Optimal Physical Diversity Algorithms and Survivable Networks

Ramesh Bhandari
AT&T Laboratories, Rm. 2B-410A
Crawfords Corner Road
Holmdel, NJ 07733, USA
Tele. No. (908)949-0693
Fax. No. (908)949-4364
rbhandari@att.com

Abstract

One way to improve the reliability of a network is through physical diversity, i.e., via routing of traffic between a given pair of nodes in the network over two or more physically-disjoint paths such that if a node or a physical link fails on one of the disjoint paths, not all of the traffic is lost. Alternatively, enough spare capacity may be allocated on the individual paths such that the lost traffic due to a node or physical link failure can be routed immediately over the predetermined paths. In this paper, we present optimal algorithms for K-disjoint paths (K≥2) in a graph of vertices (or nodes) and edges (or links). These algorithms are simpler than those given in the past. We discuss how such algorithms can be used in the design of survivable mesh networks based on the digital crossconnect systems (DCS). We also discuss the generation of optimal network topologies which permit K>2 disjoint paths and upon which survivable networks may be modeled.

1. Introduction

As fiber is increasingly deployed in networks, reliability of a network is being called into question more than ever before. This is due to the fact that as more traffic is transported over the high bandwidth fiber network, any span (physical link) cut or node failure results in the loss of a large volume of traffic. One way to increase the reliability of a given network is through physical-diversity, i.e., via routing of traffic between a given pair of nodes in the network over two or more physically-disjoint paths such that if a node or a physical link (span) fails on one of the disjoint paths, not all of the traffic is lost. Alternatively, enough spare capacity may be allocated on the individual paths such that the lost traffic due to a node or physical link failure can be rerouted over the predetermined disjoint paths.

Since most networks are bidirectional, we will represent networks (or graphs) by vertices (or nodes) and edges (or links). Unless otherwise stated, we will assume that multiple edges (two or more links between the same pair of nodes) are absent. Fig. 1 is an example of a bidirectional network of 8 nodes and 13 edges; each edge is the equivalent of two oppositely directed

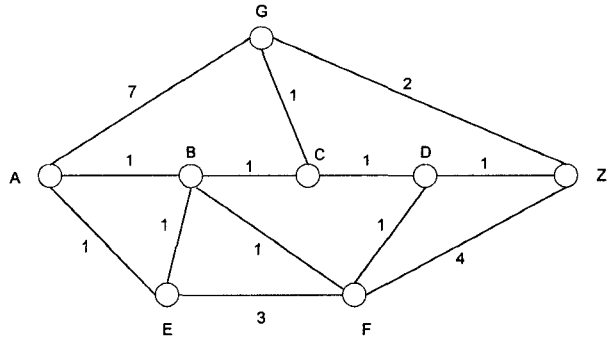

Fig. 1 A bidirectional network; the numbers are the edge lengths.

arcs, each of length equal to the edge length; length here has the general meaning in that it may represent the physical length of the edge (or link) in the network, or may be the cost of using the edge in transmission of data, and so on. A path or route between a pair of vertices is a sequence of arcs connecting them. When searching for diverse routes between a given pair of vertices, it is generally desirable to find the set of paths whose sum is a minimum. For example, if the length of an edge is the length of fiber over that physical link, then the optimal set of paths uses minimal amount of fiber between the pair of nodes. Similarly, if the length of a link in a graph represents cost of provisioning services along that link, then the optimal set of paths represents diverse routes

0-8186-7852-6/97 $10.00

over which the cost of diverse provisioning of services would be a minimum.

The paper is divided into 5 sections. Section 2 reviews the simple (but erroneous) approach of finding physically-disjoint paths and the pitfalls involved in using this approach. Section 3 gives the correct algorithms for shortest (K≥2) edge-/vertex-disjoint paths. These algorithms are novel and simpler than those given in the past [1,2]; furthermore they are easily implementable. Section 4 focuses on the utility of disjoint paths in the design of survivable networks. In particular it is shown that transporting traffic over more than two disjoint paths can sometimes be more economical and efficient than using a disjoint pair of paths. To the author's knowledge the network design approach given here, which involves the use of more than two disjoint paths, has not been considered before. In addition, given the number of nodes in a network, we discuss how optimal terrestrial networks permitting K>2 disjoint paths may be constructed.

2. Simple approach disjoint paths algorithms and shortcomings

The simple approach algorithms consist of finding first the shortest path between the pair of vertices under consideration, and the second shortest path which is disjoint from the first, the third shortest path which is disjoint from the previous two, and so on, depending upon the number of disjoint paths required. Focusing on a pair of disjoint paths, if link-disjointness is desired, all the links of the first shortest path are removed from the graph, and the shortest path algorithm rerun in this reduced graph. Likewise, if the second path is to be vertex-disjoint, then all the links incident on the vertices (except the endpoint vertices) of the first shortest path are removed. These algorithms are illustrated with reference to Fig. 1. Let us suppose a pair of disjoint paths is desired between vertices A and Z. Then the shortest path between A and Z is ABCDZ of length 4. The second shortest path edge-disjoint from the first is AEBFZ of length 7, implying a total length of 11 for the pair of edge-disjoint paths. Similarly, the shortest path vertex-disjoint from path ABCDZ in Fig. 1 is AEFZ=8, implying a total length of 12 for the pair of vertex-disjoint paths. The above simple approach of finding disjoint paths, however, has the following shortcomings:

1. Suboptimality

Note that in Fig. 1 the shortest pair of edge-disjoint paths is actually (ABCGZ, AEBFDZ) with a total length of 10. Similarly, the shortest pair of vertex-disjoint paths is (ABCGZ, AEFDZ) with a total length equal to 11. In effect, the simple approach algorithms when applied to Fig. 1 fail to provide the optimal pairs of disjoint paths between A and Z. In general, such types of algorithms may or may not provide an optimal set of disjoint paths.

2. False alarms about nonexistence of paths when such paths actually exist.

This shortcoming is illustrated via Fig. 2. Assuming ABCZ is the shortest path between the given pair of vertices A and Z, the simple approach algorithm fails to find a vertex-disjoint pair, even though two such pairs (ABFZ, ADCZ) and (ABFZ, ADCEZ) exist. Clearly, any practical implementation of the simple

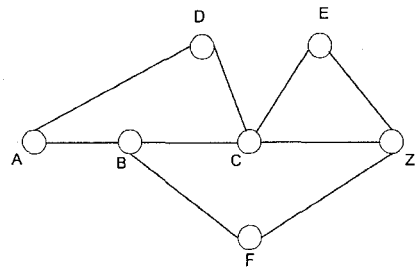

Fig. 2 The simple approach algorithm fails to produce a vertex-disjoint pair of paths between A and Z.

approach algorithm carries with it the risk of generating false alarms about the nonexistence of disjoint paths when such paths actually exist. In other words, automating such an algorithm for diverse provisioning of services in real time should be avoided, since a customer desiring diverse paths may be erroneously informed that such a service cannot be provided to him.

3. Shortest K (≥2) disjoint paths algorithms

Algorithms for shortest K(≥2) disjoint paths have been given in the past [1,2]. However, these algorithms emphasize the use of a special canonic transformation, unnecessarily making the algorithms complicated and hard for a practicing engineer to use them. In this paper, we give simpler versions of the disjoint algorithms, which circumvent the need for the special network transformation. Rather, the algorithms we construct and give below require only a slight modification of the standard Dijkstra algorithm [3,4] for finding the shortest path. This modified Dijkstra is described in Appendix A.

3.1 Edge-disjoint shortest pair of paths algorithm

The algorithm for generating the shortest pair of edge-disjoint paths between a given pair of vertices in a graph can be conveniently given as follows [5]:

1. Run the shortest path algorithm (Appendix A) for the given pair of vertices under consideration (vertices A and Z). Refer to Fig. 1 as an illustration, where A is assumed to be the source vertex and Z the destination vertex.

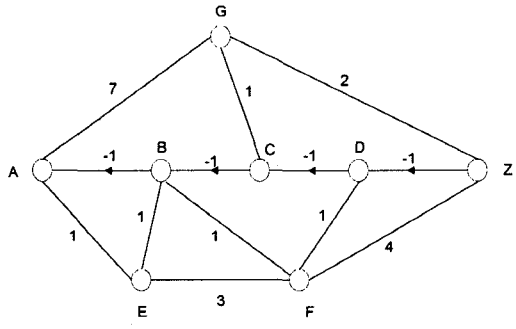

Fig. 3 The modified counterpart of Fig. 1 for the edge-disjoint shortest pair algorithm

2. Replace each edge of the shortest path by a single arc directed towards the source vertex (vertex A in Fig. 1).

3. Make the length of each of the above arcs negative.

4. Run the shortest path algorithm (Appendix A) from vertex A to vertex Z in the modified graph (Fig. 3).

5. Replace the negative arcs in the graph with the original edges (of positive length). Remove overlapping edges of the two paths found above. The desired pair of paths results.

Step 2 above ensures that the shortest path (ABCDZ of length = 4 in Fig. 1) is not reproduced when the shortest path algorithm is run in the modified graph (see Fig. 3). In addition, the arcs directed towards the source vertex permit the interlacing of the shortest path to be found (see Step 4) with the shortest path found in the original graph. Allowing for interlacing and the negativity of the arcs (Step 3) lead to optimality. Step 4 when applied to Fig. 3 yields path AEBFDCGZ of length 1+1+1+1-1+1+2=6 as the shortest path, implying a total length of 4+6=10 for the edge-disjoint path pair. However, the edge-disjoint paths themselves are obtained by erasing the interlacing part, which is DC. This step leads to (ABCGZ, AEBFDZ) as the shortest pair (of length = 10) in the original graph of Fig. 1.

3.2 Vertex-disjoint shortest pair of paths algorithm

The algorithms for vertex-disjointness also require two runs of the shortest path algorithm (Appendix A) in a modified graph, but with a crucial difference which we illustrate with reference to Figs. 1 and 4. For vertex-disjointness, all possible paths between A and Z that intersect with the shortest path ABCDZ must be excluded from consideration during the search for the second shortest path. For example, paths AEBFZ and AEBFDCGZ in Fig. 3, while candidate paths in the edge-disjoint algorithm, are not valid paths for vertex-disjointness. Exclusion of such paths is achieved via vertex-splitting along the first shortest path found. Invocation of the standard vertex-splitting technique [6] yields the following algorithm for the shortest pair of vertex-disjoint paths:

1. For the given pair of vertices under consideration, find the shortest path using the shortest path algorithm in Appendix A. For illustration, refer to the network graph of Fig.1.

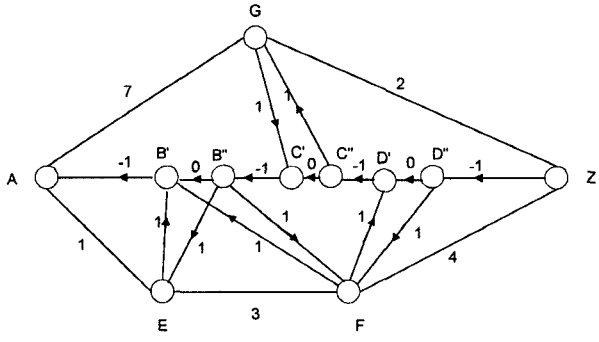

Fig. 4 The modified counterpart of Fig. 1 for the vertex-disjoint shortest pair algorithm

2. Replace each edge on the shortest path by an arc directed towards the source vertex, and make its length negative.

3. Split each vertex on the shortest path into two colocated subvertices joined by an arc of length zero. Direct this arc towards the source vertex. Replace external edges connected to vertices on the shortest path by two oppositely directed arcs of the same and original length, and connected to the two subvertices, as shown in Fig. 4; external arcs terminate on the primed subvertices, while they originate from the double-primed subvertices.

4. Run the shortest path algorithm (Appendix A) in the modified graph of Fig. 4.

5. Remove the zero length arcs; coalesce the subvertices into their parent vertices. Replace the single arcs of the shortest path with their original edges (of positive length). Remove overlapping edges of the two paths found above to obtain the shortest pair of vertex-disjoint paths.

Starting with Fig. 1, Steps 1-3 yield the graph of Fig. 4. The shortest path obtained is AEFD'C"GZ of length 1+3+1-1+1+2 = 7, which implies a total length of 4 + 7 = 11 for the shortest pair of vertex-disjoint paths. Finally, Step 5 yields (ABCGZ, AEFDZ) as the shortest pair of vertex disjoint paths (of length = 11) in the original graph of Fig. 1.

Note that the operation of vertex-splitting excluded paths AEBFZ and AEBFDCGZ from being considered, as one desired. Since the constraint of vertex-disjointness is more stringent than the constraint for edge-disjointness, one expects that the length of the shortest vertex-disjoint pair ≥ length of the shortest edge-disjoint pair; this fact is borne out in the example of Fig. 1.

It is also worthwhile to point out that in the algorithm above, although each vertex is split into two subvertices in accordance with the vertex-splitting rule [6] (which is also followed in Refs. [1,2]), splitting vertices of degree 3 is in fact redundant; one only needs to split vertices of degree 4 or more.

3.3. Shortest K-disjoint paths (K>2)

These can be obtained iteratively by using the shortest path algorithm given in Appendix A.

3.3.1 Vertex-disjoint paths. We discuss the generation of K (>2) vertex-disjoint paths from knowledge of the (K-1) disjoint paths. Suppose in Fig. 5a the shortest pair of vertex-disjoint paths between vertices A and Z obtained by the algorithm of Sec. 3.2 is (ABCZ, ADEZ). To obtain a triplet, we modify the given graph

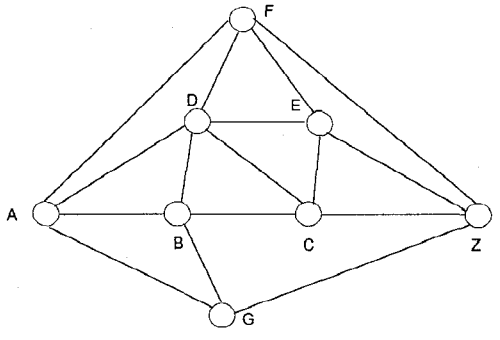

Fig. 5a (ABCZ, ADEZ) are the shortest pair of vertex-disjoint paths.

by splitting the vertices on the given shortest pair of paths, and connecting them to the other vertices in the graph by the same rules as in the construction of the shortest pair algorithm. Fig. 5b shows the modified

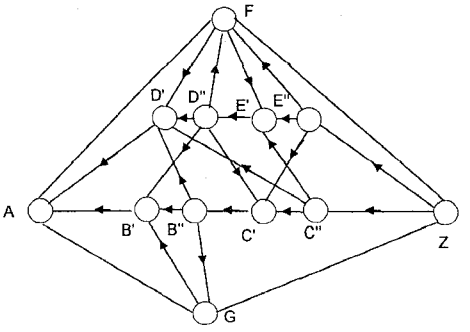

Fig. 5b Modified graph for obtaining the shortest triplet of paths in Fig. 5a

graph in which the shortest path algorithm is run from A to Z. If any interlacing takes place the common parts should be erased, and the split vertices coalesced back into their parent vertices. The final result is the shortest triplet of vertex-disjoint paths. In a similar way further iterations are performed to obtain more than three vertex-disjoint paths of minimum total length in a given network graph, provided such paths exist.

3.3.2 Edge-disjoint paths For K-disjoint paths the procedure is the same, except that vertex-splitting is not performed.

4. Survivable mesh networks

One of the major problems in today's network is making a given network survivable such that the traffic affected by a link or node failure is restored easily and quickly in the network. Restoration requires spare capacity allocation in the network. Spare capacity should be so allocated that not only is the cost minimal, but also restoration can be effected quickly so that the time of traffic outage is reduced as much as possible. In what follows, we assume that the networks are high-connectivity (mesh) networks which employ digital cross-connect systems (DCS) at nodes.

Although several algorithms exist on spare capacity allocation, they are generally based on rerouting affected traffic around the failed link. A shortest path algorithm is normally used for determining alternate routes and spare capacity assignment on each link [7]. Also in such network systems, where the traffic is routed around the failed link, the time to effect restoration may be increased due to the need to determine the location of the failed link or node in the network.

In Section 4.1, we describe an approach where the total capacity (working capacity and spare capacity) on each link is determined from the set of disjoint paths available between each pair of nodes in the network. The

traffic demand and the needed spare capacity are split over the disjoint paths between a given pair of nodes. Thus, if a node or link fails along one of the disjoint paths, the failure is detected at the endpoints of the affected traffic, and the affected traffic is switched (or crossconnected) to the available spare capacity on the remaining paths in the disjoint set.

Mesh networks, being highly connected, carry the possibility of link crossings. Fig. 6 is an example of a network where a subset of physical links cross each other. This crossing point may be a regenerator location, for example. Even though three disjoint paths exist between every pair of nodes, this is not a good network design, since damage or failure at the central point would result in the loss of half of the links in the network. Clearly, such network design should be avoided. In Section 4.2, we point out a methodology for optimal network topologies embedded in a plane, i.e., network configurations in a plane that permit a given number of disjoint paths, using the minimum number of

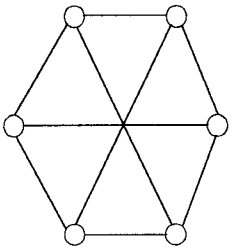

Fig. 6 A 6-node (3-connected) network; 50% of its link intersect at a single point.

links with no link crossing. In particular, we illustrate the K=3 case.

4.1 Capacity allocation

Let us suppose we are concerned with the problem of a survivable mesh network design with the requirement of 100% restoration of affected traffic when a node or link fails in the network. We assume that traffic demands between each pair of nodes in the network is given. Then the model we propose for capacity allocation (working as well as spare) in the network is illustrated by consideration of traffic demand between a single pair of vertices and its working and spare capacity allocation over the same set of disjoint paths:

a. If T denotes the traffic demand between the pair of vertices A and Z in Fig. 7, then the traffic routed over each of the M disjoint paths is T/M.

b. If a link or a node fails on any of the above paths, the affected traffic (=T/M for the node pair under consideration) is switched to the remaining M-1 paths by digital crossconnect systems (DCS) located at the end points A and Z. This implies a spare capacity reservation of T/(M(M-1)) on each of the M paths.

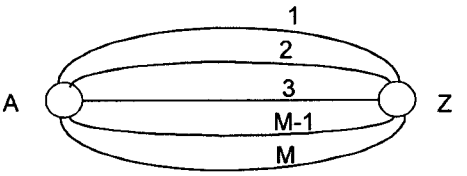

Fig. 7 Nodes A and Z between which traffic T flows over M disjoint paths.

4.1.1 Fast restoration criteria for M. Greater the number M, smaller the amount of traffic that needs to be switched between the node pair under consideration. Thus, if the DCS crossconnect time depends upon the amount of traffic to be switched, it would be desirable to have a large value of M to reduce restoration time. Clearly, $1<M\leq K$, where K is the maximum number of disjoint paths between A and Z in Fig. 7. The minimum value of M is two. However, if the restoration is to take place fast (in a time less than some threshold time τ) so that the customers do not notice the span or node failure, the value of M may have to be greater than 2. For example, if S_n denotes the restoration time (detection+signaling+crossconnect time) for an amount of traffic T/n, n (integer) >2 and S_n satisfies the inequality: $S_n < \tau < S_{n-1}$, then min(M), or the minimum number of desired disjoint paths =n; equivalently, $M\geq n$.

4.1.2 Cost constraint criteria for M. Given M set by the fast restoration criterion described above, can its value be further improved from cost considerations? For illustration, we shall assume that the measure of cost is the total capacity (working+spare) assigned to the path times the physical length of the path. Capacity determines the number of fibers needed along the path, and the length of the path is equal to the length of the fiber laid along the path. Referring to Fig. 7 and the discussion earlier, the total capacity needed on each path for 100% restoration is

$$C_{tot} = T/M + T/(M(M-1)) = T/(M-1) \qquad (1)$$

The total traffic-miles, $T_{tot}(M)$, for the traffic between the given pair of nodes in Fig. 7 is given by

$$T_{tot}(M) = C_{tot} \sum_{i=1}^{M} l_i = T/(M-1) \sum_{i=1}^{M} l_i , \qquad (2)$$

where l_i denotes the length of path i. Now let us suppose M+p paths ($p\geq 1$) also exist. Then Eq. (2) implies

$$T_{tot}(M+p)/T_{tot}(M) = (M-1)/(M+p-1) \sum_{i=1}^{M+p} l'_i / \sum_{i=1}^{M} l_i , \quad (3)$$

where l_i' denotes the length of path i within the group of M+p disjoint paths. Eq. (3) leads to

$$T_{tot}(M+p)/T_{tot}(M) \leq 1, \quad (4)$$

if

$$\sum_{i=1}^{M+p} l'_i / \sum_{i=1}^{M} l_i \leq (M+p-1)/(M-1) \quad (5)$$

Thus, if Eq. (5) is satisfied, it would be cost-effective to route the traffic over (M+p) disjoint paths instead of M disjoint paths. For example, if the ratio of the triplet pathlength to the doublet pathlength (case of M=2, p=1), is less than 2, it would be preferable to route the traffic over the triplet. Similarly, if, for M=2 and p=2, the left hand side of Eq. 5 is less than 3, the quartet is preferable to the doublet. In general, assigning capacity over M+p paths, instead of M paths, offers the following advantages:

1. More traffic can be sent between the same pair of nodes within the bound permitted by the fast restoration criterion; alternatively, for a given amount of traffic, less traffic needs to be switched when a link or node on the path fails.
2. Being greater in number, paths in the M+p set offer greater potential than paths in the M set for sharing spare capacity with the paths corresponding to other pairs of nodes.

When the above process is repeated for every pair of nodes in the network, the working capacity on each

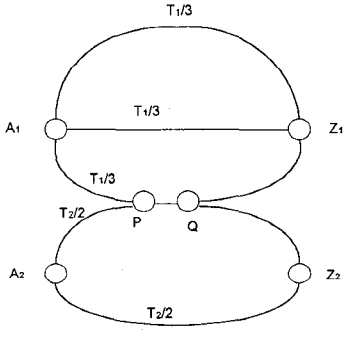

Fig. 8 Spare capacity sharing on common link PQ

individual link is obtained by summing up the working capacities corresponding to different node pairs to which the link is common (see Fig. 8). Also spare capacity allocation on each individual link involves sharing, as depicted in Fig. 8. For example, while the total working capacity on link PQ is $T_1/3 + T_2/2$, the spare capacity is $\max(T_1/6, T_2/2)$.

4.2 Network topologies for disjoint paths

Here we ask the question: given N nodes from which to construct a network, what should be the minimal topology for a K-connected network? A network that permits K-disjoint paths between every pair of nodes is called K-connected. By minimal topology is meant a topology involving the least number of links and no crossing of links. We impose the condition of no link crossing because most practical networks, being terrestrial, essentially lie in a plane, and link intersection points imply added vulnerability which should be eliminated to increase the survivability of networks (see Fig. 6). In essence, we look for planar K-connected graphs with minimum number of links.

K-connectedness implies that every node is at least K-valent, i.e., of degree K. The converse that K-valency implies K-connectedness is not necessarily true (see Fig. 9). Notwithstanding, to construct a K-connected network

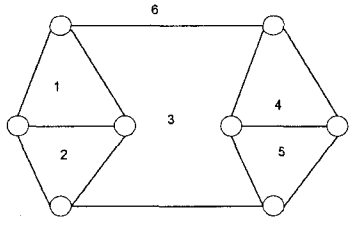

Fig. 9 A trivalent graph that is not 3-connected

with the minimum number of links, we impose the condition that each node of the graph is of degree K. Each node being K-valent leads to the minimum number of links

$$E_{min} = NK/2, \quad (6a)$$

assuming the product NK is even; the factor of 1/2 is due to the fact that each link being common to a pair of nodes contributes two to the degree sum of the nodes; if both N and K are odd,

$$E_{min} = (NK+1)/2. \quad (6b)$$

Furthermore, since the total number of links in a graph of N nodes cannot exceed N(N-1)/2 (no multiple edges assumed as before), we obtain the inequality

$$N \geq K+1 \quad (7)$$

for a K-connected network.

We now invoke the Euler Theorem [4], which states that the following equation must be satisfied by a planar graph:

$$E = N + F - 2 \qquad (8)$$

where E, N, and F are the number of links, nodes, and faces, respectively in the graph. Faces are regions, shown numbered in the example of Fig. 9. Face 6 is deemed an infinite region, and is counted in Eq. (8). Now if the planar K-connected graphs are to possess the minimum number of links given by Eqs. (6a-b), then Eq. (8) becomes

$$F = (K - 2)\, N/2 + 2, \qquad (9a)$$

$$F = (K - 2)\, N/2 + 5/2 \quad (N \text{ and } K \text{ each odd}) \qquad (9b)$$

In what follows, we discuss the optimal network topologies for K=2 and 3.

1. **K=2.** From Eq. (9a), we see that F=2; the corresponding network topology is a ring passing through each node of the network. Between every pair of nodes two disjoint paths exist. The number of links is N (Eq. 6a), where N is the number of nodes. The minimum number of nodes allowed on the ring (from Eq. 7) = K+1 =3 (the case of 2 nodes on a ring (N=2) corresponds to a 2-node graph with multiple edges, and is correctly disallowed by our equations).

2. **K=3** The minimum number of nodes for 3-connectedness=3+1=4 from Eq. (7), and for this minimum the minimum number of links is 3 x 4/2=6 from Eq. (6a). Also F=N/2+2=4. These conditions for N=4 are satisfied by the 4-node network in Fig. 10. There are three disjoint paths for every pair of nodes. As we shall see below, it is the building block for configurations for higher N; we denote it by 4' hereafter. We first assume that N is even. Then E_{min} = 3N/2, and F=N/2+2. From these expressions, we see that for every increment of 2 in N

i. the number of faces increases by 1.

ii. the number of links increases by 3.

These requirements can only be satisfied if the two new nodes are placed on two different links of the given graph, and joined by a new (nonintersecting) link. This operation of introducing two new nodes and a connecting link we call 'link insertion' (there are only two distinct ways of performing 'link insertion' in the 4' network of Fig. 10,

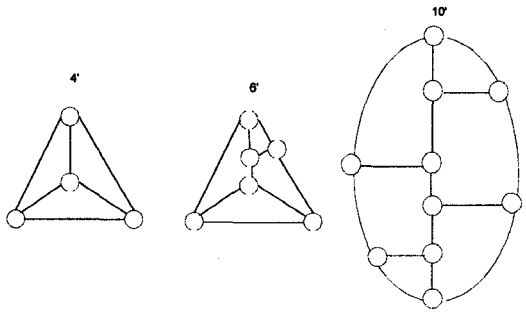

Fig. 10 Single ring topologies (denoted by N') for 3-connected planar graphs.

and these lead to the 6-node networks of Fig. 10 and Fig. 11). Clearly, a trivalent graph remains trivalent after 'link insertion'. In fact, a trivalent 3-connected graph of N nodes transforms into a trivalent 3-connected graph of N+2 nodes upon 'link insertion'. This result follows from consideration of (3-dimensional) polytope graphs [8]. Thus, starting with 4', which is the smallest trivalent 3-connected planar graph, optimal planar network topologies for N=6,8,--- can be generated by the repeated operation of 'link insertion'. We start by pointing out two types of network topologies, which we consider to be basic:

i. Single Ring

ii. Double Ring

Single Ring Network Topology: Arrange the given N nodes into a trivalent tree (each node of degree 3 or 1), and pass a ring through the univalent nodes. We illustrate the topology generation for N=4,6, and 10 in Fig. 10. We denote the generated topologies by N', where N is the number of nodes in the network.

Double Ring Topology: This topology consists of two concentric rings, each containing N/2 nodes. Each node belonging to the external ring is connected to a

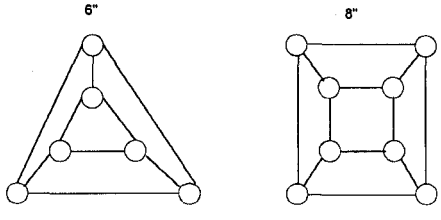

Fig. 11 Basic double ring topology (denoted by N") for N=6 and N=8

corresponding one on the internal ring. This topology leads to a series, starting with N=6, and is illustrated in

Fig. 11. It may be deemed the equivalent of superposition of an N''' network on a null ring (ring with no nodes). See Fig. 12.

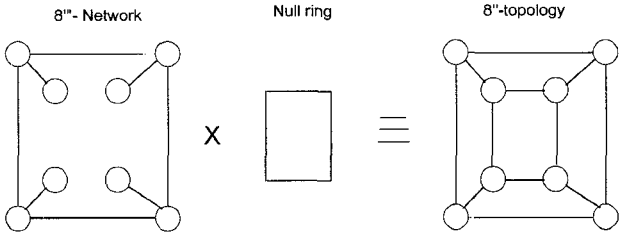

Fig. 12 The N''-topology as superposition of N''' network (with N nodes) and the null ring (with no nodes).

Hybrid Concentric Ring Topology

This 3-connected network topology is obtained by superposing the two fundamental topologies described above. For example, a network topology for a N=10 network may be obtained by applying the 6'''-topology on the 4'-topology, as illustrated in Fig. 13. Fig. 13 also

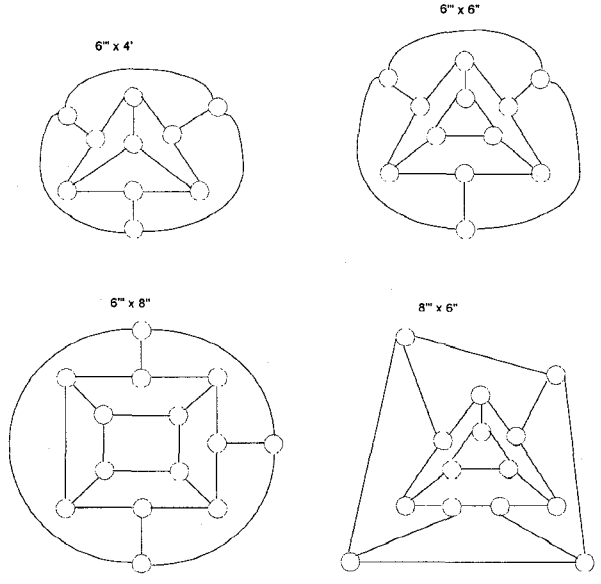

Fig. 13 Example of hybrid concentric ring topologies for N = 10, 12, and 14.

shows the cases: 6'''x6", 6'''x8", and 8'''x6" corresponding to N=12, 14, and 14 nodes, respectively. In general, given N, several network topologies may be permitted. For example, consider N/6= p + remainder, where p is an integer, and remainder = 2 or 4. If the remainder is 4, the network topology may be considered a product of p 6'''-topologies and one 4'-topology (6'''x6'''x....x6'''(p times) x4'). Similarly, if the remainder is 2, the 2 may be combined with one of the 6''',s to form an 8", resulting, for example, in the superposition of p-1 6'''-topologies and one 8"-topology. The order of 8 in the series is important, since each different ordering yields a different network topology for the N-node network (see the N=14 cases in Fig. 13). As a final example, we consider the case of N=20 network, and enumerate the various network topologies possible: 6'''x6'''x8", 6'''x6'''x8', 6'''x8'''x6", 8'''x6'''x6", 8'''x6'''x6', 8'''x8'''x4', 10'''x10', 8'''x12", 12'''x8", 10'''x10", 14'''x6", 14'''x6', 6'''x14", 16'''x4', 20", etc. Note that the primed (Fig. 10 type) and the double-primed (Fig. 11 type) always occur on the extreme right. We close the discussion above with the comment that more network topologies are permissible than have been considered so far. For example, basic series generation in Fig. 12 can be generalized to a superposition of N''' and a ring containing q chords (q≥0) (see Fig. 14); for q>0, we

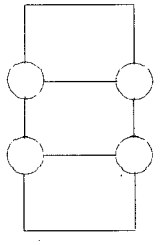

Fig. 14 Ring with two chords

denote the ensuing network topology by N''_q. Classifying it as a basic topology, it may be combined with the previously considered basic topologies to obtain more hybrid topologies.

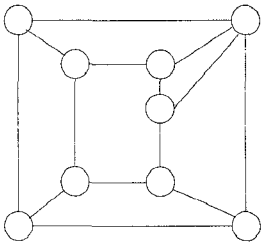

Fig. 15 N'' = 9 topology

Hitherto, the discussion has been confined to N even. When N is odd, it implies that the graph has one more node and two more links than the graph for N-1 (even) nodes (see Eqs. (6a-b)). The transformation to the N odd case from the N- 1 even case can be made by inserting a node on one of the links of the N-1 (even) topology, and connecting that node to an existing node chosen such

that the drawn link does not intersect with any existing link. The degree of the existing node changes to 4. See Fig. 15 for an illustration of one of the ways of obtaining N=9 topology from an existing N=8 topology.

5. Summary

We have presented algorithms for finding the shortest set of K-disjoint paths between a given pair of vertices in a network-graph. These algorithms are easily implementable for use in the design of survivable networks. In this paper, we have considered a model which allows for the division of traffic between a given pair of nodes in the network over more than two disjoint paths such that by suitable capacity allocation on the links the network becomes restorable 100%. We have shown that sending traffic over more than two disjoint paths can become economical and efficient in digital-crossconnect (DCS)-based networks. This approach of sending traffic over more than two paths in the design of survivable networks appears not to have been considered before. In addition, in this paper we have considered a methodology for generating network topologies that permit K>2 disjoint paths, and have illustrated it for the K=3 case. Knowledge of such topologies permits a network designer to design the links (or facility layout) for a survivable (K-connected) network, starting from the location of nodes.

References

1. J.W. Suurballe, "Disjoint Paths in a Network", Networks, 4 (1974) 125-145.
2. J.W. Suurballe and R.E. Tarjan, " A Quick Method for Finding Shortest Pairs of Disjoint Paths", Networks, 14 (1984) 325-336.
3. E.W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", Numer. Math. 1 (1959) 269-271.
4. M. Gondran and M. Minoux, *Graphs and Algorithms*, John Wiley (1984).
5. Proofs of the algorithms are not been given here due to brevity constraints.
6. L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton, University Press (1962).
7. T. Wu, *Fiber Network Service Survivability*, Artech House (1992).
8. B. Grunbaum, *Convex Polytopes*, Interscience Publishers (1967)

APPENDIX A

Modified Dijkstra Algorithm for Shortest Path from A to Z

The algorithm given below is a slight variant of the original Dijkstra algorithm [3,4]. It is different (in Step 3 below) in that it scans all the neighbors of the vertex selected (or "permanently" labeled) in Step 2.

Let d(i) denote the distance of vertex i from starting vertex A. Let P(i) denote its predecessor.

1. Start with
   $d(A)=0$, $d(i)=l(A,i)$, if $i \varepsilon \Gamma_A$,
   $= \infty$, otherwise.
   ($\Gamma_i \equiv$ set of first neighbor vertices of vertex i,
   $l(i,j) =$ length of arc from vertex i to vertex j)
   $P(i)=A \ \forall \ i \varepsilon \Gamma_A$.
   Set $\bar{S} = \Gamma_A$

2. Find $j \varepsilon \bar{S}$ such that $d(j)=\min d(i)$, $i \varepsilon \bar{S}$.
   Set $\bar{S} = \bar{S} - \{j\}$
   If j = Z (the terminal vertex), END;
   otherwise, go to 3.

3. $\forall \ i \varepsilon \Gamma_j$ if $d(j)+l(j,i) < d(i)$, set
   $d(i)=d(j)+l(j,i)$, $P(i)=j$ and $\bar{S} = \bar{S} \cup \{i\}$;
   go to 2.

After initialized in step 1, the algorithm alternates between Steps 2 and 3. In each iteration, a vertex with least pathlength is selected from the set: $\bar{S}$. The algorithm searches by making one move at a time, and terminates when the vertex selected from the set $\bar{S}$ is Z.

In the original Dijkstra algorithm, when a vertex with the least path length is selected from the list of tentatively labeled vertices, the selected vertex is said to have been labeled "permanently", i.e, the shortest path length to that selected vertex from the given origin (the source vertex A) has been found. No further scanning from any other vertex in the graph can update the label of this vertex. In our application, because of the presence of negative arcs in the modified graph (see Step 3 of Section 3.1), rescanning can update the label of the previously selected (or "permanently" labeled) vertex. The algorithm given above permits such rescanning.