NETWORK DESIGN: ALGORITHMS AND EXAMPLES

INTRODUCTION

An interconnection network connects various sources of information using a set of point-to-point links. A link is a connection using a copper wire or an optical fiber, or it may be wireless. The nodes are autonomous data sources and can request to transfer any amount of information to any other node. Figure 1 shows an example network consisting of four nodes. Node A has a link connected to node B and a link connected to node C. Node B is connected to nodes A and D. Nodes C and D are connected to nodes A and B, respectively. If node C desires to send some information to node B, it sends it to node A, which in turn routes it to node B. Node A thus acts as an intermediate node. The *capacity* of a node is the amount of information it can transmit (also called *source capacity)* or receive (also called *sink capacity).* The capacity of a link is the amount of information that can be transferred over the link in one unit of time.

The network design deals with the interconnection of various nodes and how to transmit information from one node to another. Network architecture and design both have multiple meanings. The most commonly used interpretation relates to the decisions one needs to make to design a network. The four most important aspects of network architecture and design are described below.

Network Topology

A topology defines how nodes are interconnected. For example, the topology of the NSF network is shown in Fig. 2. Most network topologies are hierarchical in nature. The design involves developing the structure of the hierarchy, structures of nodes at each level, and detailed designs of the nodes. It also involves assigning link and node capacities to transport the desired traffic. A hierarchical topology is depicted in Fig. 3. We will be studying the decisioin-making process and related algorithms and examples in detail in this artice.

A network node is placed in a hierarchical fashion in such a way that it is "close" enough to several data sources. The closeness is described in terms of suitable performance metrics such as physical distance and cost of connection. A network node serves as a service point for all data sources

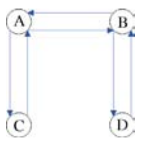


Figure 1. A four-node network.

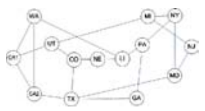


Figure 2. Topology of the NSF network.

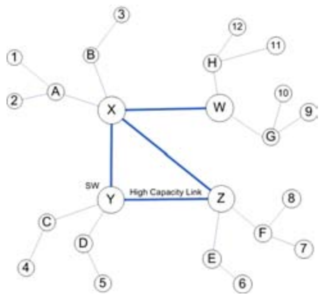


Figure 3. A hierarchical network.

connected to it. Such a node is called a "gateway" as it connects data sources to network nodes. Nodes A, B, C, are such nodes. Gateways connect to data sources, such as nodes 1, 2, ... 12 at the next lower level in the hierarchy and to routers or switches such as nodes X, Y, etc. at the next higher level. Switches and routers route information to other switches and routers on way to other gateways from where the data are delivered to destinations. There may be more levels in the hierarchy.

Node and link placement, and their capacities, in a network topology depend on the desired or required traffic flow that is defined by the traffic characteristics. This problem is well-studied, and more information can be found in References (1–7). In principle, ideal locations for both network nodes and links may be specified using algorithms (possibly complex) that would optimize network design using performance metrics of interest. In practice, these placements are also governed by factors such as existing network, ease of operation, and convenience of management, which are not always easy to accommodate in the design process.

Transmission Technology

Physical layer transmission technologies describe the characteristics of physical medium. These technologies involve signal processing techniques, modulation and demodulation techniques, coding and decoding of information, multiplexing and demultiplexing techniques employed to enhance the utilization of each link, and issues related to these techniques. Physical medium can be a wire, such as copper link, coaxial link, optical fiber, or wireless link using microwave or radio frequencies. The signals being transmitted over the physical medium can be modulated and demodulated using amplitude modulation, frequency modulation, or phase modulation where the information being transmitted modifies the shape of the waveform being transmitted. Figure 4 demonstrates examples of modulation techniques. Multiplexing techniques such as *time division, frequency division*, and *code division* multiplexing techniques are used to mix and transmit information from various sources on a single link. In time division multiplexing, each source is given a fraction of time in a given interval, called *a frame.* In frequency and code division multiplexing, frequencies and bandwidth available on a channel are effectively partitioned so that all sources can use parts of the capacity of the channel simultaneously. Figure 5 shows different multiplexing techniques.

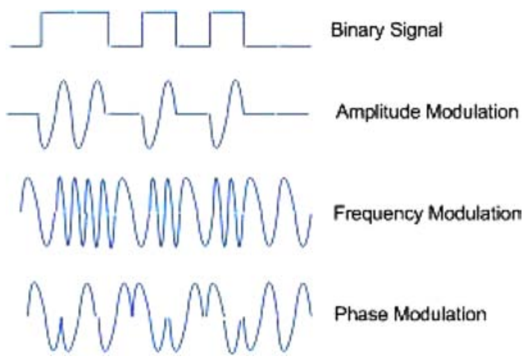


Figure 4. Different modulation techniques.

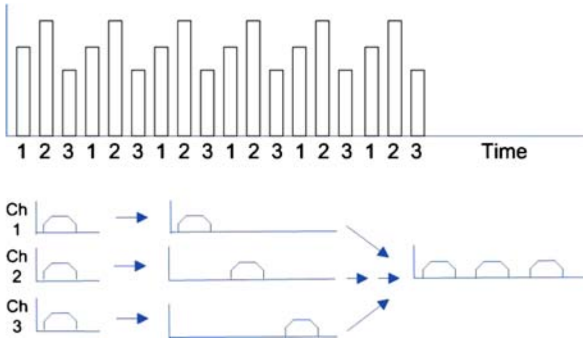


Figure 5. Time and frequency multiplexing.

Traffic Control and Network Management Techniques

This aspect of network architecture involves the control of the switching technology, flow control algorithms for smooth flow of offered loads to the network, control messages flow to set up paths, connection requests and response protocols, collection of data on actual usage, fault detection and management algorithms, and effective resource utilization algorithms.

Connection requests between nodes in a network are realized by employing a routing algorithm. Routing algorithms are designed to use existing network capacity and switching methodology. Either *circuit switching* or *packet switching* can be employed for data transfer in a communication network.

In a circuit-switched network, a complete, dedicated path from a source to a destination is established through the network before communication begins. Dedicated physical resources are reserved for the communication to take place. A signal is sent from the source to the destination node, through intermediate nodes, requesting a connection. An acknowledgment signal is sent back from the destination to the source accepting or rejecting the request. A request is accepted only if all nodes on the path have required resources available and they can reserve these resources statically. If no free path exists from end to end, the traffic is blocked and has to wait for transmission. Path establishment may take substantial time. Once the path is established, information is transmitted freely from the source node to the destination node using the path. The sender and receiver may use any format for data transfer and bit rate subject to the constraints of the physical channel. When the transfer is completed, the path is removed.

In a packet-switched transmission, no physical path is established in advance. Instead, when a source has information to transmit, it assembles it in a packet. A packet consists of data to be sent and a header. The header contains the source and destination addresses, possibly intermediate node addresses through which the packet must be routed, and some error correcting codes to check the correctness of the information. The packets are forwarded from node to node, one hop at a time. The packets are queued in buffers at the intermediate nodes along a route between a source and a destination, traveling from node to node, releasing links, and switching elements immediately after using them. A packet is received by a node, checked for correctness, A and retransmitted, if required. During the transmission, store and forward operation at each node increases the overhead and time delay of data packets.

Most networks use packet switching for smaller messages. In a packet-switched network, it is possible that the path may change from packet to packet between the same source-destination pair. Circuit switching is used when a source/destination has a substantial amount of information to transmit. The main advantage of circuit switching is that the links on a path are always available for communication. The only delay is the propagation delay. However, if no information transfer takes place for sometime during the exislence of a connection, the capacity on the path is wasted. To reduce this waste, it is possible to statistically multiplex the path. A path is established in advance between a source and a destination that is used by every packet from that source to that destination. However, actual transmission occurs as in a packet-switched network. Such a path is called a *virtual path*, as opposed to a physical path that is Established in a circuit-switched network. Such a service, in which a path is established in advance, also called a *connection-oriented service*. A pure packet-switched service is called a *connectionless* service. To set up a path, the network control sends messages to nodes on the path to request a connection. If all nodes on the path from a source node to a destination node agree, then the path is established. Flow control algorithms are employed to control the actual flow of information so that each node on the path is not overwhelmed with the information. If a fault occurs in a node on the path, then the path is established again. In case of a connectionless service, data packets are just sent to the next node along the path. If a faulty node is encountered on the path, the routing is changed on-the-fly. The Internet datagram service uses a pure packet switching protocol, whereas the telephone system, for the most part, uses circuit switching. Networks employing asynchronous transfer mode (ATM) (that are currently being developed) use the concept of virtual path for ATM cell transmission. An ATM cell is a small, 53-byte-long packet that includes 48 bytes of actual data and 5 bytes of control information. In this case, a path is established that is used by all cells, but actual transmission is cell by cell from point to point similar to a packet-switched network.

Cost

The cost of the network is viewed differently by different people. The cost includes parameters such as installation of links and nodes, including the cost of the facility to house the nodes and lay the links (copper or fiber). Laying out links is very expensive and includes buying/leasing land, digging land, laying out conduits, cost of cables, wires, or optical fibers, end-interfaces, buffering, processing hardware at each end of a link, and management of links. Additional operational and maintenance costs are needed to keep the hardware up and running, replacing faulty components and cables, and managing the resources.

For a network service provider, the cost consists of laying out and operating the network. On the other hand, the users or consumers of network resources do not concern themselves with these costs. The costs they account for are the cost quoted to them by the network providers in terms of tariffs for different *quality of service (QoS)* at different times. These tariffs are usage sensitive and depend on the volume of data being transported, time of day (morning, evening, or night), priority of transmission, tolerable delay and loss of data, and several such factors. These factors together are called QoS parameters. To provide and guarantee a specific quality of service, the network service provider has to dedicate some network resources such as bandwidth on individual links, buffer spaces at various nodes on the path, time slots for transmission of specific data, and alternative resources in case of a failure for that service. The cost of these resources forms the basis for tariffs. The development of a cost model for a link is a difficult problem. Often good approximations and simplification of cost structures are used by the network service providers to keep the complexity of the network design and service tariffs under control. In our designs, we consider both models (actual physical network cost model and consumer network cost model) of cost in our designs.

APPROACHES TO NETWORK DESIGN

If a new network is being designed from scratch with no existing capacity, well-defined traffic requirements (traffic intensities), and full freedom in selecting network components, then the designers can make the best possible decisions by balancing the cost and QoS requirements, such as throughput, delay, and other performance measures. However, more often than not, most real designs are incremental; that is, the resources are added or upgraded over the existing capabilities as required by the new demands. The network really evolves with the needs and, in general, is in response to the new requirements. This restriction restricts the optimality in design as the existing design governs the final output.

Inputs for network designs are based on the best estimates of the anticipated traffic between various sources and destinations. Such data are available in the form of a traffic matrix. Many networks are designed using current and additional anticipated needs and certain rules of thumb in an incremental fashion. The decisions are based on the experience of the designer. It is possible to make serious mistakes as part of a new design. For example, when the information transmission is from point to point as in packet switching, intermediate nodes store and forward the incoming information. By not providing enough buffer space or control for incoming traffic streams, losses may be excessive and/or delays may exceed the acceptable limits. A loss may or may not be tolerable. For example, in a voice communicatioin, a small loss may almost go unnoticed, but loss of even a byte may not be tolerable in a computer file transfer application. For a voice or real-time video communication, any significant delay may mean that the information is no more relevant at the destination.

The design process could be manual or automated using exact or heuristic-based algorithms. An automatic design process can avoid such serious design flaws in the network. Unfortunately, most known properties and optimization techniques, relate to networks that are designed new and not incrementally. Heuristic algorithms are used as part of the Automated design process to incorporate design principles used in manual algorithms. One of the most used heuristicis is the use of a *greedy* algorithm.

Sometimes a greedy algorithm may find an optimal solution. A greedy algorithm selects a feature that seems to be of immediate benefit. Consider a situation in which several nodes communicate with each other. Providing a direct link between two nodes that have a maximum amount of traffic flowing between them is a greedy approach. This may have other effects later on in the algorithm. Similarly, incorporating and using the cheapest link in a network is also a good design practice. However, this may have serious cost implications at a later stage in the design and a greedy algorithm may fail to account for them. A greedy algorithm may not always yield the best result, but nonetheless it is the most used heuristic algorithm.

To fully understand the network design process and algorithms necessary in network design, we first develop a graph model of the network. Graph models capture the exact behavior of a network and simplify the task of analysis.

GRAPH MODEL FOR NETWORK

A network is represented by a *graph* $G = (V, E)$,where V is a finite set of elements called *nodes* or *vertice* and E is a set of unordered pairs of nodes called *edges* or *arcs* (8). This graph is *undirected*. A *directed* graph is also defined similarly except that the arcs or edges are ordered pairs. For both directed and undirected graphs, an arc or an edge from a node i to a node j is represented using the notation (i, j). Examples of five-node directed and undirected graphs are shown in Fig. 6. In an undirected graph, an edge (i,j) can carry data traffic in both directions (i.e., from node i to node j and from node j to node i) Whereas in a directed graph, the traffic is only carried from node i to node j.

Graph Representations

A graph is stored as either an adjacency matrix or an incidence matrix as shown in Fig. 7. For a graph with N nodes, an $N \times N$ 0-1 matrix stores the link information in the adjacency matrix. The element (i, j) is a 1 if node i has a link to node j. An incidence matrix, on the other hand, is an $N \times M$ matrix where M is the number of links numbered

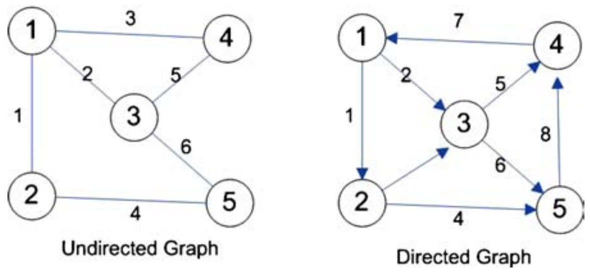
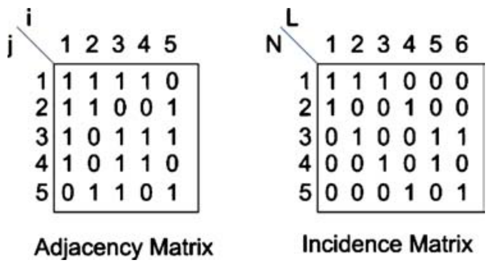


Figure 6. A directed and an undirected graph.



$$\begin{array}{c|ccccc} j \backslash i & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 1 & 1 & 1 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 \\ 3 & 1 & 0 & 1 & 1 & 1 \\ 4 & 1 & 0 & 1 & 1 & 0 \\ 5 & 0 & 1 & 1 & 0 & 1 \end{array}$$

Adjacency Matrix

$$\begin{array}{c|cccccc} N \backslash L & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 1 & 1 \\ 4 & 0 & 0 & 1 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

Incidence Matrix

Figure 7. Matrix representation for graph of Fig. 6(a).

from 0 to $M - 1$. The element *(i, j)* stores the information whether link *j* is incident on node *i* or not. Thus, the incidence matrix carries information about exactly what links are incident on a node. If a graph has more than one link from a node to another node, the incidence matrix will be able to carry this information exactly, whereas the adjacency matrix will require additional information to store the number of links.

The following terms associated with a graph are used throughout this article:

1. The *degree* of a node is the number of links incident on a node. In the case of a directed graph, we count both the number of incoming links, or *in-degree,* and the number of outgoing links, or *out-degree,* of a node. For example, in Fig. 6a, node 1 has a degree of 3, whereas in Fig. 6b, node 1 has an in-degree of one and out-degree of two.
2. A *walk* in a graph $G = (V,E)$ is a sequence of nodes $w = [v_1, v_2, \ldots v_k]$, $k > 1$, such that $(j, j+1) \in E$, $j = 1, 2, \ldots k-1$. A walk is *closed* if $k > 1$ and $v_1 = v_k$.
3. A walk without any repeated nodes in it is called a *path*.
4. A closed walk without any repeated intermediate nodes is called a *cycle.* An acyclic network does not contain any cycles as shown in Fig. 8.
5. A node *s* is said to be *connected* to node *t* if node *s* has a path to node *t* in the graph. This path is called an *(s, t)* path.
6. The length of a path is the number of links on the path.
7. An *(s, t)* path is called the *shortest path* if there is no other path of length shorter than the length of the given path.

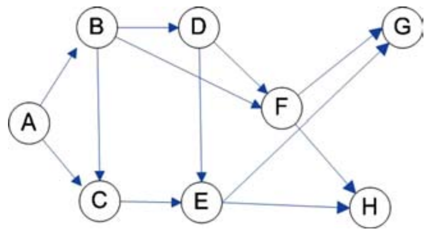


Figure 8. An example of an acyclic graph.

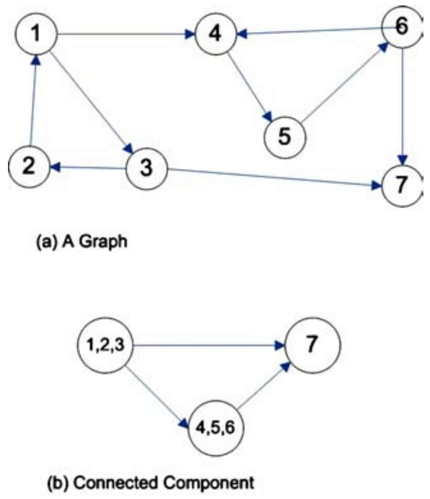


Figure 9. A graph and its connected components.

8. $\delta(i,j)$ denotes the length of a shortest path between nodes i and j. In a network, it is a measure of the maximum communication delay.
9. The *diameter* (the longest shortest path between any pair of nodes) of a graph is given by Max $\{\delta(i, j)\ \forall i, j \in V\}$.
10. A graph is said to be connected if a path exists between any of nodes, s and t.
11. A graph is said to be *strongly connected* if $\forall i, j, \in V$ a path exists, from node i to node j.
12. A *connected component* of a graph *(V,E)* is a subgraph $G' = (V', E')$, $V' \subseteq V$ and $E' \subseteq E$ with every $(i, j) \in E'$, $i, j \in V'$ such that G' is strongly connected.

    An example of connected components of a graph is shown in Fig. 9 where nodes 1, 2, and 3 form one connected component and nodes 4, 5, and 6 form another. Node 7 is a component by itself.
13. The *node connectivity* of a graph is the minimum number of nodes that should be removed from the graph in order to partition it into two disjoint subgraphs, that is, the number of node-disjoint paths. The node connectivity is a measure of the reliability of communication.

GRAPH ALGORITHMS

Once we select the graph model of a network, various algorithms can be used to efficiently design and analyze a network architecture. Some of the most fundamental algorithms among them are finding trees in a graph with minimum cost (where cost is defined appropriately) or finding a minimum spanning tree; visiting nodes of a tree in a specific order; finding connected components of a graph; finding shortest paths from a node to another node, from a node to all nodes, and from all nodes to all nodes in a distributed or centralized fashion; and assigning flows on various links for a given traffic matrix.

In the following discussion, we describe some useful graph algorithms that are important in network design. Recall that N represents the number of nodes and M represents number of links in the graph.

Shortest Path Routing

Shortest path routing, as the name suggests, finds a path of the shortest length in the network from a source to a destination (9–13). This path may be computed statically for the given graph regardless of resources being used (or assuming that all resources are available to set up that path). In that case, if at a given moment all resources on that path are in use, then the request to set a path between the given pair is blocked. On the other hand, the path may be computed for the graph of available resources. This reduced graph will be obtained after removing all links and nodes that may be busy at the time of computing from the original graph. In either case, computation of the shortest path is based on the following concepts.

Suppose for a given graph G, each arc *(i, j)* is also assigned a weight (or length) denoted by $a_{i,j}$. We are interested in finding out a path of the shortest length from a given source node to a given destination node. The path will not have any node repeated. This type of problem is fundamental in graphs, and in particular networks, as we may be interested in searching for a path from a source to a destination that is least expensive in terms of traversal. The weight or length of an arc may represent, the actual cost of traveling on that edge. The cost may be in terms of delay, dollars, or any other metric of importance.

One key ideas in computing the shortest path is that of dynamic programming. It is also based on the principle of optimality. For the shortest path computation, it has been shown that, if all edge weights are positive, then an undirected graph can be treated as a directed graph by replacing each undirected *(i, j)* by two directed edges *(i, j)* and *(j, i)*. With negative edge weights, this transformation introduces cycles of negative weights and the shortest path may go through the cycle as often as necessary to bring the total path lengths to zero or negative. Thus, it is not desirable. In the following, we will assume that all edge weights are positive. With that assumption, the shortest path can be computed using the following formulation.

Bellman's Equations. To compute the shortest path from source s to destination t, it turns out that we end up computing the shortest path from the source node s to all destinations (9). Let a_{ij} be the weight of edge *(i, j)* if the edge exists. Otherwise, it is ∞. Let u_j be the weight of the shortest path from origin s to node j. For simplicity we assume that the nodes are numbered from 1 to n and the source node is node 1. We can always renumber the nodes. It is clear that $u_1 = 0$. Let node k be the last node on the shortest path from node 1 to node j. Then we can say that $u_j = u_k + a_{kj}$, which also implies that the path from node 1 to node k with path length U_k must also be the shortest path from node 1 to node k. Otherwise, the path we selected is not the shortest path. This Concept is from the "principle of optimality." Now, we only have a finite number of choices for k. Bellman's equations use this principle concept to search for shorter paths to other nodes by using the known shortest path to node k and edge weights of direct links from node k to other nodes for all such ks. The equations state that

$$u_1 = 0$$

and

$$u_j = \min_{k \neq j}\{u_k + a_{kj}\} \quad j = 2, 3, \cdots, N$$

Using these equations, we can find a shortest path to a node as follows. First, find a node k with edge *(k,j)* such that $u_j = u_k + a_{kj}$. Then find an arc *(l,k)* such that $u_k = u_l + a_{lk}$, and continue in this fashion. Eventually, we would reach node 1. Unfortunately, Bellman's equations do not lead to a solution directly.

Shortest Path in Acyclic Network. In an acyclic network, as shown in Fig. 8, it is easy to use Bellman's equations to find a shortest path. The nodes in such a network can be renumbered in such a fashion that an edge *(i, j)* exists if and only if $i < j$. In this case we can rewrite Bellman's equations as

$$u_1 = 0$$

and

$$u_j = \min_{k < j}\{u_k + a_{kj}\} \quad j = 2, 3, \cdots, N$$

These equations can then be solved as u_1 is known, u_2 only depends on u_1, u_3 only depends on u_1 and u_2, and so on. The complexity of this problem is $O(N^2)$.

Dijkstra Method. For cyclic graphs, we need another method given by Dijkstra (11). This method is applicable to a graph for which edge weights are positive. This algorithm starts with labeling nodes in stages. At each stage of computation, some labels are designated permanent and others remain tentative. A permanent label on a node represents the true length of the shortest path from that node. After including the new labeled nodes, distances to all other nodes are computed again.

Let d_{ij} denote the distance from node i to node j. Let i be the source node. Then d_{ii} is set to zero and d_{ij}, $i \neq j$ is set to a large value if j is not a neighbor of i. Otherwise, it is set equal to the weight of the direct link a_{ij}. Next, the algorithm finds a node j with minimum d and labels it *permanent*. It then uses it to improve distances to other nodes by computing

$$d_{ik} \leftarrow \min(d_{ik}, d_{ij} + a_{jk})$$

At each stage in the process, the value of d_{ik} represents the best known shortest distance from i to k. Using these labels of the nodes, the algorithm then marks another unlabeled

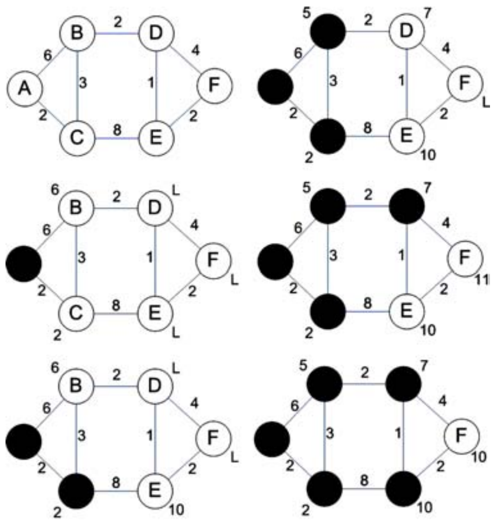


Figure 10. Example of execution of Dijkstra shortest path algorithm.

node with the mimimum value of d_{ik} as permanently labeled. The same computation is carried out again. As all edge weights are positive, in the next iteration, none of the marked nodes can have any smaller value.

An example of the execution of the algorithm is shown in Fig. 10. Node A is the source node. A dark node is a permanently labeled node. At each step, one node is marked labeled and the value associated with a node is its shortest distance from the source thus far with L being a large value. The algorithm terminates in $N — 1$ steps.

Shortest Paths Between All Pairs of Nodes

Now, suppose we want to compute the shortest path between all pairs of nodes. This process may be necessary as communication may occur between any pairs of nodes. It is desirable to use the shortest path as this reduces the requirements for network resources. Sometimes using the shortest path may cause congestion as has been shown by many researchers. For example, suppose that the network graph is such that it can be partitioned into two parts, A and B, and the two parts are only connected by two links, one with a low-weight link and the other with a high-weight link as shown in Fig. 11. All communication between the two halves will use the low-weight link and the other link remains unused. The second link should not have been included in the design, but if it exists, then its use will reduce the congestion on the low-weight link. The shortest path routing algorithm does not use the seconolink at all.

Coming back to the all-to-all communication problem, we can compute paths from every node to every other node. Thus, we need to solve the problem N times. Alternatively, we may use an integrated procedure developed separately, which may be more advantageous. We investigate the latter approach next. Let u_{ij} denote the length of the shortest path from node i to node j, and let $u_{ij}{}^{m}$ be the shortest path such that the path contains no more than m edges. It is clear that $u_{ij}{}^{N}$ will be U_{ij}, the length or the shortest path

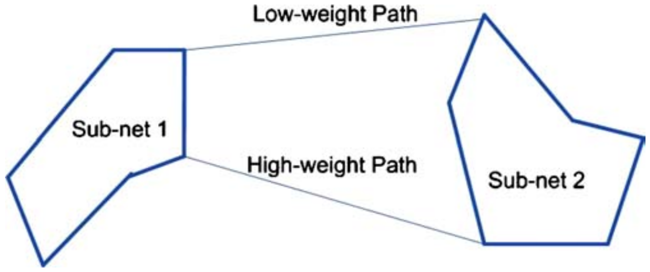


Figure 11. Two parts of a network connected by only two links.

from node i to node j. Also,

$$u_{ii} = 0$$

and

$$u^{0}_{i\,j} = \infty \quad i \neq j$$

and

$$u^{m+1}_{i\,j} = \min_{k}(u^{m}_{ik} + a_{k\,j})$$

The last equation computes the shortest path lengths for the paths that contain up to $m + 1$ edges given that we know the shortest path lengths for paths that contain up to m edges. This calculation seems to be equivalent to matrix multiplication $C = A \times B$, where element c_{ij} is computed using

$$c_{i\,j} = \sum_{k} a_{ik} b_{k\,j}$$

We modify the computation of c_{ij} as

$$c_{i\,j} = \min_{k}\{a_{ik} + b_{k\,j}\}$$

by replacing multiplication by addition and summation by minimum function. We know that $A = (a_{i\,j})$ is the matrix of arc lengths, and we let U^0 be the identity matrix; then $U^0 \times A = A$. Let $U^{m+1} = U^m \times A$. Then $U^{N-1} = A^{N-1}$ gives us the desired shorten path length matrix. It turns out that this type of matrix multiplication is also associative. Thus, we can compute $A^{2k} = A^k \times A^k$, and once $2k > n - 1$, we have U^{N-1}. A single matrix multiplication has $O(N^3)$ complexity, and we need to perform log N matrix multiplications. Therefore, the overall complexity is $O(N^3 \log N)$. This process is more complex than Dijkstra's algorithm but in practice may run faster. An example computation for the five-node graph in Fig. 6a for a given edge weight matrix A is given next:

$$A = \begin{pmatrix} 0 & 100 & 40 & 30 & \infty \\ 100 & 0 & \infty & \infty & 20 \\ 40 & \infty & 0 & 20 & 30 \\ 30 & \infty & 20 & 0 & \infty \\ \infty & 20 & 30 & \infty & 0 \end{pmatrix} \quad A^2 = \begin{pmatrix} 0 & 100 & 40 & 30 & 70 \\ 100 & 0 & 50 & 130 & 20 \\ 40 & 50 & 0 & 20 & 30 \\ 30 & 130 & 20 & 0 & 50 \\ 70 & 20 & 30 & 50 & 0 \end{pmatrix} \quad A^4 = \begin{pmatrix} 0 & 90 & 40 & 30 & 70 \\ 90 & 0 & 50 & 70 & 20 \\ 40 & 50 & 0 & 20 & 30 \\ 30 & 70 & 20 & 0 & 50 \\ 70 & 20 & 30 & 50 & 0 \end{pmatrix}$$

Floyd–Warshall method

Another method to compute the shortest paths between all node pairs is from Floyd and Warshall with a computational complexity of $O(N^3)$. In this method, $u^m_{i\,j}$ defines the length of the shortest path from node i to j such that it does not pass through nodes numbered greater than $m - 1$ except nodes i and j. Then

$$u^1_{i\,j} = a_{i\,j}$$

and

$$u^{m+1}_{i\,j} = \min\{u^m_{i\,j}, u^m_{im} + u^m_{m\,j}\}$$

$u^{N+1}_{i\,j}$ is the shortest path length matrix. Also, $u^{m+1}_{i\,j} = 0$ for all i and for all m.

This procedure has $N(N-1)(N-2)$ equations, each of which can be solved by using $N(N-1)(N-2)$ the additions and $N(N-1)(N-2)$ comparisons. This order of complexity is the same as that for Bellman's method (also known as the Bellman–Ford method as it was independently discovered by two researchers), which yields the shortest path only from a single origin. The Dijkstra method can also be applied N times, once from each source node, to compute the same shortest path length matrix. This process takes only $N(N-1)/2$ additions for each pass, for a total of $N^2(N-1)/2$ additions, but again housekeeping functions in Dijkstra's method make it noncompetitive.

The computation in the Floyd–Warshall method proceeds with $u^1 = A$ and U^{m+1} is obtained from U^m by using row m and column m in U^m to revise the remaining elements. That is, u_{ij} is compared with $u_{im} + u_{m\,j}$ and is replaced if the latter is smaller. Thus, the computation can be performed in place and is demonstrated in the following for the graph in Fig. 6a.

$$A^0 = \begin{pmatrix} 0 & 100 & 40 & 30 & \infty \\ 100 & 0 & \infty & \infty & 20 \\ 40 & \infty & 0 & 20 & 30 \\ 30 & \infty & 20 & 0 & \infty \\ \infty & 20 & 30 & \infty & 0 \end{pmatrix} \quad A^1 = \begin{pmatrix} 0 & 100 & 40 & 30 & \infty \\ 100 & 0 & 140 & 130 & 20 \\ 40 & 140 & 0 & 20 & 30 \\ 30 & 130 & 20 & 0 & \infty \\ \infty & 20 & 30 & \infty & 0 \end{pmatrix} \quad A^2 = \begin{pmatrix} 0 & 100 & 40 & 30 & 120 \\ 100 & 0 & 140 & 130 & 20 \\ 40 & 140 & 0 & 20 & 30 \\ 30 & 130 & 20 & 0 & \infty \\ 120 & 20 & 30 & \infty & 0 \end{pmatrix}$$

$$A^3 = \begin{pmatrix} 0 & 100 & 40 & 30 & 70 \\ 100 & 0 & 140 & 130 & 20 \\ 40 & 140 & 0 & 20 & 30 \\ 30 & 130 & 20 & 0 & 50 \\ 70 & 20 & 30 & 50 & 0 \end{pmatrix} \quad A^4 = \begin{pmatrix} 0 & 100 & 40 & 30 & 70 \\ 100 & 0 & 140 & 130 & 20 \\ 40 & 140 & 0 & 20 & 30 \\ 30 & 130 & 20 & 0 & 50 \\ 70 & 20 & 30 & 50 & 0 \end{pmatrix} \quad A^5 = \begin{pmatrix} 0 & 90 & 40 & 30 & 70 \\ 90 & 0 & 50 & 70 & 20 \\ 40 & 50 & 0 & 20 & 30 \\ 30 & 70 & 20 & 0 & 50 \\ 70 & 20 & 30 & 50 & 0 \end{pmatrix}$$

Multiple Shortest Paths

Many times it is useful to be able to compute additional shortest paths between a node pair, which may be longer than the first shortest path but still short in case the first shortest path is not available. The first path may be congested or may have a failed link or a node. The problem can be constrained by specific requirements such as allowing or not allowing repeated nodes and links or specific nodes and/or links. Specific methods exist to compute alternative shortest paths for all cases (see reference 14). One specific case with respect to fault tolerance is nonavailability of a node or a link. Such a path can be computed by removing the specific node or link in the original graph (removal of a node also removes all associated links) and then using the same shortest path algorithm. In another scenario, we may want another path that is mutually exclusive of the first path. In that case, all nodes and links have to be removed from the original graph before computing another shortest path. The algorithm to be used in these cases is the same as already stated.

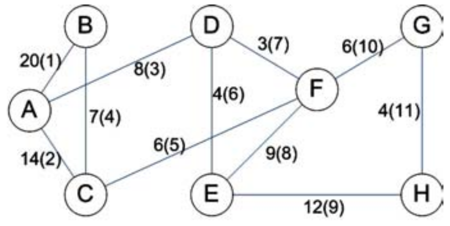
Minimum Spanning Tree (MST)

The minimum spanning tree is the "best" tree one can identify in a given graph with edge weights. Recall that edge weights represent some "cost" of communicating on that edge. The cost may be delay or expense in terms of real dollars to use the link.

The MST problem is to find a set of edges with a total minimum cost so that the nodes in the graph remain connected. A greedy algorithm can be used to find this set of edges, called MSTE. The algorithm starts with one edge with minimum weight. Then it finds an edge "e," the best candidate that has not yet been considered and adds it if it is feasible. An edge can only be added to this set if it does not create a cycle in the graph with the same set of nodes as the original graph and set of edges MSTE. MSTE is complete when it contains $N - 1$ edges in an N node graph. It is known that a greedy algorithm indeed finds an MSTE.

Several algorithms are available to find an MST. We will consider two algorithms here based on the greedy approach, but their complexities may differ slightly.

Kruskal Algorithm. This algorithm essentially requires all edges to be sorted, shortest first. Then the edges are included in set MSTE, one at a time, in an order such that the edges do not form a cycle. The test for forming a cycle can be efficiently made by maintaining a proper data structure of edges included thus far. The complexity of sorting is $O(M \log M)$), the test is of complexity $O(M + N)$ as suggested by Tarjan (12). As the process terminates once the set MSTE includes $N - 1$ edges, one may not have to sort all edges (the first few may be sufficient). This result be achieved by putting all edge weights in a heap that can be created in $O(M)$ time. An edge with the smallest weight can be removed from the heap in $O(\log M)$ time. If k edges have to be considered to select $N - 1$ edges for inclusion in MSTE, then the complexity of the selection process is $O(M + k \log M)$. Therefore, the total complexity is $O(M + N + k \log M)$. An example of execution of Kruskal's algorithm is shown in Fig. 12. Each edge is labeled with its weight and its number (shown in brackets). In each pass, the selected edge and the included nodes are shown in the table.



Edge Length	03	04	04	06	06	07	08	09	12	14	20
Edge Number	07	06	11	05	10	04	03	08	09	02	01

Pass	Edges	Nodes
1	7	B, C
2	7, 6	B, C, F
3	7, 6, 11	C, F, G, H
4	7, 6, 11, 5	C, D, E, F, G, H
5	7,6,11,5,10	C, D, E, F, G, H
6	Reject edge 4	C, D, E, F, G, H
7	....	....

Krushal Algorithm

Figure 12. Kruskal's MST algorithm.

Prim's Algorithm. For a dense network, when M is of $O(N^2)$, an alternative method to find an MST is from Prim (13). This algorithm maintains a tree and adds additional nodes to the tree using minimum cost edges. For this purpose, the minimum distance of each node that is out of the tree is maintained from the tree nodes. Each time a new node is added, the distances of nodes that are not yet in the tree from the tree change. Therefore, these distances need to be revised. In fact, the distances of the nodes outside the partial tree from the newly inserted node only need to be considered as that is the only change in the tree. The algorithm has a complexity of $O(N^2)$. We need N passes, one each to select N nodes to be included in the tree. Each time we need to find a node with minimum distance (this is an $O(N)$ procedure) and update distances of all other nodes after considering the new node (another $O(N)$ procedure if the distances are maintained in the adjacency list). Both $O(N)$ procedures can be performed in $O(d)$ if the maximum degree of each node is only d because we only need to consider d neighbors of the new node introduced in the tree. Thus, the overall complexity of the procedure is $O(dN)$.

Constrained MST. The MST computation may be constrained using some optimality criteria or requirements. In the case of constrained MST computation, the Selection of edges is constrained using appropriate selection criteria consistent with the specified constraints. For example, in the previous algorithm, it is assumed that the weight of an edge is the only criterion. But the new constraint may be that no node can have more than a certain number of edges connected to it. In that case, the algorithm may have to decide on a selectable candidate differently. If a node already has a given number of edges originating from it, then no more edges connected to that node may become part of the solution.

Tree Traversal

For a given tree graph, one may like to visit all nodes of the tree. Recall that a tree graph has no loops and the number of edges is exactly equal to $N - 1$. A node is visited after another node along a link. We will assume that no more

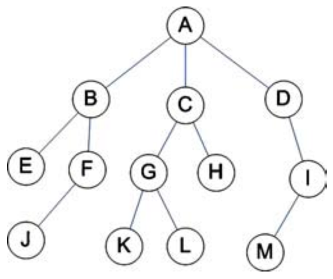


Figure 13. A 13-node tree example.

than one link exists between any pair of nodes. Nodes can be visited in two different ways. In the first case, once we are at a node, we visit all of its neighbors before visiting any other (non-neighbor) nodes. This process is called *breadth first order*. For example, for a given tree graph as shown in Fig. 13, we first visit the root node A. After that, we visit all its children, which are B, C, and D. Then we traverse children of B, C, and D, which are E and F, G and H, and I, respectively. Finally, we visit the children of these nodes and include nodes J, K, L, and M in the list of visited nodes.

In the second case, we visit nodes in *depth first order*. In this case, when we visit a node, we immediately visit its children first before visiting any of its siblings. In the example tree of Fig. 13, the nodes will be visited in order $A, B, E, F, J, C, G, K, L, H, D, I$, and M.

Depending on the application, one or the other method is used. For example, if the tree nodes represent solutions of a problem and we are interested in one solution, depth first search is likely to yield the solution faster. On the other hand, if we are interested in all possible solutions, then breadth first search is more appropriate.

Network (Max) Flow

In a given network, one may like to compute the available capacity on all paths from a source to destination. In that case we need to determine the maximum information flow possible from the source to the destination, which is accomplished by using a network flow analysis algorithms (15). The network graph is treated as a directed graph, and the maximum possible flow from a source node s to a destination node t is computed. For a given directed graph, each edge (i, j) is assigned a capacity using a nonnegative value C_{ij} that represents the available capacity to carry information on edge (i, j) from node i to node j. In addition, nodes may have additional constraints in terms of amount of information they can support in terms of buffer space and other factors from all incoming edges or links. This characteristic is the node capacity constraint. Let X_{ij} be the amount of actual flow through edge (i, j). At each node, information must be conserved as part of the total flow from s to t. That means the amount of information entering a node must be the same as the amount of information leaving that node. This information must not exceed the capacity of the node, or the following constraints must be satisfied:

$$0 \leq X_{ij} \leq C_{ij}, \quad \text{and} \quad \sum_i X_{ij} = \sum_j X_{ji}$$

Also $\sum_j X_{sj}$ is the amount of information that leaves the source node s and is equal to $\sum_j X_{jt}$, which is the amount of information that arrives at the destination node t.

Any such set of flows X_{ij} that satisfies the above constraints is called a feasible flow set. Maximizing feasible flow by increasing flow on different links while satisfying all constraints yields the max-flow value. For a given graph, this result is achieved. For a given graph, this result is achieved as follows.

First, we find a feasible flow from node s to node t (0 flow is trivial). Now, let P be an undirected path in the directed network from s to t. An edge on this path is called a *forward edge* if it is directed toward node t. Otherwise, it is a *backward edge*. A flow on this path can be augmented or increased if $X_{ij} < C_{ij}$ on all forward edges and $X_{ij} > 0$ on all backward edges. The amount of increase is given by

$$\min\{\min_{forward}\{C_{ij} - X_{ij}\}, \min_{backward}\{X_{ij}\}\}$$

If this value is greater than zero, then such a path is called an *augmentation path*. The process is repeated on all possible undirected paths. A flow is maximum if no augmentation path is available.

Figure 14 demonstrates computation of maximum flow. Figure l4a depicts a feasible flow. Each edge is marked with its capacity and the current flow value. Figure 14b shows an augmentation path with three forward edges and one backward edge. Using the relationship described above, the amount of flow that can be increased is one. Figure 14c shows the graph again with new feasible flow. Figure 14d shows another augmentation path with three forward edges. The flow can be increased by two on this path, and the new feasible flow is shown in Figure 14e. Figure 14f shows another augmentation path with four forward edges and the flow is again increased by two to obtain a maximum flow of ten as shown in Figure 14g.

The maximum value of a *s-t* flow is equal to the minimum capacity of a *s-t cut*. A cut is defined by a set of edges that partitions the network into two parts with s and t in separate partitions. A *minimum cut set* is a cut set whose total capacity of the edges is minimum.

Linear Programming Problems (LPPs)

In network design, we are mostly concerned with minimizing cost or delay in the network while maximizing the performance. Such problems can be expressed as optimization problems. The statements of such problems have an objective function that is required to be minimized or maximized subject to certain constraints. In most cases, these constraints are also linear in relation. A general linear programming problem (16, 17) is to find values of n real variables, denoted by $x_1, x_2, x_3, \cdots, x_n$, which will minimize or maximize an objective function given by

$$z = \sum_{j=1}^{n} c_j x_j$$

where C_j is a cost or reward value associated with variable X_j. The set of constraints that governs a feasible solution may vary in numbers and includes linear combinations of

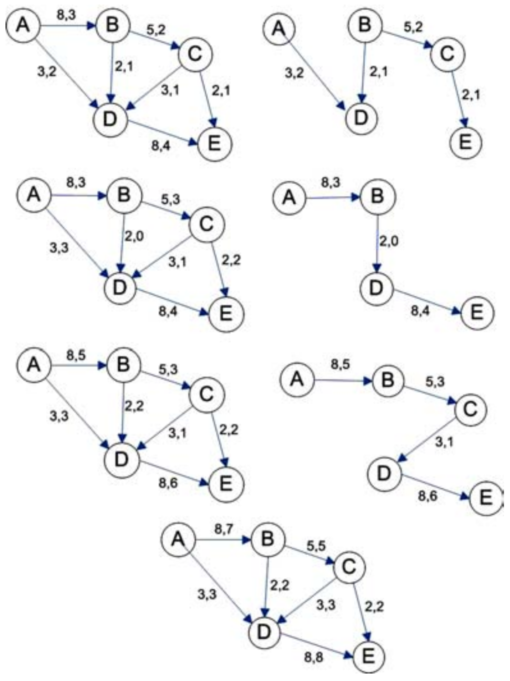


Figure 14. An example demonstrating feasible flow and augmentation paths.

variables xs and has a general form as in

$$\sum_{j=1}^{n} a_{ij}x_j > b_i \quad i = 1, 2, \cdots, m.$$

The values of variables may also be bounded by some lower and upper bounds as parts of constraints. For example, it may be desirable that all variables are positive or do not exceed a certain value. Various methods solve LPPs. The most commonly used method is the simplex method, which has no more than $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ solutions for $m \leq n$ possible solutions. The simplex method systematically searches for an optimal solution over this space.

A variation of this problem is when all variables are restricted to be integers only. This situation is called an integer programming problem (IPP), and it makes the problem more complicated. Standard packages are available to solve the two types of problems. The goal of a network designer is to formulate the proble as an LPP or IPP and then solve it using a standard package or a heuristic algorithm. If the problem size (the number of variables and constraints and therefore the number of possible solutions to search from) becomes too large, then we use heuristic methods only to solve the problem.

ROUTING ALGORITHM

A routing algorithm establishes an appropriate path from any given source to a destination. The objective of network routing is to maximize network throughput with minimal cost in terms of path length. To maximize throughput, a routing algorithm has to provide as many communication paths as possible. To minimize the cost of paths, the shortest paths have to be provided.

However, there is always a trade-off between these two objectives. Most routing algorithms are based on assigning a cost measure to each link in a network. The cost could be a fixed quantity related to such parameters as link length, bandwidth of link, or estimated propagation delay. Each link has a cost associated with it, and in most cases, it is assumed that the links have equal cost.

An interconnection network is *strictly nonblocking* if a routing algorithm exists to add a new connection without disturbing existing connections (1). A network is *rearrangeable* if its permitted states realize every permutation or allowable set of requests; here it is possible to rearrange existing connections if necessary(1). Otherwise it is *blocking*.

The *store-and-forward* operation in packet switching incurs a time delay and causes significant performance degradation. If the algorithm is used in a packet switching network, the total time delay of a data packet is obtained by summing up the time delay at each intermediate node. As the nonavailability of any link along a route causes the route not to be available, the network sees a high probability of blocking under heavy traffic, which rejects the incoming request and eventually causes data loss or delay.

The routing algorithm can be centralized or decentralized. A centralized algorithm may use a global backtracking depth first search or any other algorithm described in section on algorithms.

EMBEDDING ARBITRARY CONNECTION REQUESTS

The interconnection network should be able to embed arbitrary requests until resources are available in the network. If a set of requests is such that each node needs to communicate with a unique node, then such a set of requests is called a permutation. It is desirable to be able to satisfy this set of requests simultaneously. If each node requires communications with up to k other nodes, it may not be possible to satisfy these requests in one round and the communication requests may have to wait. Depending on the application environment, either the requests are partitioned in k disjoint permutatiqns (some may be partial permutations) or the communication needs are satisfied in k rounds without any contention. Altenatively, the network is designed to satisfy all requests up to k requests at the same time. A better solution would probably lie in between these two extreme cases. Depending on the number of transmitters and receivers, a node should be able to source and sink those many connections. The links in the network should be able to support the traffic corresponding to the requests being serviced simultaneously. The permutation routing capability of a network is extremely useful in improving the overall performance of a system.

In a permutation routing, the messages transferred from a source to a destination can be regarded as a commodity flow. For each commodity, the required flow of commodity is 1 for a single source and a single destination. In a general network, the problem of solving multicommodity integral flows is known to be NP-complete.

NETWORK TOPOLOGY DESIGN

A network can be designed using various topologies. Many interconnection networks have been proposed by the research community; some have been prototyped, but few have progressed to become commercial products. A network may be static or dynamic (18–23). The topologies can be divided into two categories: (1) regular and (2) irregular. The regular topologies follow a well-defined function to interconnect nodes. The regularity, symmetry, and most often the strong connectivity of the regular network topologies make them suitable for general-purpose interconnection structures where the characteristics of the traffic originating from all nodes are identical and destinations are uniformly distributed over the set of nodes. Thus, the link traffic is also uniformly distributed. The irregular topologies are optimized based on the traffic demands. If there is a high traffic flow between two nodes, then they may be connected using a direct link. If a direct link is not feasible, then an alternative is to provide a short path between the two nodes. Such designs are much more involved and need special attention.

We will first discuss regular topologies and then get into the design of irregular topologies. We will also discuss some specific regular topologies, such as a binary cube and its variations, in greater detail.

Regular Topologies

Several regular topologies have been proposed by various researchers in the literature. The most important among these are complete connected graphs, star, tree, ring, multiring, mesh, and hypercube. One desirable property of a structure is to be able to accommodate or embed an arbitrary permutation. We discuss various regular topologies in the following paragraphs.

Completely Connected Topologies. In a completely connected topology, every node is connecled to every other node as shown in Fig. 15a; that is, for every $\forall i,\ j,\ \in N, (i,\ j) \in E$. Thus, $N * (N-1)$ links exist. The routing is straightforward as a node i directly sends messages for node j on the corresponding link (i, j). Each node has $N-1$ transmitters and $N-1$ receivers, one for each link. The diameter of the graph is one and the reliability of the network is very high as, in addition to a direct link, $N-2$ paths of two hops exist from a node to every other node. This topology is the most expensive but most efficient. In practice, not many networks are aesigned using this topology. However, in a given network, one may set *virtual topologies* that are the eqivalent of a completely connected graph.

Star and Tree Topologies. The star and the tree are two topologies that require a minimum number of links to connect N nodes. The number of links is exactly equal to $N-1$. A star topology has a central node to which all other nodes are connected as shown in Fig. 15b. The tree topology, as shown in Fig. 15c and d, is hierarchical where the root of the tree at each level has to act as the intermediate node in any communication between nodes in the two halves of the tree (called left subtree and right subtree). In the star topology, the central node communicates with every other node using the direct link. If we consider the central node as only an intermediate node, then the routing between any two nodes is always through the central node and each path is of length two. The central node may become a bottleneck in communication. Failure of this node also causes the entire network to fail. Moreover, the central node is the most expensive node with degree $N-1$ and has to support $N-1$ other connections. On the other hand, the degree of each node is bounded and that is a big advantage. For example, in a star, each node connects to only one other node, and in a binary tree, each node only connects to three other nodes, one link to its parent node and at most two links to its Children nodes. The longest path in a binary tree can be up to $2 * \log N$. In a hierarchical structure like a tree, a different number of parallel links can be used to connect nodes at two adjacent levels to accommodate more traffic near the root node. This node is called a fat tree (24).

Rings and Multirings. The rings and multiring topologies are even simpler design's with fixed node degrees. For a simple ring, each node is connected to two other nodes. If the connections are unidirectional, then the simplest ring has one incoming link and one outgoing link. The diameter of the graph is $N-1$. In a bidirectional ring, each node has two incoming links and two outgoing links. A node i has a link to node $i+1$ and node $i-1$ (module N). The diameter of the graph is $N/2$. The multiring architecture has multiple links from each node to other nodes. Each set of corresponding links from each node forms one ring. Some examples of ring topologies are shown in Fig. 15e, f, and g.

Meshes. A node in an n-dimensional mesh structure has $2n$ neighbors, two in each dimension. A two-dimensional structure is shown in Fig. 15h. Each grid point is numbered using an *n-dimensional* tuple. Two-and three-dimension meshes are most commonly used in designing interconnection structures for multiprocessor systems. A mesh can be extended or shrunk in any dimension allowing easy reconfiguration and scalability required in many subsystem designs.

Hypercubes and its Variations. A hypercube is *n-dimensional* structure as shown in Fig. 15i. Hypercubes and its variants are popular interconnection structures because of unique properties such as symmetry, regularity, low diameter, and good fault tolerance characteristics (25). A Boolean n-cube $Q_n = (V, E)$ has $|V| = N = 2^n$ nodes. Each node is numbered using an n-bit binary string. The *Hamming distance* between two binary strings is the number of bit positions in which they differ. A pair of nodes in a Boolean cube is connected by an edge providing a bidirectional communication path between them if the Hamming distance between their binary addresses is one. An important property of an n-cube is that it can be constructed or decomposed recursively from/to two lower dimensional subcubes as is clear from its recursive definition as given next.

Definition. Boolean n-cube $Q_n = (V, E)$ is defined recursively as follows.

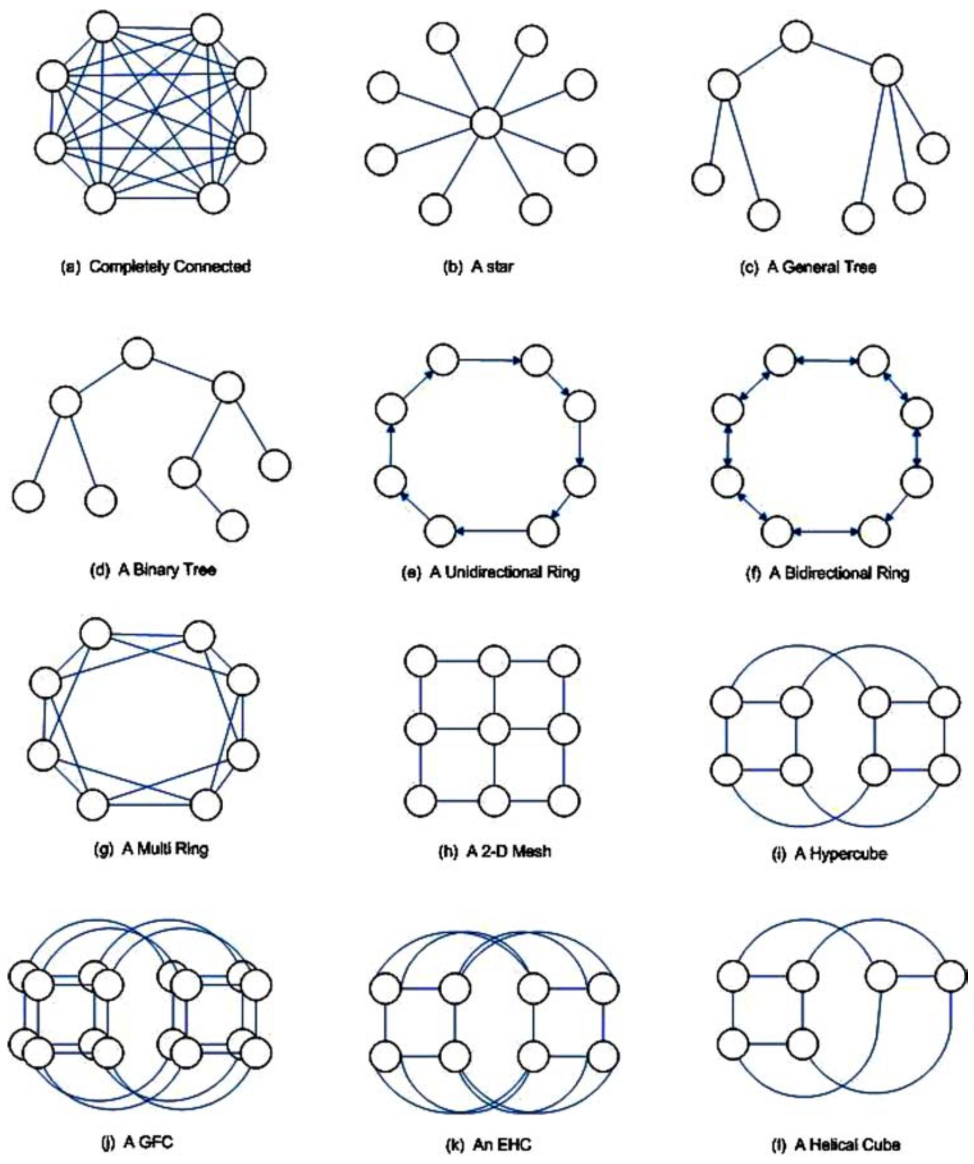


Figure 15. A hierarchical network.

1. The 0-cube Q_0 is defined as a single node with no edge.
2. $Q_n = Q_{n-1+} Q^d_{n-1}$, where the + operation is a *twofold operation* of the graph $G = (V,E)$ denoted by $G_t = G + G^d$ that yields a graph $G_t = (V_t, E_t)$, where $V_t = V \cup V^d$ and $E_t = E \cup E^d \cup \{(v, |V| + v) | \forall v \in V\}$.

The degree of each node, the diameter of the graph, and the node connectivity of the hypercube graph is n each. The length of the shortest path between any two nodes i and j in an n-cube is equal to the Hamming distance between their binary representations. There are $H(i,j)!$ shortest paths between two nodes i and j, and among them, $H(i, j)$ paths are independent (node-disjoint or parallel). In a Boolean n-cube, there are no cycles of odd length. The other regular topologies discussed above can be embedded in a binary n-cube or its variations discussed below.

Dynamic Topologies. A dynamic topology is created by modifying an existing topology as the need arises. This modification is achieved by adding links between nodes to either create more paths or point-to-point direct links to reduce delays and congestion and improve performance. Resulting networks usually look like random graphs with possibly no symmetry and very little fault tolerance.

Reconfigurable Topologies

There are two important issues in the design of a reconfigurable network: ease of embedding a given permutation and the cost of implementing the network. An $N \times N$ crossbar can realize all permutations easily but has a cost that is proportional to $O(N^2)$. To reduce the cost, a rearrangeable network (26) may be acceptable. The generalized folding-cube (GFC) and the enhanced hyper-cube (EHC) are two such topologies derived from binary cube architecture.

Generalized Folding Cube. A generalized folding cube is obtained by folding a hypercube along any dimension as follows. For a given a Boolean n-cube Q_n = (V,E), the *folding operation* of the cube Q_n denoted by $f(Q_n)$ yields a graph $Q^1_{n-1} = (V^1, E^1)$ and consists of the following two steps. (1) Split the cube into two subcubes by removing ($n-$ 1)-dimension links from $Q_n(Q'_{n-1} = (V', E')\, and\, Q''_{n-1} = (V'', E''))$. (2) Overlap the two subcubes Q'_{n-1} and Q''_{n-1} in such a way that $v' \in V'$ and $v'' \in V''$ become one and the same node $v^1 \in V^1$ if v' and v'' differ by 2^{n-1}. v^1 is numbered as $\min(v', v'')$. Either of the two links in each dimension, corresponding to v' and v'', can be used by either nodes, v' and v'', for communication.

The kth folding operation $f^k(Q_n) = f(f^{k-1}(Q_n))$ yields a graph $Q^k_{n-k} = (V^k, E^k)$. The kth unfolding operation $f^{-k}(Q^p_n) = f^{-1}(f^{-(k-1)}(Q^p_n)) = Q^{p-k}_{n+k} = (V^{p-k}, E^{p-k})$, where $k \in \{1, 2, \ldots, p\}$. The GFC denoted by $Q^p_n = (V^p, E^p)$ for $p \geq 0$, is defined from the folding operation of a hypercube: $f^p(Q_{n+p}) = Q^p_n$. The GFC consists of 2^P pairs of links in each dimension, and each node of the GFC consists of 2^P individual nodes of the original cube and a $(n+1)2^p \times (n+1)2^p$ switch. The original hypercube Q_n = (V, E) can be considered as a special case of the GFC and denoted by $Q^0{}_n = (V^0, E^0)$. Figure 15j shows 3-D GFC with 2^P pairs of links in each dimension. The rearrangeability of the GFC is shown in Reference 27.

Enhanced Hyper-Cube. If we wish to keep only one node at each vertex position and still want to design a rearrangeable network, then by duplicating links in any one dimension of the original hypercube, i.e., two pairs of links are provided instead on one, we obtain a structure that can provide conflict-free routes for every permutation (28). The EHC is shown in Fig. 15k.

A reconfigurable architecture, such as EHC or GFC, can embed other structures efficiently. The EHC and the GFC concepts can be combined to design a more cost-effective network. This design methodology has been used to design and implement the Proteus multicomputer system (29).

Helical Cube. A binary cube grows only as an integer power of two. To remove this deficiency, several of alternatives (30) have been suggested. An attractive option is a helical cube that removes $N-K$ nodes from a hypercube to obtain a K node structure while preserving all advantageous properties of the binary cube such as regularity, simplicity of routing, and fault tolerance. The degree of each node remains $n = \log K$. Only neighbors of removed nodes are affected and reconnected in such a fashion that the high graph connectivity is maintained. The links connected to nodes that are being removed are connected pairwise using a helical connection strategy, thus the name helical cube. An example of a helical cube is shown in Fig. 15l. The details of the actual connection scheme are given in Reference 31. It has been shown that this structure can have any number of nodes while maintaining a high connectivity and the same level of fault tolerance as the original cube.

Arbitrary Topology Design

If the graph structure is not constrained to be a regular topology, then the design problem can be formulated as a linear or nonlinear programming problem. Suppose we are use certain kinds of links and are given a traffic matrix. Here we assume that only one type of links is available and will consider a more complex problem in the last section. We will wish to design a network that is connected. The cost of connecting different links is different. Let $X_{i,j}$ denote whether a link between nodes i and j exists, $X_{ij} \in \{0, 1\}$, and suppose the cost to lay the link is denoted by C_{ij}. Let the original nodes be numbered as 1 to N.

One goal of the design is minimize the cost that is given by

$$Cost = \text{Min} \sum_{ij} C_{ij} X_{ij}$$

Then the existence of links has to be subjected to conditions that the network should satisfy. For example, each node should be connected by at least one link. This can be specified as

$$\sum_i X_{ij} + X_{ji} \geq 1$$

Then we may have constraints to specify that there is a path from each node to another node, or the graph should be connected. It is hard to formalize this as an equation but can be easily checked for a given X_{ij} configuration. After all the constraints are specified, one solves the problem to find a solution that is a vector of X_{ij}.

It may appear to be a simple problem but is generally very hard even for a moderate number of nodes. Therefore, it is usually solved using some heuristics. We will see a solution technique in the example section.

QUALITY-OF-SERVICE (QOS) REQUIREMENTS

Unlike conventional packet-or circuit-switched networks, some applications such as broadband integrated services/digital network (B-ISDN) require the network to provide not only connectionless traffic transportation but also connection-oriented operation for real-time data transfer between end users with multiple bit rates. Broadband packet switching based on the asynchronous transfer mode (ATM) that has a fixed packet length has been proposed for multimedia and multibit rate communications of end users by using the network resources efficiently. The most important aspect of these networks is to satisfy the (QoS) requirements. These features require a different approach to network design in comparison with the conventional packet-switched network design. For example, the cell loss probability has to be considered in an ATM network design. In circuit-switched networks, the call blocking probability is an important metric to determine the design of the circuit-switched networks.

Connection-oriented services have certain maximum delay requirements in exchanging information between the end users as given by the QoS requirements. The delay in a packet-switched network includes switching, queuing, transmission, and propagation delays. Because of the high data rate of fiber-optic links, propagation delay and node queuing delay are the dominant delay factors. In the conventional packet-switched network design problem, the average network delay and throughput have usually been used as the metrics to optimize the network cost and performance. In multimedia networks, services may have critical delay requirements; instead of the average network delay requirement, the end-to-end delay must be considered while determining the network topology.

In addition to these new requirements, the high data rates require special attention to fault management or fault tolerance. Compared with low-speed data networks, it is possible to lose many data, packets if a data link fails even for a short time. Fault management requires that the network has a control mechanisms that ensures that the existing traffic is affected as little as possible because of a failed link, and the traffic on the failed link is rerouted through the spare capacity on other links. This rerouting of traffic from a failed link to the other linkss (32) can be performed by a special facility such as the digital cross-connect system (DCS)(33).

Fault tolerance in high-speed networks is greatly needed even for short-time link failures because of large cell loss possibility. An alternative route may be longer than the original path. If a service, such as data file transfer, is not sensitive to propagation delay, the reconfiguration can be done using arbitrary available spare capacity on the other links. As voice and video service are sensitive to end-to-end delay, the reconfiguration path must be selected such that the end-to-end delay requirements are met. This performance requirement restricts the logical reconfiguration that can be embedded into the physical network. Therefore, while designing the network topology, possible failures of network links must be considered in advance.

To maintain the QoS requirements in services, we also have to consider the cell loss probability during a burst transfer. Burst cell loss can occur in several stages of the network: switch buffer overflow, cells discarded for congestion control, and physical link errors. The optical fiber link has negligible physical link errors. However, the switch buffers for each link may be of a fixed size and the packet contention for the same link may cause the output buffer to overflow in each link. Thus, we have to find the optimal link capacity assignment to meet the cell loss restrictions.

With this in mind, we investigate a fault-tolerant backbone network design algorithm and network resource management schemes while considering different (QoS) such as cell loss probability and mean end-to-end delay requirements for each call request. In the design algorithm, we ensure that these performance requirements are met even in the presence of faults in the network. We first introduce some mathematical notation and then present the design formulation, solution heuristics, and numerical examples demonstrating the goodness of the solution.

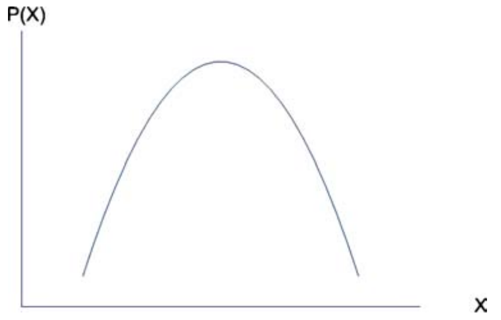


Figure 16. A bell-shaped curve.

DELAYS AND QUEUING IN NETWORK DESIGN

Probability Distributions

We will first describe three important probability distribution functions that are used in the analysis of network systems. More details can be found in Reference 34.

Normal Distribution. A random variable x is normally distributed if its probability density function is of the form

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

This is a bell-shaped curve density function as shown in Fig. 16. The peak of the bell occurs at $x = \mu$, and the width of the bell depenas on the variable σ. The random variable x is completely characterized by the two variables, mean μ, and the variance σ^2. The variable σ is the standard deviation. Three standard deviations from the mean cover about 99% of the area under the curve. That is why most of the time we are interested in $\mu + 3\sigma$ variations in the value of random variable x.

Binomial Distribution. The number of ways in which k out of n objects can be selected is given by

$$C(n, k) = \frac{n!}{k!(n-k)!}$$

If the probability of selecting a particular type of object is p (and the probability of selecting the other object(s) is $(1 - p)$), then the probability of selecting k such objects out of a total of n objects is given by

$$P(n, k, p) = C(n, k) \cdot p^k \cdot (1 - p)^{n-k}$$

The mean value of this statistic is $E(n, p) = n \cdot p$ and the variance is $V(n, p) = n \cdot p \cdot (1 - p)$. The standard deviation σ is given by $\sqrt{V(n, p)} = \sqrt{n \cdot p \cdot (1 - p)}$.

Exponential Distribution. A random variable x is exponentially distributed with parameter λ if the probability of $x \le t$ is given by

$$p(x \le t) = 1 - e^{-\lambda t} \quad t \ge 0$$

The mean and variance of x are $1/\lambda$ and $1/\lambda^2$, respectively.

An arrival of a request for service is usually modeled using a random process. Two requests are often assumed to be independent of each other. A process in which interarrival times between two consecutive requests are independent and distributed according to an exponential distribution with parameter λ is called a Poisson process (with parameter λ).

Design

How we use these distributions can be demonstrated using the following example. Suppose a network has ten nodes that want to communicate among themselves. Assume the probability that a node originates a data request is p= 0.1. The switching network can connect a call if all required links are free or not in use. How many links should we provide so that a communication request can be satisfied with high probability? In this case, the average number of requests is $E(n,\ p) = 10 * 0.1 = 1.0$ and the deviation is $\sigma = \sqrt{10 * 0.1 * 0.9} = \sqrt{0.9} = 0.95$. To satisfy most of the requests with high probability, we may like to provide $\mu + 3 * \sigma = 1 + 3 * 0.95 \approx 4$ links.

Delays in Networks

A communication link can be viewed as a bit pipe over which a given number of bits is transmitted over a unit of time. This number is called the transmission capacity of the link and depends on the physical channel and the interface at the two ends of the link. The bit pipe (link) is used to serve all traffic streams that need to use the link. The traffic of all streams may be merged into a single queue and transmitted on a first-come-first-serve basis. This process is called statistical multiplexing. It is also possible to maintain several queues for a link, one for each traffic stream or one for each priority if the incoming traffic streams have multiple priority levels assigned to them. If a packet length is L and the link capacity is C bits/s, then it takes L/C seconds to transmit a packet.

In case all incoming communication requests for a link are assigned to a queue and serviced as the resources become available, there are four different kinds of delays a packet suffers on a link. If the packet has to travel through multiple links, then the total delays will be the sum of delays on all links

1. **Queuing Delay.** The queueing delay is the delay between when a packet is assigned to a queue and when it is ready to be processed for transmission. During this time, that packet simply waits in a queue. This time depends on the number of packets waiting ahead of this packet in the queue.
2. **Processing Time Delay.** The processing time is the time between events when the packet is ready to be processed and the time it is assigned to the link for the transmission. The processing delay depends on the speed of the link processor and the actions the processor needs to take to schedule the transmission.
3. **Transmission Delay.** The time difference between the transmission of the first and last bit of the packet is referred to as the transmission delay. This delay depends on the bit transmission rate of the link.
4. **Propagation Delay.** The propagation delay refers to the time difference between the instances when the last bit is transmitted by the head of the link (source), and it is received by the tail of the link (destination). This delay depends on the physical distance of the link and speed of propagation and can be substantial for a high-speed link.

Queuing Models

To compute the queuing delay for a packet, we have to understand the nature of the packet arrival process to a link, the kind of service time it needs (amount of transmission time), and the number of links we have from the source to the destination. In most queuing systems (35, 36), we assume that the arrival process is a Poisson process. We also assume that the holding time (the amount of time a request requires to service) follows an exponential distribution with parameter μ. The mean service time is then given by $1/\mu$. If two nodes i and j are connected by m links, then m packets can be transmitted from node i to node j at the same time. Generally m = 1 and therefore packets are transmitted one at a time. In case of circuit switching, it can be observed as one request being established at a time.

M/M/m Queue. A queuing system with m servers, Poisson arrival process, and exponentially distributed service times is denoted by the M/M/m queuing system. The first letter M stands for memoryless. It can also be G for general distribution of interarrival times or D for deterministic interarrival times. The second letter stands for the type of probability distribution of the service times and can again be M, G, or D. The last number indicates the number of servers.

In a M/M/l queuing system, the average number of requests in the system in steady state is given by $\dfrac{1}{\mu - \lambda}$ and the average delay per request (waiting time plus service time) is given by $\dfrac{1}{\mu - \lambda}$. Utilization of the system is denoted by $\rho = \lambda/\mu$, and the average time for a request in a system is given by average service time/(l — ρ). The average waiting time T_w is given by the difference of the average time in system and the average service time. This time is equal to $1/(\mu - \lambda) - 1/\mu$. The average number of requests in the queue is given by $\lambda * T_w$. Also, the probability that exactly k requests are waiting is given by $P_k = (1 - \rho)\rho^k$. These results for a queuing system will be used later on.

Performance Metrics. When a request for service arrives, the server (link) may be busy or free. If the server is free, the request is serviced. If the server is busy, then there are two possibilities: 1) The request is queued and serviced when the server becomes available. In this case, we are interested in finding out how long, on average, a request may have to wait before it is serviced. In other words, we need to find out how many requests are pending in a queue or the average length of the queue. This has implications in designing queues to store requests. 2) The incoming request is denied service, which is called blocking. In this we are interested in determining the blocking probability for an incoming request. Again, this has implications in network design. We would like the blocking probability to be

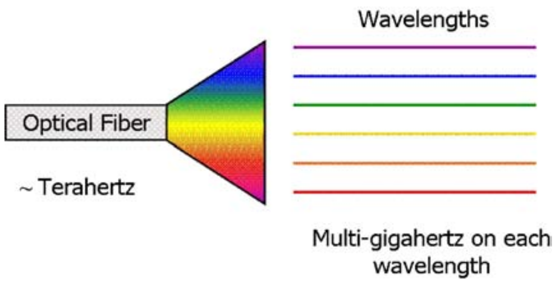


Figure 17. A fiber divided among multiple wavelengths.

as small as possible.

EXAMPLE 1: DESIGN OF A NETWORK USING WDM FIBER OPTICS

Wavelength Division Multiplexing-Based Optical Networking Technology

With the advent of optical transmission technology over optical fibers, the communication networks have attained orders of magnitude increase in the network capacity. The bandwidth available on a fiber is approximately 50 THz (terahertz). Hence, wavelength division multiplexing (WDM) was introduced that divided the available fiber bandwidth into multiple smaller bandwidth units called *wavelengths*. Figure 17 depicts the WDM view of a fiber link. Different connections, each between a single source/destination pair, can share the available bandwidth on a link using different wavelength channels. Advanced features such as optical channel routing and switching supports flexible, scalable, and reliable transport of a wide variety of client signals at ultra-high speed.

Early optical networks employed broadcast and select technology. In such networks, each node that needs to transmit data broadcasts it using a single wavelength and the receiving node selects the information it wants to receive by tuning its receiver to that wavelength. To avoid unnecessary transmission of signals to nodes that do not require them, *wavelength routing* mechanisms were developed and deployed. The use of wavelength to route data is referred to as wavelength routing, and networks that employ this technique are known as *wavelength-routed* networks. In such networks, each connection between a pair of nodes is assigned a path and a unique wavelength through the network. A connection from one node to another node established on a particular wavelength is referred to as a *lightpath*. A wavelength-routed WDM network is shown in Fig. 18. The figure shows connections established between nodes A and C, B and C, H to G, B to F, and D and E. The connections from nodes A to C and B to F share a link. Hence, they have to use different wavelengths on the fiber.

One alternative to circuit switching, described above, is to use optical packet switching (OPS) or optical burst switching (OBS) (44–46)technology in the backbone. The major advantages of OPS/OBS are the flexible and efficient bandwidth usage, which enables the support of diverse services. However, implementation technologies are not yet there for successful deployment of them in an all-optical domain.

WDM Network Design Issues

WDM network design involves assigning sufficient resources in the network that would meet the projected traffic demand. Typically, network design problems consider a static traffic matrix and aim at designing a network that would be optimized based on certain performance metrics. Network design problems employing static traffic matrix are typically formulated as optimization problems. To formulate a network design problem as an optimization problem, the inputs to the problem, in addition to a static traffic demand, are some specific reguirements, e.g., required network reliability and fault tolerance requirements, network performance in terms of blocking, and restoration time when a failure occurs. The objective of the optimization problem is to find a topology that would minimize the resources, including the number of links and fibers, the number of wavelengths on each fiber, and the number of cross-connect ports, to meet the given requirements. The outputs include the network configuration and the routes and wavelengths that are to be used for source-destination pairs. The network design problem can be formulated as an integer liner programming (ILP) or mixed integer linear-programming (MILP) problem. As the number of variables and constraints can be very large in WDM networks, heuristics are usually used to find solutions faster.

If the traffic pattern in the network is dynamic, i.e., specific traffic is not known a priori, the design problem involves assigning resources based on a certain projected traffic distributions. In case of dynamic traffic, the network designer attempts to quantify certain performance metrics in the network based on the distribution of the traffic. The most commonly used metric in evaluating a network under dynamic traffic pattern is *blocking probability*. The blocking probability is computed as the ratio of number of requests that cannot be assigned a connection to the total number of requests. With this metric, one makes decisions on the amount of resources that are needed to be deployed in a network, the operational policies such as routing and wavelength assignment algorithms, and call acceptance criteria.

Traffic Grooming WDM Networks

Data traffic in ultra-long-haul WDM networks is usually characterized by large, homogeneous data flows. The metropolitan-area WDM networks, on the other hand, have to deal with dynamic, heterogeneous service requirements. In such WAN, and MANs, equipment costs increase if separate wayelengths are used for each service. Each wavelength offers a transmission capacity at gigabit per second rates, whereas the users request connections at rates that are far lower than the full wavelength capacity. In addition, for networks of practical size, the number of available wavelengths is still lower by a few orders of magnitude than the number of source-to-destination connections that may be active at any given time. Hence, to make the network viable and cost-effective, it must be able to offer sub-wavelength-level services and must be able to pack these services efficiently onto the wavelengths. These sub-

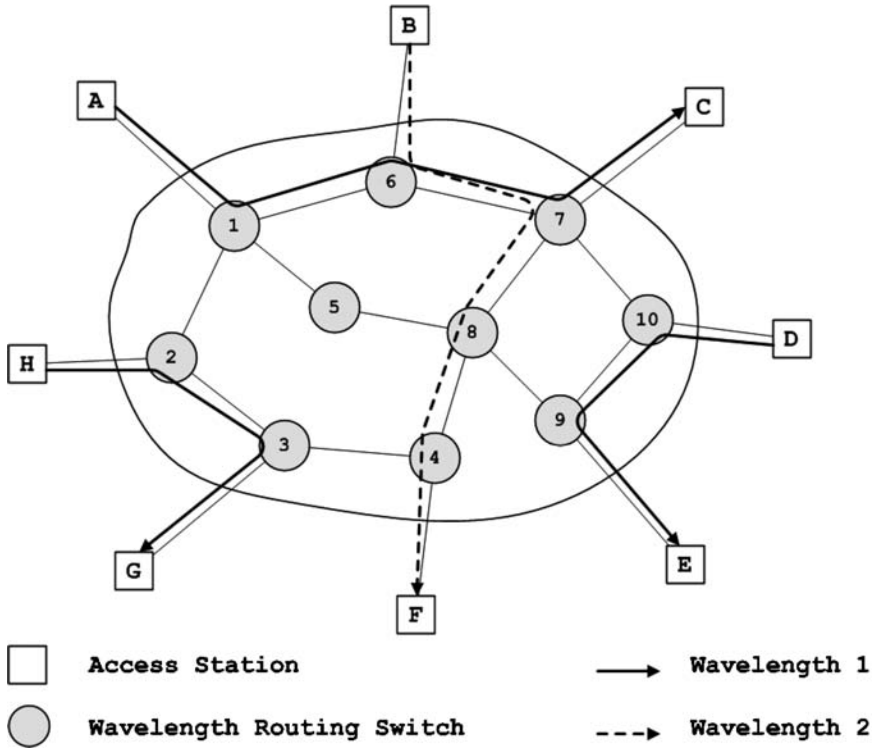


Figure 18. A wavelength-routed WDM network.

wavelength services henceforth are referred to as *low-rate traffic streams* in comparison with a full wavelength capacity. Such an act of multiplexing, demultiplexing, and switching of lower-rate traffic streams onto high-capacity lightpaths is referred to as *traffic grooming*. WDM networks offering such sub-wavelength low-rate services are referred to as *WDM grooming networks*. Efficient traffic grooming improves the wavelength utilization and reduces equipment costs.

Dynamic Traffic Grooming in WDM Network

In the future, as Internet Protocol (IP) becomes the prevailing protocol, it is the responsibility of the IP layer to effectively multiplex traffic onto wavelengths. These IP-over-WDM networks are likely to be arranged in a mesh topology rather than a ring. The traffic requirements of IP are bound to change much faster than the static scenario. It is thus important that dynamic traffic grooming is employed so that the networks can efficiently accommodate changes in traffic. Minimizing equipment costs in such a dynamic traffic grooming scenario for SONET/WDM rings is an important consideration.

It is possible to restrict traffic grooming in such a way that all traffic streams that are groomed on a path originate and terminate at the same node pair. For example, Fig. 19, traffic streams between node pair (S_1,D_1) and traffic streams between node pair (S_2,D_2) are groomed on their respective paths. In another case, it is possible that traffic streams between different node pairs share a path. For example in Fig. 19, traffic streams between node pair ((S_1,D_1)) and traffic streams between node pair ((S_1,D_2)) can share a link. Similarly, traffic streams between node pair ((S_2,D_1)) and traffic streams between node pair ((S_2,D_2)) can share a link.

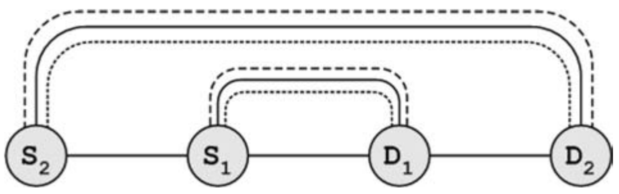
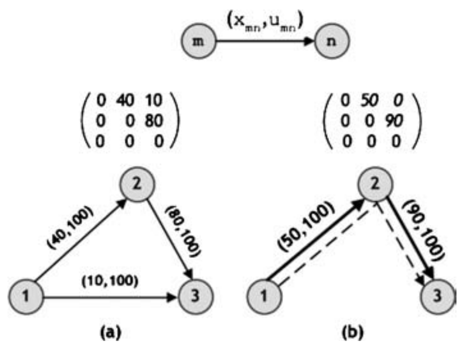


Figure 19. Example of grooming streams for same node pair.

A challenging problem for carrying IP traffic over WDM optical networks is the huge opto-electronic bandwidth mismatch. One approach to provisioning fractional wavelength capacity is to divide a wavelength into multiple subchannels using time-, frequency-, or code division multiplexing and then multiplex traffic on the wavelength, i.e., *traffic grooming*, However, optical processing and buffer technologies are still not mature enough to achieve online routing decisions at high speed. With the development of MPLS (Multiprotocol Label Switching) and GMPLS (Generalized Multiprotocol Label Switching) standards (47–49) it is possible to aggregate a set of IP packets for transport over a single lightpath. Therefore, traffic grooming in IP over WDM optical networks is performed at two layers, namely *IP traffic grooming* and *WDM traffic grooming*. IP traffic grooming is the aggregation of smaller granularity IP layer traffic streams. It is performed at MPLS/GMPLS-enabled IP routers by using transmitters and receivers. This aggregated traffic streams are then sent to the optical layer where WDM traffic grooming (or wavelength level traffic grooming) is performed by using optical add-drop



(x_{mn}, u_{mn})
m
n
$\begin{pmatrix} 0 & 40 & 10 \\ 0 & 0 & 80 \\ 0 & 0 & 0 \end{pmatrix}$
$\begin{pmatrix} 0 & 50 & 0 \\ 0 & 0 & 90 \\ 0 & 0 & 0 \end{pmatrix}$
(40,100)
(80,100)
(10,100)
(50,100)
(90,100)
1
2
3
(a)
(b)

Figure 20. Illustrative example of IP traffic grooming.

multiplexors (OADMs). The two-layered grooming reduces the workload at both IP and optical layers.

IP Traffic Grooming Issues. The main cost in IP traffic grooming is from the transmitters and receivers at the end nodes rather than number of wavelengths, which was the main cost for grooming ring network design. Minimizing the number of transmitters and receivers required is equivalent to minimizing the number of lightpaths that are needed, because each lightpath needs one transmitter and one receiver. Figure 20 demcts an illustrative example that shows how IP traffic grooming helps to reduce the number of transmitters and receivers in a three node network.

Assume that each link has a capacity of 100 units. The matrix in Fig. 20a is the original traffic matrix. It includes the location and capacity of three requests. Figure 20a depicts one solution in the absence of IP traffic grooming, and simply establishes a lightpath (connection) for each s-d pair. It requires one transmitter and one receiver at each node.

Figure 20b depicts another solution based on the fact that the capacity requested by s-d pair (1, 3) is relatively smaller. Thus, instead of reserving a separate lightpath for it, the spare capacity along lightpath $1 \rightarrow 2$ and $2 \rightarrow 3$ can be reused to accommodate the traffic of s-d pair (1, 3). That is, the traffic from Node 1 to Node 2 and 3 both take the route from Node 1 to Node 2. Node 2 receives and analyzes the traffic, drops the traffic that is destined for it, and forwards the remaining traffic (from Node 1 to Node 3) along with its own traffic (from Node 2 to Node 3) to Node 3. This add-and-drop procedure is performed by transmitters and receivers at Node 2. In this scenario, the traffic carried by the optical layer is represented by the matrix in Fig. 20b.

The scheme shown in Fig. 20b results into one less transmitter and receiver in comparison with the scheme shown in Fig. 20a. However, the lower size traffic request (1, 3) takes a longer route in IP layer to avoid reserving an entire wavelength for it. This tradeoff needs to be made in order to alleviate the wavelength underutilization in the the optical layer.

Approach to IP Grooming Problem. Let $D_{N\times N} = \{d_{st}\}$ denote the traffic matrix, where d_{st} denotes the traffic capacity required from source node s to destination node t, and represent the capacity requirement of the systems.

The IP traffic grooming problem is described as follows.

Given a traffic matrix for a network, how to aggregate the traffic requests for transporting, such that the total number of transmitters (and receivers) required in the network is minimized.

The *Physical topology* is represented by a graph $G_P(V, E)$, with V being the set of nodes and E being the set of physical links. The *Virtual topology (logical topology)* is represented by a graph $G_l(V, L)$ with nodes corresponding to the nodes in the physical network and edges corresponding to the lightpaths. Each lightpath may extend over several physical links (spans). The link flow and link capacity for link (m, n) (from node m to node n) are denoted by x_{mn} and u_{mn}, respectively.

Notice that, each request is assigned a dedicated lightpath, the virtual topology would be a full-connected network if there is a request for each node pair. The desired grooming network is the one with a minimum number of transmitters and receivers, which is a solution with a minimum set of arcs in its virtual topology that is sufficient to carry the given traffic.

For this problem to be meaningful, it is assumed that each request has a capacity smaller than or equal to the full-wavelength capacity. Note that, for a capacity requirement of more than a full wavelength, there has to be some full wavelength paths assigned to this request and its remaining capacity need would be fulfilled using the traffic grooming algorithm. The terms “link” and “arc” are used interchangeably here.

This problem is similar to a capacitated multicommodity flow design problem (50) with limited link capacities. Therefore, this problem can be formulated as an ILP optimization problem. It is assumed that a request from the same s-d pair will always take the same route. Also, it is assumed that each link has the same capacity that is given by $W \times C$, where W denotes the number of wavelengths carried by a link and C denotes the full-wavelength capacity.

1. Notations:

   1.1. Parameters:
   - ∗ *W:* Maximum number of wavelengths in each direction in a bidirectional fiber (technology-dependent data).
   - ∗ *C:* Maximum capacity of each wavelength. (It is assumed that each wavelength has the same capacity.)
   - ∗ $s, t = 1,2,\ldots, N$: Number assigned to each node in the network.
   - ∗ $l = 1,2,\ldots,L$: Number assigned to each link in the network.
   - ∗ L_{st}^{k}: (data) For each s-d node pair, list all possible routes from source node s to destination node t, excluding routes that pass through a node more than once, and number them using k as an index. That is, $r_{1,6}^{3}$ indicates the third route from Node 1 to Node 6.
   - ∗ $A_{st}^{l,k}$: (binary data) takes the value of 1 if arc l is on the kth from node s to t; zero otherwise.
   - ∗ d_{st}: Denotes the traffic capacity required from source node s to destination node t.

   1.2. Variables:

- * γ_{st}^k: Binary variable, route usage indicator, takes a value of 1 if route r_{st}^k is taken; zero otherwise.
- * u_l: Integer variable, logical link usage indicator, keeps an account of the number of lightpaths on arc l in the virtual topology.

Problem Formulation

1. Objective: The objective is to minimize the number of arcs in the virtual topology, which reflects the minimum number of lightpaths in the optical layer. Recall that variable u_l counts the number of lightpatns on arc i in the virtual topology. If the capacity carried by arc i exceeds the full wavelength capacity, multiple lightpaths between the same node pair are required. Thus, the number of transmitters (and receivers) increase.

$$\min \sum_{l \in L} u_l \tag{1}$$

2. *Fiber link capacity constraint*: Let TC^1 be the total capacity carried by link l, which is given by equation 2. Constraint 3 guarantees that the aggregated capacity on any arc does not exceed the total fiber capacity, which is bounded by $W \times C$.

$$TC^l = \sum_{(s,t),s \neq t} \sum_k \gamma_{st}^k A_{st}^{l,k} d_{st} \tag{2}$$

$$TC^l \leq W \times C \tag{3}$$

3. *Traffic routes constraint*: Equations 4 and 5 ensure that if a request from node s to t, occurs one and only one route is assigned to the request. In another word, $d_{st} \geq 0$; set $\sum_k \gamma_{st}^k = 1$. Otherwise, no traffic request occurs from node s to node t, and none of the routes from node s to node t will be taken; hence, $\sum_k \gamma_{st}^k = 0$.

$$\sum_k \gamma_{st}^k \leq d_{st} \tag{4}$$

$$NC \sum_k \gamma_{st}^k \geq d_{st} \tag{5}$$

4. *Arc usage constraint*: Recall that the arc usage indicator u_l, counts the number of lightpaths required on arc l (logical link l) in order to carry the aggregated traffic $TC^1 \cdot u_l = [TC^l/C]$, which is obtained by using equations 6 and 7. For example, if $C = 48$ and $TC^i = 62$, $[62/48] = 2$ lightpaths are required on logical link i from its start node to its end node to its end node.

$$C \times u_l \geq TC^l \tag{6}$$

$$C \times u_l \leq TC^l + C \tag{7}$$

Notice that, from equations 3 and 6, the total number of lightpaths on a logical link l is bounded by the number of wavelengths on the optical fiber.

Additional constraints, such as the limited number of transmitters on each node, can be easily added to this formulation. This process helps to capture the cost on each node in the networks.

The limitation of this exact ILP formulation is that it enumerates all possible routers for each s-d pair and searches for an optimal set of arcs in virtual topology. In a fully connected network of N nodes, up to $\sum_{h=0}^{N-2} P_{N-2}^h$ possible routes exist for each s-d pair, where P_m^n is the permutation operation. This search requires large computation time as the network size increases. The formulation can be further simplified by adding a *hop-length constraint* such that the number of possible routes is reduced to a reasonable number; consequently, the computation time is saved. However, this network design problem is still a special case of multicommodity flow problem, which becomes unmanageable even for moderata-sized networks. Therefore, a heuristic approach would be desired for obtaining "good" solutions in a reasonable amount of time that capture all constraints of the ILP solution.

Approximate Approach

For a network $G(V, E)$, in the absence of IP traffic grooming, the number of transmitters and receivers required at node s, denoted by $Tx_s^{\max}$ and $Rx_s^{\max}$, respectively, can be derived from matrix $D_{N\times N}$.

$$Tx_s^{\max} = \sum_{t:(s,t)\in E} \lceil \frac{d_{st}}{C} \rceil \tag{8}$$

$$Rx_s^{\max} = \sum_{t:(t,s)\in E} \lceil \frac{d_{ts}}{C} \rceil \tag{9}$$

where C denotes the full wavelength capacity that can be used, because request d_{st} requires at most $[d_{st}/C]$ transmitters at node s to transmit traffic d_{ts}; likewise, it requires at most $[d_{st}/C]$ receivers at node from nodes t to receive traffic d_{st} from node s.

From the perspective of network flows, the total amount of outgoing traffic flows observed by node s is $\sum_{t\neq s} d_{st}$ and the total amount of incoming flows to node s is $\sum_{t\neq s} d_{ts}$. Hence, the minimum number of transmitters and receivers ne network to carry the traffic in $D_{N\times N}$ can be derived using the following two equations:

$$Tx_s^{\min} = \lceil \frac{\sum_{t:(s,t)\in E} d_{st}}{C} \rceil \tag{10}$$

$$Rx_s^{\min} = \lceil \frac{\sum_{t:(t,s)\in E} d_{ts}}{C} \rceil \tag{11}$$

In general, $Tx_s^{\min}$ and $Rx_s^{\min}$ are loose lower bounds. The reason is that, to reduce the number of transmitters (and receivers), some s-d pairs may have to take multiple hops and hence increase the link load in the virtual topology. This overhead load is not captured in equations 10 and 11, and it is dependent on the traffic pattern.

Traffic Aggregation Algorithm

To develop a *traffic aggregation* heuristic approach, the basic idea is to merge the smaller traffic request onto bigger bundles to reduce the number of transmitters and receivers. Although the total numberof lightpaths required in the network is reduced, the finer granularity requests may take multiple-hop and longer routes. This process may introduce delay for lower-rate requests, and it would be affordable in the future slim IP-over-WDM control plane. As a matter of fact, this is a trade-off that has to be made to reduce the overall network cost.

An element in traffic matrix can be *reallocated* by merging it with other traffic streams. Thus, no need exists to establish a direct path for that s-d pair. An element in traffic matrix can be *aggregated* if it is smaller then the full capacity, i.e., has spare capacity on a wavelength channel and allows other traffic streams to be merged on it. Each element in the traffic matrix can be viewed as in one of the three states:

- State 0: If it can be reallocated or be aggregated.
- State 1: If it cannot be reallocated but can be aggregated.
- State 2: If it cannot be eliminated or aggregated. For example, if $d_{st} = 0$, no traffic exists to be reallocated and no need exists to allocate traffic.

The goal of the traffic aggregation algorithm is to choose a traffic stream d_{st} that can be merged with some other traffic streams d_{sn} and d_{nt}, so that d_{st} can be carried using a multipie-hop path and not burden the system to establish a new path for it. After selecting d_{st}, the basic traffic aggregation operation on traffic matrix D consists of the following three steps:

1. $d_{sn} \leftarrow d_{st} + d_{sn}$.
2. $d_{nt} \leftarrow d_{st} + d_{nt}$.
3. $d_{st} \leftarrow 0$.

After this operation, the traffic request between s-d pair (s, t) is aggregated on s-d pairs (s, n) and (n, t). Let $TR(T_{a,t,n})$ be the number of transmitters (equals to the number of receivers) needed after merging d_{st} with d_{sn} and d_{nt}. $TR(T^{\circ})$ is called the upper bound, where T° is the original traffic matrix.

The key here is to select d_{st} and node n to reduce the value of $TR(T_{s,t,n})$. In experimenting with the ILP formulation, described above, it is observed that the ILP solution uses multi-hop routes for smaller requests, whereas the bigger requests tend to use direct single-hop paths. This observation is used to develop a heuristic solution. Figure 21 gives the *traffic Aggregation* algorithm. The resulting new traffic matrix gives the structure of a virtual topology and the required capacity on each physical link. The idea is to integrate smaller traffic request, say d_{st}, to those bigger traffic requests, d_{sn} and d_{nt}, to saturate the existing wavelength paths before establishing a new one. This would force some smaller granularity traffic to take longer routes with multiple hops, while saving some lightpaths.

The algorithm starts by finding the s-d pair with minimum request capacity that is in state 0 (Step 2 in Fig. 21), say d_{st}. Next it searches for a set of all eligible intermediate nodes, namely K (Step 4a in Fig. 21). Define the index value of an item v in set K as $index(v) = \max(d_{sv}, d_{vt})$. The intermediate node n is selected from K to saturate some wavelengths. Hence, if K is not empty, n is chosen as the node with the maximum *index* value. One could choose

```
INPUT: Graph $G(V, E)$ and a traffic matrix $D_{N \times N}$.
OUTPUT: Rearranged traffic matrix $D_{N \times N}$.
ALGORITHM:
 1) Initialize s-d pair status:
    If $d_{st} > 0$ then $d_{st}.state = 0$,
    else $d_{st}.state = 2$.
 2) $target = min(d_{st} : d_{st}.state = 0)$.
 3) If target = NULL, terminate.
 4) else
      a) Set $K$=new stack. Pick node $v$ that satisfies:
           i) $d_{sv}.state \le 1$, $d_{vt}.state \le 1$;
          ii) $d_{st} + d_{sv} \le C$ , $d_{st} + d_{vt} \le C$;
         iii) $TR(T_{s,t,v}) < TR(T)$.
              $K$.push {v}.
      b) Define $index(v) = max(d_{sv}, d_{vt}), v \in K$.
      c) If $K = \Phi$, then $d_{st}.state \leftarrow 1$, go to 2.
      d) else $n = arg\ max_{v \in K}\{index(v) : v \in K\}$.
      e) Update traffic matrix $D_{N \times N}$:
           i) $d_{sn} \leftarrow d_{st} + d_{sn}$;
          ii) $d_{nt} \leftarrow d_{st} + d_{nt}$;
         iii) $d_{st} \leftarrow 0$, $d_{st}.state \leftarrow 2$.
 5) Go to 2.
```

Figure 21. Approximate approach: traffic aggregation.

maximum or minimum to keep an order in which nodes are explored. We choose maximum here. The algorithm then updates the current traffic matrix after an intermediate node is decided (Step 4e in Fig. 21). If K is empty, no eligible intermediate node is found for this s-d pair, $d_{st} \cdot state$ is changed from 0 to 1, which means request d_{st} cannot be reallocated, but could be aggregated. The algorithm keeps searching for the next s-d candidate for aggregation until no eligible s-d pairs in State 0 can be found.

Complexity Analysis

One s-d pair is changed from State 0 to either State 1 or State 2 in each step. Thus, the algorithm terminates after at most N^2 passes. Without any complex, the run time for searching *target* in each loop is up to N^2; it takes another N loops to find the set K. Thus, the overall computation complexity of this algorithm is $O(N^5)$. In practice one will never see this complexity and the algorithm terminates much faster. One way is to use effective data structures to make the search more efficient and faster.

Example of Traffic Aggregation

Figure 22 illustrates an example of how the traffic aggregation algorithm performs. Assume that each wavelength has a capacity of OC-48 (2.5Gbps), and the minimum allocatable unit is OC-1. Thus, $C = 48$. Consider traffic matrix that is composed of random combination of OC-1, OC-3, and OC-12. An original traffic matrix includes all possible s-d pairs, which is shown as the top-left matrix in Fig. 22.

The algorithm starts by finding the minimum eligible s-d pair that can be reallocated, which is $(1, 4)$ with $d_{1,4} = 2$ in this example. Next it finds the possible intermediate nodes to include into set K. It can be observed that $K = 2,3$ with $index(2) = 43$ and $index(3) = 37$. Among the candidate nodes in K, the one with the highest *index* value is chosen; that is, $n = 2$. Next, the current traffic matrix is updated by removing $d_{1,4}$ from the original position and aggregating it with

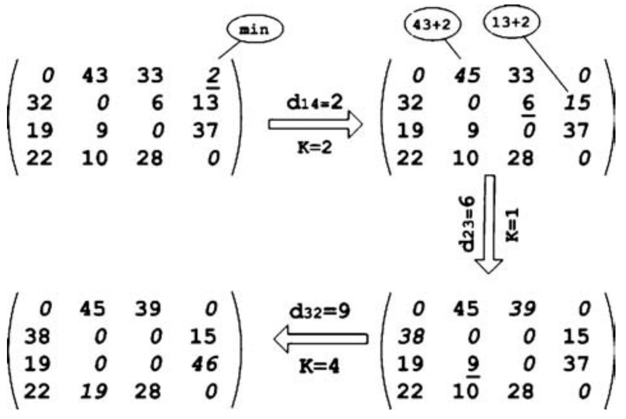


Figure 22. An illustrative example of the traffic aggregation algorithm.

$d_{1,2}$ and $d_{2,4}$, which results into the matrix on the top right in Fig. 22. Next the algorithm selects $d_{2,3} = 6$ and completes its processing by choosing $n = 1$. The algorithm continues until no more relocatable s-d pair exists as shown in Fig. 22. The botton-left matrix shows the final results. Application of equations 8 and 9 indicate that 12 transmitters (and receivers) are required for the original traffic matrix. After traffic aggregation, this number is reduced by 3.

Solutions and Resuits

The previously given ILP formulation is solved by using CPLEX Linear Optimizer 7.0. The ILP formulation and the traffic aggregation approach are applied to solve IP traffic grooming problem for a six-node network, with $W = 6$, $C = 48$. Table 1 gives a traffic matrix with randomly generated 50 requests. The integer numbers indicates the request capacity in a unit of OC-1 (51.84 Mbps). The objective is to design a network with as few logical links as possible. Notice that there are totally $P_4^0 + P_4^1 + P_4^2 + P_4^3 + P_4^4 = 65$ routes for each s-d pair in a six-node network, and this number increases dramatically as the network size increases. It would be a great burden and might be unnecessary as well to obtain optimality by searching among all possible routes. Experiments with different maximum hop-length as 3, 4, and 5 are performed on this six-node network. The results show that limiting the hop-length to 3 still yields close to an optimal solution, whereas the number of all candidate paths for each s-d pair is effectively reduced from 65 to $P_4^0 + P_4^1 + P_4^2 = 17$, which significantly reduces the size of the feasible region of this ILP formulation; hence, it reduces the computation complexity of solving the ILP optimization problem.

The results obtained from solving ILP with hop length = 3 and the traffic aggregation approach are shown in Fig. 23a and respectively.

According to Equations 10 and 11, at least nine transmitters (receivers) are required. Figure 23a shows an optimal solution consisting of 11 lightpaths by solving, the ILP formulation with a maximum hop-length limit of 3. Figure 23b shows a solution with 12 transmitters (receivers) using the traffic aggregation approach. Table 2 shows the virtual topology routing assignments obtained by solving

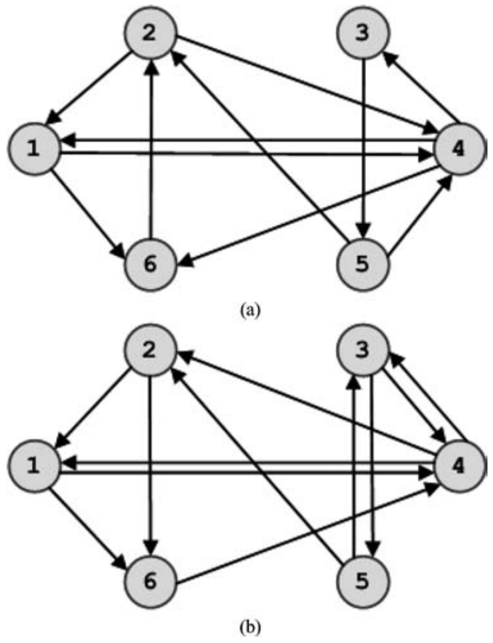


Figure 23. Comparison of the ILP solution and the heuristic approach: an illustrative example, (a) Results obtained by solving the ILP optimization problem with hop-length limit 3. (b) Results obtained from traffic aggregation approach.

the ILP formulation and the traffic aggregation heuristic algorithm.

Observations

Figure 23 also shows the similarity between the virtual topology design obtained from solving ILP formulation and the heuristic approach. More specifically, the ILP formulation tends to keep bigger requests on shorter paths in virtual topology and tries to integrate smaller traffic streams onto bigger bundles. The ILP approach provides an optimal solution by performing exhaustive search among all possible routes. The traffic aggregation heuristic algorithm also yields a very good solution in this example by just perfonning a local search, which takes much less computation time. However, as an approximate approach, the traffic aggregation heuristic cannot guarantee optimality.

The integration of the traffic helps to reduce the number of transmitters and receivers. On the other hand, it also introduces overhead traffic to the network and impacts, the resource utilization. Besides, it adds potential delays to the requests, which have been reallocated to take multiple hops in the virtual topology. From Table 2, it can be observed that the average hop-length in the ILP solution is 80/50 = 1.6. The average hop-length in the traffic aggregation heuristic is 77/50 = 1.54, whereas without grooming, given enough resource, the minimum average hop-length is 1. The more one saves on transmitters and receivers, the longer the average hop-length is, accordingly the longer is the average delay. This trade-off is an unavoidable one that would have to be faced.

The ILP approach becomes unmanageable quickly as the size of the network increases. The reason is that the number of all possible arcs in the corresponding fully con-

Table 1. Requests Matrix for A six-Node Network

	1	2	3	4	5	6
1	0	3	3+1+1	12+12	3+1+1	12+12
2	12+12+12+3	0	3	1+3	0	1+1+12
3	3	1	0	12+12	3+1+1	0
4	3	12	3+12+3+3	0	1	3+1+1+12
5	3	3+12	12	0	0	3+1
6	1+3	12	0	3+12	0	0

Table 2. Resulting Routes in Virtual Topologies

Node pair	Requested capacity	ILP formulation Route on VT	Traffic aggregation Route on VT
1-2	3	1-6-2	1-4-2
1-3	5	1-4-3	1-4-3
1-4	24	1-4	1-4
1-5	5	1-4-3-5	1-4-3-5
1-6	24	1-6	1-6
2-1	39	2-1	2-1
2-3	3	2-4-3	2-1-4-3
2-4	4	2-4	2-6-4
2-6	14	2-4-6	2-6
3-1	3	3-5-2-1	3-4-1
3-2	1	3-5-2	3-4-2
3-4	24	3-5-4	3-4
3-5	5	3-5	3-5
4-1	3	4-1	4-1
4-2	12	4-6-2	4-2
4-3	21	4-3	4-3
4-5	1	4-3-5	4-3-5
4-6	17	4-6	4-2-6
5-1	3	5-4-1	5-2-1
5-2	15	5-2	5-2
5-3	12	5-2-4-3	5-3
5-6	4	5-4-6	5-2-6
6-1	4	6-2-1	6-4-1
6-2	12	6-2	6-4-2
6-4	15	6-2-4	6-4

Table 3. Traffic Matrix for a 10-Node Network

	1	2	3	4	5	6	7	8	9	10
1	0	5	8	11	3	8	5	7	8	10
2	3	0	8	4	0	5	1	2	3	1
3	9	3	0	7	3	10	11	8	0	6
4	6	0	8	0	2	5	5	2	1	1
5	0	6	10	4	0	2	11	10	5	2
6	11	3	4	4	3	0	2	6	8	3
7	0	2	10	2	11	5	0	1	6	0
8	0	5	6	2	3	1	11	0	5	0
9	4	5	11	8	8	2	3	1	0	5
10	0	9	9	3	7	10	1	2	1	0

nected network increases dramatically as the number of nodes increases. The performance of the IP traffic aggregation heuristic approach is studied in terms of wavelength utilization in the following section.

EXAMPLE 2: LIGHT TRAIL NETWORK ARCHITECTURE FOR GROOMING

The *Light trail* architecture concept has been proposed as a novel architecture designed for carrying finer granularity IP traffic. A light trail is a unidirectional *optical trail* between the start node and the end node. It is similar to a lightpath, with one important difference that the intermediate nodes can also access this unidirectional trail. More-

Table 4. Resulting Light Trails $Tl_{\max} = 4$

No.	Light Trails	Hops	Accommodated s—d Pairs	Load
1	2, 3, 4, 7, 9	4	(3,7) (3,4) (2,7) (2,9) (4,9)	23
2	3, 2, 6, 8, 10	4	(2,6) (2,8) (2,10) (3,6) (3,8) (3,10)	32
3	4, 3, 2, 1, 5	4	(4,1) (4,3) (4,5) (3,5) (1,5) (3,1) (2,1)	34
4	4, 7, 6, 8, 10	4	(6,8) (6,10) (4,6) (4,7) (4,8) (4,10)	22
5	5, 1, 2, 3, 4	4	(1,2) (1,3) (1,4) (5,2) (5,3) (5,4) (2,4)	48
6	5, 1, 6, 7, 9	4	(1,7) (1,9) (6,9)	21
7	5, 1, 6, 8, 10	4	(1,8) (1,10) (1,6) (5,6)	27
8	5, 8, 7, 9, 10	4	(9,10) (8,9) (5,9) (5,8) (5,7) (7,9) (5,10)	44
9	9, 7, 4, 3, 2	4	(9,2) (9,3) (9,4) (7,3) (7,2) (3,2)	39
10	9, 7, 6, 1, 5	4	(7,6) (6,5) (9,1) (9,6) (6,1)	25
11	10, 8, 6, 2, 3	4	(10,3) (10,2) (8,3) (8,2) (6,3) (6,2) (2,3)	44
12	10, 8, 6, 7, 4	4	(10,6) (10,4) (7,4) (6,4) (6,7) (8,4) (8,6) (8,7)	35
13	10, 9, 7, 8, 5	4	(10,9) (10,8) (10,7) (10,5) (9,8) (9,7) (9,5) (8,5) (7,8) (7,5)	38

Table 5. Local Best-Fit: Resulting Light Trails $Tl_{\max} = 4$

No.	Light Tails	Hops	Accommodated s—d Pairs	Load
1	3, 2, 6, 8, 10	4	(3,10) (2,10) (2,8) (3,2) (6,10) (2,6) (6,8) (3,8) (3,6)	44
2	10, 8, 6, 2, 3	4	(10,3) (8,6) (10,8) (6,2) (6,3) (8,2) (8,3) (2,3) (10,2)	47
3	1, 6, 2, 3, 4	4	(1,4) (6,4) (2,4) (1,2) (3,4) (1,3) (1,6)	47
4	1, 5, 8, 10, 9	4	(1,9) (10,9) (5,10) (1,5) (8,9) (5,9) (1,8) (1,10)	41
5	2, 6, 8, 7, 9	4	(2,9) (2,7) (6,7) (7,9) (6,9) (8,7)	31
6	3, 4, 7, 8, 5	4	(3,5) (7,8) (4,5) (4,8) (8,5) (4,7) (7,5) (3,7)	38
7	4, 3, 2, 6, 1	4	(4,1) (2,1) (4,6) (4,3) (3,1) (6,1)	42
8	4, 7, 9, 10	3	(4,10) (4,9) (9,10)	7
9	5, 8, 7, 4, 3	4	(5,3) (8,4) (7,4) (5,4) (5,8) (7,3)	38
10	9, 7, 6, 2, 1	4	(9,1) (9,6) (7,2) (9,7) (7,6) (9,2)	21
11	9, 7, 4, 3	3	(9,3) (9,4)	19
12	9, 10, 8, 5	3	(9,5) (9,8) (10,5)	16
13	10, 8, 6, 7, 4	4	(10,4) (10,6) (10,7)	14
14	1, 5, 8, 6, 7	4	(1,7) (5,6) (5,7)	18
15	5, 1, 2	2	(5,2)	6
16	6, 1, 5	2	(6,5)	3

over, light trail architecture, as detailed later on, does not involve any active switching components. However, these differences make the light trail an ideal candidate for traffic grooming. In light trails, the wavelength is shared in time by the nodes on the light trail. Medium access is arbitrated by a control protocol among the nodes that have data ready to transmit at the same time. In a simple algorithm, upstream nodes have higher priorities over the nodes downstream.

Current technologies that transport IP-centric traffic in optical networks are often too expensive, because of their reliance on an expensive optical and opto-electronic approach. Consumers generate diverse granularity traffic, and service providers need technologies that are affordable and seamlessly upgradable. The exclusion of fast switching at the packet/burst level, combined with the flexible provisioning for diverse traffic granularity, makes the light trails an attractive option to conventional circuit-and burst-switched architecture.

Light Trail

A four-node light trail is depicted in Fig. 24. The light trail starts from Node 1, passes through Node 2, Node 3, and ends at Node 4. Each of the nodes 1, 2, and 3 are allowed to transmit data to any of their respective downstream nodes without a need for optical switch reconfiguration. Every node receives data from upstream nodes, but only a requested destination node(s) accepts the data packets, whereas other nodes ignore them. An out-of-band control signal carrying information pertaining to the setup, tear

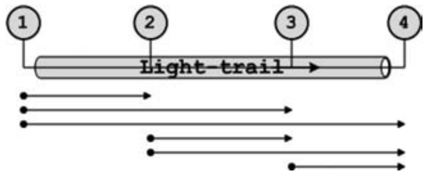


Figure 24. A light trail and possible traffic streams.

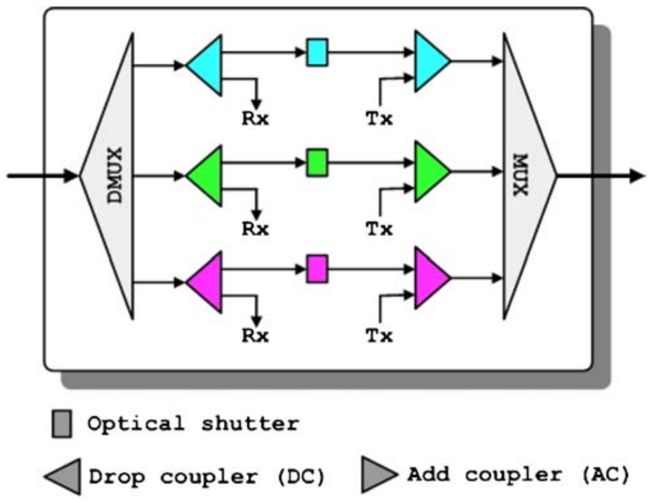


Figure 25. An example node structure in the light trail framework.

down, and dimensioning of light trails is dropped and processed at each node in the light trail. As a light trail is unidirectional, a light trail with N_T nodes can be used by up to $\frac{N_T(N_T-1)}{2}$ optical connections along the trail. The six paths for the four-node light trail are shown in Fig. 24.

Node Structure

Figure 25 provides a node structure that can be deployed in a light trail framework. In the figure, the multiple wavelengths from the input link are demultiplexed and then sent to corresponding light trail switches. A portion of the signal power is directed to the local receiver, the remaining signal power passes through an optical shutter. Such a shutter can be realized using various technologies as an AOTF (Acousto-Optic Tunable Filter). Thus, a node receives signals from all wavelengths. If a particular wavelength is not being used by an upstream node (incoming fiber has no signal), the local host can insert its own signal; otherwise, it does not use the trail. The local signal is coupled with the incoming signal as shown in the figure.

Figure 26 depicts a connection of a four-node light trail in a network and the corresponding ON/OFF switch configurations. The direction of communication is from Node 1 to Node 4. The optical shutter is set to the OFF state at the start and end nodes of the light trail such that the signal is blocked from traveling further. For an intermediate node along the light trail, the optical shutter is set to the ON state to allow the signal to pass through the node. A unidirectional light trail is thereby obtained from the start node to the end node. No switch reconfiguration is required after the initial light trail setup. From the power loss within the light trail, which mainly comes from the power splitting at each node, the length of a light trail is limited and is estimated in terms of hop-length. The expected length of a light trail is 4 to 6 hops (51).

Light Trail Characteristics

As no need exists to dynamically configure any switches when using light trails to carry IP bursts, it leads to an excellent provisioning time. Moreover, the major advantage of using light trails for burst traffic is the improved wavelength utilization. Utilization here is defined as the ratio of capacity used over time for actual data transmission to the total reserved capacity.

Multicasting in the optical layer is another salient feature of light trail architecture. Nodes in a light trail can send the same quanta of information to a set of downstream nodes without a need for a special processing or control arbitration.

In general, a light trail offers a technologically exclusive solution that enables several salient features and is practical. It exhibits a set of properties that distinguishes and differentiates it from other platforms. The following four characteristics are key properties:

- The light trail provides a way to groom traffic from many nodes to share a wavelength path to transmit their sub-wavelength capacity traffic.
- The light trail is built using mature components that are configured in such a way that allows extremely fast provisioning of network resources, which allows for dynamic control for the fluctuating bandwidth requirements on the nodes connected to a trail.
- The light trail offers a method to group a set of nodes at the physical layer to create optical multicasting, which is a key feature for the success of many applications.
- The maturity of components leads to the implementation of the light trail in a cost-effective manner resulting in economically viable solutions for mass deployment.

Light trail architecture brings up various issues in designing optical networks for transporting IP-centric traffic. These questions are as follows:

- How to is a set of light trails identified at the design phase for the given traffic?
- How hard is this problem?
- What are the new constraints introduced by the light trail architecture?
- How good can wavelength utilization be in light trail networks?
- How is survivability achieves in light trail networks?

These questions are answered in the following discussion.

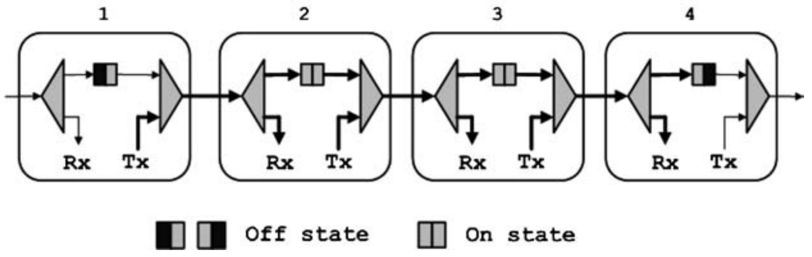


Figure 26. An example node configuration in the light trail framework.

Light Trail Design

To identify a set of light trails to carry the given traffic is one key issue in setting up light trails in a WDM network. The performance of the light trail in terms of wavelength utilization depends on the locations of the light trail. The goal of the design problem therefore is to develop an effective method to groom traffic in a light trail architecture and to come up with a set of light trails. The light trail design problem is stated as follows:

Given a graph $G(V, E)$, where $|V| = N$, and traffic matrix $D_{N\times N}$, define a minimum number of light trails to carry the given traffic.

The design problem is expected to be a hard problem. The approach to identify a set of light trail to be set up in a network presented here consists of two steps. The first step is called the *traffic matrix preprocessing* step. As stated earlier, because of the power losses on the lines, a long light trail may not be advisable. The length of a light trail is limited and is specified in terms of hop-length, denoted by $Tl_{\max}$. A reasonable hop-length of a light trail is set to 5. Therefore, in the first step, a single long hop traffic is recursively divided into multiple hops.

The second step is to formulate the design problem and to solve it as an ILP optimization problem, for a given network topology and refined traffic matrix obtained from step one. The objective is to find a minimum number of light trails that is required for the system to carry the traffic.

Step I: Traffic Matrix Preprocessing. In preprocessing of given traffic matrix, a single long hop traffic is divided into multiple hops to satisfy the hop-length constraint. For a given network physical topology $G(V, E)$, with N nodes and E links, one can apply Dijkstra' s shortest path algorithm to find the shortest path between all *s-d* pairs. This step results into a distance matrix $H_{N\times N} = \{h_{st}\}$, where h_{st} denotes the physical distance from node s to node t.

The length of a light trail is a main constraint from the loss both at nodes and over the links. Let $Tl_{\max}$ be the maximum length of a light trail. For traffic between an *s-d* pair (i,j), where $h_{st} > Tl_{\max}$, it is not possible to accommodate this traffic on a direct light trail. Thus, this traffic needs to go through multiple hops. Here one light trail is counted as one "hop", which necessitates the first step in this approach, namely *traffic matrix preprocessing*.

Let $D_{N\times N} = \{d_{st}\}$ denote the estimated traffic matrix. Traffic matrix preprocessing returns a modified traffic matrix that satisfies $D_{N\times N} = \{\, d_{st} : h_{st} \le Tl_{\max}, \quad \forall\, d_{st} > 0 \,\}$. Figure 27 provides pseudo-code for the traffic matrix preprocessing algorithm.

In this step, traffic on *s-d* pair (s, t) with $h_{st} > Tl_{\max}$ is reallocated on multiple hops. The goal is to find a node n such that path from node s to node n forms the first hop, which is less than $Tl_{\max}$ in distance. A next intermediate node n is found recursively for a new source node. Among all possible intermediate nodes, n is chosen to be as close to destination node t as possible, as shown in step 1 in Fig. 27. This is done to reduce the number of hops that the original traffic has to take.

After the preprocessing of the traffic matrix, each nonzero element in modified traffic matrix would have corresponding distance that is less than $Tl_{\max}$, the maximum length allowed for a light trail.

Step II: ILP Formulation: Given the network topology $G_p(V, E)$, and modified traffic matrix obtained from *Step* I, the next step is to list all possible paths within the hop-length limit for each *s-d* node pair, which can be accomplished by applying a *breath first search* for each node. These eligible paths form a set of *all possible light trails*. Among all possible choices, the next step is to choose an optimal set of paths to form the light trail network, such that the total number of light trails is minimized. This problem is formulated as an ILP optimization problem. It is also assumed that each request cannot be divided into different parts and transferred separately.

For the given directed graph $G_p(V, E)$, $N = |V|$, let LT be set of all the possible light trails within hop-length limit $Tl_{\max}$ and Let $T = 1, 2, \ldots, |LT|$ be the number assigned to each light trail in the LT.

Let C denote the full-wavelength capacity, represented as an integer that is a multiple of the smallest capacity requests. The smallest capacity request is denoted as 1. The integer entry in traffic matrix $D_{N\times N}$, represented by d_{st}, denotes the requested capacity from node s to node t in the units of the smallest capacity request.

A single fiber network with fractional wavelength capacity is considered. Hence, $d_{st} \le C$. In the absence of wavelength converters, the wavelength continuity constraints must be satisfied for light trail networks. The grooming helps to increase the wavelength utilization and reduces the total number of wavelengths that is required to satisfy the traffic needs. The following notations are used in problem formulation.

Variables.

- μ_{st}^{τ}: (binary variable) Route indicator takes the Value of 1 if request (s, t) takes light trail τ; zero otherwise. This also implies that nodes s and t are on trail τ and

```
INPUT: Graph $G = (V, E)$ and a traffic matrix $D_{N\times N}$.
OUTPUT: Rearranged traffic matrix $D_{N\times N}$ and the distance matrix $D_{N\times N}$.
ALGORITHM:
Step 0: Apply Dijkstra's shortest path algorithm, calculate distance matrix $D_{N\times N}$.
While ( find $(s,t) : d_{st} > 0,\ h_{st} > Tl_{max}$ )
{
   1) Pick an intermediate node $n$:
      $n = arg\ min_{v\in V}\{d_{vt} | d_{sv} \le Tl_{max}\}$;
   2) Update traffic matrix $D_{N\times N}$:
      a) $d_{sn} \leftarrow d_{sn} + d_{st}$;
      b) $d_{nt} \leftarrow d_{nt} + d_{st}$;
      c) $d_{st} \leftarrow 0$.
}
```

Figure 27. Light trail establishment step 1: Traffic matrix preprocessing.

s is t' s upstream node.

- δ^τ: (binary variable) Light trail usage indicator takes value of 1 if trail τ is used by any request; zero otherwise.

ILP Formulation.

- Objective:

$$\min \sum_{\tau} C_\tau \times \delta^\tau \tag{12}$$

When C_τ = 1, the objective is to minimize the number of light trails that is required in the network. When C_τ is defined as the *hop-length* of light trail τ, the problem becomes to minimize the total wavelength-links in the networks, which represent the total reserved capacity in the networks. This can be used to optimize the wavelength capacity utilization, although that might consume more light trails.

- *Assignment constraint*: Each request is assigned to one and only one light trail.

$$\sum_{\tau} \mu_{st}^{\tau} = 1 \quad \forall (s,t) : d_{st} \in D, d_{st} > 0 \tag{13}$$

- *Light trail capacity constraint*: The aggregated request capacity on a light trail should not exceed the full-wavelength capacity.

$$\sum_{(s,t)} \mu_{st}^{\tau} d_{st} \le C \tag{14}$$

- *Light trail usage constraint*: If any of the *s-d* pair is assigned on light trail τ, δ^τ is set to 1; otherwise, if none of the *s-d* pairs picked light trail τ, $\delta^l = 0$. Recall that δ^τ is a binary variable.

$$\delta^\tau \ge \mu_{st}^{\tau} \quad \forall\ (s,t) : d_{st} \in D \tag{15}$$

$$\delta^\tau \in \{0, 1\} \tag{16}$$

Solution Considerations

The light trail design is a challenging problem for the following reasons.

First, to use a wavelength fully, one would like to *groom* near full-wavelength capacity traffic onto the wavelength. This is similar to a normal traffic grooming problem, which is often formulated as a *knapsack problem* and is known to be an NP-complete problem. However, it might be infeasible to simply set up a light trail for any set of traffic requests that add up to C. For example, given that $d_{12} + d_{13} + d_{16} = C$, it might not be possible to establish the desired light trail because of the physical hop-length constraint. As a matter of fact, the light trail hop-length limit introduces complexity to the problem.

Second, the ILP formulation of the light trail design problem is similar to the *bin packing* problem, which is an NP-hard problem. However, if light trails are treated as the "bins," and elements in the given traffic matrix as the "items" in the bin packing problem, this problem differs from a normal bin packing problem because of a potential physical route constraint that an item cannot be put in any of the given bins but only a subset of the bins. More specifically, an *s-d* pair can be assigned to the routes that satisfy 1) nodes s and t belong to the route and 2) node s is the upstream node of node t along the route. Hence, the approximate algorithms for solving normal bin packing problems cannot be directly applied here for solving the light trail design problem.

Light Trail Design: Heuristic Approaches

As the study of Reference 52 proves that the light trail design problem is NP-hard, the following heuristic algorithms for light trail design is proposed. It is well known that the first-fit and best-fit are two common and effective heuristic algorithms for solving bin packing problems. In the following, the best-fit algorithm is used to solve the light trail design problem.

The Best-Fit Approach. Recall that, after traffic matrix preprocessing, each request in the newly obtained traffic matrix satisfies the light trail hop-length limit; that is, the shortest hop-length for each *s-d* pair is no greater than $Tl_{\max}$.

The goal of the second step is to identify a set of light trails for carrying the given traffic. To do this, first pick the *s-d* pair that has the longest distance in the distance matrix H_{st}. A light trail between this *s-d* pair is eventually required.

Once an *s-d* pair with the longest physical hop-length is found, the *head* and *tail* of a light trail is decided. The goal now is to find the *best* eligible light trail between these two end nodes, which is analogous to fully packing a "bin" in the bin packing problem. Two subproblems need to be solved. First, selection of a path (within the hop-length limit) between these two nodes is required. Second, assignment of requests to this light trail needs to be identified.

To find the best light trail between the known *head* and *tail* nodes, an exhaustive search among all possible paths between the two nodes is performed. *Best-fit* here tries to pick up the path between the head and the tail nodes that is the *best* among all paths available between the *head* and the *tail* nodes. This search; is still *local* therefore, the final results might not be globally optimal.

For each eligible path between the known *head* and *tail* nodes, all possible *s-d* pairs along this path are sorted according to their required capacities, before the routing decision is made. There are two different ways of packing them onto a path rather than doing it randomly. One is to allocate the smallest requests first, which is called the *increasing packing order,* and the other way is to allocate the biggest requests first, which is called *decreasing packing order*.

- Increasing packing order tries to allocate finer requests first, so that the number of requests that can be packed onto this path is maximized. Some capacity might still be left on this light trail, but that is not sufficient for the next smallest request. This approach grooms as many requests as possible onto the light trail, thereby leaving the rest of the network with fewer number of requests that still need to be allocated. The expectation is that this contributes to the saving on the total number of light trails that are needed in the network. However, for each light trail, the packing might not be the most efficient or the spare capacity might not be minimized.
- Decreasing packing order tries to allocate bigger requests first and leaves the light trail with minimum spare capacity. However, as the big requests are allocated first, the total number of requests that can be carried by the light trail might be smaller than that of the allocation in the *Increasing packing order.* Therefore, it could leave more requests unallocated in the network and more light trails might need to be set up later on in order to carry all requests. The spare capacity on each light trail is minimized in this approach at the time of allocating the capacity.

It is not clear which approach works better and always gives the minimum number of light trails required in the network. It depends on the traffic patterns. A preferred approach is to try both and choose the one that yields a better solution for given data.

Algorithm Design. For the given graph, all possible paths for each *s-d* pair can be computed. The paths information is stored appropriately. The data structure called $KSPath[N][N][NRoute_{\max}]$ contains the path information for each route in the network.

For efficient usage, paths are sorted according to their physical hop-length, such that *KSPath[head][tail][1]* contains the shortest path information (*hop-length, intermediate nodes along this path*) **head** to **tail**, and so on.

Figure 28 gives the pseudo-code of the best-fit algorithm. In this pseudo-code, *seq* is used to denote a route among all valid routes from which *head* and *tail* are chosen to be the trail. Also notice that only sub-wavelength level requests are considered here. Therefore, by default, a shortest path is chosen as the light trail to carry a given request if no better path can be found. That is, initially, *seq* = 1.

When there is a tie in route selection, the path that can accommodate most requests is chosen. It is possible to design and apply different criteria. As mentioned, sorting *AllRequest*[ ] in different ways yields different algorithms, namely, *best-fit decreasing packing order* and *best-fit increasing packing order*.

Discussions. The proposed heuristic algorithm has two steps, as shown in Fig. 27 and 28. Both the first step and the second step would need the information of paths between each *s-d* pairs. Therefore, one can first find out all possible paths for each *s-d* pairs. The worst-case complexity of the exhaustive searching for each *s-d* pair is $O(N^3)$, The total running time for finding all possible routes is $O(RN^3)$, where R is the number of *s-d* pairs (requests). In fact, instead of searching for all paths, it is preferable to search among the K-shortest path with K being big enough. This could reduce the complexity to $O(N(E + N\log N + KN))$ for all node pairs, which may be a promising choice for big networks.

In best-fit packing of Step 2, for each *s-d* pair, the best-fit route is chosen among all K paths. For path τ with n_τ nodes, there are a maximum of $t = (n_\tau - 1) + (n_\tau - 2) + \cdots + 1 = O(n_\tau^2)$ *s-d* pairs, where n_τ is bounded by $Tl_{\max}$. Hence, $t = O(Tl_{\max}^2)$. The sorting takes *O(tlogt)* loops, and packing takes another t loops. Thus the total complexity is *O(tlogt)* loops for each path. There are K paths, and the same procedure is performed on the selected best-fit path. Therefore, a total of $O(K(t\log t)) = O(K(Tl_{\max}^2 \log Tl_{\max}))$ loops is needed for each *s-d* pair. At least one *s-d* pair is eliminated from matrix R in Fig. 28 in each step and the program stops when R is empty.

Algorithm Performance. To evaluate the performance of the above ILP formulations and heuristic algorithms, experiments are performed on a physical topology given in Fig. 29. To simplify the problem, it is assumed that each physical link is bidirectional with the same length.

Table 3 gives a randomly generated traffic matrix for this example. The integer numbers indicate the requested capacity in a unit of OC-1 (51.84 Mbps). An entire wavelength capacity is OC-48. As aforementioned, only the fractional wavelength capacity is considered for traffic grooming in light trail networks. Intuitively, if every *s-d* pair requires a capacity greater than half of the full wavelength capacity, no two requests can be groomed on a light trail. Thus, it is assumed that most *s-d* pairs request a small fractional capacity of the full wavelength channel. Hence, integer numbers between 0 and 11 are randomly generated as requested capacities in the experiments. The resulting traffic matrix is shown in Table 3.

```
INPUT: Graph $G = (V, E)$, the rearranged traffic matrix $D_{N\times N}$ and distance matrix $H_{N\times N}$.
OUTPUT: A collection of light trail.
ALGORITHM:
Initializations: $d = 0$, $R = \{(m, n) : d_{m,n} > 0\}$.
Do {
  1) $(m, n) = arg\ max\{h_{m,n} : (m, n) \in R\}$.
     $head = m$, $tail = n$ .
  2) $Trail_{cap} = d_{m,n}$, $newstream = Trail_{cap}$,
     $best = 0$, $seq = 1$.
  3) for($\tau = 1; \tau \le NRoute_{max}; \tau + +$)
     if($KSP[head][tail][\tau].length \le Tl_{max}$)
       a) Copy all s-d pairs along path $KSP[head][tail][\tau]$ that need to be allocated
          to array $AllRequest[\,]$.
          The length of $AllRequest[\,]$ is known and denoted by $NSD$;
       b) Sort $AllRequest[\,]$ according to the capacities;
       c) for($tmp = 1;\ tmp \le NSD;\ tmp + +$)
          if ($newstream + AllRequest[tmp].cap \le C$)
                $newstream = newstream + AllRequest[tmp].cap$;
       d) if ($newstream > best$)
          {
                $best = newstream$;
                $seq = \tau$;
          }
  4) Copy all s-d pairs along path $KSP[head][tail][seq]$ that need to be allocated to
     array $AllRequest[\,]$.
     The length of $AllRequest[\,]$ is known and denoted by $NSD$;
  5) for($tmp = 1;\ tmp \le NSD;\ tmp + +$)
     if ($newstream + AllRequest[tmp].cap \le C$)
     {
           $Trail_{cap} = Trail_{cap} + AllRequest[tmp].cap$;
           $d_{AllRequest[tmp].src,AllRequest[tmp].dst} = 0$;
     }
} While ($R \neq \Phi$)
```

Figure 28. Light trail design step 2: best first approach.

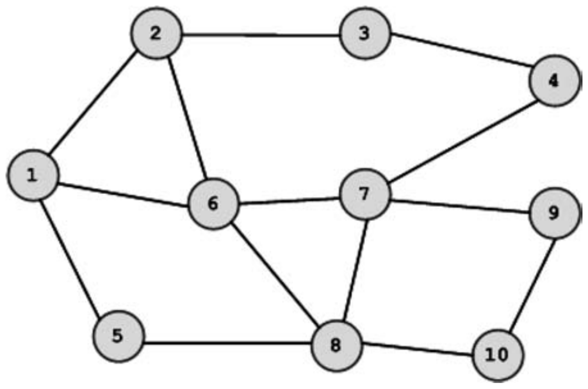


Figure 29. A 10-node example network.

The CPLEX Linear Optimizer 7.0 is used to solve the ILP formulation proposed. It is assumed that each candidate path can be used once; that is, $u = 1$. Assume that the hop-length limit $Tl_{\max} = 4$, from the topology it is observed that all *s-d* pairs have paths within this hop-length limit. Hence, the *traffic matrix preprocessing* does not make any change in the given traffic matrix.

Table 4 presents the results obtained by solving the ILP formulation with hop-length limit $Tl_{\max} = 4$. It is observed that $W = 4$ is sufficient on each link, although no constraint is imposed on the number of wavelengths.

Table 4 shows the 13 light trails that are needed to carry the given traffic. The traffic assignment obtained from solving the ILP formulation is also listed. For each light trail, the summation of all traffic it carries is calculated and shown in the right-most column in Table 4.

Table 5 depicts the results from solving the local best-fit heuristic algorithm proposed above. In this example, local best-fit increasing packing approach requires 16 light trails.

Discussions. An observation from the optimal solutions obtained by solving ILP formations is that only the longest candidate paths are chosen as light trails, because only the number of light trails is being minimized. The program stops searching further once the number of light trails does not decrease, even though it is possible to substitute some light trails with the other shorter paths.

The problem becomes unmanageable in case of the ILP approach as the problem size increases. In such a scenario, the use of relaxation techniques would be a preferred choice. When the traffic is uniform or the variation among different requests are small enough that they can be approximately treated as uniform traffic, $D_{N\times N} = \{d_{s,t} = \overline{d} \mid \forall (s,t)\}$. LP relaxation is a very effective means for obtaining fast solutions, which can be achieved by modifying the light trail capacity constraint in the ILP formulation as follow. The rest of the formulation remains the same.

$$\sum_{(s,t)} \mu^{\tau}_{s,t} \leq \lfloor C/\overline{d} \rfloor \tag{17}$$

$$0 \leq \delta^{\tau} \leq 1 \tag{18}$$

$$0 \leq \mu^{\tau}_{s,t} \leq 1 \tag{19}$$

In this formulation, the coefficient matrix of variables is totally unimodular. Hence, the LP relaxation still yields integer solutions. This effect is for the same reason as noted earlier in an LP to ILP in earlier article. Thus, this formulation can be applied to solve the light trail design problem where the traffic requests have similar capacities.

SUMMARY

The network design deals with the interconnection of various nodes and how to transmit information from one node to another. We have addressed the issues in network design. Four important factors, network topology, transfer technologies, network management and control techniques, and cost were identified and discussed. We also discussed analysis methods using graph theoretic models for a network and issues in topology design and optimization. We also introduced performance metrics such as blocking probability, throughput, and delay and how to account for them in the design of a network. An important issue to such design is fault tolerance. Two example designs, a WDM-based optical fiber network and a light trail network architecture, were presented considering all the factors together, demonstrating the concepts presented, and analyzing the trade-offs in the design, and the methods to resolve them.

BIBLIOGRAPHY

1. Benes, V. E. *Mathematical Theory of Connecting Networks and Telephone Traffic*; Academic: New York, 1965.
2. Schwartz, M. *Computer Communication Network Design and Analysis* Prentice-Hall: Englewood Cliffs, NJ, 1977.
3. Lawler, E. *Combinatorial Optimization: Networks and Matroids*; Holt, Rinehart & Winston: New York, 1985.
4. Bertsekas, D.; Gallager, R. *Data Networks*; Prentice-Hall: Englewood Cliffs, NJ, 1987.
5. Stallings, W. *Data and Computer Communication*, Macmillan: New York, 1997.
6. Kershenbaum, A. *Telecommunications Network Design Algorithms* McGraw-Hill: New York, 1993.
7. Tanenbaum, A. S. *Computer Networks*; Prentice-Hall: Englewood Cliffs, NJ, 1996.
8. Harary, F. *Graph Theory*; Addison-Wesley, Reading, MA, 1969.
9. Bellman, R. On a Routing Problem. *Q. Appl. Math*. 1958, **16**,pp 87–90.
10. Girard, A. *Routing and Dimensioning in Circuit-Switched Networks*; Addison-Wesley: Reading, MA, 1990.
11. Dijkstra, E. W. A Note on Two Problems in Connection with Graphs, *Numerische Mathematik*. 1959, **1**, pp 269–271.
12. Tarjan, R. E. Data Structures and Network Algorithms, *Society for Industrial and Applied Mathematics*, Philadelphia, PA, 1983.
13. Prim, R. C. Shortest Connection Networks and Some Generalizations. *BSTJ*, 1957, **36**, pp 1389–1401.
14. Dreyfus, S. E. An Appraisal of Some Shortest-Path Algorithms. *Oper. Res.*, 1969, **17**, 395–412.
15. Ford, L. R.; Fulkerson, D. R. *Flows in Networks*; Princeton University. Press; Princeton, NJ, 1962.
16. Hadley, G. *Linear Programming*; Addison-Wesley: Reading, MA, 1962.
17. Dantzig, G. B. *Linear Programming and Extensions*; Princeton Univ. Press: Princeton, NJ, 1963.
18. Frank, H.; Frisch, I. T.; Chou, W. Topological Considerations in the Design of the ARPA Computer Network; *Proc. Conf. Rec., 1970 Spring Joint Computer Conf. AFIPA Conf.*; AFIPS Press, 1970: Mantvale, NJ. 1970.
19. Gerla, M.; Kleinrock, L. On the Topological Design of Distributed Computer Networks. *IEEE Trans. Commun.*, 1977, **COM-25**, pp 48–60.
20. Gerla, M.; J. A. S., Monteiro; Pazos, R. Topology Design and Bandwidth Allocation in ATM Nets. *IEEE JSAC.*, 1989, **7**, pp 1253–1262.
21. Chattopadhyay, N. G.; Morgan, T. W.; Raghuram, A. An Innovative Technique for Backbone Network Design. *IEEE Trans. SMC* 1989, **19** (5):pp 1122–1132.
22. Somani, A. K. Design of an Efficient Network; *Proc. 7th International Parallel Processing Symposium*, Newport Beach, CA, April 13–16, 1993, pp. 413–418.
23. Feng, T.-Y. A Survey of Interconnection Networks. *Computer*, 1981, **14**, pp 12–27.
24. Leiserson, C. Fat-Trees: Universal Network for Hardware-Efficient Supercomputing. *IEEE Trans. Comput.*, 1985, **C-34**, pp 892–901.
25. Sullivan, H.; Bashkow, T. R. A Large Scale, Homogeneous, Fully Distributed Parallel Machine, I; *Proc. 4th Annu. Symp. Comput. Arch.*; 1977, pp 105–117.
26. Valiant, L. G. A Scheme for Fast Parallel Communication, *SIAM J. Comput.*, 1982, **11**, pp 350–361.
27. Choi, S. B.; Somani, A. K. The generalized hyper-cube; *Proc. ICPP-90*;August 1990. pp I/372–I/375.
28. Choi, S. B.; Somani, A. K. Rearrangeable Hypercube Architecture for Routing Permutations. *J. Parallel Distributed Computing*. In Press.
29. Arun K., Somani *et al.* Proteus System Architecture & Organization, *Proc. 5th International Parallel Processing Symposium*; 1991, pp 287–294.

30. Katseff, H. P. Incomplete Hypercube. *IEEE Trans. Comput.*, 1988, **37**, pp 604–607.
31. Somani, A. K.; Thatte, S. The Helical Cube Network. *Networks*, 1995, **26**, pp 87–100.
32. Gavish, B.; Neuman, I. Routing in a Network with Unreliable Components. *IEEE Trans. Commun.* 1992, **40**, pp 1248–1258.
33. Grover, W. D. The Self-Healing Network: A Fast Distributed Restoration Technique for Networks Using Digital Cross-Connect Machines; *Proc. IEEE Globecom*;Dec. 1987, pp. 28.2.1–28.2.6.
34. Allen, A. O. *Probability, Statistics, and Queuing Theory*; Academic Press: New York, 1978.
35. Kleinrock, L. *Queuing Systems Volume 1: Theory*, Wiley-Interscience, New York, 1975.
36. Kleinrock, L. *Queuing Systems Volume 1: Computer Application*; Wiley-Interscience, New York, 1980.
37. Woodruff, G. M.; Kositpaiboon, R. Multimedia Traffic Principles for Guaranteed ATM Network Performance. *IEEE JSAC*, 1990, **8**, pp 437–446.
38. Yeh, Y. S.; Hluchyj, M. G.; Acampora, A. S. The knockout Switch: A Simple Architecture for High Performance Packet Switching. *IEEE JSAC*, 1988, **SAC-5**, pp 1264–1273.
39. Yee, J. R.; F. Y. S. Lin A Routing Algorithm for Virtual Circuit Data Networks with Multiple Sessions Per O-D Pair. *Networks*, 1992, **22**, pp 185–208.
40. Song, K.; Somani, A. K. Modeling and Design of Dependable High Speed Information Networks. *Int. Assoc. Sci. Technol. Develop. J.* 1997.
41. Nishida, T.; Miyahara, H. Fault Tolerant Packet Switched Network Design Using Capacity Augmentation; *Proc. IEEE INFOCOM '88*.
42. Wu, T.-H.; Kong, D. T.; Lau, R. C. An Economic Feasibility Study for a Broadband Virtual Path SONET/ATM Self-Healing Ring Architecture. *IEEE JSAC* 1992, **10**, pp 1459–1473.
43. Lazzr, A. A.; Pacifici, G.; White, J. S. Real-Time Traffic Measurements on MAGNET II. *IEEE JSAC* 1990, **SAC-8**, pp 467–483.
44. Mahony, M.; Simeonidou, D.; Hunter, D. Tzanakaki, A., The Application of Optical Packet Switching in Future Communication Networks, *IEEE Commun. Mag.*, 2001, pp 128–135.
45. Guillemot, C. *et al.*, Transparent Optical Packet Switching: The European ACTS KEOPS Project Approach, *J. Lightw. Technol.*, 1998, **16**, pp 2117–2134.
46. Yamada Y., *et al.*, Optical Ouput Buffered ATM Switch Prototype Based on FRONTIERNET Architecture, *IEEE J. Select. Areas Commun.*, 1998, **16**, pp 1298–1307.
47. Rekhter, Y.; Davie, B.; Katz, D.; Rosen, E.; Swallow, G. Cisco Systems Tag Switching Architecture Overview; *Network Working Group Request for Comments:2105*; 1997.
48. Viswanathan, A.; Feldman, N.; Boivie, B.; Woundy, R. ARIS: Aggregrate Route-Based IP Switching; *IETF Internet Draft*;Mar. 1997.
49. Katsube, Y.; Nagami, K.; Esaki, H. Toshiba's Router Architecture Extensions for ATM Overview, *Network Design Group Request for Comments: 2098*;Feb. 1997.
50. Ahuja, R. K.; Magnanti, T. L.; Orlin, J. B. *Network Flows: Theory, Algorithms, and Applications*; Prentice Hall, Englewood Cliffs, NJ, 1993.
51. Clouqueur, M.; Grover, W. D. Quantitative Comparison of End-to-end Availability of Service Paths in Ring and Mesh-Restorable Networks; *Proceedings of the 19th Annual National Fiber Optics Engineers Conference (NFOEC 2003)*; Orlando, FL, September 2003. 7–11.
52. Fang, J.; He, W.; Somani, A. K. IP Traffic Grooming in Light Trail Optical Networks. *IEEE J. Select. Areas Commun.*, In press.

ARUN K. SOMANI
Iowa State University