Report Number: WUCS-TM-92-05

1992-02-01

Algorithms for Designing Nonblockings Communication Networks with General Topologies

J. Andrew Fingerhut

A framework is given for specifying nonblocking traffic requirements in a connection-oriented communications network. In this framework, connections may be form one point to one other point, or they may involve multiple points. Different connections may have different data rates. The communication networks are constructed from switches (or nodes) and trunks, which connect pairs of switches. This framework is intended to model Asynchronous Transfer Mode (ATM) networks and traffic. Several computational problems are formulated, each intended to find a minimum cost configuration of switches and trunks which satisfy given traffic requirements. Efficient algorithms have been found for some problems,... **Read complete abstract on page 2.**

Algorithms for Designing Nonblockings Communication Networks with General Topologies

J. Andrew Fingerhut

Complete Abstract:

A framework is given for specifying nonblocking traffic requirements in a connection-oriented communications network. In this framework, connections may be form one point to one other point, or they may involve multiple points. Different connections may have different data rates. The communication networks are constructed from switches (or nodes) and trunks, which connect pairs of switches. This framework is intended to model Asynchronous Transfer Mode (ATM) networks and traffic. Several computational problems are formulated, each intended to find a minimum cost configuration of switches and trunks which satisfy given traffic requirements. Efficient algorithms have been found for some problems, and computational hardness results for others.

Algorithms for Designing Nonblocking Communication Networks with General Topologies

J. Andrew Fingerhut

WUCS-TM-92-05

February 1993

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

This work was supported by the National Science Foundation, Bell Communications Research, Bell Northern Research, Digital Equipment Corporation, Italtel SIT, NEC, NTT and SynOptics.

February 22, 1993

Department of Computer Science
Campus Box 1045
Washington University
One Brookings Drive
St. Louis, MO 63130-4899

Abstract

A framework is given for specifying nonblocking traffic requirements in a connection-oriented communications network. In this framework, connections may be from one point to one other point, or they may involve multiple points. Different connections may have different data rates. The communication networks are constructed from switches (or nodes) and trunks, which connect pairs of switches. This framework is intended to model Aynchronous Transfer Mode (ATM) networks and traffic. Several computational problems are formulated, each intended to find a minimum cost configuration of switches and trunks which satisfy given traffic requirements. Efficient algorithms have been found for some problems, and computational hardness results for others.

This work is supported by the National Science Foundation, Bell Communications Research, Bell Northern Research, Digital Equipment Corporation, Italtel SIT, NEC, NTT, and SynOptics.

1. Introduction

The communication network design problem studied here is: given a collection of nodes (or switches), the cost of building communication links of various bandwidths between the nodes, traffic requirements for the network, and possibly other constraints, choose which bandwidth to install between each pair of nodes such that the traffic requirements and other constraints are satisfied, and the total cost is minimized.

Many variations of this problem have been studied by a variety of researchers (see references in Section 5). Most other work considers the traffic requirements to be static and between pairs of nodes only (point-to-point). This work is distinguished by its characterization of traffic requirements as sequences of requests to add and remove connections, each of which may involve two or more nodes (multipoint). The network is a candidate solution if no request to add a connection is ever refused, i.e., it is nonblocking.

This way of specifying traffic requirements is adapted from research on designing nonblocking multirate switching fabrics (e.g., Melen and Turner [MT89, MT90]). In that work, the topologies of the networks studied are highly structured. Here we desire nonblocking networks with arbitrary topology.

Some generalizations to their model are made which further restrict the traffic that can be offered to the network (point-to-point restrictions, as described in Section 2.3). This is done for two reasons. One is that it is easy to include, and could lead to cheaper networks. Another reason is that suitable choices for these restrictions (μ-bounded) show that this method of specifying traffic requirements is more general than in previous work.

It is intended that such network designs will be useful for fast packet or asynchronous transfer mode (ATM) networks, for which little is known about the traffic patterns of users. Nonblocking network designs provide strong guarantees for quality of service.

This paper is structured as follows. Section 2 defines the model we use to describe communication networks and traffic requirements, and defines what we mean by a nonblocking network. Section 3 reviews some previous results of work done with fixed traffic requirements, as opposed to nonblocking traffic requirements. Section 4 presents results that have been obtained so far on finding efficient algorithms for the nonblocking network design problem. Section 5 discusses other work known to the author that has some relationship to that here. Section 6 gives ideas for ways in which this work can be extended. Finally, section 7 gives proofs for theorems contained in section 4.

2. Definitions and Problem Statement

The following material was also presented by Fingerhut [Fin91], although there the network links and routes were represented by undirected graphs and edges. The model presented here uses directed graphs, and is not only a more accurate model of reality, but also simplifies the solution for alternate path routing (section 4.2).

2.1. Networks, connection requests, and connections

A *network* is a directed graph $G = (V, E)$ along with a function $cap : E \to \mathcal{N}$, where V is the set of nodes (or vertices), E is the set of links (or edges), $cap(e)$ is the bandwidth, or capacity, of link e, and $\mathcal{N}$ denotes the set of natural numbers.

A *connection request* (or *call*) is a triple $q = (src, dest, req\text{-}rate)$, where $src(q) \subseteq V$ are the sources of information flow, $dest(q) \subseteq V$ are the destinations of flow, and $req\text{-}rate(q) \in \mathcal{N}$ is the desired rate of the connection. Either or both of the sets $src(q), dest(q)$ must contain exactly one node. If both do, then the request is called *point-to-point.* If $|dest(q)| > 1$, then the request is called a *one to many multipoint request.* If $|src(q)| > 1$, then the request is called a *many to one multipoint request.*

A *connection* (or *route*) is a pair $r = (conn\text{-}links, conn\text{-}rate)$. *Conn-links*$(r)$ is a set of links in E which forms a tree. That is, if all of the edges E are considered as undirected edges, then the edges in E with their incident nodes form a connected, acyclic graph. A *point-to-point* connection is a directed path from a source node to a destination node. A *one to many multipoint* connection is a tree with the source node as a root, destination nodes as either leaves or internal nodes of the tree, and all edges directed away from the source. A *many to one multipoint* connection is a tree with the destination node as a root, source nodes as either leaves or internal nodes of the tree, and all edges directed towards the destination. $conn\text{-}rate(r)$ is the bandwidth which is used on each link of the connection.

A connection r *realizes a request* q under one of several conditions. If q is point-to-point, then r realizes q if $conn\text{-}links(r)$ is a directed path from $src(q)$ to $dest(q)$. If q is a one to many multipoint request, then r realizes q if $conn\text{-}links(r)$ is a directed tree with root $src(q)$, all links are directed away from the root, all $dest(q)$ are in the tree, and all leaves are contained in $dest(q)$. If q is a many to one multipoint request, then r realizes q if $conn\text{-}links(r)$ is a directed tree with root $dest(q)$, all links are directed toward the root, all $src(q)$ are in the tree, and all leaves are contained in $src(q)$. In all cases, $conn\text{-}rate(r) = req\text{-}rate(q)$ should also hold.

A *state* of the network is a (multi)set of connections.

Given a network G with link capacities $cap : E \to \mathcal{N}$ and a state s, we define the *usage* and *available capacity* of a link e in state s to be

$$\begin{aligned} usage(s,e) &= \sum_{r \in s,\, r \text{ uses link } e} conn\text{-}rate(r) \\ avail(s,e) &= cap(e) - usage(s,e) \end{aligned}$$

A state s is *compatible with network* G if

$$(\forall e \in E)\,(usage(s,e) \leq cap(e))$$

In other words, every link is used in connections that have a total rate which is at most the link's capacity.

For example, the point-to-point request $(e, b, 3)$ is realized by the connection $(\{(e,a),(a,b)\},3)$, which is the dotted path superimposed on the network in Figure 1. Similarly, the one to

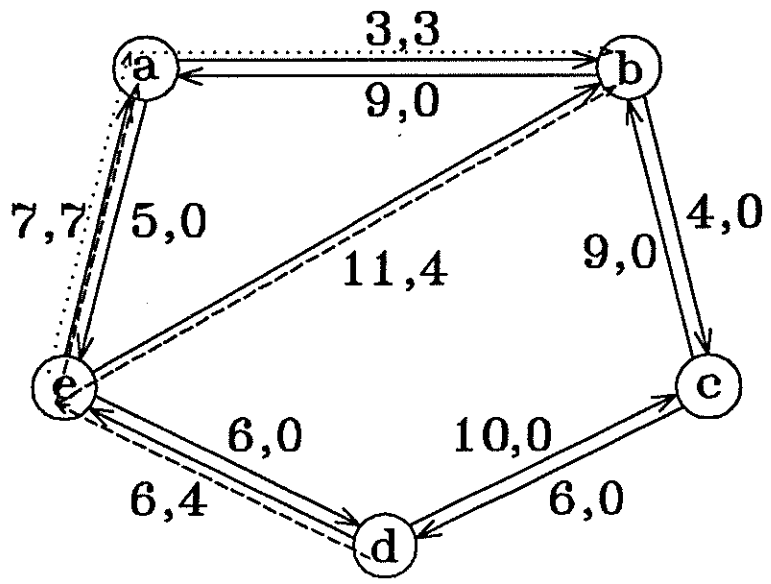

Figure 1: An example state

many multipoint request $(d, \{a, b, e\}, 4)$ is realized by the connection $(\{(d, e), (e, a), (e, b)\}, 4)$, which is the dashed tree in Figure 1. The links are labeled with two numbers, first capacity and then usage. Note that the link (e, a) has $usage(s, (e, a)) = 7 = cap(e, a)$ and $avail(s, (e, a)) = 0$. The link (e, b) has $usage(s, (e, b)) = 4$ and $avail(s, (e, b)) = 7$.

2.2. Fixed traffic requirements

Fixed traffic requirements describe an unchanging pattern of traffic among terminals. Although they are not suitable in all cases, they are useful for describing a single worst case load on the network. If the network can handle the worst case load, then it should also handle any load which is a "subset" of the worst case.

Fixed traffic requirements are specified by giving a (multi)set of connection requests REQ. Let $G = (V, E)$ be a network with link capacities cap. We say that G is *routable* for request set REQ if there is a state s compatible with G such that for each request $q \in REQ$, there is one and only one route $r \in s$ which realizes q.

2.3. Nonblocking traffic requirements

We will define a nonblocking network formally in a moment, but intuitively it is one which, given any state s it can get into and an additional connection request q, we can find a connection r which realizes q such that $s \cup \{r\}$ is a compatible state. It should be clear from this rough definition that unless there is some way to restrict the new requests that come in, any network with finite link capacities will eventually not be able to handle the new requests. We will restrict new requests by introducing the concept of termination capacity.

Every node v has a *source termination capacity* $\alpha(v) \in \mathcal{N}$ and a *destination termination capacity* $\omega(v) \in \mathcal{N}$. $\alpha(v)$ ($\omega(v)$) is a number which represents the maximum total rate of all connections in which a node may be a source (destination).

For example, if $\alpha(v) = 5$, then v may be a source simultaneously in connections with rates 1, 2, and 2, but then it could not be a source in any more connections until an existing connection is removed.

The source termination capacity $\alpha(u)$ may be interpreted as the bandwidth of the interface to the switch u from the collection of terminals connected to u (similarly for $\omega(u)$, but in the opposite direction). This interface has a finite bandwidth, and if it is ever all used, then it is impossible for any more connections to be made which have one of those terminals as a source. Note that additional connections may pass *through* the node, they just cannot start there. Also note that when we say a network is nonblocking, we mean that if a new request does not exceed any of the termination capacities, then there will be enough capacity on the links of the network to satisfy the request. From the point of view of a user operating a terminal, a request may block because other users connected to the local node through the same interface may be using all of the termination capacity. A network designer using the algorithms developed here may wish to vary the values of termination capacity to see how it affects the cost of the network. They may also wish to engineer the network to be nonblocking for lower values of α and ω than are actually present in the network. This could lower the cost of the network, but then the nonblocking guarantee only applies when the network operates within the designed values. Users could try to exceed those termination capacity values, and then blocking may occur. It is possible that some requests like this could succeed, and then other users operating within their termination capacities may block because someone else is "breaking the rules". This idea of a termination capacity was inspired by the "maximum port weight" β used by Melen and Turner [MT89, MT90].

An additional idea which may be useful in lowering the cost of network designs is *point-to-point restrictions*. It is also straightforward to add these restrictions to some of the network design algorithms efficiently. They are specified by giving a value $\mu(u, v) \in \mathcal{N}$ for each $u, v \in V$. The value $\mu(u, v)$ is the maximum total rate of all connections that may exist from u to v simultaneously.

Given a network G with termination capacities $\alpha : V \rightarrow \mathcal{N}$, $\omega : V \rightarrow \mathcal{N}$, point-to-point restrictions $\mu : V \times V \rightarrow \mathcal{N}$, and a (multi)set of connection requests REQ, define the *point-to-point usage*, *source usage*, and *destination usage* under requests REQ as

$$\begin{aligned}
\textit{pp-usage}(u, v, REQ) &= \sum_{q \in REQ,\, u \in src(q),\, v \in dest(q)} \textit{req-rate}(q) \\
\textit{source-usage}(u, REQ) &= \sum_{q \in REQ,\, u \in src(q)} \textit{req-rate}(q) \\
\textit{dest-usage}(u, REQ) &= \sum_{q \in REQ,\, u \in dest(q)} \textit{req-rate}(q)
\end{aligned}$$

A set of requests REQ is *compatible with traffic requirements* α, ω, μ if

$$(\forall u \in V)\ (\textit{source-usage}(u, REQ) \leq \alpha(u)) \wedge$$

$$(\forall u \in V)\ (\textit{dest-usage}(u, REQ) \leq \omega(u)) \wedge$$
$$(\forall u, v \in V)\ (\textit{pp-usage}(u, v, REQ) \leq \mu(u, v)) \quad (1)$$

That is, no node is involved in more requests than its termination capacity will allow, and no pair of nodes is involved in more requests than their point-to-point restriction will allow.

There are two restrictions on the values of α, ω, and μ which are sometimes useful. The first is the condition

$$(\forall u, v \in V)(\mu(u, v) \geq \min\{\alpha(u), \omega(v)\}) \quad (2)$$

It can be proven that when this condition holds, the first two lines of condition 1 imply the third line. Therefore the compatible sets of requests depend only upon the values of α and ω. When condition 2 holds, we will call the traffic requirements *termination capacity bounded*, or *α, ω-bounded.*

The second restriction is

$$(\forall u \in V)\ (\alpha(u) \geq \sum_{v \in V} \mu(u, v)) \wedge$$
$$(\forall u \in V)\ (\omega(u) \geq \sum_{v \in V} \mu(v, u)) \quad (3)$$

When this condition holds, the third line of condition 1 implies the first and second lines. Therefore the compatible sets of requests depend only upon the values of μ. When condition 3 holds, we will call the traffic requirements *point-to-point bounded*, or *μ-bounded.*

It is possible for traffic to be neither α, ω-bounded nor μ-bounded. This will be referred to later as the *general* case of nonblocking traffic requirements.

Some computational problems are made significantly easier when we restrict the types of connection requests that may be made. For example, we may restrict all requests to have the same rate. This is called the *single rate* case. If requests may be made with more than one rate, we call it the *multirate* case. In the multirate case, the complexity of the problem further depends on the set of allowable rates $\mathcal{W}$ that may be requested. The only case studied here is where $\mathcal{W}$ is a discrete set, and all rates are equal to an integer multiple of the minimum rate.

Note that we can handle problems where link capacities, termination capacities, and point-to-point restrictions are arbitrary real numbers. For the single rate case with rate r, just replace any of the above quantities x (i.e., the α, ω, and μ values) with the integer $\lfloor x/r \rfloor$. Then act as if the rate is 1. This "scales" all of the rates in the problem, and now we can think of every quantity in units of whole numbers of connections, rather than rates. The multirate case where all rates are integer multiples of the minimum rate r can be scaled similarly to the single rate case. The set of allowable rates should also be scaled, so that it would end up being a set of integers containing 1. The multirate case where more general rate sets are used is more complicated, and it is currently unknown how to handle such cases efficiently. See section 7.1.1 for more details.

Another restriction that may be made is to disallow multipoint requests. This is called the *point-to-point* case. If requests may be either point-to-point or multipoint we call it the *multipoint* case.

2.4. Nonblocking networks

One way of defining a nonblocking network is given below. It is based on a definition given by Pippenger [Pip82].

Consider a game between two players called the *blocker* and *nonblocker.* The parts of the game which are fixed and cannot be changed by either player are (1) a network G, link capacities $cap : E \to \mathcal{N}$, termination capacities $\alpha, \omega : V \to \mathcal{N}$, and point-to-point restrictions $\mu : V \times V \to \mathcal{N}$; (2) a deterministic connection selection algorithm A which, given the network description above, a state s, and a single connection request q, produces a set of connections R. For each $r \in R$, r must realize q and $s \cup \{r\}$ must be a compatible state for G (i.e., no link capacities exceeded). All of this is known completely by both players. The blocker will try to add and remove connection requests so that the nonblocker cannot realize the requests.

At the start of the game, both the set of connection requests REQ and the state s are empty. The blocker has the first move.

At the beginning of the blocker's move, s realizes REQ. The blocker either adds a single connection request q to REQ (this new set of requests $REQ \cup \{q\}$ must be compatible with G), or it removes a single request. It is then the nonblocker's move.

At the beginning of the nonblocker's move, s does not realize REQ. If a request was removed from REQ, then the nonblocker moves by removing the corresponding connection from s. If a request q was added, then the nonblocker performs algorithm A given s and q, producing a set of connections R. If R is empty, then the game is over and the blocker wins. If R is not empty, then the *blocker* selects one of these connections and adds it to s. If the game is not over, then it is the blocker's move again. It is critical that the blocker is the one to choose the connection from R.

If the blocker can make a sequence of moves and choices from R such that it eventually wins, then the network G is *blocking for routing algorithm* A, or simply *blocking.* If the nonblocker can prevent R from ever being empty, then the network is *nonblocking (for routing algorithm A).*

It should be clear that the only power the nonblocker has is given to it by the routing algorithm. A network with given link capacities may be blocking for one routing algorithm, but nonblocking for another.

2.5. Routing Algorithms

In this section we present several examples of routing algorithms for point-to-point connection requests (some have natural extensions to multipoint requests). In each case, the algorithm knows the network $G = (V, E)$ with link capacities cap, the current state s, and a point-to-point connection request $q = (u, v, \textit{req-rate})$. The algorithm must compute a set R of connections. Recall from the definitions that for a connection r, $s \cup \{r\}$ is compatible if and only if $(\forall e \in r)\ (\textit{req-rate} \leq \textit{avail}(s, e))$. It is implicitly assumed that any connections not satisfying this condition are removed from the set returned.

Fixed path routing (FP)

> FP has access to a table *path*. For each ordered pair of endpoints u, v, $path(u, v)$ is a directed path from u to v. Return $\{path(u, v)\}$.

Alternate Path Routing (AP)

> AP is similar to FP, but in this case each entry of the table contains two paths $prim(u, v)$ and $sec(u, v)$. If $s \cup \{prim(u, v)\}$ is a compatible state, then return $\{prim(u, v)\}$, otherwise return $\{sec(u, v)\}$.

3. Solved problems for fixed traffic requirements

All of the results in this section are presented in more detail by Fingerhut [Fin91]. Both network links and traffic are considered undirected in that paper, but similar results hold for directed links and traffic.

There are two major types of problems that have been studied for fixed traffic requirements. The first is a network analysis problem, in which we are given a network and traffic, and we simply wish to determine if the network can handle the traffic.

Network Analysis with Fixed Traffic Requirements

> INSTANCE: A network $G = (V, E)$ with link capacities $cap : V \rightarrow \mathcal{N}$. Fixed traffic requirements, given by a set of connection requests REQ.
>
> QUESTION: Is G routable for REQ? That is, find a state s compatible with G that realizes the connection requests in REQ, or determine that no such state exists.

The second is a network design problem, in which we are given traffic and the costs of building various links, and we wish to find the cheapest network which can handle the traffic.

Network Analysis with Fixed Traffic Requirements

> INSTANCE: Network $G = (V, E)$ (without link capacities). For each link $(u, v) \in E$, a function $cost_{(u,v)} : \mathcal{N} \rightarrow \mathcal{N}$, where $cost_{(u,v)}(x)$ is the cost of installing a link of capacity x from u to v. Fixed traffic requirements given by REQ.
>
> SOLUTION: A capacity $cap(u, v) \in \mathcal{N}$ for each $(u, v) \in E$. This assignment of capacity should make the network G routable for REQ.
>
> SOLUTION COST: The cost of the network is the sum of the costs of each link:
>
> $$\sum_{(u,v)\in E} cost_{(u,v)}(cap(u, v))$$
>
> OBJECT: Find a solution with minimum cost.

The complexity of the problem above depends on two kinds of restrictions. One is whether the connection requests REQ are restricted to be point-to-point, and the other is the kind of link cost functions *cost* allowed. Link cost functions considered here are

		Point-to-point	Multipoint
Analysis		NP-complete [Fin91, Sec. 4.1]	
Design	linear	P [Hu69]	NP-hard,2 [Fin91, Sec. 4.2]
	startup	NP-hard,2 in general P for a restriction [Fin91, Sec. 4.3]	
	both	NP-hard	

P – polynomial time (efficient) algorithm known
NP-hard,2 – NP-hard, but there is a known polynomial time algorithm to approximate within a factor of 2 of optimal

Table 1: Summary of results for fixed traffic requirements

(1) linear, where $cost_{(u,v)}(x) = b_{(u,v)} \cdot x$ and $b_{(u,v)}$ may be different for each u, v pair; (2) startup, where $cost_{(u,v)}(x) = a_{(u,v)}$ if $x > 0$, and 0 if $x = 0$; and (3) both, where $cost_{(u,v)}(x) = a_{(u,v)} + b_{(u,v)} \cdot x$ if $x > 0$, and 0 if $x = 0$. Some results are only known to hold for *symmetric* link cost functions, where $cost_{(u,v)}(x) = cost_{(v,u)}(x)$ for all u, v pairs and all x. Also, some results are only known to hold for symmetric traffic, where for all u, v pairs, there must be positive traffic from u to v if and only if there is positive traffic from v to u. All results hold for undirected networks and traffic, as presented by Fingerhut [Fin91].

The known results are summarized in table 1. Both NP-hard,2 results are only known to hold for symmetric link cost functions, and all results for startup costs are only known to hold for symmetric traffic as well. The ratio of the approximation algorithm for linear costs and multipoint traffic is as good as the ratio one can find for the Steiner tree problem on weighted undirected graphs. Several approximation algorithms are known which achieve a ratio of 2 [KMB81, WI88], Zelikovsky has found an algorithm which achieves a ratio of 11/6 [Zel91], and Berman and Ramaiyer have done further work in this direction [BR92].

4. Solved problems for nonblocking traffic requirements

There are several variations of the network design problem studied here. Here is the most general formulation.

Nonblocking network design with routing algorithm A

INSTANCE: Network topology $G = (V, E)$ (without link capacities). For each link $(u, v) \in E$, a function $cost_{(u,v)} : \mathcal{N} \rightarrow \mathcal{N}$, where $cost_{(u,v)}(x)$ is the cost of installing a link of capacity x from u to v. Traffic requirements are given by termination capacities $\alpha(v), \omega(v) \in \mathcal{N}$ for each $v \in V$, and point-to-point restrictions $\mu(u, v) \in \mathcal{N}$ for each $u, v \in V$.

SOLUTION: A capacity $cap(u, v) \in \mathcal{N}$ for each $(u, v) \in E$. This assignment of capacity should make the network nonblocking for routing algorithm A, where connection requests may be multirate and multipoint.

SOLUTION COST: The cost of the network is the sum of the costs of each link:

$$\sum_{(u,v)\in E} cost_{(u,v)}(cap(u,v))$$

OBJECT: Find a solution with minimum cost.

In each of the following subsections, we will present a more specific problem formulation with additional restrictions.

4.1. Results for fixed path routing

The results known for fixed path routing are divided on one major difference. This is whether the paths to be used are given as part of the problem instance, or they are to be determined by the algorithm.

4.1.1. Fixed path routing with paths given.

The following problem and the auxiliary results derived during its solution form the cornerstone of this work.

Network design with fixed path routing, fixed paths given

INSTANCE: Network topology $G = (V, E)$. For each link $(u, v) \in E$, a nondecreasing function $cost_{(u,v)} : \mathcal{N} \to \mathcal{N}$, where $cost_{(u,v)}(x)$ is the cost of installing a link of capacity x from u to v. Traffic requirements are given by termination capacities $\alpha(v), \omega(v) \in \mathcal{N}$ for each $v \in V$, and point-to-point restrictions $\mu(u, v) \in \mathcal{N}$ for each $u, v \in V$. For each $u, v \in V$, a directed path $path(u, v)$ from u to v.

SOLUTION: A capacity $cap(u, v) \in \mathcal{N}$ for each $(u, v) \in E$. This assignment of capacity makes the network nonblocking for routing algorithm FP with fixed paths $path$, where connection requests may be multirate and multipoint.

SOLUTION COST: The cost of the network is the sum of the costs of each link:

$$\sum_{(u,v)\in E} cost_{(u,v)}(cap(u,v))$$

OBJECT: Find a solution with minimum cost.

This problem can be solved efficiently.

THEOREM 4.1. *The problem above can be solved in $O(m \cdot maxflow(n, U))$ time, where m is the number of links used in fixed paths, n is the number of nodes, U is a parameter depending on the paths which is at most $n^2 + n$, and $maxflow(n, U)$ is the time required to solve a maximum flow problem [Tar83, Chapter 8] with n nodes and U edges.*

Proof: See Section 7.1 ∎

The best asymptotic bound for $maxflow(n, m)$ known to the author is $O(mn + n^{2+\epsilon})$, for any constant $\epsilon > 0$, given by King, Rao, and Tarjan [KRT92]. A relatively easy to implement algorithm described by Goldberg and Tarjan [GT88, Section 4] can solve the problem in $O(n^3)$ time.

A more efficient solution can be obtained if we restrict the paths to form a tree, and only allow α, ω-bounded traffic requirements.

THEOREM 4.2. *Let α, ω-bounded traffic requirements be given by α and ω, and suppose that the fixed paths path are all along a tree. Then the problem above can be solved in $O(n)$ time, where n is the number of nodes.*

Proof: See Section 7.2 ■

4.1.2. Fixed path routing with paths to be computed. It is currently unknown whether there are efficient algorithms to simultaneously determine the fixed paths and select the cheapest link capacities, given some traffic requirements. However, the problem of finding a tree network with minimum total capacity given α, ω-bounded traffic requirements is efficiently solvable.

Tree network design, minimize total capacity

INSTANCE: Network topology $G = (V, E)$. Traffic requirements are given by termination capacities $\alpha, \omega \in \mathcal{N}$ for each $v \in V$. ($\mu(u, v)$ values are assumed to be any values such that α, ω, μ are α, ω-bounded, so μ may be ignored.)

SOLUTION: A capacity $cap(u, v) \in \mathcal{N}$ for each $(u, v) \in E$, where the links with non-zero capacity form a tree. This assignment of capacity makes the network nonblocking for routing algorithm FP with fixed paths defined by the tree structure, where connection requests may be multirate and multipoint.

SOLUTION COST: The cost of the network is the sum of the *capacities* of each link:

$$\sum_{(u,v) \in E} cap(u, v)$$

That is, $cost_{(u,v)}(x) = x$ for all $u, v \in V$.

OBJECT: Find a solution with minimum cost.

For the above problem, there is a particularly easy solution.

THEOREM 4.3. *Every tree solution to the problem above costs at least as much as the tree with node C as the "center" and all other nodes connected directly to C. Here C may be any node with the maximum value of $\alpha(C) + \omega(C)$ among all nodes.*

Proof: See Section 7.3. ■

The center node C may be found in $O(n)$ time, and optimum link capacities may also be found in $O(n)$ time (by Theorem 4.2).

We conjecture that for an instance of the problem above, a tree solution is always optimum even if we allow solutions which are not trees. If this is true, then the problem is

			Restriction on Traffic Requirements		
Fixed paths given?	Form of fixed paths	Link cost functions	α,ω-bounded	μ-bounded	general
Y	any	nondec.	$O(m \cdot M(n,U))$ (Thm.4.1)	$O(m \cdot U)$	$O(m \cdot M(n,U))$ (Thm.4.1)
Y	tree	nondec.	$O(n)$ (Thm.4.2)		
Y	star tree	nondec.		$O(n^2)$	
N	any	linear	?	$AP(n,m)$	?
N	tree	cost = cap.	$O(n)$ (Thm.4.3)	$O(n \cdot M(n,m))$ [Hu74]	

Table 2: Summary of results for nonblocking network design with fixed path routing

still efficiently solvable when the restriction "where the links with non-zero capacity form a tree" is removed.

If we change the restriction on traffic requirements from α, ω-bounded to μ-bounded, and require that $\mu(u,v) = \mu(v,u)$ for all $u, v \in V$, then the problem is solvable in $O(n \cdot \mathit{maxflow}(n,U))$ time, where $U \leq n^2$ is the number of non-zero μ values. This was proven by Hu [Hu74].

The problem in the previous paragraph becomes NP-hard if the link cost functions are of the form $\mathit{cost}_{(u,v)}(x) = b_{(u,v)} \cdot x$, where each $b_{(u,v)} \in \mathcal{N}$ may be different from one node pair to another [GJ79, problem ND7, p. 207].

Note that when traffic requirements are μ-bounded and link cost functions are as in the previous paragraph, an optimum solution is not always a tree. The optimum solution is a solution in which every fixed path is a shortest path in the graph with edge lengths given by the $b_{(u,v)}$ values [Fin91, Hu69].

4.1.3. Summary of results for fixed path routing. Table 2 summarized the results known for designing nonblocking networks with fixed path routing. Here $M(n,m)$ represents the time required to find a maximum flow in a graph with n vertices and m edges, and $AP(n,m)$ represents the time required to find shortest paths between all pairs of nodes in a similar graph. Best known times for $M(n,m)$ were given in Section 4.1.1, and the best known time for $AP(n,m)$ is $O(n^2 \log n + nm)$ [FT87].

The $O(mU)$ result for μ-bounded traffic requirements holds because the instances of maximum flow created in the algorithm of Section 7.1 are simple and can each be solved in linear time (and need not even be explicitly constructed). Similarly, the $O(n^2)$ result holds because each maximum flow instance can be solved in linear time, which is $O(n)$ for all such instances.

The $AP(n,m)$ result is the same as that for the fixed point-to-point traffic requirements with linear costs given in Table 1. There is a close similarity between fixed point-to-point traffic requirements and μ-bounded nonblocking traffic requirements with fixed path routing. Intuitively, this is because the worst possible traffic in the nonblocking network can be exactly modeled by appropriate fixed point-to-point traffic requirements.

4.2. Results for alternate path routing with paths given

Let $prim(u,v)$ and $sec(u,v)$ be given for all $u,v \in V$ (each is a set of links). A simple path is a path which visits each switch at most once. It is not a good idea for a path from one switch to another to be anything other than a simple path, as it only wastes network resources.

Note that if $prim(u,v)$ is a simple path from u to v and $prim(u,v) - sec(u,v) = \emptyset$, then either $prim(u,v) = sec(u,v)$ or $sec(u,v)$ is not a simple path from u to v. If all paths are simple, this implies that either $prim(u,v) - sec(u,v) \neq \emptyset$ or $prim(u,v) = sec(u,v)$, for all $u, v \in V$.

It is unknown whether there are efficient algorithms for the general network design problem where alternate path routing is used. However, careful choices of restrictions yields a tractable problem. In particular, the following restriction on the primary and secondary paths will be useful:

$$(\forall u,v \in V)(\text{no link in } prim(u,v) - sec(u,v) \text{ is contained in any path except } prim(u,v)) \quad (4)$$

When condition (4) holds, it is useful to define:

$$\begin{aligned} E' &= \bigcup_{u,v \in V} (prim(u,v) \cup sec(u,v)) \\ p(u,v) &= prim(u,v) - sec(u,v) \\ s(u,v) &= sec(u,v) - prim(u,v) \\ \mathcal{P} &= \bigcup_{u,v \in V} p(u,v) \\ \overline{\mathcal{P}} &= E' - \mathcal{P} \end{aligned}$$

The solution found for the problem below has only been proven to work when link cost functions are nondecreasing and concave. A function $f(x)$ is concave on the interval $[a,b]$ if for all $x_1, x_2 \in [a,b]$ and $\lambda \in [0,1]$, $f(\lambda x_1 + (1-\lambda)x_2) \geq \lambda f(x_1) + (1-\lambda) f(x_2))$. That is, if we draw the curve of the function f on a graph and draw a straight line between any two points on the curve, then f lies on or above this straight line.

Network design with alternate path routing, paths given

INSTANCE: Network topology $G = (V,E)$. For each link $(u,v) \in E$, the cost function $cost_{(u,v)}(cap)$ is a nondecreasing and concave function of cap. Traffic requirements are given by termination capacities $\alpha(v), \omega(v) \in \mathcal{N}$ for each $v \in V$, and point-to-point restrictions $\mu(u,v) \in \mathcal{N}$ for each $u,v \in V$. For each $u,v \in V$, there are directed paths $prim(u,v)$ and $sec(u,v)$ from u to v. These paths satisfy condition (4).

SOLUTION: A capacity $cap(u,v) \in \mathcal{N}$ for each $(u,v) \in E$. This assignment of capacity makes the network nonblocking for routing algorithm AP with paths *prim* and *sec*, where connection requests must be single rate and point-to-point.

SOLUTION COST: The cost of the network is the sum of the costs of each link:

$$\sum_{(u,v)\in E} cost_{(u,v)}(cap(u,v))$$

OBJECT: Find a solution with minimum cost.

THEOREM 4.4. *The problem above can be solved in $O(2^d \cdot m \cdot maxflow(n,U))$ time, where $m = \left|\overline{\mathcal{P}}\right|$, n is the number of nodes, U is a parameter depending on the paths which is at most $n^2 + n$, and d is a parameter depending on the paths which is at most the number of u,v pairs such that $p(u,v) \neq \emptyset$.*

Proof: See Section 7.4. ■

We conjecture that the running time can be improved to $O(m \cdot maxflow(n,U) + 2^d \cdot U)$. The above theorem yields a polynomial time algorithm if we restrict the instances so that $d = O(\log n)$.

5. Other Related Work

Most work on network design known to the author ([MW84, Min89, GW90, KKG91, and references therein]) falls in the category of fixed point-to-point requirements. This would correspond roughly to our case where traffic requirements are μ-bounded. Variations in problems studied include: the set of link capacities allowed to be installed (discrete or continuous), different link (and sometimes node) cost functions, different ways of satisfying a single point-to-point requirement (e.g., single path, multiple paths with discrete choices for traffic splitting, multiple paths with continuous choices for traffic splitting), average packet delay constraints for packet networks, and reliability restrictions (e.g., network can still satisfy requirements when any single link fails). Most of these problems are NP-hard, and solution techniques are either heuristic or based on mathematical programming theory.

The current research is distinguished from all of the above in the way traffic requirements are specified. Instead of fixed point-to-point requirements, traffic is dynamic and restricted by the termination capacity and point-to-point restriction values (the α's, ω's, and μ's). We also consider multicast traffic. The emphasis in this work has been on finding efficient algorithms or proving intractibility.

The relation to Hu's work should be clear from the problem in Section 4.1.2. He considers only μ-bounded traffic requirements, however.

Gibbens and Kelly [GK90] consider a given network topology which is completely connected (links between all node pairs). The network traffic consists of stochastically arriving connection requests which are point-to-point and single rate. The routing algorithm is like the alternate path routing algorithm considered here, except that the primary path is the direct link between the pair of nodes (which exists, because of complete connectivity), and if that link has no bandwidth available, then a random two link path is chosen as the secondary path (independently for each connection request). They do not require that the network never block a request, only that the probability of blocking is low. This work is distinguished by its requirement that the network never block a call request. This alleviates the need to choose a probability model for the arriving connection requests.

Multicast routing algorithms are considered by Waxman [Wax88, Wax89] and Klein and Powell [KP91]. Klein and Powell consider a problem in which the network topology and link bandwidths are given, as well as a schedule of connection requests that will arrive in the future. Their goal is to choose routes for all requests so that none of them will block, and they give heuristics for doing so. Here we do not have a schedule of connection requests in advance, and we are designing the link capacities rather than working within the constraints of given capacities.

Waxman considers a model of traffic which is stochastic. There are several classes of requests, each with their own rate of arrival, average holding time, average number of endpoints, and bandwidth used. Endpoints can enter and leave connections dynamically during the lifetime of the connection. The routing algorithm used is one which returns the shortest path which has enough spare capacity, along with some variations of this. Waxman uses simulation to determine how these routing algorithms perform (in terms of blocking probability) in several particular network topologies, some randomly generated.

This work was influenced by that of Melen and Turner [MT89, MT90]. They consider some networks which have very regularly structured topologies, such as Beneš and Cantor networks. They introduce a parameter β which restricts the maximum total rate of all connections sharing the same input port or output port, and it is the same for all such ports. All links except the input and output ports have bandwidth 1. They present conditions relating β and other network parameters such that the network can perform as a multirate nonblocking connector (point-to-point connections) or distributor (multipoint connections). This work extends Melen and Turner's to include general network topologies.

6. Future Work

There are several solutions in section 4 which only solve special cases of the general problem. The author believes that more general versions can be solved. For example, the tree network design problem discussed in section 4.1.2 will also solve the more general problem in which the fixed paths are not restricted to be on a tree if, among all possible solutions, a tree solution were always optimal. This is conjectured to be true.

The work already done on alternate path routing suggests that more complex routing algorithms could very well cause the corresponding network design problems to be intractable. "More complex" in this context does not necessarily mean that the routing algorithm is hard to specify or compute. It means that its choice of routes is very dependent on the current state of the network, or that many choices for a feasible path are possible. For such routing algorithms, it may be intractible to determine simply whether a given choice of network topology *and link bandwidths* can block. In such cases, we desire algorithms which design networks guaranteed not to block, even though they may not have the minimum cost possible. The most desirable such algorithms would give solutions guaranteed to cost no more than a constant factor times more than an optimal solution (e.g., 2 times optimal).

In this work, we assume that a link from a node u to a node v may take any capacity desired. When building a network, however, capacity is installed by putting a number of cables between the nodes. The individual cables may have large capacities (e.g., 155 megabits/sec). The problem in section 4.1.1 can formulate this restriction by making the link cost function have discontinuous "jumps" in cost when a new discrete capacity is reached, because the algorithm works for any nondecreasing cost functions. None of the other algorithms can currently handle this discrete capacity restriction.

Another fact that is not modeled by any of this work is one relating to the operation of ATM networks Between nodes there is an integer number of trunks. When we wish to add a connection, the entire bandwidth of the connection must be assigned to a single trunk. This restriction will be called *individual trunking.* The model which has been used so far assumes that a connection's bandwidth may be divided arbitrarily among the trunks. This model will be called *aggregate trunking.*

In the individual trunking model, restricting the traffic to be single rate makes the problem no harder than the aggregate trunking model. They are the same. It is when multirate traffic is allowed that the problem changes significantly. The author believes that results can be found to solve problems using the individual trunking model.

7. Proofs

In the following, we use the notation $\alpha(X)$ to denote $\sum_{u \in X} \alpha(u)$ for any $X \subseteq V$. Similarly for $\omega(X)$. Also, we use $\overline{X}$ to denote $V - X$, for any $X \subseteq V$.

The computational problems and theorems have been restated in this section to aid the reader.

7.1. Proof of Theorem 4.1

An undirected version of this problem was considered by Fingerhut [Fin91].

Network design with fixed path routing, fixed paths given

INSTANCE: Network topology $G = (V, E)$. For each link $(u, v) \in E$, a nondecreasing function $cost_{(u,v)} : \mathcal{N} \to \mathcal{N}$, where $cost_{(u,v)}(x)$ is the cost of installing a link of capacity x from u to v. Traffic requirements are given by termination capacities $\alpha(v), \omega(v) \in \mathcal{N}$ for each $v \in V$, and point-to-point restrictions $\mu(u, v) \in \mathcal{N}$ for each $u, v \in V$. For each $u, v \in V$, a directed path $path(u, v)$ from u to v.

SOLUTION: A capacity $cap(u,v) \in \mathcal{N}$ for each $(u,v) \in E$. This assignment of capacity makes the network nonblocking for routing algorithm FP with fixed paths *path*, where connection requests may be multirate and multipoint.

SOLUTION COST: The cost of the network is the sum of the costs of each link:

$$\sum_{(u,v)\in E} cost_{(u,v)}(cap(u,v))$$

OBJECT: Find a solution with minimum cost.

THEOREM 7.1. *The problem above can be solved in $O(m \cdot maxflow(n,U))$ time, where m is the number of links used in fixed paths, n is the number of nodes, U is a parameter depending on the paths which is at most n^2+n, and $maxflow(n,U)$ is the time required to solve a maximum flow problem [Tar83, Chapter 8] with n nodes and U edges.*

7.1.1. Point-to-point requests only. The special case in which only point-to-point requests are allowed is handled first. This solution also works when multipoint requests are allowed, as shown in section 7.1.2.

The first fact to note about fixed path routing is *the state of the network is a function of the current set of requests.* This may sound trivial, but note that if the routing algorithm gives a choice of routes to use, the state of the network depends not only on the set of requests, but also on the choices made. Even if the routing algorithm does not return more than one route, it may return a different route depending on the current state (e.g., the alternate path routing algorithm AP). In this case, the current state depends not only on the current set of requests, but also the *sequence* of add and delete connection requests.

Assume for the moment that all links which are used in some fixed path have a very large capacity which could not be exceeded even if every connection used it (e.g., the value $\min\{\alpha(V),\omega(V)\}$ would be sufficiently small). Let f be the function that maps request sets to states (this function depends only upon the table of fixed paths *path*). Given any set of requests REQ, we can determine the state of the network $f(REQ)$, and therefore the usage of each link e, $usage(e, f(REQ))$. The set of all request sets $ALLREQ$ which are compatible with α, ω, μ is finite. Therefore we can determine the maximum possible usage of any given link e:

$$maxusage(e) = \max_{REQ\in ALLREQ} usage(e, f(REQ))$$

Note that this value is independent of any other link capacity. It depends only upon *path*, α, ω, and μ.

If we assign any link capacities cap that satisfy $cap(e) \geq maxusage(e)$ for all links e, then the network will be nonblocking. If any link e has capacity less than $maxusage(e)$, the network can block. Therefore, when the link cost functions $cost_{(u,v)}$ are *any* nondecreasing functions of capacity, the link capacities $cap(e) = maxusage(e)$, for all $e \in E$, will be the cheapest solution to the computational problem described in Section 4.1.

		v				
		a	b	c	d	e
u	a		ab	abc	aed	ae
	b	ba		bc	bcd	bae
	c	cdea	cb		cd	cde
	d	dea	deb	dc		de
	e	ea	eb	ebc	ed	

$path(u, v)$

		v				
		a	b	c	d	e
u	a		4	1	4	4
	b	4		1	4	7
	c	2	6		3	6
	d	4	5	1		5
	e	2	2	1	2	

$\mu(u, v)$

u	$\alpha(u)$	$\omega(u)$
a	4	4
b	7	7
c	6	1
d	5	5
e	2	8

Figure 2: Problem instance for fixed path routing with paths given

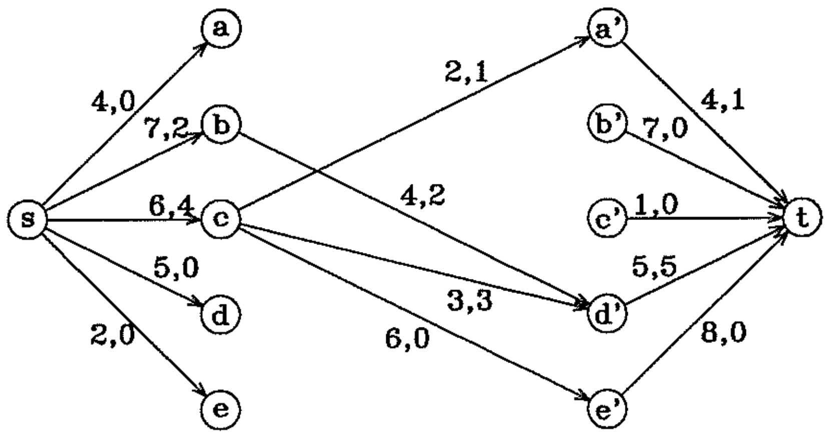

Figure 3: $END_{(c,d)}$ for the instance in Figure 2

Let $e \in E$ be any link of the network. Computing *maxusage*(e) by generating every compatible request set in *ALLREQ* would be computationally prohibitive. It may be computed more efficiently as follows.

Construct a directed graph $END_e = (V_{END_e}, E_{END_e})$, where

$$\begin{aligned} V_{END_e} &= \{u, u' : u \in V\} \cup \{s, t\} \\ E_{END_e} &= \{(s, u) : u \in V\} \cup \{(u', t) : u \in V\} \cup \{(u, v') : path(u, v) \text{ uses link } e\} \end{aligned}$$

Assign a capacity to each arc as follows:

$$cap\,(u, v) = \begin{cases} \alpha(x) & \text{if } (u, v) = (s, x) \text{ for some } x \in V \\ \omega(x) & \text{if } (u, v) = (x', t) \text{ for some } x \in V \\ \mu(x, y) & \text{if } (u, v) = (x, y') \text{ for some } x, y \in V \end{cases}$$

For example, for the instance described in Figure 2, the graph $END_{(c,d)}$ and its arc capacities are given in Figure 3. The arc capacities are the numbers before each comma. Ignore the numbers after each comma for the moment.

Let $q = (u, v, req\text{-}rate)$ be a point-to-point connection request. If $path(u, v)$ contains link e, then END_e contains an arc from u to v'. We can model the request q and its corresponding connection $r = (path(u,v), req\text{-}rate)$ by adding flow *req-rate* along the arcs (s,u), (u,v'), and (v',t). If $path(u,v)$ does not contain link e, then do not add any flow to END_e. To model a set of compatible connection requests REQ, simply add all of the flows induced by each individual request. For example, for the previously given instance and the compatible request set $\{(b,d,2),(c,d,3),(c,a,1),(a,b,4)\}$, the flow assigned to the graph $END_{(c,d)}$ is shown in Figure 3. The numbers before the commas are the arc capacities, and the numbers after the commas are the arc flows.

The usage of link (c,d) in this state is 6, which is also the value of the flow (i.e., the total flow out of the source node s).

Note that given any compatible set of point-to-point requests REQ, the flow induced on an endpoint graph END_e is a network flow from s to t, and this flow does not exceed any of the arc capacities. This follows from the definition of a compatible request set, and the construction of END_e.

Suppose we restrict ourselves to the single rate case, or multirate cases where the set of allowable rates is one in which all rates are an integer multiple of the smallest rate. Then we may scale all of the rates and all of the values α, ω, and μ so that they are integers (see end of section 2.3). After scaling, any integer network flow on END_e from s to t is induced by at least one compatible request set. It is easy to verify that the value of such a flow is exactly the usage of link e in state $f(REQ)$ for any such compatible request set REQ. Therefore, $maxusage(e)$ may be computed by finding a maximum value integer flow from s to t in END_e.

The statement of the theorem claims that this network design problem may be solved in $O(m \cdot maxflow(n,U))$ time, where m is the number of links used in fixed paths, n is the number of nodes, and U is a parameter depending on the paths which is at most n^2+n. Here we see that computing $maxusage(e)$ for a single link e takes $maxflow(2n+2,U)$ time, where U is the number of arcs in END_e. U can be at most $2n+n(n-1) = n^2+n$ as claimed. The $2n+2$ problem can be resolved by noting that $maxflow(2n+2,U) = O(maxflow(n,U))$ for any polynomial time maximum flow algorithm. The design algorithm must also construct END_e for each e contained in some fixed path, but this time is dominated by the time to compute maximum flows.

Note that if we allow more general sets of allowable rates in the multirate case, the problem may not be so easy. For example, if all rates are integer multiples of some rate r, but r is not itself an allowable rate, we can still perform the "scaling" as before, but now the set of allowable rates is a set of integers which does not contain 1. In this case, it is still true that any compatible set of requests induces an integer flow on END_e, but it is no longer always true that every integer flow on END_e is induced by a compatible set of requests. If the set of allowable rates is also finite, then the problem of finding the maximum usage of a single link e can be shown to be (weakly) NP-hard by a simple reduction from the problem INTEGER KNAPSACK [GJ79, problem MP10, p. 247].

7.1.2. Handling multipoint requests. We only considered point-to-point requests and connections in the previous section. In this section we show how to extend the fixed path

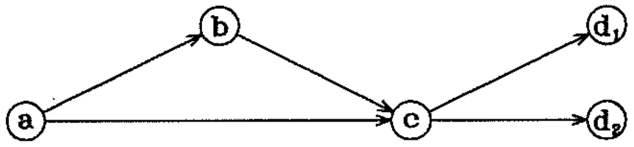

Figure 4: Simple network to demonstrate dynamic multipoint connections

routing algorithm to handle multipoint requests in such a way that the solutions produced by the algorithm in the previous section will still be nonblocking. We consider only one to many multipoint requests. Many to one multipoint requests can be handled similarly.

Consider a one to many multipoint request $q = (u, \{v_1, \ldots, v_k\}, \textit{req-rate})$. Assume for the moment that we route this request by using bandwidth *req-rate* on all links in the set $\bigcup_{i=1}^{k} \textit{path}(u, v_i)$, which is not necessarily a tree. Consider any link e in this set and its graph END_e. This request and its connection can be modeled by adding flow *req-rate* on the arcs (s, u), (u, v_i), and (v_i, t), where $v_i \in \{v_1, \ldots, v_k\}$ is some vertex (chosen arbitrarily) for which the arc (u, v_i) is in END_e.

With this induced flow, it is still true that for any set of compatible requests REQ, the flow is a flow from s to t that does not exceed any arc's capacity. Also, there is still at least one compatible request set REQ corresponding to each integer flow. Therefore, the same method of determining $\textit{maxusage}(e)$ as for point-to-point requests still works.

To make the multipoint connections into trees, just pick an arbitrary tree which is a subset of $\bigcup_{i=1}^{k} \textit{path}(u, v_i)$ that contains a path from u to each $v_i \in \{v_1, \ldots, v_k\}$. Note that some of the links for which we have reserved bandwidth for this connection may not be used at all.

The intuitive idea for why this works is that termination capacity of the nodes is "used up" more quickly for multipoint requests than for point-to-point requests. Link capacities, however, are "used up" at the same rate, or less.

The results of this section also imply that we may add and remove destinations from a one to many multipoint connection dynamically, without removing the entire connection and replacing it with a new one. However, some rearrangement of the connection may be necessary when removing an endpoint, depending on the structure of the fixed paths.

For example, consider the network of Figure 4, where $\textit{path}(a, d_1) = acd_1$ and $\textit{path}(a, d_2) = abcd_2$. Suppose there is first a connection request $(a, d_1, 1)$, which can be satisfied by the path acd_1. Now suppose that we wish to add the destination d_2 to this connection. One way would be to remove the existing connection and then build a connection for the request $(a, \{d_1, d_2\}, 1)$. Another would be to add links to the existing connection so that the result is a tree which satisfies the connection request $(a, \{d_1, d_2\}, 1)$.

The following way makes as small a change as possible to the existing connection. Step backwards along the path $\textit{path}(a, d_2)$ until a switch is reached which is already in the connection. In this example, add the link (c, d_2) and then stop, because c is in the connection. This method will work in general for adding any destination to an existing one-to-many connection, because the result is always a tree, and a subset of the links $\bigcup_{i=1}^{k} \textit{path}(u, v_i)$.

However, suppose that we now wish to remove d_1 as a destination from this existing connection. In the general case, we could try to update the connection in an analogous way to adding a destination. That is, work backwards from the destination to be removed until a switch is reached which must remain in the connection (because of other remaining destinations). If we tried that in the example, we would remove the link (c, d_1) from the tree, ending up with the path $acd_2 \neq abcd_2 = path(a, d_2)$.

As far as the existing connection requests are concerned, there is only $(a, d_2, 1)$. However, the link (a, c) is used in the current state, and so future (a, d_1) connection requests may block. Therefore, the minimal modification to the connection, described in the previous paragraph, does not guarantee nonblocking behavior. It is necessary to make certain that the links in the connection are a subset of $\bigcup_{i=1}^{k} path(u, v_i)$ at all times. This may require removing links from and adding links to the existing connection when a destination is removed. In our example, we must remove links $(a, c), (c, d_2)$ and add links $(a, b), (b, c)$.

Note that if the network is a tree, this problem does not occur, and removing endpoints can be done by removing links but not adding any.

7.2. Proof of Theorem 4.2

THEOREM 7.2. *Let α,ω-bounded traffic requirements be given by α and ω, and suppose that the fixed paths path are all along a tree. Then the problem above can be solved in $O(n)$ time, where n is the number of nodes.*

Call a link e a *directed cut edge* if the removal of e leaves no directed path from any $u \in X$ to any $v \in \overline{X}$, for some set of nodes $X \subset V$.

LEMMA 7.1. *Let α,ω-bounded traffic requirements be given by α and ω, and let A be any routing algorithm. If e is a directed cut edge, then its usage in any state will be at most $\min\{\alpha(X), \omega(\overline{X})\}$. Furthermore, there is a state in which the usage will be that much.*

Proof: Very similar to the proofs of Lemmas 5.1 and 5.2 given by Fingerhut [Fin91]. ■

Let T be any spanning tree of the nodes of G, and let u, v be adjacent nodes in T. Both of the links (u, v) and (v, u) are directed cut edges, so the minimum necessary capacities on these links are given by

$$cap(u, v) = \min\{\alpha(X), \omega(\overline{X})\} = \min\{\alpha(X),\ W - \omega(X)\}$$
$$cap(v, u) = \min\{\alpha(\overline{X}), \omega(X)\} = \min\{A - \alpha(X),\ \omega(X)\}$$

where $A = \sum_{v \in V} \alpha(v)$, $W = \sum_{v \in V} \omega(v)$, and X is the set of nodes in the component of $T - \{u, v\}$ which contains u.

An efficient algorithm for finding all link capacities is:

```
Compute $A = \sum_{u \in V} \alpha(u)$ and $W = \sum_{u \in V} \omega(u)$
while there are at least 2 nodes left in $T$ do
        Pick any leaf $u \in T$ and let its neighbor be $v$
        $cap(u,v) := \min\{\alpha(u),\ W - \omega(u)\ \}$
        $cap(v,u) := \min\{A - \alpha(u),\ \omega(u)\}$
        $\alpha(v) := \alpha(v) + \alpha(u)$
        $\omega(v) := \omega(v) + \omega(u)$
        Remove $u$ from $T$
end
```

This algorithm can be implemented to run in only $O(n)$ time.

7.3. Proof of Theorem 4.3

Tree network design, minimize total capacity

INSTANCE: Network topology $G = (V, E)$. Traffic requirements are given by termination capacities $\alpha, \omega \in \mathcal{N}$ for each $v \in V$. ($\mu(u,v)$ values are assumed to be any values such that α, ω, μ are α,ω-bounded, so μ may be ignored.)

SOLUTION: A capacity $cap(u,v) \in \mathcal{N}$ for each $(u,v) \in E$, where the links with non-zero capacity form a tree. This assignment of capacity makes the network nonblocking for routing algorithm FP with fixed paths defined by the tree structure, where connection requests may be multirate and multipoint.

SOLUTION COST: The cost of the network is the sum of the *capacities* of each link:

$$\sum_{(u,v) \in E} cap(u,v)$$

That is, $cost_{(u,v)}(x) = x$ for all $u, v \in V$.

OBJECT: Find a solution with minimum cost.

THEOREM 7.3. *Every tree solution to the problem above costs at least as much as the tree with node C as the "center" and all other nodes connected directly to C. Here C may be any node with the maximum value of $\alpha(C) + \omega(C)$ among all nodes.*

Let C be a node with maximum value of $\alpha(C) + \omega(C)$ among all nodes. In a star tree with C at center, where all nodes other than C have direct link to and from C, the minimum total capacity of the links (u, C) and (C, u) necessary for nonblocking operation is

$$\begin{aligned}
cap(u,C) + cap(C,u) &= \min\{\alpha(u),\ W - \omega(u)\} + \min\{A - \alpha(u),\ \omega(u)\} \\
&\qquad\qquad\qquad\qquad \{ \text{ from previous section } \} \\
&= \min\{\alpha(u) + \omega(u),\ A + W - (\alpha(u) + \omega(u)),\ A,\ W\} \\
&\qquad\qquad\qquad\qquad \{ \text{ properties of min } \} \\
&= \min\{\alpha(u) + \omega(u),\ A,\ W\} \qquad \{ \text{ see below } \}
\end{aligned}$$

The last step is justified by the observation

$$\begin{array}{rcll} \alpha(u)+\omega(u) & \leq & \alpha(C)+\omega(C) & \{\text{ by choice of } C\ \} \\ & \leq & \sum_{v\in V-\{u\}}(\alpha(v)+\omega(v)) & \{C \in V-\{u\}\} \\ & = & A+W-(\alpha(u)+\omega(u)) & \{\text{ defns. of } A \text{ and } W\ \} \end{array}$$

Therefore, $A+W-(\alpha(u)+\omega(u))$ can be dropped from the collection of values to minimize, since $\alpha(u)+\omega(u)$ is always no larger.

Now consider any spanning tree T for the nodes of G. Pick the same node C used before and draw T as a rooted tree with C as the root and all other nodes going down. Let D_u be the set of descendants of u in this rooted tree (including u), for any $u \in V$. The minimum total capacity necessary on the links $(u, p(u))$ and $(p(u), u)$ between a non-root node u and its parent node $p(u)$ is

$$\begin{array}{rcl} cap\,(u,p(u))+cap\,(p(u),u) & = & \min\{\alpha(D_u),\ W-\omega(D_u)\}+\min\{A-\alpha(D_u),\ \omega(D_u)\} \\ & & \hfill \{\text{ from previous section }\} \\ & = & \min\{\alpha(D_u)+\omega(D_u),\ A+W-(\alpha(D_u)+\omega(D_u)),\ A,\ W\} \\ & & \hfill \{\text{ properties of min }\} \end{array}$$

Now if we can show that $cap\,(u,p(u))+cap\,(p(u),u)$ is always at least as large as $cap\,(u,C)+cap\,(C,u)$ then the total capacity of the tree T, $\sum_{u\in V-\{C\}}(cap\,(u,p(u))+cap\,(p(u),u))$, is at least as large as the total capacity of the star tree with center C, $\sum_{u\in V-\{C\}}(cap\,(u,C)+cap\,(C,u))$.

$\alpha(D_u)+\omega(D_u)\geq\alpha(u)+\omega(u)$ follows immediately from $u \in D_u$. Also

$$\begin{array}{rcll} A+W-(\alpha(D_u)+\omega(D_u)) & = & \alpha(\overline{D_u})+\omega(\overline{D_u}) & \{\text{ defns. of } A \text{ and } W\ \} \\ & \geq & \alpha(C)+\omega(C) & \{C\in\overline{D_u}\} \\ & \geq & \alpha(u)+\omega(u) & \{\text{ choice of } C\ \} \end{array}$$

Therefore $\min\{\alpha(D_u)+\omega(D_u), A+W-(\alpha(D_u)+\omega(D_u))\}\geq\alpha(u)+\omega(u)$. Now $x \geq y$ implies that $\min\{x,z\}\geq\min\{y,z\}$. With $x=\min\{\alpha(D_u)+\omega(D_u), A+W-(\alpha(D_u)+\omega(D_u))\}$, $y=\alpha(u)+\omega(u)$, and $z=\min\{A,W\}$, we know that $cap\,(u,p(u))+cap\,(p(u),u)\geq cap\,(u,C)+cap\,(C,u)$ for all $u\in V-\{C\}$, and we have proved that all trees have at least as much total capacity as the star tree with node C as the center.

7.4. Proof of Theorem 4.4

Condition 4 is restated here to aid the reader.

$(\forall u,v\in V)$(no link in $prim(u,v)-sec(u,v)$ is contained in any path except $prim(u,v)$)

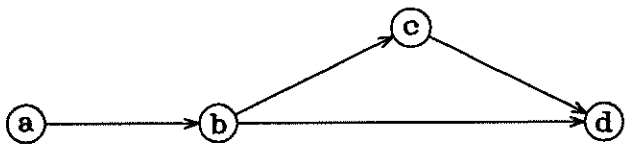

Figure 5: A simple instance with AP routing

Network design with alternate path routing, paths given

INSTANCE: Network topology $G = (V, E)$. For each link $(u,v) \in E$, the cost function $cost_{(u,v)}(cap)$ is a nondecreasing and concave function of cap. Traffic requirements are given by termination capacities $\alpha(v), \omega(v) \in \mathcal{N}$ for each $v \in V$, and point-to-point restrictions $\mu(u,v) \in \mathcal{N}$ for each $u, v \in V$. For each $u, v \in V$, there are directed paths $prim(u,v)$ and $sec(u,v)$ from u to v. These paths satisfy condition (4).

SOLUTION: A capacity $cap(u,v) \in \mathcal{N}$ for each $(u,v) \in E$. This assignment of capacity makes the network nonblocking for routing algorithm AP with paths *prim* and *sec*, where connection requests must be single rate and point-to-point.

SOLUTION COST: The cost of the network is the sum of the costs of each link:

$$\sum_{(u,v)\in E} cost_{(u,v)}(cap(u,v))$$

OBJECT: Find a solution with minimum cost.

THEOREM 7.4. *The problem above can be solved in $O(2^d \cdot m \cdot maxflow(n, U))$ time, where $m = \left|\overline{\mathcal{P}}\right|$, n is the number of nodes, U is a parameter depending on the paths which is at most $n^2 + n$, and d is a parameter depending on the paths which is at most the number of u, v pairs such that $p(u,v) \neq \emptyset$.*

First, a couple of trivial results. If we restrict $prim(u,v) = sec(u,v)$ for all $u, v \in V$, then the alternate path routing algorithm AP behaves exactly like fixed path routing, and the results in section 7.1.1 apply.

If we apply the algorithm in section 7.1.1 with fixed paths given by *prim*, then we will find link capacities for which the network will never block when we use the alternate path routing algorithm AP. This is because when AP tests the primary path to see if it has enough capacity available, it will always be so. This idea will also work for any routing algorithm like AP which first checks a single path to see if it has enough capacity, and returns it as the only path if so. However, this choice of link capacities may not be as cheap as possible for routing algorithm AP.

Before presenting a solution for more general instances, it is helpful to examine a small instance to understand the issues involved. Consider the network of Figure 5, where $\alpha(a) = 8$, $\alpha(c) = 4$, $\omega(d) = 6$, $\mu(a,d) = 5$, $\mu(c,d) = 4$, and all other α, ω, and μ values are 0. In this case, only a and c can be the source of connections, and only d can be the destination

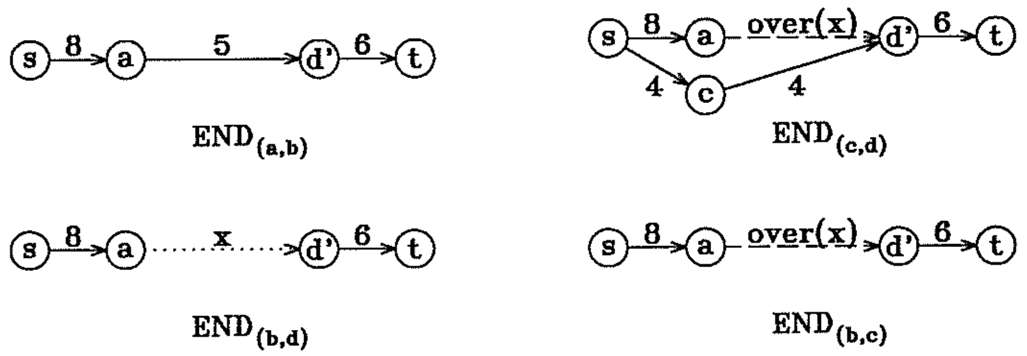

Figure 6: *END* graphs for the simple *AP* routing instance

of connections. Let $prim(a,d) = abd$, $sec(a,d) = abcd$, and $prim(c,d) = sec(c,d) = cd$. The other paths do not matter because they can not be used.

Let us generalize the definition of END_e for alternate path routing. Let $END_e = (V_{END_e}, E_{END_e} \cup O_{END_e} \cup A_{END_e})$, where

$$\begin{aligned} V_{END_e} &= \{u, u' : u \in V\} \cup \{s, t\} \\ E_{END_e} &= \{(s,u) : u \in V\} \cup \{(u',t) : u \in V\} \cup \{(u,v') : \text{both } prim(u,v) \text{ and } sec(u,v) \text{ use link } e\} \\ O_{END_e} &= \{(u,v') : prim(u,v) \text{ uses link } e\text{, but not } sec(u,v)\} \\ A_{END_e} &= \{(u,v') : sec(u,v) \text{ uses link } e\text{, but not } prim(u,v)\} \end{aligned}$$

The graphs $END_{(a,b)}$, $END_{(b,c)}$, $END_{(b,d)}$, and $END_{(c,d)}$ are shown in Figure 6. All arcs that have 0 capacity have been omitted. The dotted arc (a,d') in $END_{(b,d)}$ signifies that link (b,d) is in the primary path from a to d, but not in the secondary path. The dashed arc (a,d') in $END_{(b,c)}$ and $END_{(c,d)}$ signifies that links (b,c) and (c,d) are in the secondary path from a to d, but not in the primary path. The solid arc (a,d') in $END_{(a,b)}$ signifies that link (a,b) is in both the primary and secondary path from a to d. Similarly for the solid arc (c,d') in $END_{(c,d)}$.

The arc capacities require some explaining. For solid arcs $(u,v') \in END_{(x,y)}$, we know that whichever path the routing algorithm *AP* finds, that path *must* contain the arc (x,y). Therefore, if such a graph $END_{(x,y)}$ contains *all* solid arcs, as $END_{(a,b)}$ does in this example, we may find the maximum integer flow on $END_{(x,y)}$ as described in section 7.1.1 and know that it corresponds to the maximum usage of link (x,y), which equals the minimum necessary capacity.

When a graph $END_{(x,y)}$ contains dotted or dashed arcs, however, it introduces dependencies between link capacities. In this simple instance, we may think of assigning a capacity x, $0 \le x \le \mu(a,d)$, to link (b,d), and this will allow us to determine the minimum necessary capacities of links (b,c) and (c,d). How can this be done?

Consider a sequence of requests to add and delete single rate connections between a and d, all having rate 1. As long as no more than x requests are using link (b,d), the routing algorithm *AP* will find that there is enough capacity available on the path *abd*, and the

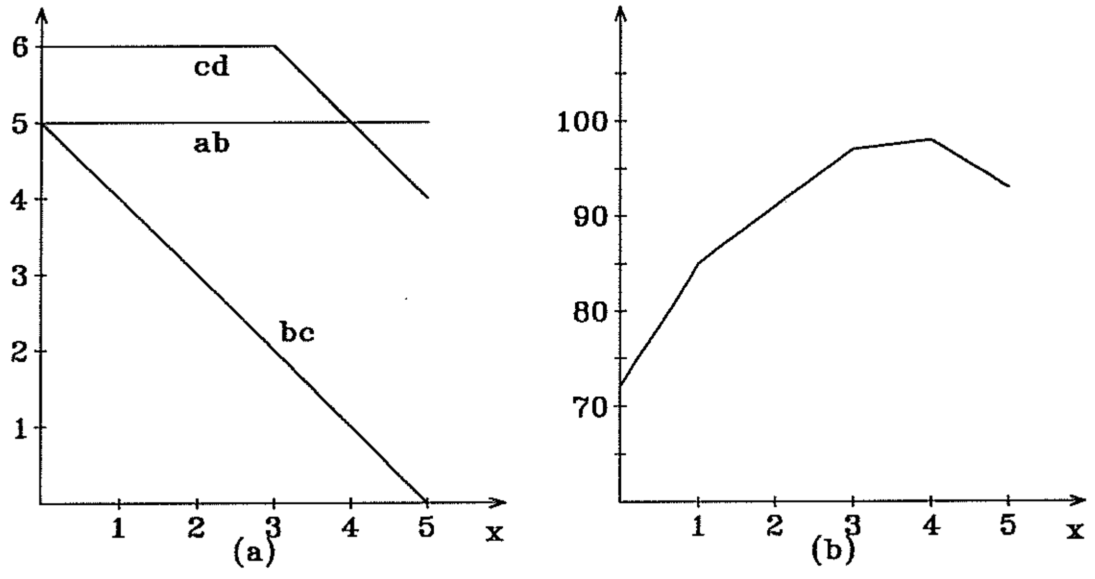

Figure 7: (a) Minimum capacities and (b) total network cost as a function of $x = cap(b,d)$

connections will follow that path. However, as soon as x requests are using link (b,d) and an additional request comes in (this can only occur if $x < \mu(a,d) = 5$), *AP* will find that the path *abd* cannot be used, and so it returns *abcd* instead. We say that the request *overflows* onto its secondary path in such a case.

The quantity $over(x)$ is used to express the maximum total rate of requests that can simultaneously overflow onto the secondary path, given that the primary path can handle a total rate x. In the single rate case, it should be easy to see that $over(x) = \mu(a,d) - x$.

In $END_{(b,d)}$, we assign the arc (a,d') the capacity x to denote that this is the maximum total rate of all connections from a to d that may use link (b,d) simultaneously. Therefore the arc capacity "means the same thing" as it did in the fixed path case, but now x can be any quantity at most $\mu(a,d)$. Similarly, we assign the arc (a,d') in $END_{(b,c)}$ the capacity $over(x)$ to denote that this is the maximum total rate of all connections from a to d that may use link (b,c) simultaneously.

Given that the capacity of link (b,d) is x, where $0 \le x \le \mu(a,d)$, we can now find the minimum necessary capacities for links (b,c) and (c,d). Thus we can also find the costs of each link, and the total cost of the network, as a function of x.

For example, let the cost functions of each link in the example have the form $a_{(u,v)} + b_{(u,v)} \cdot cap$ if the capacity $cap > 0$, and 0 otherwise (these functions are nondecreasing and concave when $a_{(u,v)}, b_{(u,v)} \ge 0$). Let $a_{(a,b)} = 0$, $b_{(a,b)} = 3$, $a_{(b,c)} = 6$, $b_{(b,c)} = 4$, $a_{(b,d)} = 10$, $b_{(b,d)} = 7$, $a_{(c,d)} = 1$, and $b_{(c,d)} = 5$. Then the link capacities are given as functions of x in Figure 7(a), and the total cost of the network is given as a function of x in Figure 7(b).

Note that this function is minimized when $x = 0$, one of the "extreme" values that x may take. The other extreme value is 5. x may be larger than 5, but if so, any capacity over 5 will never be used. It will be shown that under the conditions given in the problem instance, the minimum cost network will always be found by assigning extreme values to the links in $\mathcal{P}$.

LEMMA 7.2. *Suppose we have primary and secondary paths which satisfy condition (4). Then all links $e \in \mathcal{P}$ will have END_e with only a single dotted arc of the form (u, v'). All other links $e \in \overline{\mathcal{P}}$ will have END_e where all arcs are either solid or dashed.*

Proof: Follows directly from the modified definition of END_e given earlier in this section. ■

Note that it is possible for $|p(u,v)| > 1$ to hold for some $u, v \in V$. If so, then it never makes sense to assign different capacities to links in $p(u,v)$. It is always the link in $p(u,v)$ with the smallest capacity which determines when connections overflow onto the secondary path. Any such links with larger than minimum capacity will never have their "excess" capacity used. When link cost functions are nondecreasing with capacity, it is never cheaper to assign unusable capacity to links. Similarly, all links in $p(u,v)$ should never have a capacity larger than $\min\{\mu(u,v), \alpha(u), \omega(v)\}$.

Let $\mathcal{D} = \{(u,v) : p(u,v) \neq \emptyset\}$.

For convenience, let the elements of $\mathcal{P}$ be denoted $\{l_1, \ldots, l_p\}$. The exact numbering is unimportant.

LEMMA 7.3. *Let every link in $p(u,v)$ have capacity $x_{(u,v)}$, for all $(u,v) \in \mathcal{D}$. Then the maximum usage, which equals the minimum necessary capacity, of every link in $\overline{\mathcal{P}}$ can be determined in $O(maxflow(n, U))$ time per link, where $U \leq n^2 + n$.*

Proof: The key idea here is that once capacities are chosen for links in $\mathcal{P}$, this is essentially the same as the fixed-path problem. The only difference is that some of the point-to-point restrictions μ have been reduced for links in $\overline{\mathcal{P}}$.

Let e be an arbitrary link in $\overline{\mathcal{P}}$. By Lemma 7.2, all arcs in its graph END_e are either solid or dashed. Assign all solid arcs (u, v') a capacity of $\mu(u,v)$, and all dashed arcs (u, v') a capacity of $\mu(u,v) - x_{(u,v)}$.

When arc capacities in END_e are assigned in this way, then for every state s which the network can reach given single rate point-to-point connection requests, there is an integer flow in END_e obeying the arc capacities where the usage of link e in state s equals the value of the flow. Conversely, for every integer flow in END_e obeying the arc capacities, there is a sequence of adding and removing connection requests (all point-to-point with rate 1) where the value of the flow equals the usage of e in the resulting state.

Therefore, the maximum usage of e in any state is equal to the value of a maximum flow in END_e with arc capacities as described above. ■

Lemma 7.3 tells us that the minimum necessary capacity of each link $e \in \overline{\mathcal{P}}$ is functionally dependent on a subset of $X = \{x_{(u,v)} : (u,v) \in \mathcal{D}\}$. For each $e \in \overline{\mathcal{P}}$, denote this subset of X by X_e, and denote the minimum necessary capacity of e by $cap_e(X_e)$.

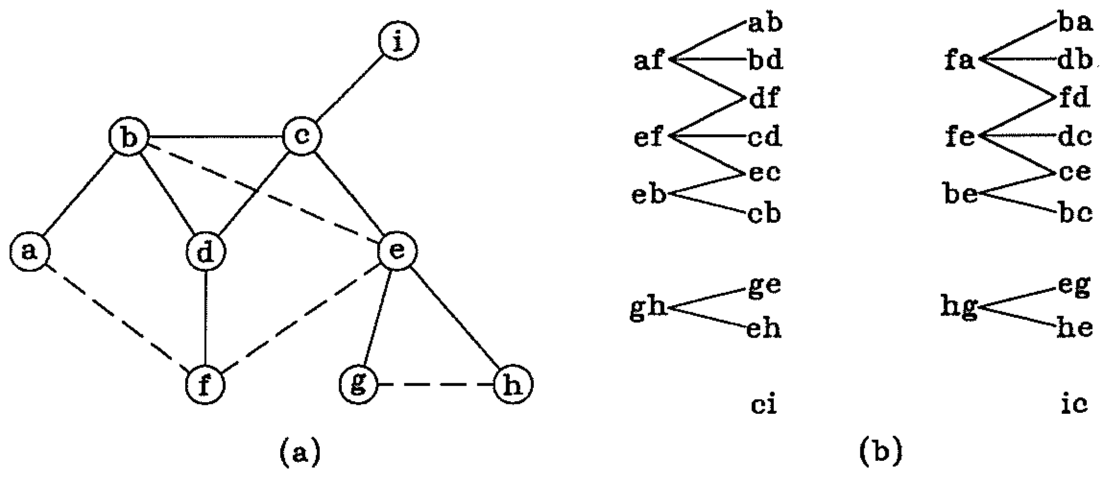

Figure 8: (a) Alternate path network instance and (b) dependency graph

Recall that the total cost of the network is $\sum_{(u,v)\in E} cost_{(u,v)}(cap(u,v))$, which can be expanded to

$$\sum_{(u,v)\in\mathcal{D}} \sum_{e\in p(u,v)} cost_e(x_{(u,v)}) + \sum_{e\in\overline{\mathcal{P}}} cost_e(cap_e(X_e))$$

We want to minimize this function while allowing each $x_{(u,v)}$ to range from 0 to $\mu(u,v)$.

It may be possible to break this sum into several parts, each independent of the others. To determine these parts, construct a *dependency graph*, which will represent the functional dependencies between link capacities. The graph is bipartite, with one node partition containing a node for each $(u,v)\in\mathcal{D}$, and the other containing a node for each link $e\in\overline{\mathcal{P}}$. There is an edge between two nodes (u,v) and e if the minimum necessary capacity of e depends on $x_{(u,v)}$, i.e., $e\in s(u,v)$.

For example, in the network of Figure 8(a), each edge represents two oppositely directed links. The dashed edges are the links in the set $\mathcal{P}$. Let $prim(a,f) = af$, $sec(a,f) = abdf$, $prim(a,e) = abe$, $sec(a,e) = abce$, $prim(f,e) = fe$, $sec(f,e) = fdce$, $prim(g,h) = gh$, and $sec(g,h) = geh$. For the opposite directions, reverse the paths above (e.g., $prim(f,a) = fa$ and $sec(f,a) = fdba$). For all other switch pairs $u,v\in V$, $prim(u,v) = sec(u,v)$ and all links in these paths use only solid links. These paths satisfy condition 4. The dependency graph for this instance is shown in Figure 8(b). For example, the capacities of links ec and cb depend on $x_{(e,a)}$ because $s(e,a) = \{ec, cb\}$.

After the dependency graph is constructed, find the connected components of this graph. Each component contains a subset $\mathcal{D}'$ of the nodes $\mathcal{D}$ and a subset $\overline{\mathcal{P}}'$ of the nodes $\overline{\mathcal{P}}$. The portion of the total network cost represented by a single component,

$$\sum_{(u,v)\in\mathcal{D}'} \sum_{e\in p(u,v)} cost_e(x_{(u,v)}) + \sum_{e\in\overline{\mathcal{P}}'} cost_e(cap_e(X_e))$$

can be minimized by choosing values for the variables in $\mathcal{D}'$ completely independently of the variables in other components of the dependency graph. This technique does not change the worst-case running time of this algorithm, but it can speed it up dramatically for some instances. The value of the exponent d in the running time is equal to the size of the largest subset $\mathcal{D}'$ induced by the components.

For example, in the dependency graph of Figure 8(b), we may minimize the total cost of links gh, ge, and eh independently of all other links, and this may be done by optimizing over the single variable $x_{(g,h)}$. Note that the capacities of links ci and ic depend on no other link capacities, and their minimum necessary capacities may be determined exactly as in Section 7.1.1 because the graphs END_{ci} and END_{ic} contain all solid arcs. In this example, this technique of finding independent subproblems reduced the exponent d from $|\mathcal{P}| = 8$ to 3.

LEMMA 7.4. *Let $G = (V, E)$ be a directed graph with source $s \in V$ and sink $t \in V$. Let $a \in E, a \neq (s,t)$ be a distinguished arc with a capacity given by the variable $x \geq 0$. All other arcs $e \in E$ have a fixed capacity $cap(e) \geq 0$. Then the value of a maximum flow from s to t as a function of x, $m(x)$, has slope 1 for some interval $[0, c]$, and slope 0 afterwards. More formally, there exist constants $s, c \geq 0$ such that*

$$m(x) = \begin{cases} s + x & \text{if } 0 \leq x \leq c \\ s + c & \text{if } c \leq x \end{cases}$$

Proof: The max-flow min-cut theorem of Ford and Fulkerson [FF64] states that the value of a maximum flow in a network is equal to the capacity of a minimum $s - t$ cut. An $s - t$ cut is denoted $(X, \overline{X})$, where $s \in X$ and $t \in \overline{X}$, and is defined as the set of all arcs $(u, v) \in E$ for which $u \in X$ and $v \in \overline{X}$. The capacity of a cut is the sum of the capacities of the arcs within it. Let $\mathcal{C}$ be the set of all $s - t$ cuts.

Some cuts contain the arc a. Every such cut $(X, \overline{X})$ has a capacity of the form $s_X + x$. Let s' be the minimum of all s_X values.

There exist cuts which do not contain the arc a, because $a \neq (s,t)$. Every such cut $(X, \overline{X})$ has a capacity c_X. Let c' be the minimum of all c_X values.

Then $m(x) = \min\{s' + x, c'\}$. If we choose $s = \min\{s', c'\}$ and $c = \max\{c' - s', 0\}$, then this is exactly the function described in the lemma. ■

Note that all of the capacity functions in Figure 7(a) are of the form $m(5 - x)$.

LEMMA 7.5. *If the cost function $cost(cap)$ of a link is a nondecreasing and concave function of cap, and $m(x)$ is a function of the form described in Lemma 7.4, then $cost(m(x))$ is a nondecreasing and concave function of x.*

Proof: Let $m(x)$ have the form described in Lemma 7.4 with constants $s, c \geq 0$. Then on the interval $[0, c]$, $m(x)$ is increasing with slope 1, so $cost(m(x))$ is nondecreasing and concave on $[0, c]$ because $cost(cap)$ is nondecreasing and concave on $[m(0), m(c)] = [s, s + c]$. On the interval $[c, \infty)$, $m(x)$ is constant, and so is $cost(m(x))$. It is easy to see from this that $cost(m(x))$ is concave on $[0, \infty)$ given that $cost(cap)$ is nondecreasing and concave. ■

Corollary 7.1. *$cost(m(\mu - x))$ is a nonincreasing and concave function of x on the interval $[0, \mu]$.*

Fact 1. *If $f_1(x), \ldots, f_k(x)$ are all concave functions of x, then so is $\sum_{i=1}^{k} f_i(x)$.*

This can be easily proven from the definition of concave functions.

Fact 2. *If $f(x)$ is concave on the interval $[a, b]$, then $\min_{a \le x \le b} f(x) = \min_{x \in \{a,b\}} f(x)$. That is, the minimum function value over the interval must occur at one of the endpoints of the interval.*

This is a well-known fact about concave functions. See Bazaraa and Shetty [BS79, Thm. 3.4.6, p. 99], for example.

Lemma 7.6. *Let $\mathcal{D}'$ and $\overline{\mathcal{P}}'$ be all nodes in one component of the dependency graph. Then the minimum total cost for all links in $\overline{\mathcal{P}}' \cup \bigcup_{(u,v) \in \mathcal{D}'} p(u,v)$ can be determined in $O(2^{|\mathcal{D}'|} \cdot \left|\overline{\mathcal{P}}'\right| \cdot maxflow(n, U))$ time, where n is the number of nodes in the network and $U \le n^2 + n$.*

Proof: Let e be any link in $\overline{\mathcal{P}}'$. For all $(u,v) \in \mathcal{D}'$, either END_e has a dashed arc (u, v') with capacity $\mu(u,v) - x_{(u,v)}$, or it has no arc (u, v'). Any other (u, v') arcs in END_e must be solid (with a fixed capacity). If $(u,v) \in \mathcal{D}'$ and (u, v') is in END_e, and if we let $x_{(u,v)}$ change while leaving the other x values fixed, Corollary 7.1 tell us that the cost of link e as a function of $x_{(u,v)}$ is nonincreasing and concave on $[0, \mu(u,v)]$. If $(u,v) \in \mathcal{D}'$ and (u, v') is not in END_e, then the minimum necessary capacity of e as a function of $x_{(u,v)}$ is a constant, and so is its cost. A constant function is concave.

For all $(u,v) \in \mathcal{D}'$, each link in $p(u,v)$ has a capacity $x_{(u,v)}$, and by the restriction in the problem instance, each of their costs is a concave function of $x_{(u,v)}$.

The cost of this portion of the network is the sum of costs of each link in $\bigcup_{(u,v) \in \mathcal{D}'} p(u,v) \cup \overline{\mathcal{P}}'$. Since the cost of each individual link is a concave function of each $x_{(u,v)}$ variable, Fact 1 tells us that the total cost of all of these links is also a concave function of each $x_{(u,v)}$.

We wish to find values for all $x_{(u,v)}, (u,v) \in \mathcal{D}'$, such that the cost of this portion of the network is minimized. If we fix the value of all x variables except for $x_{(u,v)}$, then Fact 2 tells us that the minimum can be found simply by evaluating the total cost at $x_{(u,v)} = 0$ and $x_{(u,v)} = \mu(u,v)$. Therefore, the overall minimum can be found by trying all $2^{|\mathcal{D}'|}$ choices of assigning the x variables either 0 or their maximum value.

By Lemma 7.3, we can find the costs of each of these $2^{|\mathcal{D}'|}$ possibilities in $O(\left|\overline{\mathcal{P}}'\right| \cdot maxflow(n, U))$ time (assuming the cost functions are easy to compute). ■

Theorem 4.4 follows directly from Lemma 7.6 by adding the running times for each component of the dependency graph. This result implies that alternate path routing with these constraints gives no lower cost than fixed path routing. The author considers it likely that if point-to-point multirate traffic is allowed, there will be some instances for which alternate path routing will give a lower cost than fixed path routing.

References

[BR92] Piotr Berman and Viswanathan Ramaiyer. Improved approximations for the Steiner tree problem. In *Proc. 3rd ACM-SIAM Symp. Discrete Algorithms*, pages 325–334, January 1992.

[BS79] Mokhtar S. Bazaraa and C. M. Shetty. *Nonlinear Programming.* John Wiley & Sons, 1979.

[FF64] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton Univ. Press, Princeton, NJ, 1964.

[Fin91] J. Andrew Fingerhut. Designing communication networks with fixed or nonblocking traffic requirements. Technical Report WUCS-91-55, Washington University, St. Louis, Missouri, 1991.

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[GJ79] Michael R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to NP-Completeness.* Freeman, 1979.

[GK90] R. J. Gibbens and F. P. Kelly. Dynamic routing in fully connected networks. *IMA J. Mathematical Control and Information*, 7:77–111, 1990.

[GT88] Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, October 1988.

[GW90] Alexander Gersht and Robert Weihmayer. Joint optimization of data network design and facility selection. *IEEE Journal on Selected Areas in Communications*, 8(9):1667–1681, December 1990.

[Hu69] Te Chiang Hu. *Integer Programming and Network Flows.* Addison-Wesley, 1969.

[Hu74] Te Chiang Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, 3(3):188–195, September 1974.

[KKG91] Aaron Kershenbaum, Parviz Kermani, and George A. Grover. MENTOR: An algorithm for mesh network topological optimization and routing. *IEEE Transactions on Communications*, 39(4):503–513, April 1991.

[KMB81] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[KP91] J. H. Klein and P. L. Powell. Simple heuristic methods for network routeing: a case study. *Omega*, 19(2/3):137–148, 1991.

[KRT92] V. King, S. Rao, and Robert Endre Tarjan. A faster deterministic maximum flow algorithm. In *Proc. 3rd ACM-SIAM Symp. Discrete Algorithms*, pages 157–164, January 1992.

[Min89] Michel Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, 19:313–360, 1989.

[MT89] Riccardo Melen and Jonathan S. Turner. Nonblocking multirate networks. *SIAM J. Comput.*, 18(2):301–313, April 1989.

[MT90] Riccardo Melen and Jonathan S. Turner. Nonblocking multirate distribution networks. In *Proc. INFOCOM*, pages 1–8?, 1990.

[MW84] Thomas L. Magnanti and Richard T. Wong. Network design and transportation planning: models and algorithms. *Transportation Science*, 18(1):1–55, February 1984.

[Pip82] Nicholas Pippenger. Telephone switching networks. In Stefan A. Burr, editor, *The Mathematics of Networks (Proc. of symposia in applied mathematics - v. 26)*, pages 101–133. American Mathematical Society, 1982.

[Tar83] Robert Endre Tarjan. *Data Structures and Network Algorithms.* SIAM, 1983.

[Wax88] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[Wax89] Bernard M. Waxman. *Evaluation of Algorithms for Multipoint Routing.* PhD thesis, Washington University, St. Louis, August 1989.

[WI88] Bernard M. Waxman and Makoto Imase. Worst-case performance of Rayward-Smith's Steiner tree heuristic. *Information Processing Letters*, 29:283–287, December 1988.

[Zel91] A. Z. Zelikovsky. The 11/6 approximation algorithm for the Steiner problem on networks. to appear in Information and Computation, 1991.