DYNAMICS OF DISTRIBUTED SHORTEST-PATH ROUTING ALGORITHMS

William T. Zaumen, J.J. Garcia-Luna Aceves

zaumen@nisc.sri.com
garcia@sri.com
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025*

ABSTRACT

The dynamics of shortest-path routing algorithms based on distance vectors and link states are investigated. Detailed quantitative comparisons of the distributed Bellman-Ford algorithm used in several routing protocols in the past, an ideal link-state algorithm similar to the one used in OSPF and in the OSI intradomain routing protocol, and a loop-free distance-vector algorithm, are made for the network topologies of the 1988 ARPANET, LOSNETTOS, DOE-ESNET, and the NSFNET T1 Backbone. Comparisons include the response of the algorithms to link-cost changes and to link and node failures and recoveries. A variety of quantities, including the length of messages and the average number of paths affected by routing loops, are computed as a function of time after a link or node change. Probabilities of various conditions are also obtained as a function of time, including the existence of loops. As expected, the distributed Bellman-Ford algorithm behaves poorly compared with the other two. However, deciding between a routing protocol based on a link-state algorithm and one based on a loop-free distance-vector algorithm depends on the particular network in which it will perform.

1. INTRODUCTION

The shortest-path routing algorithms used in computer networks today can be classified as distance-vector or link-state algorithms (also called topology-broadcast algorithms). In a distance-vector algorithm, a node knows the length of the shortest path (distance) from each node linked to it to every network destination, and uses this information to compute the distance and next node in the path to each destination. In a link-state algorithm, a node must know the entire network topology to compute its distance to any network destination. Each node broadcasts update messages, containing the state of each of the node's adjacent links to every other node in the network.

* This work was supported by SRI IR&D funds.

Well-known examples of routing protocols based on distance-vector algorithms, which we call *distance-vector protocols* (DVP), implemented in internetworks are the Routing Information Protocol (RIP) [HEDR-88], the HELLO protocol [MILL-83], the Gateway-to-Gateway Protocol (GGP) [HIND-82], and the Exterior Gateway Protocol (EGP) [MILL-84]. The old ARPANET routing protocol [MCQU-74] and the routing protocol of the Digital Network Architecture (DNA) Phase IV [SCHW-86] are examples of DVPs used in computer networks. All of these DVPs have used variants of the distributed Bellman-Ford algorithm for shortest-path computation [FORD-62, BERT-87]. The primary disadvantages of this algorithm are *routing-table loops* and *counting to infinity* [JAFF-82, GARC-89a]. A routing-table loop is a path specified in the nodes' routing tables at a particular point in time, such that the path visits the same node more than once before reaching the intended destination. A node counts to infinity when it increments its distance to a destination until it reaches a predefined maximum distance value.

On the other hand, link-state algorithms are free of the counting-to-infinity problem. However, each node needs to receive up-to-date information on the entire network topology. Well-known examples of routing protocols that use link-state algorithms, which we call *link-state protocols* (LSP) are the new ARPANET routing protocol [MCQU-80], the ANSI proposal for IS-IS routing [ISO-89], and the Open Shortest Path First (OSPF) protocol [COLT-89].

Because of the poor performance of DVPs implemented using the distributed Bellman-Ford algorithm, because of the clear improvements obtained with an LSP over a DVP based on the Bellman-Ford algorithm, and because efficient distance vector algorithms had not appeared until recently, DVPs have been recently considered to be inferior to LSPs. This view has been reinforced by previous unsuccessful attempts to solve the Bellman-Ford counting-to-infinity and routing-table-loop problems [CEGR-75, NAYL-75, SHIN-87, STER-80]. Recently, however, a number of efficient distance-vector algorithms have been proposed to either eliminate counting to infinity [CHEN-89, LOUG-89, SHIN-87], or eliminate routing-table loops altogether [GARC-89b, JAFF-82, MERL-79].

This paper provides further insight into and understanding of the dynamics of DVPs and LSPs. The algorithms used in our

analysis were the distributed Bellman-Ford algorithm, which has been used in several DVPs; an ideal link-state algorithm based on Dijkstra's shortest-path algorithm, which provides an upper bound on the performance of today's LSPs; and the loop-free distance-vector algorithm proposed by Garcia-Luna [GARC-89b], which performs at least as well as any other loop-free distance-vector algorithm proposed to date.

Previous papers addressing the performance of shortest-path algorithms have focused on their worst-case behavior (or complexity), or have been limited to the analysis of the average number of messages and the time required for their convergence. This paper presents a detailed quantitative comparison of the dynamic behavior of three of the most relevant shortest-path algorithms for the first time. It shows that counting to infinity and looping problems are no longer an issue when choosing between an LSP and a DVP. Rather, the choice involves a multifaceted tradeoff that can be driven by a variety of factors, including the performance and type of the operating system used at network nodes. Determining the mean or worst-case values for such quantities as the time to convergence or the total number of packets sent will not be sufficient and, as will be shown, can be misleading.

It is not now possible to sample enough different topologies to make statistically justifiable statements about how an algorithm's performance scales with parameters that can characterize a topology (e.g., number of nodes and links, diameter, or node degree). Therefore, we used the topologies of four well-known networks (ARPANET, LOS-NETTOS, DOE-ESNET, and the NSFNET-T1 Backbone) to analyze by simulation the behavior of the three algorithms mentioned.

Sections 2 through 5 describe the design, validation, instrumentation, and test cases of our simulation. Sections 6 and 7 describe the static and dynamic behavior of the algorithms, and Section 8 compares them.

2. SIMULATION DESIGN

We developed the simulations using an actor-based, discrete-event, simulation language called *Drama* [ZAUM-91] together with a network-simulation library. The library treats both nodes and links as actors. Nodes send packets over links by using the functional-call interface to the links actor, but they receive packets by responding to messages delivered from the event queue. Link failures and recoveries are handled by sending a link-status message to the nodes at the end points of the appropriate link. In the link models used in the simulation, link propagation time is an input parameter (which can be changed during the course of the simulation), although all runs were made with unit propagation time. If a link fails, packets in transit are dropped.

For the routing-algorithm simulations in this study, a node receives a packet and responds by running a routing algorithm, then queuing any outgoing updates, and finally, by waiting for some processing time. If any incoming packets arrive before the processing time expires, the routing algorithm is run again and any new packets generated are queued. Once the processing time for all these events has expired, redundant updates are removed (removal is algorithm dependent) and the queues are sent over the links. In the runs actually made, processing time was set to zero; however, *Drama*s internal mechanisms ensured that all updates due to arrive at the current simulation time were processed before any new updates were generated. The advantage to this approach is that simulation puts multiple updates into the same packet and, for many networks, the number of packets is a more critical measure of performance than the number of bits actually transmitted.

Three shortest-path algorithms were implemented—the Diffusing Update Algorithm (DUAL) [GARC-89b], the Distributed Bellman-Ford algorithm (DBF) [FORD-62, BERT-87], and an Ideal Link-State algorithm (ILS) using Dijkstra's shortest-path algorithm [DIJK-59] at each node. In all cases, the algorithms produced a routing table giving a successor for each destination. The successor is either a neighboring node or null (a null indicates that no current path leads to the destination).

In the DBF simulations, when an update packet is built, only the latest update for a given destination is sent. This reduces the length of the routing packets. Similarly, redundant updates were removed in the DUAL simulations. The ILS simulations slightly underestimate the number of updates that must be sent: in practice, a link-state algorithm will rely on periodic topology broadcasts and will need some mechanism to allow sequence numbers to be reused (a fixed, preferably small,, number of bits are available for such purposes). We have ignored these complexities because the mechanisms needed to handle them can be spaced long enough apart to avoid a severe impact on performance.

Because the overhead imposed by a fail-safe link-state algorithm is very high, however, existing LSPs may simplify the mechanisms needed to provide fail-safe topology broadcast. Consequently, the performance of ILS constitutes an upper bound on the performance that can be achieved with LSPs. For example, to reduce its overhead, OSPF requires only adjacent gateways to remain synchronized [COLT-89]; no broadcast of link-state updates takes place when a failed link is placed back in operation. In theory, this simplification cannot ensure that every gateway will receive all the necessary updates in a dynamic internet [JAFF-86]; in practice, it means that OSPF can be slower to converge after topological changes than predicted by the performance of ILS.

In the following, we occasionally refer to the Dijkstra-LS algorithm, instead of ILS, in contexts where the running time of the shortest-path algorithm at a node is important.

3. NETWORK MODEL AND ASSUMPTIONS

A network is considered as an undirected connected graph in which each link has two lengths or costs associated with it—one for each direction—and in which any link exists in both directions at any one time. The three algorithms implemented assume the existence of a link-level protocol assuring that

- Every node knows its neighbors, which implies that a node detects within a finite time the existence of a new neighbor or the loss of connectivity with a neighbor.
- All packets transmitted over an operational link are received correctly and in the proper sequence within a finite time.
- If a link fails, all packets transmitted before the failure occurred but not yet delivered when the link failed are lost.
- All messages, changes in the cost of a link, link failures, and new-neighbor notifications are processed one at a time, within a finite time and in the order in which they occur.

In this model, each node has a unique identifier and link costs can vary in time but are always positive. The distance between two nodes is measured as the sum of the link costs in the path of least cost or *shortest path* between them. Each node adjacent to a node i is called a *neighbor* of node i.

This same model is directly applicable to an internet, in which gateways are the nodes of the graph and networks are the edges of the graph. For this case, the services listed above are provided by

a datagram service at the network level and a transport-level protocol similar to the transmission control protocol (TCP).

4. VALIDATION

To validate the simulation, we performed a series of tests, initially checking small cases manually using a trace facility that prints out detailed debugging output. We then performed a comprehensive set of tests using the 1988 ARPANET topology, in which we compared how all three algorithms responded to a variety of topological changes—node and link failures and recoveries, random link cost changes, and random sequences of the above. After each set of changes, we let all algorithms converge, and compared the distances produced by each to every destination. Since all are shortest-path routing algorithms, all must produce the same distances after convergence.

During the final test, whenever a bug in any of the simulations was uncovered after the halfway point, we doubled the size of the test. At the end, we ran approximately 250 cases, each consisting of a long sequence of link changes. Tests that measured the response of the algorithm to a change in just a single link or node were replicated to exhaustively cover all nodes and links, but could not meaningfully be extended. Such tests were rerun after any program modification. Finally, all the test cases were rerun for DUAL, using a test showing that the algorithm as implemented never produced a routing loop, in accordance with the algorithm's theoretical development [GARC-89b]. The routing-loop test was not applied to the other algorithms because they are not loop free.

The simulation of DUAL was based on a pseudocode description of the algorithm that was developed concurrently with the simulation itself. Algorithm complexity made a formal proof of correctness of the pseudocode (as opposed to a purely mathematical description of the algorithm) not feasible. Whenever a discrepancy arose between DUAL and the other two (ILS and DBF), we used the simulation output, consisting of snapshots of various tables defined by the algorithm, to determine if a node behaved incorrectly. The pseudocode was then checked and modified as necessary without reference to the simulation source code. The simulation's source code was modified only if the pseudocode was changed or if it was determined that the simulation did not follow the pseudocode. In almost all cases, the algorithm's developer made modifications to the pseudocode without reference to (or even knowledge of) the simulation source code. In all cases in which the pseudocode had to be modified, it was because it did not agree with the mathematical theory on which the algorithm is based.

5. INSTRUMENTATION

The simulation was instrumented in two ways. The first and simplest requires no more than a set of counters that can be reset at various points. These counters determine such statistics as the total number of times all the nodes in the simulation responded to a packet and the total number of messages sent. When the event queue empties (implying that an algorithm has converged), the values of the counters are printed. Some counters are associated with individual nodes or links in the simulation whereas others are associated with all nodes or links.

In addition, the simulation gathers information after each event is processed. At certain points in the main event loop, Drama allows the programer to provide a function that will be called with a calling convention specified by the run-time library. In particular, a copy of the routing tables was analyzed at each step of the simulation thereby allowing us to characterize the routes each algorithm produced while the algorithm was still running. Each copy of every routing table was generated jut before a node's script returned. This user-verified function can perform a series of actions. For each algorithm, just before a node's script returned, the node copied its routing tables into a global area containing the successor for each destination. This new table was then analyzed at each step of the simulation, and which allowed us to characterize the routes each algorithm produced while the algorithm was still running.

6. TEST NETWORKS AND STATISTICS

The simulations were run on several network topologies, the 1988 ARPANET and MILNET topologies, the LOS-NETTOS topology, the DOE-ESNET topology, and the NSFNET-T1-Backbone topology as shown in Figure 1. We chose these topologies to compare the performance of routing algorithms for well-known cases, given that we cannot yet sample a large enough number of networks to make statistically justifiable statements about how an algorithm's performance scales with network parameters.

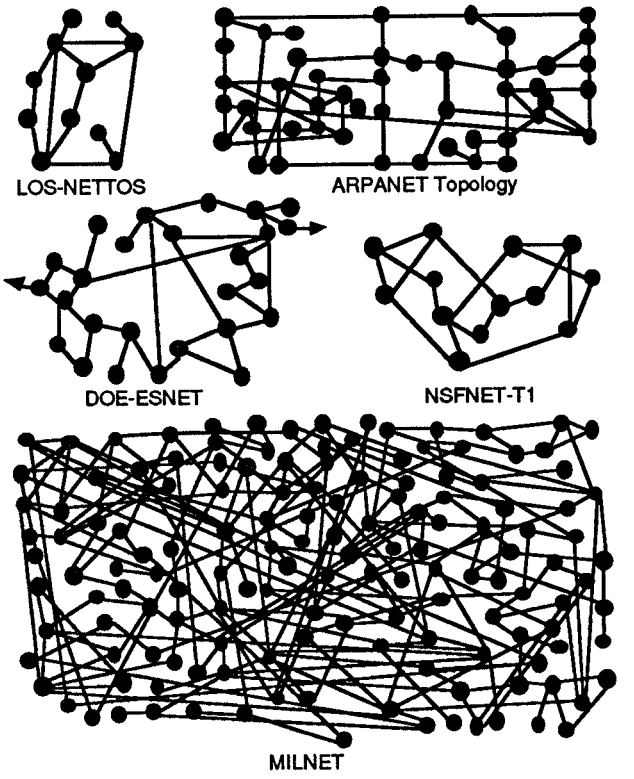

Figure 1. Network topologies

For each network, we generated test cases consisting of all single failures and recoveries for both links and nodes in which the routing algorithms were allowed to converge after each change. We also generated random link-cost changes; links were chosen at random, with link costs chosen from the interval *(0,1]* and with Poisson-distributed interarrival times. The link and node failures and recoveries test runs exhaustively, covering all cases, thus precluding sampling error. For the cases where link costs changed, five independent runs were made and the averages and standard deviations of all quantities measured were determined overall.

Some runs required the use of a random-number generator, either to randomly choose a link, a link cost (from 0 to 1), or a

Poisson-distributed interarrival time. The UNIX library function `random` was used with a state size of 256. Additional functions used this random number generator to produce a double-precision random-number between 0 and 1, and to produce a random integer in the range *[0, N]* where *N* is the number of links in a simulation. We determined that the random numbers produced were uniformly distributed and ran an autocorrelation test to determine that successive changes would not be correlated.

In all cases, nodes were assumed to perform computations in zero time, and links were assumed to provide one time unit of delay. The link models allow link delay and link cost to be set independently. Each unit of time therefore represents a step in which all currently available packets are processed.

Although the choice of input parameters causes the simulation to proceed synchronously, the node models treat each incoming packet asynchronously. Each input event is thus processed by a node independently of other events received during the same simulation step. Such treatment provides a basis for algorithm comparison that is independent of link delay and that accurately reflects the problems caused in any asynchronous distance-vector algorithm when a node reacts to new information from one neighbor using outdated information obtained from another.

7. TOTAL RESPONSE TO CHANGES

After failure or recovery of a node or link, or a change in link cost, each algorithm was allowed to run to convergence. Node failures were modeled as the simultaneous failure of all links attached to that node, starting from a topology in which all links were up. Node recoveries were modeled by bringing the attached links back up simultaneously. Link failures and recoveries behaved similarly, in that at most one link was down at any time. As in the validation runs, nodes ran each algorithm in zero time and link delay was set to unity. For the failure and recovery runs, costs were set to unity; otherwise, they were set to a random number in the interval *(0,1]*. Several quantities were then measured.

- *Events.* The total number of updates (including queries and replies for the DUAL algorithm) and changes in link status processed by nodes.
- *Packets.* The total number of packets transmitted over the network. Each packet may contain multiple updates.
- *Duration.* The total elapsed time it takes for an algorithm to converge.
- *Operations.* The total number of operations performed by each algorithm. The operation count is incremented whenever an event occurs, and whenever the statements within a "for" or "while" loop are executed. (For the Dijkstra-LS algorithm, one step inserts data into a priority queue. The number of operations required to do so is estimated to be $log_2 n$, where n is the length of the queue.)

The results appear in Tables 1–5. Both the mean and the standard deviation of the value distributions are given. There is no sampling error for the results shown in Tables 1–4 because the statistics covered all possible cases. For the rest, errors were determined based on repeated trials (by computing the mean and sample standard deviation for data collected over five trials, each containing approximately 500 points).

In every case, the operation count for the Dijkstra-LS algorithm (replicated at each node) was substantially higher than for the other two algorithms, often more than an order of magnitude higher. For node and link failures, DBF sends significantly more packets than DUAL, except for link failures in NSFNET-T1 Backbone case. The existence of such an exception indicates that we cannot yet compare the average performance of routing algorithms in general, but only with respect to particular topologies. For node recoveries, DUAL sent fewer packets on the average than ILS. The packet count values for DBF varied considerably with respect to the values for DUAL and ILS. For link-cost changes, depending on the network, DBF sent from as many packets to about twice as many packets as the other algorithms, and DUAL sent about as many packets as ILS in some cases and about 35% more in the others.

DBF was by far the worst for mean duration of node failures. For both link and node recoveries, DBF took approximately one time unit less than DUAL, at least partly because of an implementation detail: in DUAL's implementation, when a node recovers, it broadcasts its distance of 0.0; in DBF's implementation, neighboring nodes assume that this distance will be zero because the distance of any destination to itself is zero. The mean duration for node failures for DUAL was 1.7 to 3.1 times as long as for ILS, depending on the network. For link failures, the mean duration ratio of DUAL to ILS varied from about 1.3 to 2.0 and for link changes, from about 1.4 to 1.9.

8. DYNAMIC RESPONSE TO CHANGES

The mean duration in Tables 1–5 is a very crude measure of an algorithm's performance because it ignores the possibility that routing tables may provide some usable routes while the algorithm is still running. This in fact turns out to be the case. To study the dynamics of routing algorithms, we ran an exhaustive series of test cases for all node failures and recoveries, and recorded, at each step of the simulation, the number of nodes that (a) reached each destination, (b) had no successor, (c) could not reach a given destination because of loops, and (d) could not reach a given destination because of a downstream node with no successor for that destination. We also recorded the queue lengths and the number of outstanding messages. Finally, we recorded the number of paths containing a node or a link that is part of a routing loop (possibly for a different destination).

Treating every node change as a separate case, we obtained a statistical characterization of each routing algorithm's performance by computing a distribution as a function of time. For example, the 1988 ARPANET topology has 47 nodes, and one can compute the probability $P(n)$ that n nodes (for $n \in [0,46]$) can reach some arbitrary destination (we do not consider paths from a node to itself, because they do not require a network). When all nodes and links are up, 46 nodes will reach their destination when the routing algorithms converge. When a node fails, it will not be able to reach any other destination, and other nodes cannot reach it. In most cases, 45 nodes will be able to reach an arbitrary destination after some node has failed, except for the cases in which a node failure partitions the remainder of the network. As a result, $P(45)$ after the algorithms converge is slightly less than 1.0. During convergence, $P(45)$ varies with time. In cases where one slice of the distribution function is shown, the other slices did not contain anything of interest—small nonzero probabilities, but with no discernable pattern.

In addition, some of the statistics have been characterized by the probability as a function of time that some condition is true, and by an average value given that it is true. For example, we show the probability that a loop exists and the average number of loops given that at least one loop exists. This is useful because, in

Table 1

ROUTING-ALGORITHM RESPONSE TO A SINGLE NODE FAILURE

Parameter	DUAL		DBF		ILS	
	mean	sdev	mean	sdev	mean	sdev
ARPANET Node-Failure Cases						
Even count	1050	661	6510	1840	218	64.1
Packet count	382	97.5	5690	63.3	212	62.5
Duration	17.8	7.71	47	0	8.66	0.974
Operation count	1320	721	6770	1870	34000	9890
MILNET Node-Failure Cases						
Even count	4500	4070	79300	51400	686	284
Packet count	1480	720	60500	335	680	282
Duration	31.6	21.8	144	0	10.1	1.31
Operation count	5370	4420	80200	51600	492000	204000
LOS-NETTOS Node-Failure Cases						
Even count	73	25.4	273	72.1	31.8	9.61
Packet count	45.5	3.26	217	20.2	26.7	7.2
Duration	6.91	0.996	11	0	4.09	0.514
Operation count	124	50.2	324	89.2	673	195
DOE-ESNET Node-Failure Cases						
Even count	321	273	1950	915	72.2	28.3
Packet count	135	23.7	1380	43.4	67.2	26
Duration	11.1	1.42	26	0	6.27	0.901
Operation count	444	328	2080	961	5080	1980
NSFNET-T1-Backbone Node-Failure Cases						
Even count	176	76.7	472	17.6	65.1	9.89
Packet count	97.2	24.5	443	13.4	59.1	8.8
Duration	12.6	5.58	14	0	5.21	0.558
Operation count	254	89.2	550	28.1	2070	282

Table 2

ROUTING-ALGORITHM RESPONSE TO A SINGLE NODE RECOVERY

Parameter	DUAL		DBF		ILS	
	mean	sdev	mean	sdev	mean	sdev
ARPANET Node-Recovery Cases						
Even count	692	317	823	353	301	83.6
Packet count	207	41.6	242	65	295	82.1
Duration	8.45	0.82	7.57	0.765	9.62	1.35
Operation count	958	394	1090	433	50100	14000
MILNET Node-Recovery Cases						
Even count	2390	1680	2850	1900	854	309
Packet count	644	166	743	259	848	307
Duration	10.2	1.11	9.32	1.11	10.4	1.18
Operation count	3260	2080	3720	2300	625000	227000
LOS-NETTOS Node-Recovery Cases						
Even count	94.3	40.5	118	53.4	49.4	18.6
Packet count	41	12.4	39.7	12.5	44.3	16.2
Duration	4.73	0.445	3.73	0.445	4.09	0.514
Operation count	145	66.5	169	79.3	1520	610
DOE-ESNET Node-Recovery Cases						
Even count	276	173	341	210	102	45
Packet count	97	28.1	106	38.7	97.2	42.6
Duration	6.5	0.5	5.58	0.494	6.42	1.01
Operation count	399	235	464	273	8550	3830
NSFNET-T1-Backbone Node-Recovery Cases						
Even count	153	40.2	186	45.9	104	13.8
Packet count	60.9	9.47	61.8	11.9	98.3	12.7
Duration	4.5	0.5	3.5	0.5	6	0.926
Operation count	231	56.3	264	62.2	4150	597

Table 3

ROUTING-ALGORITHM RESPONSE TO A SINGLE LINK FAILURE

Parameter	DUAL		DBF		ILS	
	mean	sdev	mean	sdev	mean	sdev
ARPANET Link-Failure Cases						
Even count	721	503	649	1300	160	20.9
Packet count	266	94.1	437	1310	158	20.9
Duration	15.1	2.63	9.13	9.55	8.56	0.793
Operation count	813	503	741	1300	25600	3390
MILNET Link-Failure Cases						
Even count	2540	2120	11300	35200	506	90.2
Packet count	767	431	8010	20200	504	90.2
Duration	20.3	10	25.2	45.5	9.92	1.26
Operation count	2830	2120	11600	35200	366000	66000
LOS-NETTOS Link-Failure Cases						
Even count	49.9	18.6	65.7	90.3	29.7	5.87
Packet count	32.6	11.8	58.6	92.8	27.7	5.87
Duration	6.71	1.33	4.29	3.57	4.21	0.558
Operation count	69.9	18.6	85.7	90.3	724	169
DOE-ESNET Link-Failure Cases						
Even count	265	207	552	746	63.1	14.8
Packet count	111	37	432	617	61.1	14.8
Duration	11.1	1.58	10.8	9.58	6.34	0.922
Operation count	315	207	602	746	4750	1220
NSFNET-T1-Backbone Link-Failure Cases						
Even count	91.6	47.8	37.7	21.3	53	2.32
Packet count	53	19.1	22.3	10.6	51	2.32
Duration	6.95	0.785	3.29	0.452	5.43	0.495
Operation count	118	47.8	63.7	21.3	1840	61.6

Table 4

ROUTING-ALGORITHM RESPONSE TO A SINGLE LINK RECOVERY

Parameter	DUAL		DBF		ILS	
	mean	sdev	mean	sdev	mean	sdev
ARPANET Link-Recovery Cases						
Even count	362	176	363	174	163	21.3
Packet count	79.4	26	118	43.9	161	21.3
Duration	7.31	1.19	6.6	1.13	7.74	0.699
Operation count	454	176	455	174	26900	3450
MILNET Link-Recovery Cases						
Even count	1180	704	1200	698	510	90.8
Packet count	232	118	341	168	508	90.8
Duration	8.71	1.67	8.17	1.61	9.36	1.05
Operation count	1460	704	1480	698	371000	66000
LOS-NETTOS Link-Recovery Cases						
Even count	45.7	7.45	46.1	9.8	33.3	6.66
Packet count	17	7.25	19.6	7.53	31.3	6.66
Duration	3.71	0.881	2.93	0.799	3.86	0.35
Operation count	65.7	7.45	66.1	9.8	944	173
DOE-ESNET Link-Recovery Cases						
Even count	160	73.7	166	71.4	66.5	15.7
Packet count	45.4	16.6	63.2	24.1	64.5	15.7
Duration	5.69	0.916	5.12	0.74	5.56	0.496
Operation count	210	73.7	216	71.4	5410	1230
NSFNET-T1-Backbone Link-Recovery Cases						
Even count	67.6	19	65.6	19	57	2.32
Packet count	22	6.28	28.4	11.6	55	2.32
Duration	3.86	0.35	2.9	0.294	4.71	0.452
Operation count	93.6	19	91.6	19	2140	76.8

Table 5

ROUTING-ALGORITHM RESPONSE TO A CHANGE IN LINK COST

Parameter	DUAL		DBF		ILS	
	mean	sdev	mean	sdev	mean	sdev
ARPANET Node-Recovery Cases						
Even count	435 ± 7.7	655 ± 17.3	182 ± 7.5	375 ± 15.8	83.4 ± 0.21	83.9 ± 0.19
Packet count	109 ± 1.0	132 ± 1.26	699 ± 29	1630 ± 83.6	84.4 ± 0.213	84.9 ± 0.19
Duration	14 ± 0.18	5.6 ± 0.205	18.8 ± 0.9	15.8 ± 0.67	7.81 ± 0.010	0.781 ± 0.017
Operation count	481 ± 7.7	687 ± 17	715 ± 29	1650 ± 83	13600 ± 71	13700 ± 68
MILNET Node-Recovery Cases						
Even count	1880 ± 54	3450 ± 69			274 ± 0.15	276 ± 0.18
Packet count	366 ± 5.6	471 ± 6			273 ± 0.15	275 ± 0.18
Duration	18.5 ± 0.32	8.12 ± 0.36			9.49 ± 0.028	1.07 ± 0.011
Operation count	2020 ± 54	3530 ± 70			199000 ± 729	200000 ± 743
LOS-NETTOS Node-Recovery Cases						
Even count	43 ± 2.0	59.4 ± 3.6	39.3 ± 2.5	64.1 ± 6.0	18.5 ± 0.061	18.5 ± 0.051
Packet count	19.2 ± 0.41	23.3 ± 0.61	18.1 ± 1.14	28.1 ± 2.5	17.5 ± 0.061	17.5 ± 0.050
Duration	5.91 ± 0.12	2.64 ± 0.16	4.91 ± 0.23	3 ± 0.22	4.05 ± 0.045	0.458 ± 0.030
Operation count	53 ± 2.1	67 ± 3.5	49.3 ± 2.5	70.7 ± 5.8	479 ± 3.1	480 2.9
DOE-ESNET Node-Recovery Cases						
Even count	150 ± 1.9	229 ± 3.6	202 ± 15	427 ± 45	38.6 ± 0.076	38.7 ± 0.070
Packet count	47.7 ± 0.37	55.7 ± 0.30	75.3 ± 6.0	142 ± 12	37.6 ± 0.076	37.7 ± 0.070
Duration	9.38 ± 0.093	3.57 ± 0.036	11.5 ± 0.44	8.8 ± 0.31	5.83 ± 0.0070	0.612 ± 0.0073
Operation count	175 ± 1.9	246 ± 3.4	227 ± 15	440 ± 446	3010 ± 25	3020 ± 259
NSFNET-T1-Backbone Node-Recovery Cases						
Even count	67.7 ± 1.2	94 ± 3.1	62.4 ± 4.2	107 ± 12	28.5 ± 0.024	28.5 ± 0.023
Packet count	28.4 ± 0.32	35 ± 0.54	25.9 ± 1.7	40.8 ± 3.8	27.5 ± 0.024	27.5 ± 0.023
Duration	6.99 ± 0.15	3.08 ± 0.19	6.12 ± 0.30	3.81 ± 0.30	4.74 ± 0.0094	0.438 ± 0.0049
Operation count	80.7 ± 1.15	104 ± 3.0	75.4 ± 4.16	116 ± 12	1010 ± 1.9	1020 ± 1.9

some cases, the probability of a loop is fairly low. The average number of loops over all cases would be heavily biased by the cases where there are no loops at all, and is not a particularly interesting number.

Figure 2–15 show the transient response of routing the algorithms after a node failure. The remaining graphs (Figures 16–18) show the transient response for a node recovery. All the results shown in these graphs are for the ARPANET topology. Graphs for the other topologies occasionally differed in shape. The most notable difference is that for the MILNET topology, the number of loops showed a prominent spike just before the BF and ILS algorithms terminated. This occured as the probability of loops dropped, suggesting that the spike is related to worst-case behavior.

9. ALGORITHM COMPARISON

Determining which algorithm is best for a particular network really depends on a wide variety of considerations:

- Does responding to a routing packet require a context switch? If so, how long does a context switch take?
- To what extent are CPU time and memory overhead important parameters?
- How much traffic will be routed before the routing algorithm responds. Do routing updates have priority over user traffic?
- Does each outgoing link at a router have its own buffer pool, or is a common pool used for all links?

For example, for high-speed networks, the delay in moving routing packets between neighboring nodes may be large enough that even a transient routing loop can be significant—before the routing algorithm can respond, enough user packets may be in the loop to fill up available buffers.

For the networks simulated, ILS seems to be best in most cases for minimizing the number of routing packets that need to be sent, albeit at the cost of considerably more CPU time compared to DUAL. ILS also runs to completion for node failures in considerably less time than DUAL does. This total elapsed time is somewhat misleading, however. As can be seen in Figures 2, 3, and 8, most of the routes are determined correctly in approximately the same time as for the BFD and ILS.† For node and link recoveries, the corresponding figures are similar for all three algorithms, with running time roughly proportional to network diameter. Figure 2 shows that immediately after a single node failure, fewer nodes have paths to their destination using DUAL than for the other algorithms. This is the consequence of the mechanisms used by DUAL to prevent routing loops for all cases, including those in which multiple node failures occur at about the same time. For a particular application, a tradeoff has to be made to determine whether this behavior is worth the reduction in congestion, which also affects paths to nodes other than the one that failed.

One should also note that, although additional messages are sent by DUAL, compared to ILS's, DUAL is loop free, whereas the ILS is not. Figures 13–15 provide a comparison. At any time after a node failure, ILS is usually loop free, but in some cases it produces a loop. That loop can lead to congestion if enough packets travel along it while it persists. Figure 14 shows the number of paths that will reach a node that is part of a loop for some

† We have similar results for link failures.

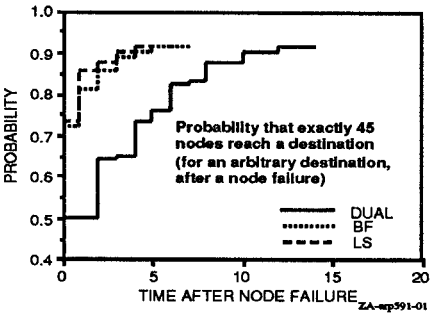

Figure 2. For DUAL, probabilities are lower than for ILS or DFB immediately after the node failure, because DUAL will not use alternative routes to a destination until it can determine that there are no loops.

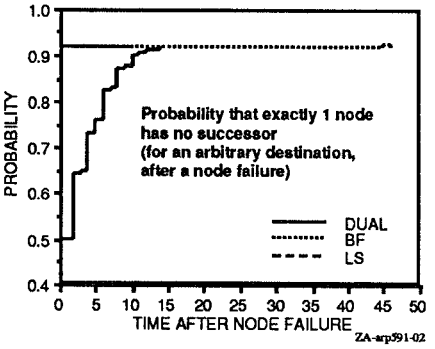

Figure 3. Directly after a node failure, DUAL is more likely than the others to have multiple nodes with no successor as an artifact of the mechanism that produces loop-free routes, reducing the probability that only one node has no successor to a destination.

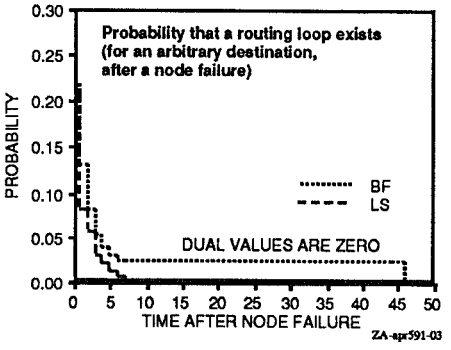

Figure 4. Time values for DBF values extend well beyond those for LSI because of the counting to infinity problem.

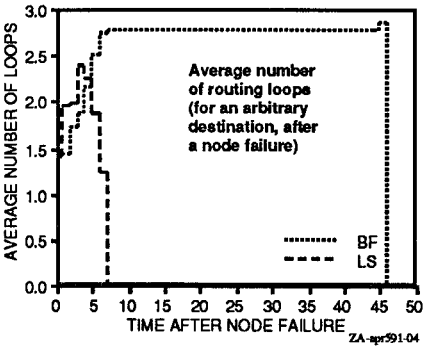

Figure 5. Average number of loops for routes to an arbitrary destination, given that at least one loop exists.

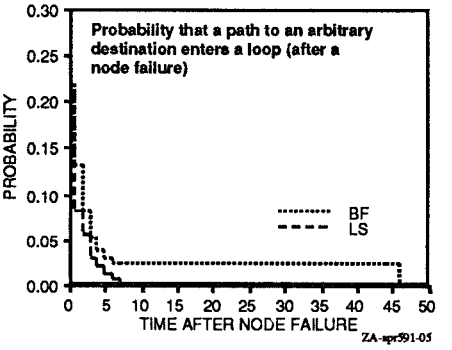

Figure 6. The probability of a DBF loop is low for times between 5 and 45, because a node failure does not cause a loop for all destinations.

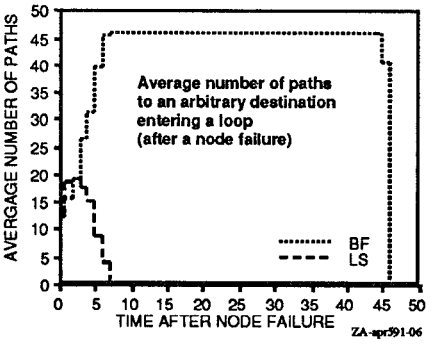

Figure 7. Average number of paths to an arbitrary destination that enter a loop, given that at least one loop exists for that destination.

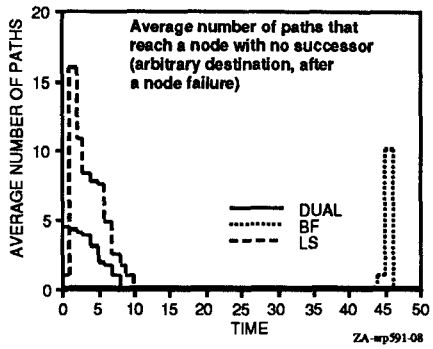

Figure 8. The peaks at times around 45 for DBF occur because near that point, the algorithm has counted to infinity and some nodes will determine that there is no successor before others.

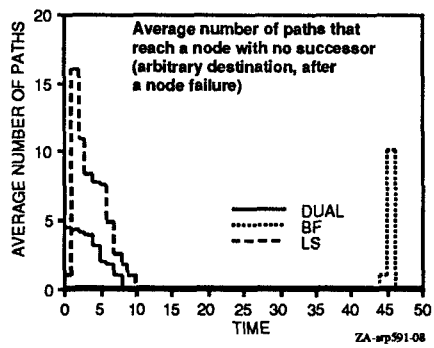

Figure 9. Average number of paths to an arbitrary destination that reaches a node with no successor given that at least one node for that destination has no successor.

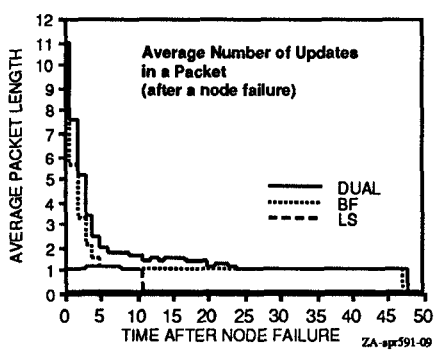

Figure 10. Probability of packets in transit after a node failure.

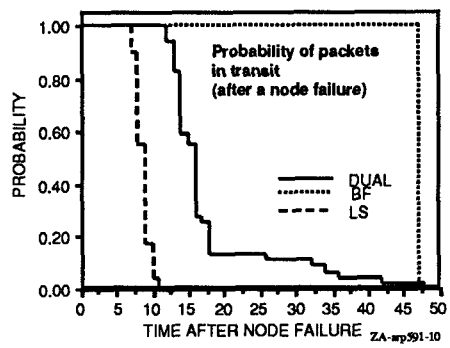

Figure 11. Average number of updates in a packet after a node failure. As time progresses, the number of updates in a packet drop, so the packets get shorter.

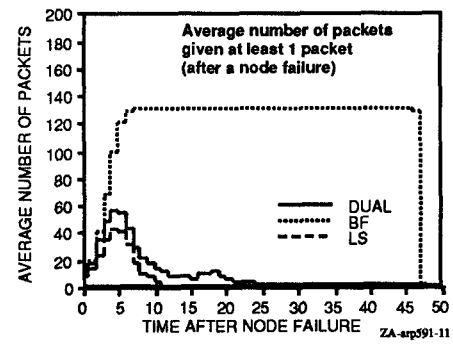

Figure 12. Average number of packets given that at least one packet was sent. DBF algorithm sends more packets than the others for much longer.

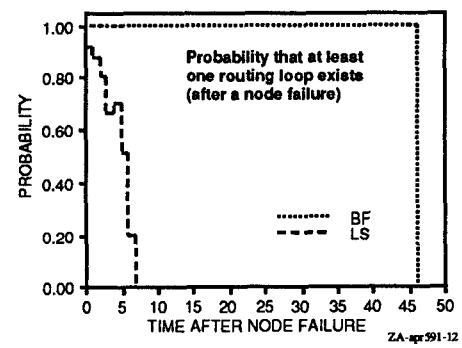

Figure 13. Probability that at least one routing loop exists, regardless of the destination.

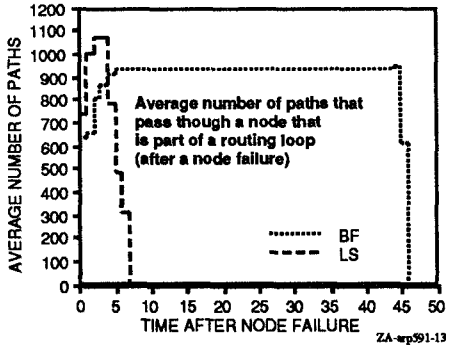

Figure 14. Average number of paths that pass though a node that is part of some loop, including loops for other destinations.

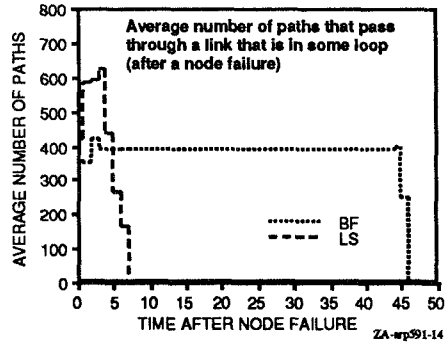

Figure 15. Average number of paths pass though a link in some loop, including links for other destinations.

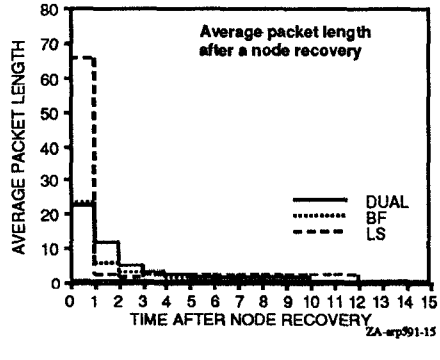

Figure 16. Probability of packets in transit after a node recovery.

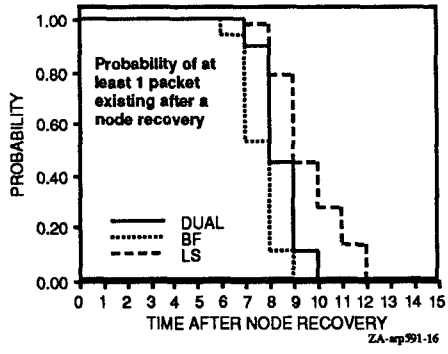

Figure 17. Average number of updates in a packet.

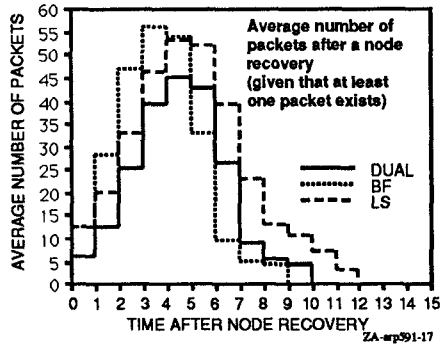

Figure 18. Average number of packets given that at least one packet was sent.

destination, and Figure 15 shows the number of paths that will reach a link that is part of a loop. There are 2162 paths that may exist at any one time (2162 is the number of source-destination pairs, ignoring cases in which the source is the destination), so a sizable fraction of the paths may be affected by the congestion caused by loop. Whether or not congestion will build up fast enough to cause problems is, of course, dependent on many factors, some of which were described above.

In terms of the number of messages sent, DUAL has an interesting property—the length of time that messages are sent may exceed the time needed for the routing tables to converge. This extra time is needed for synchronization and to provide loop free routes when multiple topological changes occur while the algorithm is running. This is evident from Figures 12 and 18. It may be possible to use a heuristic approach to minimize the number of such packets. For example, the use of hold-downs can help minimize the number of packets sent over a network, but may increase an algorithm's time to convergence. Once it is likely that all the routing tables have converged, the use of such mechanisms may increase the time to termination, but not the time the routing tables need to converge.

Finally, we note that DUAL sends shorter messages at later times after a node failure or recovery than after earlier times, as shown in Figures 11 and 17. This is interesting, because it indicates that the longer messages affect only a small number of nodes, because each time unit represents packets that propagate farther; and that a heuristic approach based on the local topology for improving performance can be used to improve its performance.

10. CONCLUSIONS

As expected, DBF performs very poorly after resource failures or link cost increases. The results of this study show that a choice between a DVP based on loop-free distance-vector algorithm and an LSP involve a multifaceted tradeoff that can be driven by a variety of factors, including the performance and type of the operating system used in the routers. Determining the mean or worst-case values for such quantities as the time to convergence, or the total number of packets sent are not sufficient and in some cases (as seen above) can be misleading. The statistical techniques, especially the time series analysis, used in our analysis do, however, provide a way of characterizing the performance of various algorithms, and can be used as a basis for a tradeoff analysis during network design: by changing the weights assigned to various properties, tradeoffs can be made without re-running the simulations. It also appears that a heuristic approach to improving the performance of routing algorithms can be effective—the time behavior of loop-free distance-vector algorithms shows that parameters such as packet size can change as updates propagate, thereby suggesting the possibility of heuristics that can exploit local conditions.

The principal limitations for these results involve the idealized network parameters (unit link delay and negligible processing time at the nodes) together with ignoring multiple node failures. The later assumption is reasonable in practice—even for a 144 node network such as MILNET, the mean time between node failures is much larger than the time it takes for an algorithm to converge, so the probability of multiple failures occurring while an algorithm is responding to a previous failure is negligible. These limitations are not inherent in the simulation and statistical analysis techniques used in this work, and were imposed primarily to constrain the number of parameters that had to be varied so as to reduce processing time for the simulations.

BIBLIOGRAPHY

[BERT-87] D. Bertsekas and R. Gallager, 1987: *Data Networks*. Englewood Cliffs, NF: Prentice-Hall, Inc.

[CEGR-75] T. Cegrell, 1975: "A Routing Procedure for the TIDAS Message-Switching Network," *IEEE Transactions on Communications*, Vol. COM-23, No. 6, pp. 575–585.

[CHEN-89] C. Cheng, 1989: "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect," *ACM Computer Communication Review*, Vol. 19, No. 4, pp. 224–236.

[COLT-89] R. Coltun, 1989: "OSPF: An Internet Routing Protocol," *ConneXions*, Vol. 3, No. 8, pp. 19-25.

[DIJK-59] E.W. Dijkstra, 1959: "A Note on Two Problems in Connection with Graphs," *Numer. Math.*, Vol. 1, pp. 269–271.

[DIJK-80] E.W. Dijkstra and C.S. Scholten, 1980: "Termination Detection for Diffusing Computations," *Information Processing Letters*, Vol. 11, No. 1, pp. 1–4.

[FORD-62] L.R. Ford and D.R. Fulkerson, 1962: *Flows in Networks*. Princeton, NJ: Princeton University Press.

[GARC-89a] J.J. Garcia-Luna-Aceves, 1989: "A Minimum-Hop Routing Algorithm Based on Distributed Information," *Computer Networks and ISDN Systems*, Vol. 16, No. 5, pp. 367-382.

[GARC-89b] J.J. Garcia-Luna-Aceves, 1989: "A Unified Approach for Loop-Free Routing Using Link States or Distance Vectors," *ACM Computer Communication Review*, Vol. 19, No. 4, pp. 212–223.

[HEDR-88] C. Hedrick, 1988: "Routing Information Protocol," RFC 1058, Network Information Center, SRI International, Menlo Park, CA 94025 (June).

[HIND-82] R. Hinden and A. Sheltzer, 1982: "DARPA Internet Gateway," RFC 823, Network Information Center, SRI International, Menlo Park, CA 94025, (September).

[ISO-89] International Standards Organization, 1989: "Intra-Domain IS-IS Routing Protocol," ISO/IEC JTC1/SC6 WG2 N323, (September).

[JAFF-82] J.M. Jaffe and F.M. Moss, 1982: "A Responsive Routing Algorithm for Computer Networks," *IEEE Transactions on Communications*, Vol. COM-30, No. 7, pp. 1758–1762.

[JAFF-86] J.M. Jaffe, A.E. Baratz, and A. Segall, 1986: "Subtle Design Issues in the Implementation of Distributed, Dynamic Routing Algorithms," *Computer Networks and ISDN Systems*, Vol. 12, No. 3, pp. 147-158.

[LOUG-89] K. Lougheed and Y. Rekhter, "A Border Gateway Protocol," 1989: RFC 1105, SRI International, Menlo Park, CA 94025 (June).

[MERL-79] P.M. Merlin and A. Segall, 1979: "A Failsafe Distributed Routing Protocol," *IEEE Transactions on Communication*, Vol. COM-27, No. 9, pp. 1280–1288.

[MCQU-74] J. McQuillan, 1974: "Adaptive Routing Algorithms for Distributed Computer Networks," BBN Rep. 2831, Bolt Beranek and Newman Inc., Cambridge MA (May).

[MCQU-80] J. McQuillan, et al., 1980: "The New Routing Algorithm for the ARPANET," *IEEE Transactions on Communications*, Vol. COM-28, (May).

[MILL-83] D. Mills, 1983: "DCN Local-Network Protocols," RFC 891, Network Information Center, SRI International, Menlo Park, CA 94025 (December).

[MILL-84] D. Mills, 1983: "Exterior Gateway Protocol Formal Specification," RFC 904, Network Information Center, SRI International, Menlo Park, CA 94025, (December).

[NAYL-75] W.E. Naylor 1975: "A Loop-Free Adaptive Routing Algorithm for Packet Switched Networks," *Proceedings of the Fourth Data Communications Symposium*, Quebec City, Canada, pp. 7.9–7.15 (October).

[SCHW-86] M. Schwartz, 1986: *Telecommunication Networks: Protocols, Modeling and Analysis*, Chapter 6, Menlo Park CA: Addison-Wesley Publishing Co.

[SHIN-87] K.G. Shin and M. Chen, 1987: "Performance Analysis of Distributed Routing Strategies Free of Ping-Pong-Type Looping," *IEEE Transactions on Computers*, Vol. COMP-36, No. 2, pp. 129–137.

[STER-80] T.E. Stern, 1980: "An Improved Routing Algorithm for Distributed Computer Networks," *IEEE International Symposium on Circuits and Systems*, Workshop on Large-Scale Networks and Systems, Houston, Texas (April).

[ZAUM-91] W.T. Zaumen, 1991: "Simulations in Drama," Network Information System Center, SRI International, Menlo Park, CA 94025 (January).