the old one, some results can be summarized as follows, according to McQuillan et al [55].

1) Better utilization of resources (line and processor bandwidth).

2) Quicker and more correct response to topology changes.

3) Better congestion control.

4) Less instability or oscillations due to feedback effects.

5) No significant impact of loops on the average delay of the network.

6) More capability of coping with heavy load.

7) Tendency to route traffic on minimum hop paths.

As they pointed out in [55], there is a sense that the old routing computation is a distributed, global one in that the inputs to the computation at one node are the outputs of the computation at the neighboring nodes. Since the nodes perform the computation in an unsynchronized manner, the output of the global computation at any instant depends more on the history of events around the network than on the network traffic at that instant. The new algorithm, on the other hand, is a local computation. It does depend on measurements made all around the network, but the updating protocol provides these measurements to all nodes unchanged and unprocessed. The SPF computation at one node never learns of the results of the SPF computation at any other node. In this way, the new algorithm keeps the advantages of distributed routing while dispensing with the disadvantages of distributed computation. For this reason, the new algorithm is also viewed as "partially centralized" method by Schwartz in [73].

Finally, it should be noted that the new algorithm does take about three times the memory as the old one, but this point does not alter the conclusion that the new algorithm is indeed a good improvement.

B. CHU: ALGORITHM FOR TOPOLOGY UPDATE PROBLEM

The old ARPANET algorithm and all others of its type -- built on repeated distributed minimization or maximization -- share a flaw: They have the property that the reachability algorithm reacts very quickly to "good news" but very slowly to "bad news". Take the old ARPANET algorithm for example. If the number of hops to a given node decreases, the nodes soon all agree on the new, lower number. If the hop count increases, however, the nodes will not take action on the reports of higher counts while they still have neighbors with the old, lower values. They simply increase their hop counts by two in each update cycle.

One early solution to this adaptivity problem is the "hold down" method [51]. It works by "purging" the surrounding nodes of any out-of-date information before the nodes will accept any new information. Because the entire hold down mechanism is rather ad hoc, researchers have been looking for better ways to propagate information about changes in the topology. Among several algorithms which make explicit use of the concept of sink tree, Chu's research report [18] is a good representative, and will be reviewed in this section.

A sink tree is a tree rooted at the destination with all the other nodes connected on their shortest paths to the root. Based on the optimality principle*, the set of optimal paths from all sources to a given destination form one sink tree [86]. Figure 8 illustrates a network with

---

* The optimality principle of dynamic programming states that the optimal path between two points in a network is the sum of optimal subpaths. To put it another way, if node J is on the optimal path from node I to node K, then the optimal path from J to K also falls along the same route.

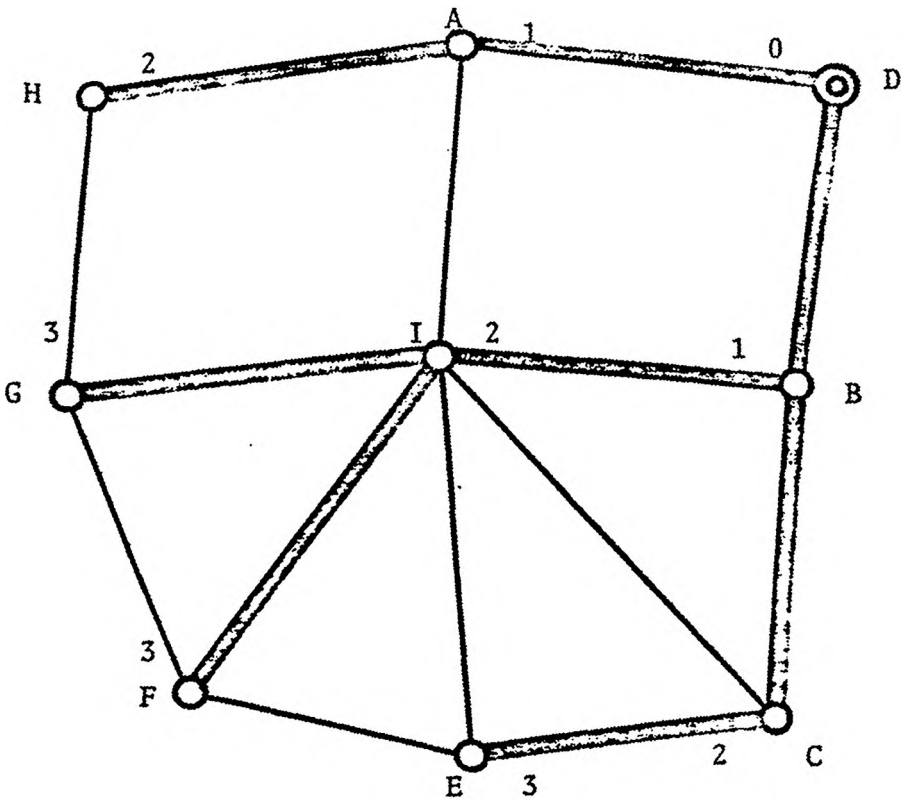


Figure 8. Sink tree for destination node D

nine nodes and the sink tree for destination node D. Since a tree does not contain any loops, each packet will be delivered within a bounded number of hops.

Chu's algorithm makes particular efforts to recognize interdependent relations from the information exchanged among neighboring nodes. For each destination, a sink tree (also called shortest path spanning tree here) is established to identify its downstream neighbor and upstream neighbors. Such trees are implemented by means of "flow labels" used in each node's "critical distance table", in which the current distances estimates to each of its possible destinations over each of its outgoing lines are recorded. The under-bar flow label for the entry at row D, column B implies that node I has chosen node B as its downstream neighbor in the sink tree for node D. The upper-bar flow labels for the entries at row D, columns F and G imply that node I realizes that it is the downstream neighbor of node F and of node G in the sink tree for node D. The distance is measured in terms of hop count. For a network with N nodes, the longest path can be no longer than (N-1) hops. If the path is selected by the shortest distance and the downstream-upstream relations are consistently designated for all nodes, there should be a sink tree rooted from each destination node. Figure 9 shows the critical distance table at node I for destination D, corresponding to part of the sink tree in Figure 8.

Let the shortest distance from node I to node J be denoted by d(I,J), so that all the adjacent nodes of J should have their distances from I as d(I,J)+1. As Chu noted in [18], the following rules hold for the above structure.

1) There can be only one downstream node J for a given node I in its sink tree for a particular destination D, so that the shortest distance from node I to destination D, d(I,D) should be the entry at row D and column J in the table at node I.

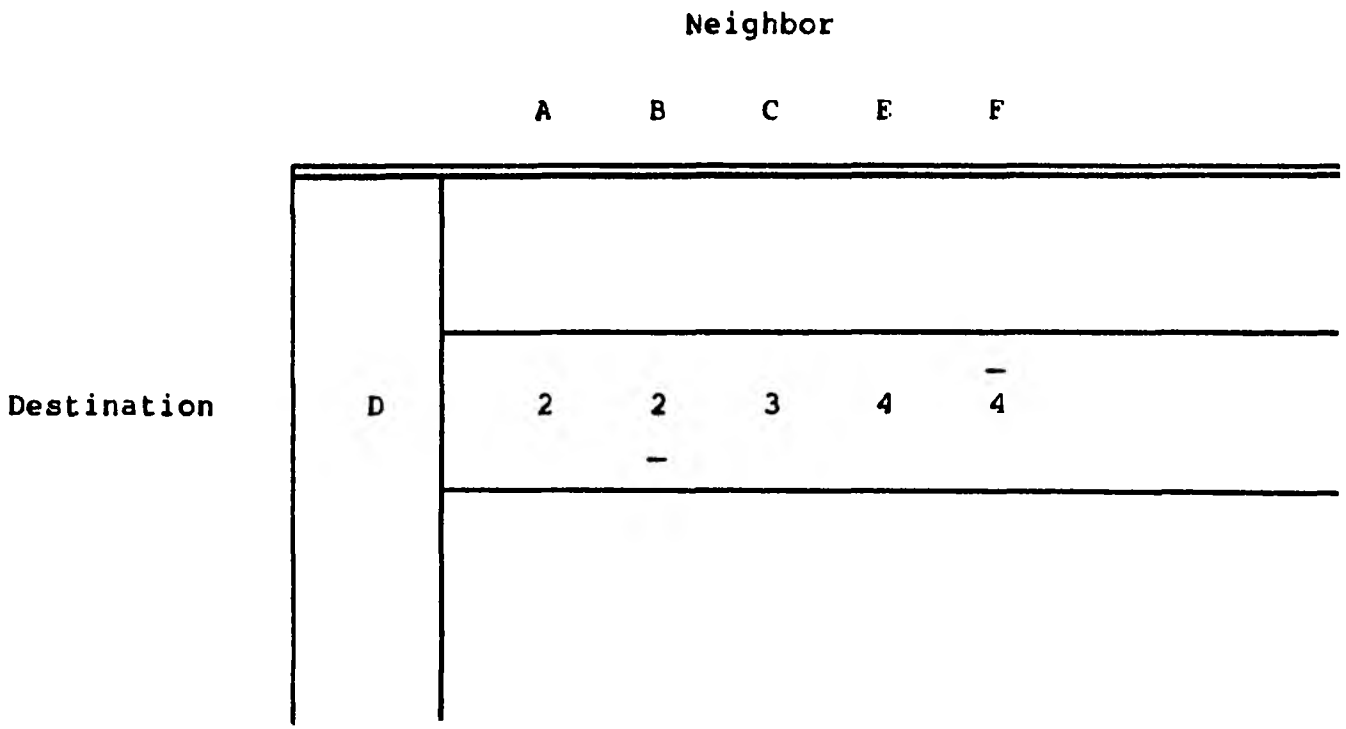


Figure 9. Critical distance table at node I

2) The distance from node I via any upstream node to destination D should be d(I,D)+2.

3) All the distances from node I to destination D via other neighboring nodes should be either d(I,D), d(I,D)+1, or d(I,D)+2.

A set of procedures are designed to deal with various situations of topology changes. If there are any inconsistencies according to the rules, certain procedures will be activated to make them consistent while exchanging messages about topology information. Each such messages includes a bit, telling whether or not the sending node desires to take the receiving node as its downstream node in the sink tree for a particular destination.

If a node detects a failure from its downstream line, it chooses a new downstream node from the set of unlabelled neighbors. If there are no unlabelled neighbors, the former upstream node is chosen to be the new downstream node. If the failure detected is not from its downstream lines, the node simply erases the corresponding column in the table.

If a node detects the coming-up of a new line, it adds a new corresponding entry in its table and sends the information around for choosing new possible paths.

If node A receives a topology message from a neighbor, say B, and B did not request to have A as downstream node, A will update its table, choose a new downstream node and propagate the news to its neighbors.

If node A receives a message from its former downstream node B and B did request to have A as a downstream node, A will seek a way to alter the direction of traffic flow as in the case of downstream line failure. If the distance of the new path is larger than the number of nodes in the network, the procedure will quit. The updating is stopped until some new change happens.

Chu's algorithm provides a good way to solve the so called topology update problem or adaptivity problem, but

it is only concerned about topology changes. For the purpose of adapting the sink tree to changes in traffic as well as in topology, some other algorithms were developed. Segall et at devised a number of failsafe algorithms, which will appear in next section. Some of them use the concept of sink tree [59], [76], [77] to maintain loop-free routing.

C. GALLAGER: MINIMUM DELAY ALGORITHM AND SEGALL ET AL: FAILSAFE ALGORITHMS

In 1981, Segall and Sidi published a protocol [81] possessing the following features:

1) Distributed computation.

2) Loop free routing for each destination at all times.

3) Adaptability to slow load changes.

4) For stationary input traffic and fixed topology, the protocol reduces network delay during each cycle, and minimum average delay is obtained in steady state.

5) After arbitrary number, location and sequence of topology changes, the network recovers in finite time in the sense of providing routing paths between all connected nodes. In addition, nodes that are not affected by the topology change continue the algorithm and adapt to the new load pattern in a smooth way.

This algorithm is designed after some early ones such as minimum delay algorithm [29], optimal distributed algorithm [75], recoverable algorithm [80] and failsafe distributed algorithms [24], [59], [76], [77]. It will be helpful to first review the minimum delay algorithm due to Gallager.

In 1977, Gallager proposed a minimum delay routing algorithm using distributed computation. It is an algorithm for a quasistatic environment, where the traffic statistics for each source-destination pair change slowly over time and furthermore individual traffic samples do not frequently exhibit large and persistent deviations from their averages. The algorithm was defined for establishing routing tables in the individual nodes of a network. The routing table at a node i specifies, for each other node j, what fraction of the traffic destined for node j should leave node i on each of the outgoing lines of node i. The algorithm is applied independently at each node. It

successively updates the routing table at that node, calculates the marginal delay (incremental delay estimated by means of partial derivative with respect to traffic flow) of each outgoing line based on information exchanged between adjacent nodes, reduces the fraction of traffic sent on nonoptimal lines, and increases the fraction on the best line by some small quantities properly selected. Such flow deviation will produce a net delay saving. For stationary input traffic the average delay per message through the network converges, with successive updates of the routing tables, to the minimum average delay over all routing assignments.

In order to guarantee the traffic to each destination to be loop free at each iteration of the algorithm, some rules are enforced that the updating must start from the destination node and propagate back to the source node, i.e. a node cannot update its tables until it has received the delay information from all its downstream neighbors. After a node has completed the update, it will broadcast its delay information to all its neighbors. This is different from the old ARPANET algorithm where the transmissions of updates are unordered.

Comparison between this algorithm and the ARPANET algorithm also shows some other differences. Gallager's algorithm is intended for static or quasistatic environments, where the time required to converge to the optimal solution is not critical. Topological changes are not successfully coped with by this algorithm. The ARPANET algorithm, on the other hand, is adaptive in the sense that it takes into account all the above factors. Besides, the ARPANET algorithm attempts to send each packet over a route that minimizes that packet's delay with no regard to delay of other packets, while with Gallager's algorithm, the packets are sent over routes to minimize the overall delay of all messages. This is a difference between the "user optimization" and "system optimization". Another point is

that the ARPANET algorithm uses actual delay, and the transmissions of delay are unordered, so that many updates are required for changes to propagate through the network, but Gallager's algorithm uses marginal delay of each line, and changes are propagated completely in one update.

Gallager's algorithm, as he claimed, is the first one possessing the property of being loop free at each iteration. After this, Segall, Merlin and Gallager jointly developed, in 1978, a recoverable loopfree distributed routing protocol, which extended Gallager's minimum delay algorithm into one insuring recovery from arbitrary topology changes [80]. At about the same time, Segall published an optimal distributed algorithm [75]. Segall's later extensions are called failsafe distributed algorithms, with improvements and increments made time and again until the latest version mentioned at the beginning of this section [81].

The failsafe algorithm is run for each destination independently, updating the routes from all nodes to that destination. When an update cycle is triggered by a destination node, it will change the routes to that destination according to the new weights of lines. The partial ordering of updates is insured by defining a sink tree for each destination. Each cycle can be viewed as proceeding in two phases. In Phase 1, control messages propagate upstream from destination to the leaves of the current tree, while updating the line weights. In Phase 2, control messages propagate downstream to the destination, each node selecting its "preferred neighbor" (downstream node), thereby updating the tree. The path through the preferred neighbor to the destination provides the minimum distance.

In the latest version [81], the failsafe protocol is applied to minimum delay routing, i.e. it uses marginal delay as line weight. Multiple paths, instead of single path, are used. Each node may have a number of "sons",

rather than only one preferred neighbor, for routing traffic to a destination. In Phase 2 of each cycle, routing table at each node is updated as increasing traffic flow to its "preferred son" and decreasing traffic flows to other sons.

Update cycles corresponding to a given destination are nondecreasingly numbered. During normal operation, a cycle started will be properly completed within finite time, and the destination can start a new cycle with the same number as the previous cycle. When a failure or a recovery of lines or nodes happens, however, the destination will have to be informed not to wait for the completion of the current cycle and to immediately start a cycle with a higher number in order to propagate the news throughout the network. The cycle number is carried by the control messages belonging to that cycle. Each node, say i, keeps track of the highest cycle number it has known. This number is denoted by mx_i. Except for messages indicating failures, all control messages with cycle numbers strictly lower than mx_i are discarded. Node i participates in Phase 1 of a cycle after receiving control messages with cycle number mx_i from all its current sons. It goes from an "idle state" to a "waiting state" after updating its incremental delay coefficient and its blocking status and sending the results to all its neighbors except its sons. The node will stay in waiting state until it receives control messages with cycle number equal to mx_i from all its current neighbors. At this time, it performs its part of Phase 2 by sending control messages to all sons, updating routes, and going back to idle state in order to wait for the next cycle.

If the failure is on a line carrying traffic flow, the node immediately upstream from the failure has to redistribute the traffic flow among its remaining sons, if any, without waiting for control messages on this line. The redistribution is arbitrary, since later cycles will

improve the routing until a new optimum is reached. If there are no other sons to which the traffic can be redistributed, the node, and then possibly other nodes upstream, have to consider that they have lost all their current paths to the destination.

The failsafe quality is guaranteed by the special request message REQ generated by nodes adjacent to the topological change. The REQ carries the number of the last cycle handled by this node and is forwarded towards the destination. Whenever a failure or recovery occurs, the destination will be notified so as to be able to start a new cycle to cope with the situation. The REQ is forwarded by the node to its preferred son if it has one. If the line to its preferred son has previously failed and the node has lines to other sons, it sends the REQ to one of the other sons. If the node has no sons (because of previous failures), it discards REQ. Since the failure that causes the discarding of a REQ will induce generation of another REQ, it is guaranteed that at least one of all the REQs carrying a given cycle number will indeed arrive at the destination.

When a failure is detected on an adjacent line, the corresponding node is deleted from the list of neighbors, and from the list of sons, if appropriate. Each node that has lost one of its sons stops the flow to that son, redistributes it among its remaining sons, if it still has any, and modifies its routing variables correspondingly. The redistribution is arbitrary, since later cycles will improve the routing until the new optimum is reached. The nodes at the ends of a line that is ready to be added to the network due to recovery or initialization have to coordinate their operations for bringing the line up. The coordination is achieved by having both nodes bring the line up as soon as they start to perform their part of the same new cycle.

Details of this algorithm described as a finite state

machine can be found in [81].

As Segall et al claimed, this latest version extends and improves the previous ones in the sense that it adapts to both slow load changes and arbitrary topology changes, and the adaptability to the new load pattern is smooth for nodes that are not affected by topology changes.

The failsafe minimum delay algorithms are presented in separate papers for the cases of circuit switching networks and packet (or message) switching networks. The former case is dealt with in [81], the latter in [82]. The difference lies in that in circuit switching networks, the quantities to be controlled are the total flows between source-destination pairs, while in packet (or message) switching networks, they are fractions of the flows corresponding to packets (or messages) from their sources to destinations.

D. OTHER ALGORITHMS FOR QUASISTATIC ROUTING

In addition to the ones discussed in the last section, some other quasistatic routing algorithms are to be reviewed here.

In 1979, Bertsekas et al first generalized Gallager's algorithm into a nonlinear multicommodity network flow problem [3] and conducted an extensive numerical study of five distributed routing algorithms of this type and their properties [8]. One year later, he published a new optimal algorithm of this type [5]. In his algorithm, each node maintains a list of paths along which it sends traffic to each destination together with a list of fractions of total traffic that are sent along these paths. At each iteration, a minimum marginal delay path to each destination is computed and added to the current list, if it is not already there. The corresponding fractions are thus updated in a way that reduces average delay per message.

The algorithm is similar to Gallager's method and its generalization in that it relates to the gradient projection method for nonlinear programming. The new points, however, are that it operates in the space of path flows rather than in the space of line flows, and therefore is also well suited for virtual circuit networks, and that it utilizes a shortest path computation to obtain a search direction rather than an upstream summation of line marginal delays, hence the smaller amount of computation per iteration.

It is possible to distribute the computation involved in each iteration among the nodes of the network, resembling the new ARPANET algorithm in that information providing length for each line is propagated throughout the network, each node computes shortest path from itself to its destination on the basis of these lengths, and shifts flow to the shortest path. While ARPANET type algorithms

cannot provide optimal routing because of their inability to send data along more than one path for any one source-destination pair, Bertsekas' algorithm retains a portion of flow in previous shortest path, resulting in asymptotic convergence of flow pattern into optimal without oscillations.

Bertsekas et al also found it possible to employ second derivatives of line delay functions within the context of this method, thereby providing automatic stepsize scaling with respect to traffic input level. In 1984, they jointly published a paper [7] elaborating on such second derivative algorithms. The advantages of employing second derivatives are of crucial importance for the practical implementation of the algorithms using distributed computation in a quasistatic environment.

Another algorithm designed by Chen and Meditch [13] is also related to the theoretical work of Gallager [28]. This is a distributed adaptive algorithm, comprising two separate but coordinating processes, termed NUP (Normal Updating Process) and DAP (Disturbance Adaptive Process), respectively.

The NUP is an iterative process that updates the flow for one destination at a cycle, providing minimum average delay given an initial loop-free routing assignment. For the flow to each destination, say j, NUP starts at node j and is followed successively by its upstream nodes with respect to destination j. After each node has received the information about delay and flow computed from all of its immediately downstream nodes, it does its own computation and propagates the result to all of its adjacent nodes. When all nodes involved in the j-destination flow have completed the update, a cycle is completed. The cycles are initialized by the destination nodes, either on some prespecified timing basis, or whenever a destination node determines it necessary on account of its average delay estimates of its traffic flow. The computational process

is essentially the same as Algorithm 1 in [8] except for the formula used in calculating the line delay. Details of the derivation of these formulas appear in the appendix of [13].

The second part, DAP, is activated when network disturbances (changes in traffic load or topology) occur. It generates a new loop free routing assignment subject to the new constraints arising from the disturbances. Upon achieving the new assignment, control is tranferred back to NUP. For the coming-up of lines or nodes, some protocols are used to inform all the relevant nodes of such changes so as to make new assignments respectively. For the case of line failure, the affected node first tries to find an alternative outgoing line to accommodate the flow originally routed over the failed line. If that is possible, the newly adjusted assignment is established, and control reverts from DAP to NUP to reduce the delay as much as possible. If no alternative lines can be found, the information of failure is propagated to all of its immediately upstream nodes and they try in the same way as the previous node. In this manner the trial is made in the upstream direction until some node finds some alternative lines, then the flow on the failed line is turned back onto the newly chosen alternative route. As long as there is at least one upstream node having alternative lines, DAP will succeed.

Now we turn to algorithms using another type of distance computation -- minimum hop algorithms. With this method, the distance between any pair of adjacent nodes is one hop. The weight (or length) of a path is evaluated as the number of hops between the source-destination pair.

Minimum hop computation finds its important application in reachability detection. When the hop distance to a destination exceeds (N-1), where N is the number of nodes in the network, that destination is unreachable, since no path without loops in an N-node

network can be longer than (N-1) hops. The ARPANET algorithm is an example of such an application.

In addition, minimum hop computation is also used in route selection. The effect of minimizing the number of hops that messages make in proceeding from source to destination is to minimize the number of times that a given message must undergo nodal processing, which involves buffering, error detection, line control, acknowledgement and routing decisions. It is particularly useful in those environments, where nodes are very vulnerable (as in the case of some military applications). As an alternative to minimum delay computation, this is conceptually simple and computationally efficient.

In 1980 and 1981, Meditch and Gorecki developed a theory and procedures for constrained minimum hop routing in message switching networks, particularly the centralized minimum hop routing algorithm in which one or more end-to-end average delays serve as a constraint set [56], [37]. Another distributed algorithm which achieves the same result was presented by the same authors in 1981 [57].

The use of a set of end-to-end average delay constraints will serve to meet user requirements for timely delivery of messages, particularly important where critical source-destination pairs are involved.

This distributed algorithm is composed of two parts, the first part providing unconstrained minimum hop routing and the second adjusting this routing to satisfy the end-to-end delay constraints.

The first part first determines the lengths (in hop count) of all source-destination paths and assigns routing variables to them. Then it calculates all the line flows of the network to minimize the average path length subject to the capacity constraints and the conservation of flow, and uses the line flows to calculate the routing variables assigned to all the paths. These routing variables now indicate the fractions of flow for the paths of each

source-destination pair.

The second part iteratively calculates the path delays, compares them with the end-to-end delay constraints and recalculates the lengths and flows for those paths violating the constraints, until all are satisfied.

Both parts are implemented distributively by each node, requiring information only from adjacent nodes. Operations can be carried out either synchronously or asynchronously. The present algorithm minimizes the average path length with respect to a set of paths. Investigations are under way of algorithms that minimize the average path length over the entire network. Further efforts are also made by Meditch and Gorecki to develop such algorithms by incorporating the two parts into one.

E. JAFFE ET AL: RESPONSIVE ALGORITHM

The idea of using sink trees to define the partial ordering of routing updates among nodes is, as discussed in Chu's algorithm and Segall et al's failsafe algorithms, a significant step towards resolving the adaptivity problem. This has been used by Jaffe and Moss as a criterion to classify the distributed algorithms into two generations [39]. As they noted, the old ARPANET algorithm and the MERIT algorithm (to be mentioned later) belongs to the first generation, where no control of update ordering is exercised, and adaptation to line/node failure is slow. On the other hand, the second generation ones, such as the failsafe algorithms, can deal better with those problems by using sink trees.

To add to the second generation ones, Jaffe and Moss developed a responsive distributed algorithm, as they named it, in 1982. While it is similar in many respects to failsafe algorithms, its major contribution is that the control of update ordering is only exercised over the cases where line weights increase, rather than over all kinds of line weight changes. Moreover, coordination in those instances need only occur among a subset of the nodes, instead of the whole tree. Also after a failure, coordination is only needed briefly, not for all subsequent updates. As they observed, this can result in improvement in computational complexity for failure recovery, so that the algorithm can be very responsive, ideal for situations where changes in line weights are relatively infrequent and yet fast recovery is needed upon changes.

The above design philosophy is based on a fact, pointed out by McQuillan [51] and Stern [84], that the first generation algorithms maintain loop free paths in the presence of static or decreasing line weights. In [39], Jaffe and Moss presented their algorithm in two parts. The first part, named IUP (Independent Update Procedure) is one

common to first generation algorithms. As they proved, IUP is capable of maintaining loop free paths in the presence of nondecreasing line weights.

Each node maintains a routing table. Figure 10 is the routing table at a node, say A, with K adjacent nodes B_1, B_2, ..., B_K. Entry $C(A,DES,B_i)$ is the estimated minimum weight from A to DES via B_i. $HOP(A,DES,B_i)$ is the hop number of that path. NN(A,DES) is the adjacent node on the path that provides minimum estimated weight. $C^*(A,DES) = C(A,DES,NN(A,DES))$. d(A,B) is the weight of the line from A to B.

Initially, each $C(A,DES,B_i)$ is set to infinity, and each $HOP(A,DES,B_i)$to zero, except for B_i=DES, in which case, $C(A,DES,B_i) = d(A,B_i)$ and $HOP(A,DES,B_i) = 1$.

When $C^*(A,DES)$ changes, node A sends an update message MSG(DES,C,h) to all its neighbors, where $C = C^*(A,DES)$. Upon receiving such a message, a node, say B, updates its table by setting C(B,DES,A) = C + d(B,A) and HOP(B,DES,A) = h + 1. Any message of the form MSG(DES,C,N-1), where N is the number of nodes in the network, is ignored. The other items in the table are accordingly reevaluated before node B, in turn, sends the MSGs to its neighbors. If line weights change, the table at each node is also updated and the update messages sent to neighbors.

While the above mentioned IUP deals with cases of nonincreasingly changing line weights, the second part, CUP (Coordinated Update Procedure), will take care of cases of increased line weights.

A sink tree is defined. When a weight increase occurs on a line, all nodes upstream of this line are progressively "frozen" starting at the node adjacent to the line and proceeding upstream. The "freeze state" for node A is with respect to a particular destination, DES, and means that A may update its weight entries to DES but may not change NN(A,DES). Node A is not "unfrozen" until all upstream nodes have increased their costs and sent back

DES	NN(A,DES)	$C^{*}(A,DES)$	$C(A,DES,B_1)$	$C(A,DES,B_2)$	...	$C(A,DES,B_K)$
⋮	⋮	⋮	⋮	⋮		⋮
D	B_2	2	5	2		17
⋮	⋮	⋮	⋮	⋮		⋮

Figure 10. Routing table at node A

their acknowledgements. A single bit added to the update message can indicate whether or not the line weight increases. In this fashion, node A never causes a loop to form by choosing an upstream node, because the only time an upstream node may have lower cost is when the downstream node is in freeze state. This is in distinction to IUP, where an upstream node may have lower cost due to the fact that the news of the increase has yet to propagate upstream.

In order to discuss the speed of recovery, Jaffe and Moss assumed a hypothetical synchronization of the algorithm, so that every node executes a "step" of the algorithm simultaneously at fixed points in time. At each step a node may receive and process one message from each neighbor. The question of "how fast?" is then equivalent to "how many steps?"

Their analysis showed that the algorithm has worst case speed of recovery of O(X) where X is the number of nodes affected by the failure. This is favorable in comparison with the first generation algorithms and with the failsafe algorithms. In terms of the same assumption, the number of steps required for the first generation algorithms to recover is O(N) where N is the number of nodes in the network, and the failsafe algorithms take $O(h^2)$ steps where h is the height of the shortest path tree at the start of a cycle.

F. CHIN ET AL: PPD ALGORITHM

According to Davis and Barber [21], most existing distributed routing algorithms are "branch-directed", which means that the routing decision of a packet is determined from node to node, i.e. each node selects an outgoing line to be the next branch to route a packet toward its destination. Another method, "path-directed", on the other hand, predetermines the entire path of each packet at its source node. Usually, path-directed routing is applied in centralized-control networks [73]. To simplify packet routing in distributed-control networks, an algorithm using path-directed method was proposed by Chin and Hwang in 1983 [14].

This algorithm is named as PPD (Probabilistic Path-Directed). Probabilistic indicates that for routing a packet, one out of multiple paths is chosen, instead of a single path. Each path is an entry in the routing table associated with the source node. Paths to the same destination are grouped into a subtable. The use of each path is periodically checked and recorded in the subtables. A source node distributes packets among selected paths to achieve balanced and nearly minimum-delay performance. To allow immediate routing at intermediate nodes, each packet being transmitted is tagged with a particular "path code".

The key parameter used in the computation of this algorithm is the "effective capacity", which is defined together with a set of other terms and notations by the authors as follows.

The packet generation rate r_g is measured on the packets that are generated (enters the network) at node i (source), destined for node j, and routed via path g. Both the generation rate $r_{i,k}$ and passing rate $s_{i,k}$ are measured on {i,k}, which is the set of all possible paths from node i to node k. $r_{(i,k)}$ and $s_{(i,k)}$ are subterms of $r_{i,k}$ and $s_{i,k}$, respectively, if there exists a line (i,k). ϕ_g =

$r_g/r_{i,k}$ is the assignment probability of path g = (i,j,...,k) in set {i,k}, $\phi_g \geq 0$ and $\sum_{g\in\{i,k\}} \phi_g = 1$. Each packet, generated at any node, is preassigned with a path by examining the ϕ_g's of all paths in {i,k}. The line capacity $C_{(i,k)}$ is an undirected quantity, i.e. $C_{(i,k)} = C_{(k,i)}$. Assume the exponential packet length with an average 1/L bits and the Poisson distribution of packet generation. If the traffic is light, the packet arrival rate at each node will not be affected by those at other nodes, thus the arrival rate at each node can also assume a Poisson distribution. The average line delay per packet transmitted from node i along line (i,k) is denoted by D(i,k) in sec/packet, and D_g is the average path delay along path g = (i,j,...,k). Since each line (i,k) can be considered as an M/M/1 queueing model, the effective line capacity of (i,k) at node i can be defined as

$$E_{(i,k)} = L \cdot C_{(i,k)} - s_{(i,k)} - r_{(k,i)} - s_{(k,i)}.$$

In other words, the effective line capacity is the service rate of that line dealing with $r_{(i,k)}$, and E_g is the effective path capacity of path g = (i,j,...,k) at node i. The line (i,k) is considered as an M/M/1 queueing model, and the path queue is also approximated as an M/M/1 model. Thus, the average line delay per packet

$$D_{(i,k)} = 1/(E_{(i,k)} - r_{(i,k)})$$

and the average path delay per packet

$$D_g = 1/E_g - r_g.$$

From the latter equation, the effective path capacity is calculated with D_g measured by each packet routed via path g and sent back with the acknowledgement packet.

The PPD algorithm is described in the following two

parts.

The first part is its routing scheme. To expedite the routing process, they use trunk numbers to encode the paths. The outgoing lines of a node are called trunks of that node. All the trunks of a node are numbered as 1, 2, 3, ... The encoding scheme finds every outgoing trunk number of the desired path in the order from its source to destination and concatenates these numbers from right to left.

Each node maintains a routing table, which contains a number of subtables, one for each of the other nodes in the network as the possible destination. All possible paths to the same destination have entries in the same subtable. Recorded in the subtable are the path code, the packet generation rate, the path delay, the path capacity and the assignment probability for each path.

After a packet destined for node k has been generated at node i, the source node probabilistically assigns a path code according to the assignment probabilities in the corresponding subtable. The probabilistic distribution can be implemented by either software or hardware mechanisms. Once a path code is assigned, the packet carrying its path code can be routed through the network by a simple algorithm at each of the intermediate nodes. The node simply checks the path code. If the path code is zero, then the destination is reached. Otherwise, it updates the path code by right-shifting out one trunk number, and transmits the packet through the outgoing line having the shifted out trunk number.

Upon receiving a packet, the destination node sends back an acknowledgement to the source node through the same path in the reverse direction. The acknowledgement enjoys the highest priority to pass through the network, so that the source node can quickly receive it and record the path delay information.

The second part of PPD algorithm is the routing table

update policies. The routing table is updated at each node locally, and different subtables do not have to be updated at the same time.

For a source node i to update its subtable for destination node k, it first calculates every effective path capacity E_g for all g in {i,k}, and finds out their maximum E_{max}. It then calculates $F_{i,k}$, the total of the effective path capacities for all paths that are in the set S, which includes every path g that E_g/E_{max} exceeds a threshold. Finally, the assignment probability is set as $E_g/F_{i,k}$ for all paths in S, and as zero for all paths not in S.

The update period for the subtable is adjusted by $r_{i,k}$, the packet generation rate for this source-destination pair. The greater the $r_{i,k}$, the shorter the interval. Since the subtable is updated individually, among the newly generated packets, only those destined for the node corresponding to the subtable being updated will be blocked for a short time.

In [21], they gave some analysis, which shows the saving of search time at each intermediate node by using this path-directed method. The worst time complexity for routing a packet is O[s+(n-1)c], while for branch-directed methods it is O[(n-1)(s+c)], where s is the worst routing table search time, and c is the execution time for the routing algorithm. If binary search is used, this time complexity for PPD method can be improved to O[log n + (n-1)c].

Simulation results of the PPD algorithm were also given in [21]. They show a favorable comparison with the new ARPANET algorithm with respect to delay performance. The ARPANET algorithm uses global line delay information to find the shortest path trees. The PPD algorithm uses the path capacities to determine the assignment probability of each path. The ARPANET algorithm has higher average delay than the PPD algorithm. Under heavy traffic condition, PPD

algorithm can handle the traffic better than ARPANET algorithm.

According to Chin and Hwang, the superiority of the PPD algorithm over the new ARPANET algorithm is due to the probabilistic nature in routing a packet. Inspired by the result, they proposed, in the same paper, a PPD-generalization of the new ARPANET algorithm. This generalized algorithm uses the same delay measurement and update method as the ARPANET algorithm, i.e. periodically updating all routing tables at the same time based on the same delay information. Instead of a single shortest path, m shortest paths from one node to any other node are selected during the table update process. The packet destined for the same node are probabilistically distributed among the m paths, based on the path assignment probabilities. The entire routing path of a packet is determined by its source node. The assignment probabilities are determined in proportion to the inverse of the corresponding path delays. Hopefully, the proposed method will be capable of balancing the load among multiple paths and reducing congestion in heavy traffic, thus giving better delay performance than the ARPANET algorithm.

Yet, everything has its pros and cons. Like other multiple path routing algorithms, it could suffer from increased complexity for keeping information. The multiple paths may also affect stability. And for making routing decisions, the path-directed approach will take more time to collect information about the whole network, thus it may be less responsive than branch-directed approaches. If the paths are very long, it will be very probable that topological change will occur while packets are in transit. Then, there could be more problems in rerouting these packets. All these defects may not be compensated for by the delay performance improvement. Therefore, whether or not the generalized algorithm is feasible is a question still open to discussion.

G. RUDIN: DELTA ROUTING AND OTHER SIMILAR ALGORITHMS

While centralized and decentralized methods have their respective advantages, they both have their drawbacks. In 1976, Rudin described an interesting hybrid algorithm, Delta routing, combining the strengths of the centralized and distributed classes of algorithms [67]. The centralized portion of it can keep track of the global state of the network in a relatively lethargic way based on average values of past performance and use this information to ensure all overall, consonant routing strategy for the entire network. Within this overall strategy established by the centralized NRC (Network Routing Center), further decisions could be delegated to the individual nodes which could react instantaneously and in a distributed manner, responding even to the absence and presence of single packets on the lines to which they are attached. The past global information and instantaneous local information could thus be used to best advantage.

This algorithm was named after the parameter Delta, which regulates the relative amount of decision making authority the NRC delegates to the nodes. Using the information sent to it from nodes, the NRC computes the K best paths from node i to node j, for all i and all j (only the paths that differ in their initial lines are considered). Let C^1_{ij} be the total cost of the best i-j path. If $C^n_{ij} - C^1_{ij} < \delta$, path n is considered as equivalent to path 1. Upon finishing the computation, the NRC sends each node a list of all the equivalent paths for each of its possible destinations. The node is thus free to choose any of the equivalent paths to do actual routing, basing its decision on various methods such as at random or use the current measured value of the line costs, etc.

By adjusting K and δ , the authority can be transferred between the NRC and the nodes. As δ approaches to zero, the NRC makes all the decisions, since all other paths are

deemed inferior to the best path. As δ approaches infinity, however, all the paths will be considered equivalent, and the decisions are made by the node based on local information only.

By simulations, Rudin showed that δ could be adjusted to provide better performance than either pure centralized routing or pure decentralized routing. Hence the name "ultra dynamic" or "super adaptive" routing. As for whether or not the improvement justifies the complexity, the choice of routing strategy may depend on the cost efficiency, delay or availability of lines. Anyway, it was a thought provoking idea. The French public packet switching network, Transpac, uses Delta routing [20], [73]. A similar idea has been applied to some other routing algorithms designed later. We next describe one of them, JBQ-BS routing by Yum and Schwartz [93].

Before talking about the JBQ-BS routing, the concepts of JBQ rule and BS rule should be introduced. According to Yum and Schwartz [90], [92], [93], the routing rules can be classified as fixed and adaptive. The simple SP (Shortest Path) rule is a fixed one. A more sophisticated one is the BS (Best Stochastic) rule which allocates traffic flows stochastically (i.e. by fixed probability assignment) through the network so as to minimize the overall average delay. Better overall delay performance can be obtained by bifurcating the flow adaptively. One way to do this is the JBQ (Join-Biased-Queue) rule, in which a biased term is used in comparing the queue lengths. By adjusting the biased term, the proportions of traffic bifurcation can be regulated at will. The difference between the BS rule and the JBQ rule lies in their message arrival processes. For the BS rule, the message arrival process of each queue remains Poisson distributed because random bifurcation of Poisson processes remains Poisson. For the JBQ rule, on the other hand, the message arrivals are state dependent because traffic bifurcation is based on the instantaneous

queue lengths, so the queue length distribution is not analytically known.

The essence of JBQ-BS routing is to superimpose local JBQ adaptivity on the fixed BS rule base. As the centralized portion of Delta routing, the BS rule determines the traffic flow on each line based on the global traffic input rate information. Like the local portion of Delta routing, the JBQ rule, with its inherent bifurcation ability, determines the instantaneous traffic flow in the local environments.

Details of the JBQ-BS routing can be found in [93]. Also discussed there are three problems that remain unsolved in the analysis of JBQ-BS rule.

The concept of bifurcation of traffic flow mentioned above deserves a few words of comment here. Interestingly, single paths turn out not to be the optimum if the long-term average delay of the whole network is to be minimized. On this account, arised the bifurcation -- packets at a node are assigned to one of several outgoing lines on a probabilistic basis [27]. A weighting system is used to determine, on topological grounds, the proportion of traffic to use the respective routes. Using a random number generator a node can distribute its traffic according to the ratio of the weights. Price gave a good discussion on bifurcation [64]. The factors to be taken into account are the length of queue for each outgoing line as well as the topology. Furthermore, as Price maintained, it is possible to increase the amount of information available by making the routing decision depend not only on outgoing queue lengths, but also on the number of packets already transmitted but as yet unacknowledged. He gave an account of the experimental work using simulation to investigate the performance of such bifurcated routing algorithms. Definite benefit was detected in the case where a very heavy stream of traffic needs to pass between a particular source and destination, while the rest of the

network carries a moderately heavy general load.

In his analysis, Price noted that the ability to split loads is inherent in many of the routing strategies used in practice today, but in many cases, load splitting, though theoretically possible, does not in fact take place to any useful degree. He also found from simulation results that successful bifurcation can be carried out using only local information.

Before finishing this section, another adaptive routing technique proposed by Boorstyn and Livne [9] is to be reviewed. It is a two-level scheme. In some sense, it bears strong resemblance to Delta routing.

At each node, a subset of the outgoing lines is specified as allowable for each message with a certain destination, and the message may use any allowable lines according to some discipline. Each message appearing at the node has its own allowable set of lines. The assignment of allowable lines at each node for each message is one level of the routing scheme. These assignments are based essentially on global information of topology, traffic flows and long-term average delays, and may be adaptive in a quasistatic way, responding to average statistics of congestion and traffic and alarms due to line failures, onset of congestion, new traffic, etc. Some mechanism is assumed to exist for making adjustments, and that these will be made relatively infrequent compared to the rate of second level adaptivity.

The second level, on the other hand, is truely dynamic and local, involving queue disciplines at each node. It is the task for the second level to choose among the set of allowable paths of the same or similar quality. At this level, several strategies for the multiple server queueing system were suggested. The more alternative paths, the better the second level may contribute to the average delay performance.

Some analytic approximations to estimate the

performance improvement of this technigue over nonadaptive routing were given that, in heavy traffic, it could improve almost by a factor on the order of k, where k is the number of outgoing lines a node has, and for moderate traffic, good improvement could still be achieved.

Described in this section are three routing algorithms. All of them share a common essence, i.e. their local adaptability coupled with a globally quasistatic scheme considerably improve the delay performance. Of the two-level hierarchy, the lower level is the locally adaptive decision making as to which outgoing line to select for waiting packets with alternate routing options. The optimal local policies were analyzed and compared with a newly proposed one by Marglaris [46], which can hopefully improve the delay performance for such two-level routing.

As is seen in the last section, for networks of large size, the overhead for distributed adaptive routing can become quite excessive. This is due to the fact that the memory and updating cost of such routing procedures increase with the number of nodes, since the size of the routing table to be maintained at each node becomes very large. Furthermore, the computation of routing updates needs to be done at each node and the required exchange of status information conducted on an adjacent node basis might take considerable time to reach certain nodes. One way to mitigate this problem is to reduce the imformation costs by requiring the updates to be computed with only a subset of the global network information at the price of a degradation of overall performance. This trade-off between information requirements and routing efficiency can be used to design hierarchical structures for routing.

The basic idea of hierarchical routing is to partition the nodes into clusters, with each node knowing all the details about how to route packets to destinations within its own cluster, but knowing nothing about the internal structure of other clusters. When different networks are connected together, it is natural to regard each one as a separate cluster in order to free the nodes in one network from having to know the topological structure of the other ones. For huge networks, more than two levels of hierarchy may be needed. For example, the clusters may be grouped into regions, the regions into zones, and so on. That is multilevel hierarchy.

An early attempt at the design of hierarchical routing schemes is due to Kleinrock and Kamoun [42], [45]. They employed a hierarchical clustering of nodes to reduce the length of the routing table. The basic idea used is that the node maintains a detailed routing information for these nodes close to it and coarse aggregated information for

those nodes located farther. The network nodes are partitioned into m levels, where any level, say k-th level, is defined in terms of the clusters at the (k-1)th level. This scheme results in a reduction of the cost of nodal storage and processing capacity. As they found, the optimal number of levels for an N-node network is ln(N), requiring a total of e*ln(N) table entries per node (e is the base of natural logarithm). Also discovered was that the increase in effective message path length caused by hierarchical routing is fairly small and that it is tolerable in most cases.

The above scheme, in some cases, still suffers from the increase in the message path length. In attempt to overcome this limitation, Muralidhar and Sundareshan propostd a different approach recently [60]. In this scheme, a part of the overall decision-making is done at the lower level of network nodes where nominal routing tables, which provide satisfactory routing under nominal load and network conditions, are established, and another part at the higher level of "supervisors" (or "coordinators") who provide the control of updates to account for variations in traffic load and topology. Specific optimization problems are formulated. Solutions to them at different hierarchical levels comprise the overall control scheme.

As they noted, one of the major merits of this scheme is that it permits consideration of multiple objective functions (throughput, delay, hop count, etc.) in performance optimization, and that it provides a mechanism for integrating routing and flow control functions for efficient control of traffic congestion. The traditional development of routing schemes within an optimization framework is with respect to a single performance objective, with a few exceptions such as those techniques that use the "generalized power" as a performance measure which attempts to provide a compromise between maximizing

the throughput and minimizing the delay [36], [44]. Studies of flow control and routing are traditionally conducted independent of each other. Not until 1979, had the interrelations existing between the two been identified [53], [29]. Methods for designing efficient control algorithms that take into consideration the coupling of routing and flow control are being investigated only currently.

For the lower level decision-making, any kind of optimal routing algorithms available in the literature such as the Dijkstra algorithm [22], the flow deviation method [27], etc. can be used, since this computation is only done once. They can be selected based on specific performance criterion to be optimized at this level.

The two modes of action for the supervisor to provide the required updates are identified as "periodic mode" and "interrupt mode". In the periodic mode of operation, the supervisor for each cluster attempts to solve the higher level problem to improve the network throughput and utilization at periodic intervals of time. From the global congestion measure for the cluster, the supervisor is able to deviate the line flows to permit the routing of any increased traffic load at a source node within its cluster. If the destination is also in that cluster, this can be done simply by a depth-first search, which identifies all the paths between that source-destination pair and determines the "capacity slackness" in them (The capacity slackness of a line is the difference between the line capacity and the sum of the average flows on that line towards various destination). If the destination is in a different cluster, the congestion measure in that cluster as well as in the intermediate clusters through which this traffic needs to pass must be broadcast to each supervisor periodically.

The interrupt mode of operation of the supervisor is similar to that of periodic mode, except for two

differences. The first point is that the supervisor action is initiated by an interrupt from a node when the traffic load at that node increases considerably above the nominal value. Secondly, instead of broadcasting the congestion tables, the interrupted supervisor identifies the paths from the source to the destination and requests the updated congestion tables from the supervisors of clusters through which these paths pass.

Unlike some other hierarchical schemes, this scheme requires the supervisors to participate only in making routing decisions by computation of updates while not necessarily getting involved in the actual routing of data messages, thus avoiding the chances of routing the messages on possibly longer paths via supervisors.

I. BRAYER: SURVIVABLE ALGORITHM

In 1982 Brayer proposed a survivable routing algorithm with autonomous decentralized control [10], [11]. This routing strategy was based on the mathematical algorithm for finding shortest paths between node pairs due to Chyung and Reddy [19] and its implementation [12]. As he introduced, the algorithm is characterized by the property that it permits nodal computers to autonomously create a network and then continue to adapt to changes in network topology, i.e. changes in the interconnections between nodes and changes in the sign-on of addressees* of various nodes. No routing center is used to centrally control the network. No overhead traffic for nodes to exchange routing table is required, either. Instead, a small amount of information about the path is appended to each packet as it is going through that path in the network. This information is what the nodes use to continually recompute the nature, shape and topology of the network and the location of addressees.

Brayer designed the algorithm as containing two major parts, addressee finding and packet routing.

When a node is to send a packet, it must first know to which node the target addressee is signed on. The addressee finding part serves this purpose. Before a packet is actually transmitted, a separate "header" is generated by the source node, and sent to any one of its adjacent nodes. If the receiving node does not have the addressee, it appends its own identification to the header and sends the header to another node. Headers are sent from node to node in this fashion until the addressee is found. As the node having the addressee receives the

---

* A user on a terminal signed on to a nodal computer is the addressee of a packet if the packet is meant to be destined to that user.

header, an end-to-end acknowledgement is sent back to the source node through the path on which the header was sent, thus every node on the path can update its own addressee table. Upon receiving the acknowledgement to the header, the source node proceeds to forward the packet. If no acknowledgement is received and the addressee is not located after a specified number of retransmissions of header upon time-out, the source node will stop looking for the addressee.

After the network has run for a period of time, most nodes will have built up their full addressee tables, and headers will appear occasionally only when new addressees sign on. In the event that an addressee changes from one node to another, the latter will generate an "update" and send it to the former. Again, all the nodes on the way the update is passing can update their addressee tables.

The second part, packet routing, allows the packets to be forwarded in two fashions: one is via the routing algorithms, the other random. A routing table is maintained by each node, containing shortest paths. If paths can be found in the routing table, the packet is sent to the next node on such path, and the next node repeats the same process, and so on. Otherwise, the source node randomly sends the packet to an adjacent node in the hope of finding a path. Node-by-node acknowledgements are given as the packet goes down its path, and end-to-end acknowledgement is given when it reaches the destination node. Time-out is also used for retransmission in case the packet is not acknowledged.

As with the header, when a packet, acknowledgement, or update goes through its path, the identification of each node on the path is appended to it, and the nodes being passed can update their routing tables to reflect the current connectivity. As traffic passes through the network, the nodes learn better and better about the network's connectivity, and the connectivity is defined in

terms of unidirectional paths.

Alternate paths are used, instead of single path, for retransmission. Over time, the nodes keep track of which nodes repeatedly fail to give acknowledgement. Eventually, a node will simply determine that a line has failed, depending on some specified parameters. The alternate path is also applied to random routing mode.

Many other routing algorithms for packet switching networks depend on some form of routing information exchange or a central-control node. Neither of these occur in this algorithm. Therefore, the network does not have to suffer from the vulnerability due to the failure of a central-control node, or the performance degradation of other nodes if one fails to propagate its current routing table. Such adaptive learning without overhead results in the most important characteristic of this algorithm -- survivability, though it does not seek to provide minimum delay or maximum throughput.

With this survivable algorithm, a "cold start" with no prior knowledge can be assumed for the network system. At start up, each node has a set of lines connecting to its adjacent nodes, and transmits a "start-up" message to its neighbors identifying itself. After a few seconds, all nodes know their own neighbors, and get ready to accept traffic. Packets addressed to specific users come into the nodes. If the node knows to which node the target addressee has signed on, it directly goes to execute the packet routing part of the algorithm. Otherwise it first resorts to the addressee finding part. After going through its learning stage, the algorithm can stablized if the connectivity of network and signing on of addressees are not changing continually.

The way the algorithm deals with failure of lines or nodes is using a "node-link-out" message being sent node by node just like the header. As for the coming-up of lines or nodes, no special message needs to be sent, because the

routing algorithm's self-learning mechanism will become aware of this after a little while.

For networks of large scale, there is one problem with appending a semi-infinite path to every packet. The way out is to divide the network into smaller subsets organized with multiple gateways in between. When a message passes through a gateway, the previous subnet's paths are replaced by the previous subnet name. In order to prevent the subsets from being disjoined from the network by gateway failure, they suggested to have topologies such that all nodes of a subset are gateways to another subset.

This algorithm was not tested by simulation. Instead, actual implementation on physical computers helped demonstrate the performance in real world.

Since the algorithm is oriented for survivability, it is best suited for situations such as airborne or spaceborne relay systems and mobile ground systems.

Another adaptive routing algorithm of similar characteristics was proposed by Meketon and Topkis [58]. It also emphasizes recoverability from damage, i.e. it only adapts to topology changes. The major part of this algorithm is a learning mechanism that reorders the routing tables of all nodes in real-time, which guarantees the network to work well even when the network configuration is not fully known. Messages can find their paths to destinations through the learning experience in past routing. Three possible strategies for the learning mechanism were suggested. They are "success-to-top", "failure-to-bottom", and "success-up-one".

J. GERLA ET AL: UNIDIRECTIONAL ALGORITHM

The survivable routing algorithm due to Brayer described in the last section is good for unidirectional networks [11]. In this section, another distributed routing algorithm for unidirectional network due to Gerla et al [34] is introduced.

A unidirectional communication network is one in which some (or all) of the lines are unidirectional (simplex) as opposed to bidirectional (full duplex). In other words, the presence of a channel from node A to node B does not necessarily imply the presence of another channel from node B to node A. A subsequent constraint in distributed routing algorithms is that the routing updates can be transmitted only to downstream nodes. Because of this fact, conventional distributed routing algorithms thus cannot be generally applied to unidirectional network, special routing algorithms must be developed.

This algorithm evaluates the distances of paths in terms of hop count between "two-way connected" node pairs in a unidirectional network. Maintained at each node, say v, is a list of nodes with which v is two-way connected, i.e. node v has both a directed path to and a directed path from which. The knowledge of two-way connectivity here is essential to determining if two-way communication is possible between node pairs in a unidirectional network.

Every node participates in the routing computation and periodically propagates its routing and distance information to its adjacent nodes. Stored at each node is the local topology information, instead of the global one. In this respect, the algorithm is reminiscent of the old ARPANET routing algorithm. The reason this algorithm does not follow the new ARPANET algorithm is that the procedure for keeping and flooding the global information is too complicated and storage consuming, and the entire network topology is vulnerable to intruders.

The algorithm consists of two phases. In the first phase, each node constructs its sink tree, represented by the "PDL" table. In the PDL table, P(i) denotes the ID of "parent", the immediately upstream node in the path from node i, D(i) denotes the path length in hop count from node i, and L(i) denotes the ID of the line from its parent of node i. In Figure 11 is a sample network configuration. The thick lines define the sink tree for node 1, as is denoted in the corresponding PDL table.

Initially, the D(s) is set to positive infinity for all $s \neq i$.

The PDL table are periodically transmitted by each node on each of its outgoing lines. When node i receives the PDL tables from all its immediately upstream nodes, it updates each entry, say for node s, of its PDL table as follows.

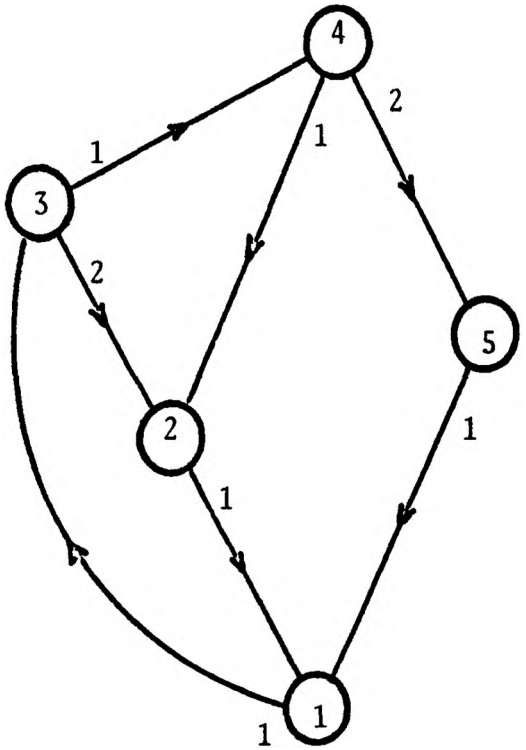
$$D(s) = \min [D_k(s)+1] \text{ for all } k$$

$$L(s) = L_m(s) \text{ where m is the immediately upstream node yielding the minimum distance}$$

$$P(s) = P_k(s) \text{ if } s \neq m$$
$$= i \text{ if } s = m$$

The second phase uses the standard minimum hop routing algorithm [31], in which each node propagates to its immediately upstream nodes its minimum hop estimates to all two-way connected destinations.

Upon receiving the PDL table from node k, node i also proceeds to inspect $D_k(i)$. If $D_k(i) < N$, where N is the total number of nodes in the network, node i concludes that it has a directed path to k as well as one from k. Node i then determines the "shortest cycle" through k and the sequence of lines associated with the cycle by simply tracing the parents through the PDL table received from k.



At Node 1:

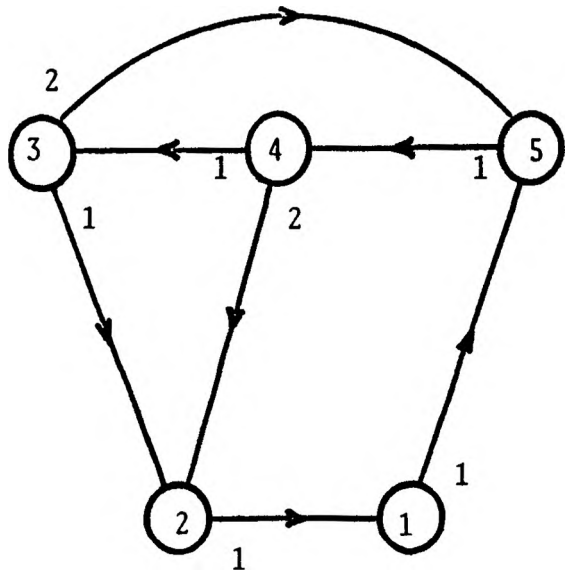
	P	D	L
1	0	0	0
2	1	1	1
3	2	2	2
4	2	2	1
5	1	1	1

Figure 11. PDL table for a sample network configuration

In the example of Figure 12, node 1 just received a PDL table from node 2, it generates the cycle from 1 through 2 (5,4,2) and the line sequence for this cycle (1,1,2,1). Node K is called a two-way connected neighbor of node I. Then the equivalent of the old ARPANET algorithm can be carried out in the unidirectional network. Namely, the routing table and distance table are computed distributively with each node updating its tables using the information received from its immediately upstream nodes and propagating its tables to its immediately downstream nodes. The distance table is sent from each node to its two-way connected destination using the line sequence for the cycle kept in message header to direct the transmission. This is called "path driven" routing. Besides the transmission of distance tables, other information (including data packets) is transmitted by means of "destination driven" routing, as is done in the old ARPANET algorithm.

Their analysis shows that when this unidirectional algorithm is applied to a bidirectional network, it converges in the same number of steps as the bidirectional algorithm, and produces twice the overhead of the latter (the additional overhead being mainly for processing the PDL tables). These results are comparable to that of conventional, bidirectional algorithms. Thus, the unidirectional algorithm can be efficiently applied to networks with a mix of unidirectional and bidirectional channels.

Research on the unidirectional algorithms is still under way. Several extensions to the algorithm described above were suggested by the same authors. One possible way is the incremental table updating as soon as a table is received instead of waiting until all the tables have been received from all upstream neighbors. Also possible is to use more general measurement for line distance changing this minimum hop routing into minimum delay routing.



Cycle from Node 1 through Node 2: (5,4,2)
Line sequence for this cycle: (1,1,2,1)

Node 2 sends to Node 1 the PDL table below:

	P	D	L
1	5	3	1
2	0	0	0
3	2	1	1
4	2	1	2
5	4	2	1

Figure 12. Path tracing to upstream neighbors

K. SNA AND TYMNET ALGORITHMS

Although distributed dynamic routing seems superior to static or centralized ones in many aspects, the latter is not without value. Due to their simplicity in implementation and some historical reason, many of the commerically available networks or network architectures have adopted static or centralized semidynamic routing methods. In this section, two representives among them, i.e. SNA and TYMNET, will be described for the convenience of comparison.

SNA (Systems Network Architecture) is a network architecture intended to allow IBM customers to construct their own private networks, both hosts and subnet [1], [2], [40], [47], [48]. Among the seven layers of SNA, the path control layer provides virtual circuit service to its higher layer, transmission control layer. Not exactly corresponding to ISO's OSI model, the path control layer encompasses some functions of transport layer as well as the network layer in the OSI model. This goal is accomplished by using end-to-end session routing (a route remains in force for an entire user session), with an elaborate system of alternate routes and backup routes. In essence, the network dynamically chooses from among the static routes, which are prepared by the network manager a priori.

Jaffe et al in [40] gave a comprehensive review of the evolution of SNA. As they noted in that paper, SNA's ene-to-end static routing mechanism has evolved from the initial anouncement in 1974 through the present. The routing structure utilizes two physical addresses, called the origin and destination addresses, each containing two parts, the "subarea" (major node) and the "element" (minor node) fields. The address is contained within the "transmission header" preceding the user message, and remains unchanged from the beginning to the end of a

session. Routing was based only on the destination subarea field regardless of the origin. The routing table was organized by destination subarea number and indicated the "next leg of the journey" on the way to the destination subarea. Each of the subarea routing tables was statically created by the system administrator or system programmer via a system generation process on a node-by-node basis. The routing was nonadaptive to topology changes. Topology changes required regeneration of the routing table and reloading of subarea nodes.

This kind of routing remained until 1978 when the capability to establish multiple or alternate paths between two subarea nodes was announced. This support satisfied several requirements such as load distribution, better path selection for better service needs, and circumvention of network component failures. The path between two subarea nodes was called "explicit route", defining an ordered set of nodes and "transmission groups" (A transmission group is a user designated set of parallel lines between two subarea nodes) from one subarea to another. During system generation, eight explicit routes were allowed to be defined between two subarea nodes. The explicit route identifier was added to the routing table and the transmission header to be used in conjunction with the destination subarea number as an index. A virtual route was used to manage a source-destination subarea protocol without being concerned with the explicit route in between. The virtual route number was mapped at activation to an explicit route number. Multiple virtual routes could be mapped to the same explicit route. If all the lines in a transmission group fail, all the explicit routes using that group must be rerouted using another explicit route corresponding to the same virtual route. If none can be found, another virtual route must be chosen. If no virtual route is available, the session must be aborted. The multiple route function reduced but did not eliminate the

problem of network availability. The mathematical algorithm for selecting the optimal route was elaborated on by Gavish et al in [30]. The way the routing tables are generated and three associated problems were addressed in detail by Maruyama in [47].

To satisfy the need for greater network availability, the SNA Network Interconnection technique was announced in 1983. This allows SNA sessions to be established between resources that could span multiple SNA networks. Routing for these internetwork sessions still utilizes the destination subarea and explicit route number, except that they are changed in the gateway nodes as an internetwork session proceeds from network to network. Thus a large number of network interconnections are permitted. They include two networks interconnected at one or multiple gateway nodes, two or more networks interconnected to the same gateway nodes, and cascaded interconnected networks. Each individual network generates its own static routing table. Changes to one network can be masked from changes in other networks.

According to Jaffe et al [40], in addition to the configurations for larger networks, one potential evolution for SNA routing is to provide dynamicity while preserving predictability, controllability and integrity of having sessions assigned to end-to-end routes which do not change during the lifetime of the session. This way would avoid problems often related to dynamic routing, such as message looping, lost messages, and ping-ponging of traffic, while allowing automatic on-line generation of end-to-end routes, overcoming problems of system generation burden and poor network availability often associated with static routing schemes.

The following are the ways conceived by them as feasible to realize the dynamicity.

One possible approach under consideration is the use of a control message, ROUTE-SETUP, which traverses the best

path calculated by an "oracle", allowing each node along the path to make an entry in its routing table to represent the explicit route being established. A reply to the ROUTE-SETUP message is sent back by the destination along the reverse of the path. When the reply reaches the source, the explicit route becomes active. After the virtual route is also established, message flow can begin on the new session between the source-destination pair.

As for the placement, form and function of the oracle, many alternatives are feasible. One way is centralized, like that of TYMNET (to be described shortly). A data base of global topological information is maintained and continually updated by the centralized oracle. The oracle calculates for a source-destination pair the path of minimum cost. This information, along with an explicit route number, will then be given to the source node and inserted into its ROUTE-SETUP message.

The oracle can also be distributed. Again, several forms are possible. One is similar to that of the new ARPANET in that each node keeps the identical global topology data base. With this data base, the source node calculates the best path by itself before the route setup. (Note that in ARPANET, the oracles are used not only to calculate the best paths, but also to route the data packets directly without route setup.)

Another possibility is to have distributed oracles with local information, i.e. for each destination the next transmission group to be taken on the best path along with the cost of the path. The ROUTE-SETUP traverses node by node as each of them consults its local oracle to find the appropiate transmission group and forwards the message on the path. Algorithms for this type of oracles are represented by the responsive algorithm due to Jaffe and Moss described in section F.

Tanenbaum had an interesting comment on the evolution processes of ARPANET and SNA [86]. The two major networks

originally with radically different routing algorithms, have moved closer in the course of time. The original ARPANET algorithm was completely dynamic, but later revised to base the routing on explicit knowledge of topology (see Section A). The original SNA algorithm, on the other hand, was completely static, but has been moving in the direction toward more dynamicity. This somehow indicates that "good routing algorithms should be dynamic and based on knowledge of global topology."

TYMNET is a commercial value added network. It has been in operation since 1971. Like SNA, it is also a session-based network, but it does differ from SNA in that it uses dynamic routing [65], [73], [87].

In TYMNET, all complexity that could be centralized, such as routing control, was put into a supervisor program, which maintained an image of the internal routing tables of all the nodes and explicitly read and wrote the tables in the nodes. This was the original version, TYMNET I. As design considerations changed over time, TYMNET II came into use, gradually displacing TYMNET I, first in high-density areas and new installations. In TYMNET II, the tables are maintained by the nodes, and there is much less interaction between the node and supervisor.

The virtual circuit in TYMNET is defined as full duplex data path between two nodes in the network. All routing is done by the "supervisor". When a user requests building of a virtual circuit, the supervisor hashes the user name into the "master user directory" to get access control and accounting information, and then assigns a "cost" to each line in the network. This cost reflects the desirability of including a certain line in the virtual circuit. Assigning costs is mostly a matter of indexing into correct tables. After that assignment, the path of lowest cost is to be found by an algorithm similar to Dijkstra [22]. Details of the specific algorithm appeared in [65]. This path is defined by backward pointers. If

the cost of the chosen path is too high, the supervisor may choose to reject the user rather than tax the network to provide poor service. Whenever the network conditions change, e.g. line failure or overload, the supervisor is notified and ready to take this into account for the next virtual circuit to be built. The next step is to send to the source node a "needle", which contains the routing information and threads its way through the network, building the virtual circuit as it goes, with the user data following behind.

SNA and TYMNET are both session-based networks using virtual circuits. A good classification of route selection algorithms for session-based networks, both static and dynamic, was proposed by Maruyama and Shorter [49]. Their work is based on the network work-load information available for making decisions. A reliable distributed route set-up procedure using LPID (local path identifiers) was introduced by Segall and Jaffe [79].

L. OTHER PRACTICAL EXAMPLES

In this section, routing algorithms of some other practical networks are introduced.

Digital Equipment Corporation's Digital Network Architecture (DNA) is the standard structure for DECNET network products first introduced in 1973. In its Phase III implementation, the transport layer of DECNET, corresponding to the network layer of ISO's OSI model, provides pure datagram service to its higher layer, network services layer. Packets may be delivered out of sequence, may loop, may be duplicated, and may be discarded by the congestion control mechanism. All these problems are taken care of by the network services layer [73], [88].

The routing algorithm used in DECNET is essentially a copy of the original ARPANET algorithm, i.e. it is a distributed adaptive algorithm. Routing tables kept at each node contains two matrices, HOPS and COST, HOPS(i,j) denoting the path length to node i via line j, and COST(i,j) the path cost to node i via line j. From these can be calculated the existence of a path to a given destination i if there are "reachable values" (to be explained below) in some entry in row i of HOPS, and the best next hop to that destination, i.e. the line corresponding to the minimum value in row i of COST with a reachable value for that entry in HOPS. Each individual node thus knows the best next hop to each of the destinations. Data messages are delivered along such lines, which constitute the best path from source to destination.

The path lengths and costs are exchanged among adjacent nodes as "routing messages." Whenever an event that potentially changes paths occurs (e.g. a line or node going down or coming up, or the reception of new path information from adjacent nodes), a node determines if its paths have changed, or if its HOPS and COST matrices should

be updated. If anything changed, the node sends its new routing message to all its adjacent nodes. The routing messages are exchanged either upon such changes or at periodic time intervals. Figure 13 shows a typical routing data base.

Another usage of the path length information, HOPS, is to detect routing loops computed by routing algorithm. Such loops may take place when, in reality, the destination is unreachable. They may also be due to the time delay in transmitting HOPS and COST and the subsequent improper sequence in which they are received. When the hop count exceeds the longest possible nonredundant path length in the network, the algorithm stops circulating routing messages, marking the node unreachable in the HOPS matrix.

Though the routing matrices are updated in much the same way as the original ARPANET algorithm, DECNET only attempts to adapt to topology changes, not to traffic fluctuations. Instead of delay, the inverse of the line bandwidth is used as cost metric. Because of the use of additional event-driven updating process (triggered by line or node coming-up or going-down), the frequency of the periodic updates can be much less than that of ARPANET (15 seconds). Another difference is that each node in ARPANET maintains estimated delay and hop count only for the best line, while nodes in DECNET maintain the information for every outgoing line, thus allowing the possibility of bifurcation, if desired, or if necessary.

Packets for an unreachable destination are discarded. If a line fails, packets queued on that line are discarded. To maintain end-to-end integrity, acknowldgements and time-outs are employed by the higher level network service layer. The lower level data link layer provides line (or node-to-node) error control.

How does a node know if a line has failed? This is based on the number of retransmissions of packets needed. In addition, if a neighboring node has not been heard of

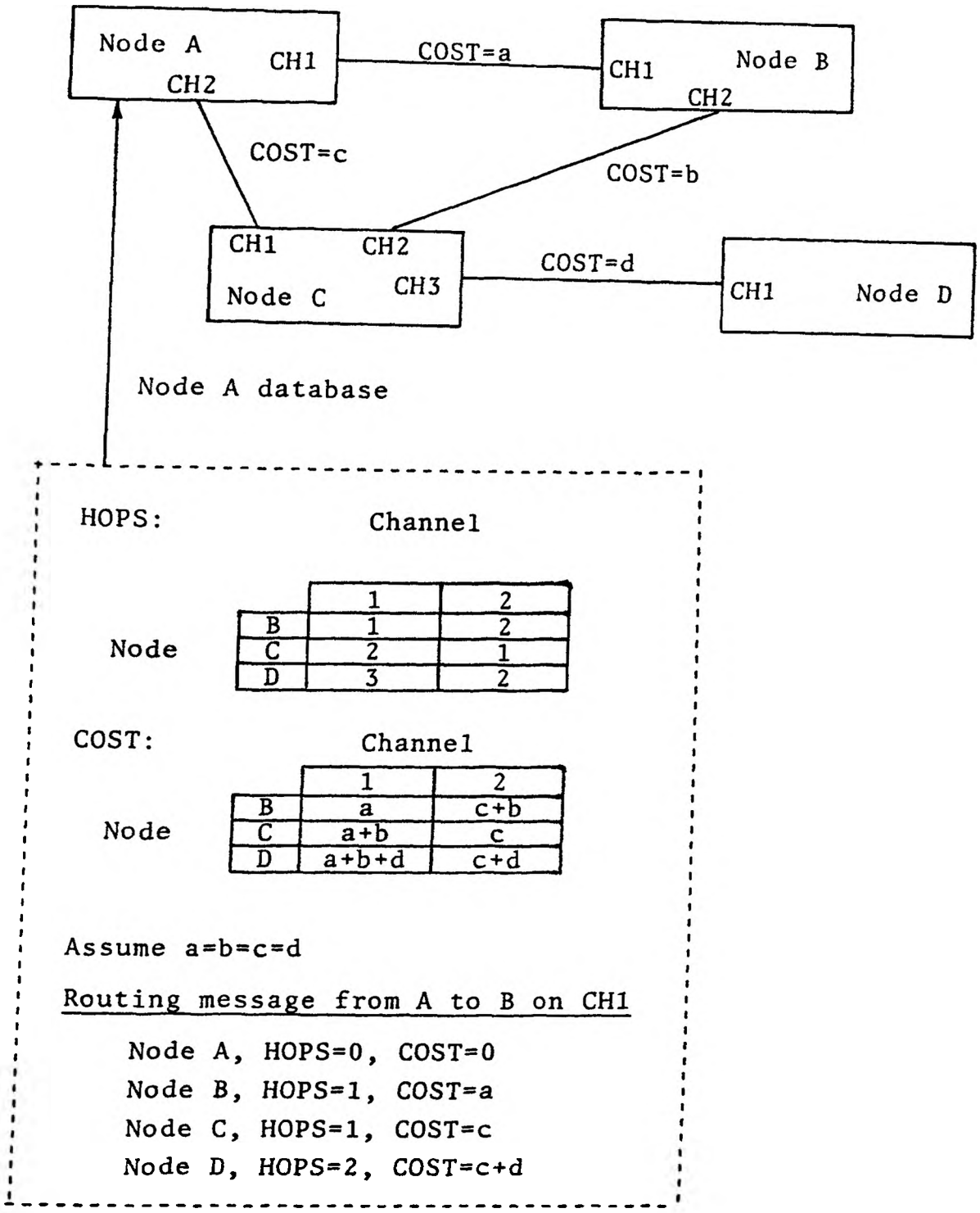


Figure 13. DECNET routing database and routing message

for a while, a low-priority "Hello" message is sent to it. If there is no acknowledgement, this node is considered to have failed.

The actual implementation of the routing involves three parts: a "decision process" which receives routing messages, an "update process" which updates the routing tables, and a "forwarding process" which routes the packets on minimum cost paths. The first two are run only when changes in network topology dictates changes in routing tables. Only the third one is used normally.

The Canadian public network DATAPAC began commercial service in 1977. It employs a distributed adaptive routing algorithm [83]. Based on Northern Telecom SL-10 Packet Switching System, DATAPAC has its communication facilities in three layers. At the core, the datagram subnet layer provides a basic internodal communication facility. On top of that layer, a virtual circuit (VC) communications layer is built to provide the basic DATAPAC VC service. Customer access to VC service is provided by the network access layer.

For routing in subnet, a routing vector table (RVT) is maintained by a global routing information process at each node. This process communicates with similar processes at each of its neighboring nodes by means of "routing updates", which provide information about what nodes can be reached by the neighboring nodes and the delay estimates in reaching them, based on the number and speed of trunks traversed to get to the destination. This information is used to build up another table giving the delay estimate for each trunk group (TG) (collection of all trunks that connects to the same adjacent node) on the node to reach each of the nodes when using that TG. Then the TG of minimum delay estimate is selected for reaching each of the possible destinations, and this information is used to update the RVT. This method of selecting routing update delay estimates is called the "split-horizon" method, and

is intended to minimizing routing loops.

The VC routing relies primarily on the lower level subnet routing. Once the destination VC process is established and the addresses exchanged, the two VC processes can communicate directly through the subnet. If intermediate trunks or nodes fail during a call and an alternate path is available, the subnet can automatically adapt to the topology change without affecting the established VC.

Another higher level of internetwork routing is performed using an adjacent network routing table (RT) at each node to route to the nearest gateway serving the network. The process maintaining the adjacent network RT is the same as that controls the RVT.

Both internodal and internetwork routing are "topology adaptive", i.e. the selected route will not be altered until there is a topology change. This is similar to DECNET. The French public packet switching network TRANSPAC began operation in 1978. It is a virtual circuit oriented system [20], [73]. As mentioned in Section G, it is similar to Delta routing due to Rudin. It is partially decentralized through six local control points which handle a certain amount of statistics gathering and perform test and reinitialization procedures in case of node or line failures. The general network supervision, including the bulk of routing computation, is exercised through a single Network Management Center (NMC).

The algorithm assigns the routes on a single-path-per-VC basis. To establish a VC, a call request in the form of "call packet" is emitted by the source node, requesting connection to a specified destination. The path that eventually will be retained by the switched VC is identical to that taken by the call packet as it is forwarded through the network. Routing of the call packet is directed by each node's routing table, which contains a unique outgoing line for each destination node.

The routing tables are constructed in an essentially centralized fashion, using a minimum cost criterion. Line costs are defined in terms of line resource utilization (a function of line capacity and line buffers) evaluated both by estimation and by measurement. Thus the cost of line varies dynamically with network load.

The major part of routing computation takes place at the NMC, but some local information is used at each node. The procedure can be illustrated by an example in Figure 14. A call packet arriving at node 1 is to be forwarded through one of the adjacent nodes 2, 3, 4 to node 5. Consider a full duplex line k connecting nodes m and n. Let $C_m(k)$, $C_n(k)$ be the cost of line k as perceived by node m and node n, respectively. Let $C(k) = \text{Max}[C_m(k), C_n(k)]$, and let C(k,n) (computed by NMC) be the total cost associated with the minimum cost path between nodes k and n. Node 1 determines the best path to node 5 by choosing the value of k which minimizes $C(k,5) + \text{Max}[C_1(k), C(k)]$ where k = 2, 3, 4. In this way, the final routing decision is made locally, rather than using purely centralized procedure.

Some other networks are briefly mentioned below. Telenet [38] initially duplicated ARPANET technology and later modified its internal transport technology into one similar to that used in TYMNET. The small private network, MERIT, connecting three Michigan universities [85] uses a distributed adaptive shortest path routing algorithm, very similar to that used in the old ARPANET except that it measures the distance of a path by its hop number. SITA operates a worldwide network [41] which uses predetermined routing tables containing a primary path and several alternate paths for each "destination group", keeping the routing separate from the failure recovery. The Cyclades network in France [63], [95] was designed initially with static routing, but was subsequently changed to an ARPANET-like algorithm.

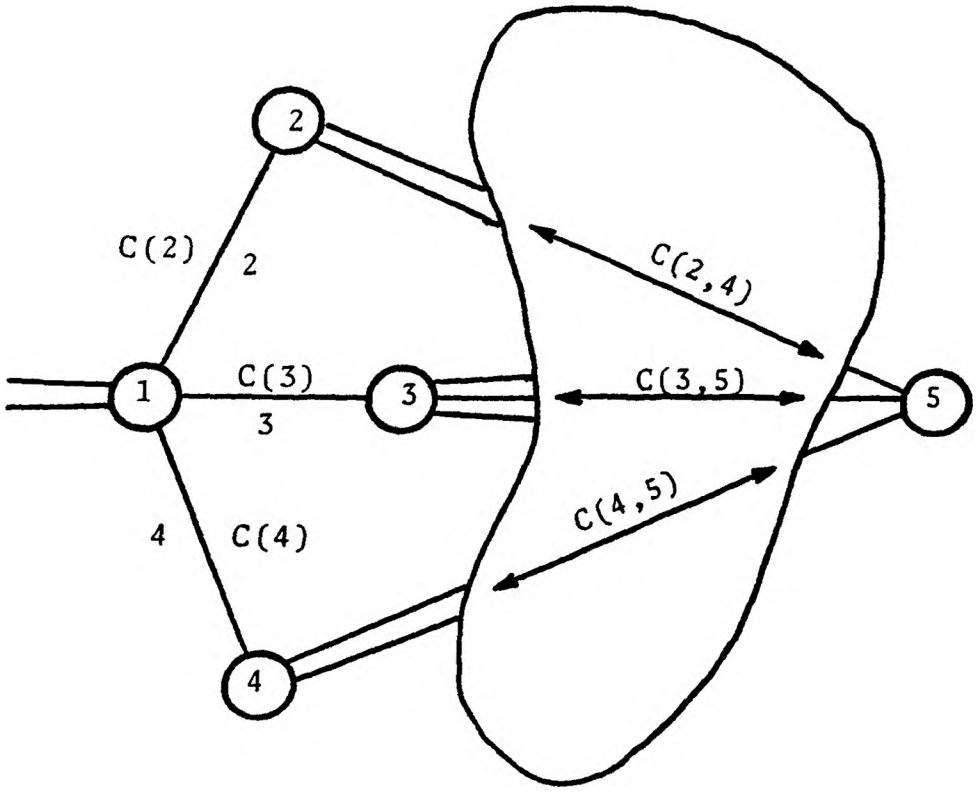


Figure 14. TRANSPAC routing example

XII. OTHER RESEARCH IN THIS AREA

Packet switching networks are an efficient means for transmitting bursty traffic, because extensive resource sharing is allowed through routing and flow control.

Flow control is the mechanism for regulating the rate at which the sender generates messages so that the receiver can process them. From the network user's point of view, flow control prevents those messages that cannot be delivered in a predefined time from entering the network. When messages from the sender exceed the capacity of the receiver to process or forward, congestion arises. In this respect, flow control is a mechanism for preventing congestion. Many authors do not distinguish flow control from congestion control, though the two terms are differently defined by some [86].

In most of the available literature, routing and flow control have been studied as separate problems. Consequently, routing procedures have been designed independently of flow control schemes. As a result, when a packet is submitted to a network, the flow control procedure will first determine whether or not to accept it, generally based on the buffer availability. If the packet is accepted, it is then the task of the routing procedure to find a path to deliver the packet to its destination. This path, however, will not always exist. A better approach would, therefore, be to combine the routing and flow control decisions, i.e. to ascertain that a feasible path does exist before accepting a packet into the network.

The interactions between routing and flow control (congestion control) has been dealt with in depth by McQuillan [53] and Gerla et al [35]. Other authors have also pointed out that it could be misleading to try to study routing and flow control as isolated mechanisms [68], [69]. A number of contributions were made towards achieving the integration of routing and flow control [6],

[29], [31], [33], [35].

A new scheme was proposed by Gerla et al [32] as another step in the direction of that goal. One feature of this proposal is that the fairness issue is also included in the problem. As pointed out in this paper, efficiency of routing and flow control algorithms in sharing resources does not always imply fairness, because the network may favor some users over the others in order to achieve better overall efficiency. Unfairness is undesirable especially in public networks where users pay the same tariff, supposedly, for equal services. In this sense, the new solution has considerable significance.

As Gerla et al reported, work is now under way on the implementation of the integrated routing and flow control algorithm in actual networks, both centralized and distributed solutions being considered. Another issue under investigation is to find and use different fairness as objective functions in the optimization [32].

Although nonadaptive routing does not seem to be as much preferred in present and future networks, it is not without value. Some authors have argued that only nonadaptive (or semidynamic) routing will be effective in the future environment of very large networks, because a fully dynamic approach may require enormous overhead. Traditionally, nonadaptive routing is associated with centralized routing. Examples of such studies include the paper by Pesic and Lewis [62], in which three heuristics for improving centralized routing in large long-haul networks have been developed and applied in the construction of fast routing algorithms.

The basic problem of routing analysis is the fact that the adaptive routing involves the time-varying behavior of a set of interactive queues. Examples of theoretical work carried out in the last few years are as follows.

Yum et al studied the design and analysis of semidynamic routing rules [90]. These rules were also

studied as the load balancing problem in the queueing system perspective [91], which is useful in multidestination routing algorithms. Research on networks with multiple destinations also includes analysis of the dynamic behavior of shortest path algorithms for such networks by Bertsekas [4].

In 1977, Segall introduced a new model for dynamic and quasistatic routing [74]. In 1983. he presented a unified approach to the formal description and validation of several distributed protocols [78].

Foschini et al analyzed a basic dynamic routing strategy using diffusion theory. A heavy traffic diffusion method and the limitations of an ad hoc approach to applying diffusion were explored in [25], [26].

Schoute et al approached the problem of distributed routing by separating the information problem from the control problem [71]. As they noted, under the assumption of perfect information, the control problem has a simple solution, which is optimal with respect to minimizing delay for individual packets. Perfect information, however, is not possible, because the actual values of delay change rapidly. For finding a good practical information policy, they examined several classes of information policies corresponding to a cascade of stochastic processes.

A problem that may occur in virtual circuit networks is routing instability, if the rerouting of virtual circuits is allowed. This problem was investigated, and a method for achieving a stable rerouting rate was proposed by Wunderlich et al in [89].

It was recognized by Rybczynski et al that the integration of circuit and packet switching may be a long-term objective. The implementation of integrated circuit/packet switching networks by common carrier is likely some years in the future. Opportunities to apply integrated voice/data access systems to packet switching networks exist today. They studied the interworking

between packet networks and various integrated voice/data access systems and demonstrated that the provision of integrated voice/data access systems to packet switching networks is an important step towards the integration of circuit and packet switching technologies.

Routing in integrated voice/data networks has also been investigated. A strategy to handle adaptive routing, flow control and buffer allocation as a whole in the integrated voice/data networks has been proposed by Nassehi et al [61]. A distributed algorithm similar to one proposed by Bertsekas [5] is used to implement this optimum strategy.

XIII. CONCLUSION

A brief discussion of routing for computer networks in general was given, followed by an overview of the typical routing algorithms reported or used in the past few years. Although the algorithms were oriented towards a broad spectrum of operational characteristics and optimization criteria, it is interesting to note that there are many similarities in them. At the same time, there is a great deal of diversity in the manner in which these algorithms are designed or implemented. The author's point of view is biased towards distributed adaptive algorithms.

Generally speaking, distributed adaptive routing procedures perform the following five functions in one way or another.

1) Measurement or estimation of network parameters pertinent to routing strategy, including traffic load, states of lines, line weights, available resources (line capacity, nodal buffer), etc.

2) Forwarding of the measured or estimated information to the nodes where routing computation takes place.

3) Computation of routing tables.

4) Conversion of routing table information into packet routing decisions.

5) Transmission of packets.

The adaptability ranges from the bare mimimum necessary to react to line failures to more sophisticated procedures sensing and responding to queueing delays, error rates and line loading. A still larger variety is represented by the rich set of alternative schemes for information gathering, routing computation and packet forwarding. One can conclude from this survey that while the routing function is essential to the efficient and smooth operation of networks, no one scheme can deserve the name as "best".

It is also evident from this survey that the problem

of designing good routing algorithms is an active research area. On the one hand, many new algorithms have achieved various improvements over their predecessors. On the other hand, they still have inefficiencies, limitations, and undesirable properties. In the future, it seems that much more attention should be focused on topics such as

-- accurate routing and correct adaptation based on uncertain and imprecise traffic information,

-- routing in multidestination networks,

-- routing in large networks,

-- routing in heterogeneous network environments, combining different types of traffic, differenttransport mechanisms and different media,

-- internetwork routing,

-- integration of routing with flow control.

Above all, there is a need for convincing methods of proving the effectiveness of routing algorithms. All too often, analytic and simulation work relies on simplifying assumptions which weaken the applicability of the results.

BIBLIOGRAPHY

[1] Ahuja, V.: "Routing and Flow Control in Systems Network Architecture", IBM Systems Journal, Vol. 18, No. 2, 1979, pp. 298-314

[2] Atkins, J. D.: "Path Control: The Transport Network of SNA", IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980, pp. 527-538

[3] Bertsekas, D. P.: "Algorithms for Nonlinear Multicommodity Network Flow Problems", Proceedings of International Symposium of System Optimization and Analysis, 1979, pp. 210-224

[4] Bertsekas, D. P.: "Dynamic Models of Shortest Path Routing Algorithms for Communication Networks with Multiple Destinations", Proceedings of 18th IEEE Conference on Decision and Control, 1979, pp. 127-133

[5] Bertsekas, D. P. : "A Class of Optimal Routing Algorithms for Communication Networks", Proceedings of International Conference on Computer Communications, October 1980, pp. 71-75

[6] Bertsekas, D. P.: "Optimal Routing and Flow Control Methods for Communication Networks", Proceedings of 5th International Conference, 1982, pp. 615-621

[7] Bertsekas, D. P., Gafni, E. and Gallager, R. G.: "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks", IEEE Transactions on Communications, Vol. COM-32, No. 4, August 1984, pp. 911-919

[8] Bertsekas, D. P., Gafni, E. and Vostola, K. S.: "Validation of Algorithms for Optimal Routing of Flow in Networks", Proceedings of 17th IEEE Conference on Decision and Control, January 1979, pp. 220-227

[9] Boorstyn, R. and Livne, A.: "A Technique for Adaptive Routing in Networks", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 474-480

[10] Brayer, K.: "Implementing Computer Communications with OEM Microprocessors: Survivable Network Routing System", IEEE GLOBCOM, November 1982, pp. 1166-1170

[11] Brayer, K.: "Implementation and Performance of Suvivable Computer Communication with Autonomous Decentralized Control", IEEE Communications Magazine, July 1983, pp. 34-41

[12] Brayer, K. and Lafleur, V. S.: "A Testbed Approach to the Design of a Computer Communication Networks", Computer, No. 10, Octobet 1982

[13] Chen, M. S. and Meditch, J. S. : "A Distributed Adaptive Routing Algorithm", Proceedings of IEEE International Communications Conference, 1983, pp. 484-492

[14] Chin, C. and Hwang, K.: "A New Probabilistic Routing Algorithm for Packet-Switched Computer Networks", Proceedings of AFIPS National Computer Comference, May 1983, pp. 705-719

[15] Chou, W., Bragg, A. W. and Nilsson A. A.: "The Need for Adaptive Routing in the Chaotic and Unbalanced Traffic Environment", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 481-490

[16] Chou, W., Nilsson, A. A. and Bragg, A. W.: "The Need for Dynamic Routing in a Network Spanning Several Time Zones", IEEE Communications Magazine, July 1982, pp. 13-17

[17] Chou, W., Powell, J. D. and Bragg, Jr. A. W.: "Comparative Evaluation of Deterministic and Adaptive Routing", Flow Control in Computer Networks, IFIP, North-Holland Publishing Company, 1979, pp. 257-279

[18] Chu, K. C.: "A Distributed Protocol for Updating Network Topology Information", Report RC 7235 (#31163), IBM T. J. Watson Research Center, July 1978

[19] Chyung, D. H. and Reddy, S. M.: "A Routing Algorithm for Computer Communication Networks", IEEE Transactions on Communications, Vol. COM-23, No. 11, November 1975

[20] Danet, A., Despres, R., LaRest, A., Pichon, G. and Ritzenthaler, S.: "The French Public Packet Switching Service: The Transpac Network", Proceedings of 3rd International Conference on Computer Communications, August 1976, pp. 251-260

[21] Davis, D. W. and Barber, D. L.: Computer Networks and Their Protocols, Wiley, 1979, pp. 89-107

[22] Dijkstra, E. W.: "A Note on Two Problems in Connection with Graphs", Numerical Mathematics, Vol. 1, 1959, pp. 269-271

[23] Dowdy, L. W. and Tripathi, S. K.: "Routing Strategies: Classification and Comparison". Technical Report TR-798, Department of Computer Science, University of Maryland, College Park, August 1979

[24] Finn, G.: "Resynch Procedures and a Fail-Safe Network Protocol", IEEE Transactions on Communications, Vol. COM-27, No. 6, June 1979, pp. 840-845

[25] Foschini, G. J.: "On Heavy Traffic Diffusion Analysis and Dynamic Routing in Packet Switched Networks", Computer Performance, North Holland Publishing Company, 1977, pp. 499-513

[26] Foschini, G. J. and Salz, J.: "A Basic Dynamic Routing Problem and Diffusion", IEEE Transactions on Communications, Vol. COM-26, No. 3, March 1978, pp. 320-327

[27] Fratta, L., Gerla, M. and Kleinrock, L.: "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design", Networks, No. 3, 1973, pp. 97-133

[28] Gallager, R. G.: "A Minimum Delay Routing Algorithm Using Distributed Computation", IEEE Transactiocs on Communications, Vol. COM-25, No. 1, January 1977, pp. 73-85

[29] Gallager, R. G. and Golestaani, S. J.: "Flow Control and Routing Algorithms for Data Networks", Proceedings of 5th International Conference on Computer Communications, October 1980, pp. 779-784

[30] Gavish, B. and Hantler, S. L.: "An Algorithm for Optimal Route Selection in SNA Networks", IEEE Transactions on Communications, Vol. COM-31, No. 10, October 1983, pp. 1154-1160

[31] Gerla, M.: "Routing and Flow Control", Protocols and Techniques for Data Communications Networks, Prentice Hall, 1981, pp. 122-174

[32] Gerla, M., Chan, H. W. and DeMarca, J. R. B.: "Routing, Flow Control and Fairness in Computer Networks", Proceedings of IEEE International Conference on Communications, May 1984

[33] Gerla, M. and Kleinrock, L.: "Flow Control: A Comparative Survey", IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980, pp. 553-574

[34] Gerla, M., Kleinrock, L. and Afek, Y.: "A Distributed Routing Algorithm for Unidirectional Networks", IEEE GLOBCOM 1983, pp. 654-658

[35] Gerla, M. and Nilsson, P. O.: "Routing and Flow Control Interplay in Computer Networks", Proceedings of 5th International Conference on Computer Communications, October 1980, pp. 84-89

[36] Giessler, A., Hanle, J., Koing, A. and Pade, E.: "Free Buffer Allocation -- An Investigation by Simulation", Computer Networks, Vol. 2, 1978, pp. 191-204

[37] Gorecki, F. D. and Meditch, J. S.: "On Minimum Hop Flow Assignment in Message-Switched Telecommunication Networks", Proceedings of 8th Triennial World Congress of the International Federation of Automatic Control, August 1981, pp. 1727-1732

[38] Hovey, R. B.: "The User's Role in Connecting to a Value-Added Network", Data Communication, May/June 1974

[39] Jaffe, J. M. and Moss, D. H.: "A Responsive Distributed Routing Algorithm for Computer Networks", IEEE Transactions on Communications, Vol. COM-30, No. 7, July 1982, pp. 1758-1762

[40] Jaffe, J. M., Moss, F. h. and Weingarten, R. A.: "SNA Rotuing: Past, Present and Possible Future", IBM Systems Journal, Vol. 22, No. 4, 1983, pp. 417-434

[41] Kaliszewski, J. M.: "Routing Algorithm and Route Optimization on SITA Data Transport Network", IEEE GLOBECOM 1983, pp. 892-897

[42] Kamoun, F. and Kleinrock, L.: "Stochastic Performance Evaluation of Hierarchical Routing for Lange Networks", Computer Networks, Vol. 3, 1979, pp. 337-353

[43] Kleinrock, L.: Queueing System, Vol. 2, Computer Application, Wiley, 1976

[44] Kleinrock, L.: "Power and Deterministic Rules of Thumb for Probabilistic Problems in Computer Communications", Proceedings of International Conference on Communications, June 1979

[45] Kleinrock, L. and Kamoun, F.: "Hierarchical Routing for Large Networks", Computer Networks, Vol. 1, 1977, pp. 155-174

[46] Maglaris, B. S.: "An Optimal Local Policy for Two-Level Adaptive Routing in Computer Networks", Proceedings of IEEE INFOCOM 1984, pp. 284-290

[47] Maruyama, K.: "Defining Routing Tables for SNA Networks", IBM Systems Journal, Vol. 22, No. 4, 1983, pp. 435-450

[48] Maruyama, K. and Markowsky, G.: "On the Generation of Explicit Routing Tables", Proceedings of 5th International Conference on Computer Communications, October 1980, pp. 90-95

[49] Maruyama, K. and Shorter, D.: "Dynamic Route Selection Algorithms for Session Based Communication Networks", ACM SIGCOMM Computer Communication Review, Vol. 13, No. 2, 1983, pp. 162-169

[50] McQuillan, J. M.: "Design Considerations for Routing Algorithms in Computer Networks", Proceedings of 7th Hawaii International Conference System Science, January 1974, pp. 22-24

[51] McQuillan, J. M.: "Adaptive Routing Algorithms for Distributed Computer Networks", Bolt Bernek and Newman Inc., Report-2831, May 1974

[52] McQuillan, J. M.: "Routing Algorithms for Computer Networks -- A Survey", Conference Record of National Telecommunications Conference, 1977, Vol. 2

[53] McQuillan, J. M.: "Interactions between Routing and Congestion Control in Computer Networks", Flow Control in Computer Networks, IFIP, North-Holland Publishing Company, 1979, pp. 63-75

[54] McQuillan, J. M., Falk, G. and Richer, I. : "A Review of Development and Performance of the ARPANET Routing Algorithm", IEEE Transactions on Communications, Vol. COM-26, No. 12, December 1978, pp. 1802-1810

[55] McQuillan, J. M.,Richer, I. and Rosen, E. C.: "The New Routing Algorithm for the ARPANET", IEEE Transactions Communications Vol. COM-28, No. 5, May 1980, pp. 711-719

[56] Meditch, J. S. and Gorecki, F. D.: "Minimum Hop Flow Assignment and Routing in Computer-Communication Networks", Proceedings of 19th IEEE Conference on Decision and Control, December 1980, pp. 634-636

[57] Meditch, J. S. and Gorecki, F. D.: "A Distributed Minimum Hop Routing Algorithm", Proceedings of 20th IEEE Conference on Decision and Control, December 1981, pp. 392-397

[58] Meketon, M. S. and Topkis, D. M.: "Adaptive Routing for Damaged Networks", Proceedings of IEEE Military Communications Conference, October 1983, pp. 282-288

[59] Merlin, P. and Segall, A. : "A Failsafe Distributed Routing Protocol", IEEE Transactions on Communications, Vol. COM-27, No. 9, September 1979, pp. 1280-1287

[60] Muralidhar, K. H. and Sundareshan, M. K.: "Adaptive Routing and Flow Control in Large Communication Networks", Proceedings of IEEE INFOCOM 1984, pp. 299-308

[61] Nassehi, M. M. and Hayes, J. F.: "On Routing, Flow Control and Buffer Allocation in Integrated Voice-Data Networks", Proceedings of Canadian Communications and Energy Conference, October 1982, pp. 291-294

[62] Pesic, I. M. and Lewis, D. W.: "Three Heuristics for Improving Centralized Routing in Large Long-Haul Computer Communication Networks, Proceedings of AFIPS National Computer Conference, May 1983, pp. 694-701

[63] Pouzin, L.: "Cigale, The Packet Switching Machine of the Cyclades Computer Network", Proceedings of IFIP Congress, 1974, North Holland, pp. 155-159

[64] Price, W. L.: "Adaptive Routing in Store-and-Forward Networks and the Importance of Load-Splitting", IFIP Information Processing 77, North Holland, pp. 309-313

[65] Rajaraman, A.: "Routing in TYMNET", European Computing Conference, London, England, May 1978

[66] Rosen, E. C.: "The Updating Protocol of ARPRNET's New Routing Algorithm", Computer Networks, Vol. 4, 1980, pp. 11-19

[67] Rudin, H.: "On Routing and 'Delta Routing': A Taxonomy and Performance Comparison of Techniques for Packet-Switched Networks", IEEE Transactions on Communications, Vol. COM-24, No. 1, January 1976, pp. 43-59

[68] Rudin, H. and Muller, H.: "On Routing and Flow Control", Flow Control in Computer Network, IFIP, North Holland Publishing Company, 1979, pp. 241-255

[69] Rudin, H. and Mueller, H.: "Dynamic Routing and Flow Control", IEEE Transactions on Communications, Vol. COM-28, No. 7, July 1980, pp. 1030-1039

[70] Rybczynski, A. M., Shechtman, G. I. and Kayser, L. S.: "Interworking between Packet Networks and Integrated Voice/Data Access Systems", Proceedings of IEEE International Conference on Communications, May 1984

[71] Schoute, F. C. and MqQuillan, J. M.: "A Comparison of Information Policies for Minimum Delay Routing Algorithms", IEEE Transactions on Communications, Vol. COM-26, No. 8, August 1978, pp. 1266-1270

[72] Schwartz, M. and Cheung, C. K.: "The Gradient Projection Algorithm for Multiple Routing in Message Switched Networks", IEEE Transactions on Communications, Vol. COM-24, No. 4, April 1976

[73] Schwartz, M. and Stern, T. E.: "Routing Techniques Used in Computer Communication Networks", IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980, pp. 539-552

[74] Segall, A. : "The Modeling of Adaptive Routing in Data Communication Networks", IEEE Transactions on Communication, January 1977, pp. 85-95

[75] Segall, A. : "Optimal Distributed Routing for Virtual Line-Switched Data Networks", IEEE Transactions on Communication, January 1979, pp. 201-208

[76] Segall, A. : "Failsafe Distributed Algorithm for Routing in Communication Networks", Flow Control in Computer Networks, IFIP, North Holland Publishing Company, 1979, pp. 235-240

[77] Segall, A. : "Advances in Varifiable Failsafe Routing Procedures", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 491-497

[78] Segall, A: "Distributed Network Protocols", IEEE Transactions on Information Theory, Vol. IT-29, No. 1, January 1983, pp. 23-35

[79] Segall, A. and Jaffe, J. M.: "A Reliable Distributed Route Set-Up Procedure", IEEE GLOBCOM 1983, pp. 644-648

[80] Segall, A., Merlin, P. M. and Gallager, R. G.: "A Recoverable Protocol for Loop-Free Distributed Routing", Proceedings of IEEE International Conference on Communications, 1978, pp. 3.5.1.-3.5.5.

[81] Segall, A. and Sidi, M.: "A Failsafe Distributed Protocol for Minimum Delay Routing", IEEE Transactions on Communications, Vol. COM-29, No. 5, May 1981, pp. 689-695

[82] Sidi, M. and Segall, A: "Failsafe Distributed Optimal Routing in Data Communication Networks", Dept. of Electrical Engineering, Technion, Haifa, EE Pub. 342, december 1978

[83] Sproule, D. E. and Mellor, F.: "Routing, Flow and Congestion Control in the Datapac Network", IEEE Transactions on Communication, Vol. COM-29, No. 4, April 1981, pp. 386-391

[84] Stern, T. E.: "An Improved Routing Algorithm for Distributed Computer Networks", ICCAS/80 Workshop on Large Scale Systems, 1980

[85] Tajibnapis, W. D.: "A Correctness Proof of a Topology Maintenance Protocol for a Distributed Computer Network", Communications of the ACM, July 1977, pp. 477-485

[86] Tanenbaum, A. S.: Computer Networks, Prentice-Hall, 1981

[87] Tymes, L. R. W.: "Routing and Flow Control in TYMNET", IEEE Transactions on Communications Vol. COM-29, No. 4, April 1981, pp. 392-398

[88] Wecker, S.: "DNA -- The Digital Network Architecture", Computer Network Architectures and Protocols, Plenum Publishers, 1982, pp. 249-296

[89] Wunderlich, E. F. and Printis, R. S.: "Rerouting Stability in Virtual Circuit Data Networks", Proceedings of International Conference on Communications, June 1980, pp. 13.5.1.-13.5.5.

[90] Yum, T. P.: "The Design and Analysis of a Semidynamic Deterministic Routing Rule", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 498-504

[91] Yum, T. P. and Lin, H. C.: "Adaptive Load Balancing for Parellel Queues", Proceedings of IEEE International Conference on Communications, May 1984

[92] Yum, T. P. and Schwartz, M.: "Comparison of Adaptive Algorithms for Computer Communication Networks", Proceedings of National Telecommunications Conference, December 1978, pp. 4.1.1.-4.1.4.

[93] Yum, T. P. and Schwartz, M.: "The Join-Biased-Queue Rule and Its Application to Routing in Computer Communication Networks", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 505-511

[94] Zimmermann, H.: "OSI Reference Model", IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980, pp. 425-432

[95] Zimmermann, H.: "The Cyclades Experience -- Results and Impacts", IFIP Information Processing 77, North Holland, pp. 465-469