TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

APPROXIMATION ALGORITHMS FOR CONFIGURING NONBLOCKING COMMUNICATION NETWORKS

1. INTRODUCTION

For any communications technology, those who wish to build, operate, and use (or sell the use of) networks based on this technology confront a network design problem. In its most basic form, the problem is to find a way to construct a network that meets the desires of network users, and does so as cheaply as possible.

For many years, researchers have studied telephone networks to find ways of installing and upgrading them less expensively, while still providing point-to-point voice connections with a low probability of blocking [6, 14, 48, 52, 61]. Much of this work models call arrivals as a Poisson process, and either finds exact or approximate blocking probabilities.

In the late 1960's and early 1970's, researchers began studying the problem of efficiently constructing packet switched data networks, particularly the ARPANET [13, 22, 23, 24, 33, 54]. Here there are no circuits, so blocking of connection attempts is not a consideration. The main source of user dissatisfaction is packet delay, which is higher, on average, when communication links are heavily loaded.

Much work has taken a more deterministic view of the node to node traffic [34, 44, 45, 50, 53, 55]. Here the only requirement on the network is that it is possible to fit

the traffic onto the links of the network. With any of these algorithms, one can set a maximum utilization for all links. For example, one could specify that no link should be utilized over 75%. This can be used as a heuristic to keep the delay or blocking probability low.

In all of the previously cited work, the "desires of the network users" are represented as magnitudes of traffic from particular network nodes to other nodes. These can be written in a table, where each row corresponds to traffic originating at a particular node, and each column corresponds to traffic received at a particular node. We call this a traffic matrix.

Another common kind of network that has been studied is sometimes called a "local access network." In data processing networks, this is the network that connects terminals to a central processing site [7, 29, 30, 31, 32]. In telephone networks, it is the collection of cables called the "outside plant" that connect telephone subscribers to their local telephone switch, called the central office [11, 46]. In such a problem, the traffic is specified as a traffic magnitude for each terminal. The network must be capable of carrying traffic from all terminals to the central site simultaneously.

About 15 years ago, the idea of making a network that was capable of supporting many kinds of services was introduced, and has since gained support. ATM is an emerging set of network standards that can support many kinds of services [18].

It is sufficient for our purposes to describe ATM networks as connection-oriented networks in which connections may have different rates from one another, and they may be point-to-point or multipoint. A multipoint connection involves more than two nodes.

It would be desirable to construct and operate ATM networks with low probability of blocking connection requests. However, before we can even begin to estimate such a blocking probability, we must have a probability model for incoming connection requests.

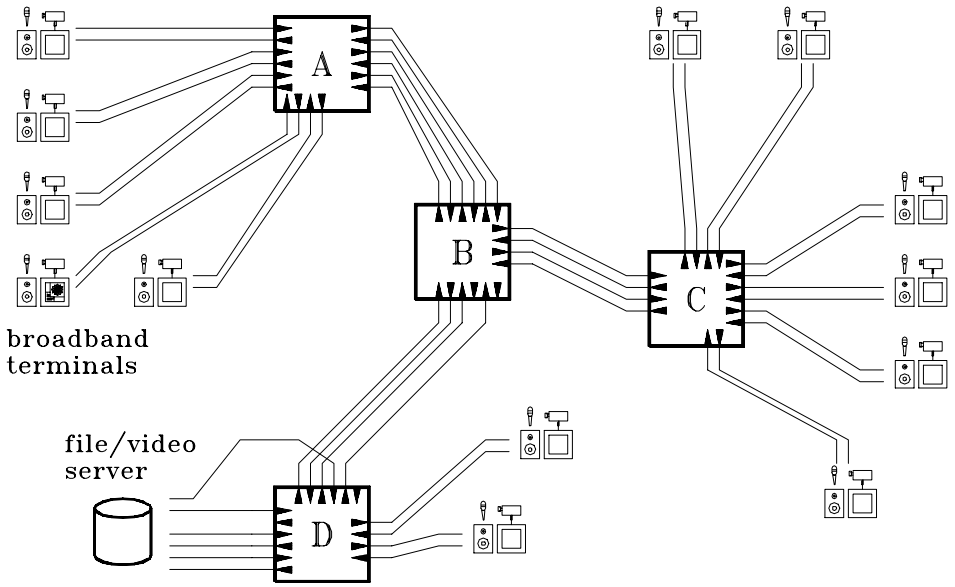

Figure 1.1: A network

For ATM networks, this is significantly more complex than for telephone networks, because requests may have multiple rates, and may be multipoint. Even if such a mathematical model were proposed, there would be serious questions about whether the model was accurate, because few ATM networks have been deployed, and very little is known about the way they will be used. Even if the model were accurate for one network, it might be inaccurate for another, because its users may use different applications.

For these reasons, we consider models of traffic that are worst case. We desire to construct networks that never block a connection request, subject to certain limits on the requests.

For example, Figure 1.1 shows an ATM network. The boxes labeled A, B, C, and D are 8 port nonblocking ATM switches, where each port runs at 150 Mb/s. By nonblocking we mean that if the links have enough bandwidth to support a new connection, then the switch is able to set up that connection. The lines drawn between the switches

are either fiber optic or twisted pair copper links. Each link is unidirectional, only used for carrying data in the direction of the arrow. Switches A, C, and D are attached to workstations, and D is also attached to a file and video server.

During the operation of the network, the following sequence of events is possible. First, a user at a workstation attached to switch C, call it C_1, requests a connection from his/her workstation to workstation D_1 (attached to switch D) with a rate of 35 Mb/s. This request can be satisfied by setting up a connection along the path C_1CBDD_1, reserving 35 Mb/s out of 150 Mb/s on each link in the path. We denote the request with the tuple $(C, D, 35)$, and the connection with the tuple $(CBD, 35)$. It is convenient to ignore the particular terminals involved in each request and connection, as we are concerned with preventing requests from blocking due to insufficient link capacity between switches, not between the terminals and their local switch.

After the initial request and connection setup, we could have the request $(A, D, 90)$, satisfied by $(ABD, 90)$, followed by the pair of requests $(C, A, 100)$ and $(C, A, 125)$, satisfied by $(CBA, 100)$ and $(CBA, 125)$. At this time, the link from B to D has $35 + 90 = 125$ Mb/s of its 150 Mb/s capacity used, and 25 Mb/s available.

If the next request made is $(C, D, 40)$, then the network cannot satisfy the request. We say that the request *blocks*. Our goal in this research is to configure networks so that they never block a "compatible" connection request.

What do we mean by a compatible request? In the network shown, a network manager may realize that while it is possible that all of the workstations attached to switches A and C could simultaneously make requests for large rate connections to workstations attached to D, it is very unlikely. The manager may make this knowledge concrete by saying, "I want the network to be nonblocking, but only as long as the total

rate of requests that have their destination attached to D, and their source attached to some other switch, is at most 150 Mb/s. If it goes over that, I don't care if it blocks."

We denote this limit by $\omega(D) = 150Mb/s$, and call it a *destination termination limit*, or simply a *destination limit*. In our example sequence of requests, the last one, $(C, D, 40)$, violates this limit. Similar limits can be specified for the other switches in the network. In addition, we can specify a *source termination limit*, or *source limit*, $\alpha(D) = 450$ for switch D, which restricts the total rate of all requests that can begin at D and end at some other switch. In this example, $\alpha(D)$ is larger than $\omega(D)$ because of the server attached to D.

This method of specifying the traffic is similar to the way it is done for the local access network design problems mentioned earlier. However, here there is no a priori assumption that all of the traffic will go between the various switches and some central site. We show that a similar network structure, a star network, is often very close to optimal.

There are two important objections to star networks. One is that the switch in the center becomes overloaded, and thus it requires several physical switching systems (e.g., routers), highly interconnected in a mesh topology, to implement the center switch.

In ATM switching systems, the largest component of the cost is currently located in the ports of the switch. Furthermore, there exist parallel switch architectures whose port interconnection hardware costs grow at a rate of $n \log n$, where n is the number of ports [65]. Thus the total cost of even quite large switches can be made roughly linear with the number of ports.

Another objection is reliability. If a link or switch fails in a tree network, it becomes disconnected. Several researchers have explicitly accounted for such failures in their network design algorithms (see Chapter 9 for references), although all of them either

ignore the capacity required on links, requiring only a two-connected network, or they require a traffic matrix to specify the network traffic. This area is a topic of future research.

As mentioned before, most previous work specifies limits with a traffic matrix. For each ordered pair of distinct switches u, v, a limit $\mu(u, v)$ is specified by the network manager, where $\mu(u, v)$ is the maximum total rate of requests from switch u to switch v that must be accepted without blocking.

Specifying point-to-point limits is reasonable when the offered traffic has been observed to have stable and predictable traffic between each pair of switches, and measurements have been collected so that these traffic levels are known. However, there is very little experience in characterizing the offered traffic in ATM networks. We would expect the traffic in most local area networks to be highly variable, not just in its rates, but in which pairs of switches are communicating (or subsets, for multicast traffic).

In Chapter 2, we define the mathematical model we use to describe networks and connection requests. Chapter 3 presents "simple" traffic limits, which are similar to the way traffic is specified in local access networks, and several results that give strong evidence that a star network is either optimal or nearly so. Chapter 4 generalizes these traffic limits. It shows how one may further restrict the offered traffic by specifying a traffic matrix. Experiments show that unless the traffic matrix restrictions are significantly smaller than the termination limits, the cost of the network cannot be reduced significantly below a star network. Chapter 5 generalizes the traffic limits further, by allowing one to specify "clusters" of switches, each with high traffic among the switches within the clusters, but with lower traffic between switches in different clusters. This grouping of switches into clusters is useful for specifying traffic in a network larger than a small LAN. Chapter 6 shows that all of the previously obtained results hold

for multipoint traffic as well as point-to-point. Chapter 7 presents some extensions to the network configuration problem that can be handled with our solution methods. In most of this dissertation, we consider a simple method of routing connections called fixed path routing, where there is only one choice for routing connections. Chapter 8 examines two other routing algorithms that allow choices in how connections are routed, shows that they are more difficult to analyze, and gives evidence that fixed path routing is just as good at producing low cost networks. Chapter 9 concludes this dissertation.

2. DEFINITIONS

2.1. Networks, connection requests, and connections

A *network* $\mathcal{N}$ is a directed graph with switch set N, link set L, and a function *cap* on the links, where *cap*(l) is the bandwidth, or capacity, of link l. For example, the network in Figure 1.1 is represented by the directed graph in Figure 2.1. Each link is labeled with its capacity in Mb/s.

For simplicity of exposition, we confine our attention to point-to-point connections for most of this dissertation. However, all of the link dimensioning algorithms also work when we allow the following kinds of multipoint connections: one source and many destinations, many sources and one destination, or many sources to the same set of destinations. See Chapter 6 for more details.

A *connection request* is represented by a triple $r = (u, v, \rho)$, where switch u is the source of information flow, switch v is the destination of flow, and ρ is the desired rate of

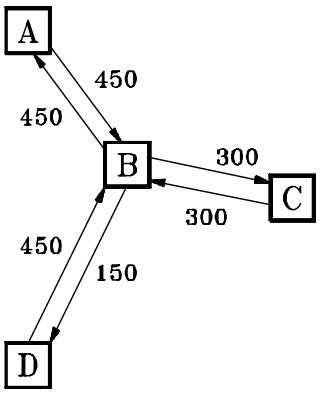

Figure 2.1: Directed graph representing the network of Figure 1.1

the connection. A *connection* is a pair $c = (\Pi, \rho)$, where $\Pi \subseteq L$ is a path. A connection $c = (\Pi, \rho)$ *realizes a request* $r = (u, v, \rho)$ if Π is a path from u to v, and $\rho(c) = \rho(r)$. A *state* $\mathcal{S}$ of the network is a collection (formally, a multiset) of connections. Given a network $\mathcal{N}$ and a state $\mathcal{S}$, we define the *usage* of a link l in state $\mathcal{S}$ to be

$$\lambda(l, \mathcal{S}) \quad = \quad \sum_{c \in \mathcal{S},\, c \text{ uses link } l} \rho(c)$$

A state $\mathcal{S}$ is *legal for network* $\mathcal{N}$ if $(\forall l \in L)$ $(\lambda(l, \mathcal{S}) \leq cap(l))$. In other words, every link is used in connections with a total rate that is at most the link's capacity.

For example, recall the request set $\{(C, D, 35), (A, D, 90), (C, A, 100), (C, A, 125)\}$. This set is realized by the state $\{(CBD, 35), (ABD, 90), (CBA, 100), (CBA, 125)\}$, which is legal for the network of Figure 2.1. To realize the additional request $(C, D, 40)$ would require us to add the connection $(CBD, 40)$ to the state, but the resulting state $\mathcal{S}$ is not legal because $\lambda((B, D), \mathcal{S}) = 35 + 90 + 40 = 165 > cap(B, D)$.

2.2. Fixed path routing and nonblocking networks

A routing algorithm is a method for taking a connection request and the current state of the network and determining a connection that satisfies the request. The new connection should not exceed any of the link capacities when it is added to the current state.

For most of this dissertation, we consider one simple routing algorithm, called *fixed path* routing. All of the definitions in this section have been simplified with this routing algorithm in mind. See Chapter 8 for definitions of other routing algorithms.

When fixed path (FP) routing is used, there is a table π. For each ordered pair of switches u, v, $\pi(u, v)$ is a directed path from u to v. Whenever a connection request $r = (u, v, \rho)$ is given, FP returns the path $\pi(u, v)$. Note that this path may not have enough available capacity to handle the new connection, and thus will cause the request

to block, even though other paths from u to v may have enough available capacity. The name of this routing algorithm comes from its behavior: the path to be used from u to v is *fixed* for the lifetime of the network (or at least for a long time). It ignores the current state of the network. Surprising as it may seem, this inflexibility is not a hindrance, at least for the kinds of worst-case traffic models studied here. The cheapest network configurations we have found all use fixed path routing.

We define what it means for a set of connection requests $\mathcal{R}$ to be compatible with traffic limits $\mathcal{T}$ in Sections 3.1, 4.1, and 5.1. The traffic limits restrict the number and rate of connection requests that can be in a compatible set $\mathcal{R}$.

Consider a network $\mathcal{N} = (N, L, cap)$, and a sequence of add and drop requests $r_1, \ldots, r_k$. Each r_i is either a request to add a new connection, or to remove a previously added connection that has not yet been removed. Define $\mathcal{R}_i$ to be the subset of $\{r_1, \ldots, r_i\}$ from which all drop requests and their corresponding add requests have been removed. $\mathcal{R}_i$ is the set of *active* connection requests after request r_i has been completed. Define the sequence $r_1, \ldots, r_k$ to be compatible with traffic limits $\mathcal{T}$ if for all i, $1 \leq i \leq k$, the set $\mathcal{R}_i$ is compatible with $\mathcal{T}$.

The network $\mathcal{N}$ is *blocking* for traffic $\mathcal{T}$ if there exists a sequence of requests $r_1, \ldots, r_k$ compatible with $\mathcal{T}$ such that $r_k = (u, v, \rho)$ is an add request, and there are less than ρ units of bandwidth available on the path $\pi(u, v)$. We say that the request r_k *blocks*. The network $\mathcal{N}$ is *nonblocking* for traffic $\mathcal{T}$ if there is no such request sequence.

Note that for most of this dissertation, we are not concerned with the sequence of add and drop requests $r_1, \ldots, r_k$, but with a set of active requests $\mathcal{R}$, for which no order is implied among the requests. The symbol $\mathcal{R}$ denotes a set of active requests at some unspecified time. If a sequence of add and drop requests is important, it will be explicitly stated.

2.3. The Network Configuration, Link Dimensioning, and Network Analysis Problems

We now describe a computational problem that models the following scenario. A network manager has several switches in given locations, and knows how much it costs to install links of various capacities between these switches.

These costs are represented by *link cost functions*. For example, one link cost function is "Between switches A and B, the cost of the link is the capacity in Mb/s times \$100." Such a function is called a *linear* link cost function, since the cost is linear with the link capacity. The value \$100 is called the *link cost coefficient* from A to B, denoted $\gamma(A, B)$. It is estimated by the network manager and specified as part of problem instance.

The coefficients $\gamma(u, v)$ of these linear link cost functions satisfy the triangle inequality:

$$(\forall u, v, w \in N) \quad \gamma(u, v) \leq \gamma(u, w) + \gamma(w, v) \tag{2.1}$$

That is, it is never more expensive to build a link directly between a pair of switches than it is to build links on an indirect path between the switches. If it is impossible to install a link in a "straight line" between the switches, one may always install it on an indirect path, possibly through places where other switches are located. Such a link need not be attached to the switches in question; it may simply occupy a small amount of space in those locations. As long as a shortest indirect path is chosen, the coefficients $\gamma(u, v)$ satisfy the triangle inequality.

Another kind of link cost function is a *step* or *staircase* cost function. These are of the form $b\lceil cap/a \rceil$, where cap is the desired capacity of the link. This is a realistic model, since links are often offered in discrete chunks of a given capacity a (e.g., T3 $\approx$ 45 Mb/s, OC-3 $\approx$ 155 Mb/s), where each chunk costs b dollars.

We will focus on linear link cost functions for most of this dissertation. However, several of the results, particularly the lower bounds, apply to step cost functions (see Section 3.4). Also, linear link cost functions are a close approximation to step cost functions when the traffic magnitudes are at least a few multiples of the chunk size b.

The network manager also knows the traffic limits, and wants to know how much capacity to install between each pair of switches so that the resulting network is nonblocking with respect to this traffic. We call this the *network configuration* problem.

The network configuration problem described below is actually a whole class of problems, one for each kind of routing algorithm. For example, there is the network configuration problew with fixed path routing, studied in Chapters 3, 4, and 5. The complexity of the problem might vary significantly with the routing algorithm chosen. It would be ideal if an efficient network configuration algorithm could determine the best routing algorithm to use, but this appears unrealistic.

Nonblocking network configuration with routing algorithm A

> INSTANCE: A set of switches N. For each ordered pair of distinct switches u, v, a cost coefficient $\gamma(u, v)$. These coefficients should satisfy the triangle inequality (2.1). Traffic limits $\mathcal{T}$.
>
> SOLUTION: A capacity $cap(u, v)$ for each switch pair u, v. This assignment of capacity should make the network nonblocking for traffic $\mathcal{T}$ when routing algorithm A is used.
>
> SOLUTION COST: The cost of the network is the sum of the costs of each link:
>
> $$\sum_{u,v \in N} \gamma(u, v) cap(u, v)$$
>
> OBJECT: Find a solution with minimum cost.

When working towards a solution to the network configuration problem, it is often useful to consider the following more restricted *link dimensioning* problem. The main difference is that now we are given more restrictions on the routing algorithm as part of the instance. For example, in a link dimensioning problem with fixed path routing, the problem instance could also specify a particular table π of paths. The solution cost and the objective are the same as before.

Nonblocking network link dimensioning with routing algorithm A

INSTANCE: A set of switches N. For each ordered pair of distinct switches u, v, a cost coefficient $\gamma(u, v)$. These coefficients should satisfy the triangle inequality (2.1). Traffic limits $\mathcal{T}$. Additional parameters constraining routing algorithm A.

SOLUTION: A capacity $cap(u, v)$ for each switch pair u, v. This assignment of capacity should make the network nonblocking for traffic $\mathcal{T}$ when the routing algorithm A, constrained as specified, is used.

SOLUTION COST:

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

OBJECT: Find a solution with minimum cost.

Finally, a still more restricted problem is the *network analysis* problem. In this problem we are given the network completely, with link capacities, as well as the traffic limits and routing algorithm. It is a decision problem, in which the object is to determine whether the network can ever block. Some routing algorithms are more difficult to analyze than fixed path routing, and it is useful to study this problem in such cases. See Chapter 8.

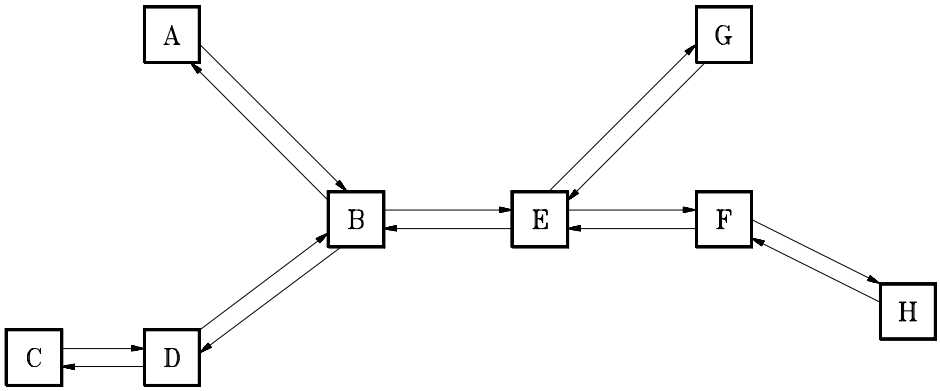

Figure 2.2: A tree network

Nonblocking network analysis with routing algorithm A

INSTANCE: A network $\mathcal{N} = (N, L, cap)$. Traffic limits $\mathcal{T}$. Additional parameters constraining routing algorithm A.

QUESTION: Is there a sequence of requests compatible with $\mathcal{T}$ such that at least one of the requests blocks?

2.4. Tree and star networks

A tree network is a directed graph obtained by starting with a tree (i.e., a connected, acyclic, undirected graph) and replacing each undirected edge with two oppositely directed edges between the same pair of vertices. For example, the network of Figure 2.1 is a tree network, and so is any network with the links shown in Figure 2.2.

For any link l in a tree network $\mathcal{N}$, let X_l be the set of switches on its "source side." That is, if $l = (u, v)$, then X_l is the set of switches in the connected component of $\mathcal{N} - \{(u, v), (v, u)\}$ that contains switch u. All other switches, those in $N - X_l$, are on the "destination side" of l. For example, the link (B, E) in the network of Figure 2.2 has the switches $X_{(B,E)} = \{A, B, C, D\}$ on its source side and $\{E, F, G, H\}$ on its destination side.

A star network is a special case of a tree network, one in which there is a "center" switch, and all other switches are directly attached to the center by two oppositely directed links. The network of Figure 2.1 is a star network with center B, but the one in Figure 2.2 is not a star.

The *underlying undirected graph* of a network is the undirected graph obtained by replacing each pair of directed edges $(u, v), (v, u)$ with the undirected edge $\{u, v\}$.

3. SIMPLE FLAT TRAFFIC

3.1. Definition

Every switch u has a *source termination limit* $\alpha(u)$ and a *destination termination limit* $\omega(u)$. The termination limit $\alpha(u)$ is a number that represents the maximum total rate of all connections in which u may be a source, and $\omega(u)$ is the maximum total rate of all connections in which u may be a destination. For example, if $\alpha(u) = 50$ Mb/s, then u may simultaneously be a source in connections with rates 5, 10, and 35 Mb/s, but then it could not be a source in any more connections until an existing connection is removed.

Given a set of switches N and a collection of connection requests $\mathcal{R}$, define the *source usage* and *destination usage* of a switch u under requests $\mathcal{R}$ as

$$\alpha\text{-usage}\,(u,\mathcal{R}) = \sum_{(u,v,\rho)\in\mathcal{R}} \rho \tag{3.1}$$

$$\omega\text{-usage}\,(u,\mathcal{R}) = \sum_{(v,u,\rho)\in\mathcal{R}} \rho \tag{3.2}$$

Let flat traffic limits $\mathcal{T} = (\alpha,\omega)$ be given. The set of requests $\mathcal{R}$ *is compatible with traffic limits* $\mathcal{T}$ if

$$(\forall u \in N) \quad \Big(\alpha\text{-usage}\,(u,\mathcal{R}) \le \alpha(u) \ \wedge \ \omega\text{-usage}\,(u,\mathcal{R}) \le \omega(u)\Big) \tag{3.3}$$

That is, no switch is involved in more requests than its termination limits allow.

Recall from the example in Chapter 1 that if the network in Figure 2.1 is presented with the set of requests $\mathcal{R}_1 = \{(C,D,35),(A,D,90),(C,A,100),(C,A,125),(C,D,40)\}$,

Table 3.1: Simple traffic limits

u	$\alpha(u)$	$\omega(u)$
A	450	450
B	0	0
C	300	300
D	450	150

the last request blocks. We also saw there that if $\omega(D) = 150$, then the last request would not be allowed. Now we can see that $\omega\text{-}usage(D, \mathcal{R}_1) = 35 + 90 + 40 = 165 > \omega(D)$. Hence $\mathcal{R}_1$ is not compatible with the traffic limits. The request set $\mathcal{R}_2 = \{(C, D, 35), (A, D, 90), (C, A, 100), (C, A, 125)\}$ is compatible, however, and any network that is nonblocking for these example traffic limits should be able to satisfy all connections in $\mathcal{R}_2$ simultaneously. The α, ω values for all switches are given in Table 3.1. Note that $\alpha(B) = \omega(B) = 0$. This is because B has no terminals attached to it, and therefore cannot be a source or destination of connections. It may only be an intermediate switch in a connection. In telephony, such switches are called tandem switches.

3.2. Link dimensioning in tree networks

Given a particular tree network and traffic limits α, ω, we can independently compute the minimum necessary capacity for each link so that the network is nonblocking. The minimum necessary capacity also happens to be the maximum possible usage of the link. Define $\alpha(X) = \sum_{u \in X} \alpha(u)$ for any set of switches X, and similarly for $\omega(X)$, and let $\lambda^*(l)$ denote the maximum usage of link l in any state of the network.

Lemma 3.1 *Let l be a link in tree network $\mathcal{N}$ with source side X_l, destination side $N - X_l$, and traffic limits α, ω. Then $\lambda^*(l) = \min\{\alpha(X_l), \omega(N - X_l)\}$.*

Proof: First we show that the usage of link l can never be more than $\min\{\alpha(X_l), \omega(N\text{–}X_l)\}$, and then show that it can be that large.

Consider any compatible set of requests $\mathcal{R}$. In a tree network, there is exactly one state $\mathcal{S}$ that realizes this set of requests, because there is exactly one path between any pair of switches. Let $\mathcal{S}' \subseteq \mathcal{S}$ be the connections that use link l. The requests that caused these connections to appear are exactly those with some switch in X_l as their source and some switch in $N\text{–}X_l$ as their destination. Denote this set of requests by $\mathcal{R}'$. Now we can see that

$$\begin{aligned}
\lambda(l, \mathcal{S}) &= \sum_{c \in \mathcal{S},\, c \text{ uses link } l} \rho(c) && \{\text{defn. of } \lambda\} \\
&= \sum_{c \in \mathcal{S}'} \rho(c) && \{\text{defn. of } \mathcal{S}'\} \\
&= \sum_{(u,v,\rho) \in \mathcal{R}'} \rho && \{\text{defn. of } \mathcal{R}'\} \\
&= \sum_{u \in X_l} \sum_{(u,v,\rho) \in \mathcal{R}'} \rho && \{\text{rearranging sums}\} \\
&= \sum_{u \in X_l} \alpha\text{-}usage\,(u, \mathcal{R}') && \{\text{defn. of } \alpha\text{-}usage\} \\
&\leq \sum_{u \in X_l} \alpha(u) && \{\mathcal{R}' \subseteq \mathcal{R} \text{ is compatible, and defn. (3.3)}\} \\
&= \alpha(X_l)
\end{aligned}$$

We can give an analogous derivation using ω-*usage* and ω. The result is $\lambda(l, \mathcal{S}) \leq \omega(N\text{–}X_l)$. Therefore $\lambda(l, \mathcal{S}) \leq \min\{\alpha(X_l), \omega(N\text{–}X_l)\}$.

To construct a state in which $\lambda(l, \mathcal{S}) = \min\{\alpha(X_l), \omega(N\text{–}X_l)\}$, simply make requests with source in X_l and destination in $N\text{–}X_l$ until it is no longer possible. ■

If we set the capacity of any link l to be less than its maximum possible usage $\lambda^*(l)$, then the network can block. Simply make a set of requests that gives the maximum usage on l, and one or more of the last requests made block because of insufficient capacity on link l. If we set the capacities of all links in a tree network to their maximum possible

usage, then obtain a nonblocking network. It is impossible to overflow a link because of the traffic limits. Thus the following link dimensioning algorithm works when the link costs are any nondecreasing functions of capacity, not just linear functions.

For all links l in the tree, let $cap(l) = \lambda^*(l)$

The most obvious way of implementing this algorithm requires $\Theta(n)$ time per link, for a total of $\Theta(n^2)$ time. The following algorithm is faster.

Compute $\alpha(N) = \sum_{u \in N} \alpha(u)$ and $\omega(N) = \sum_{u \in N} \omega(u)$

While there are at least 2 nodes left in the tree network $\mathcal{N}$ do

Pick any leaf switch u and let its only neighbor be v

$cap(u, v) := \min\{\alpha(u),\ \omega(N) - \omega(u)\}$

$cap(v, u) := \min\{\alpha(N) - \alpha(u),\ \omega(u)\}$

$\alpha(v) := \alpha(v) + \alpha(u)$

$\omega(v) := \omega(v) + \omega(u)$

Remove u, (u, v), and (v, u) from $\mathcal{N}$

End while

This algorithm can be implemented to run in $\Theta(n)$ time.

The result above solves the link dimensioning problem for a given tree network, but does not determine which tree network is cheapest. One heuristic we analyze later is to compute the cost of each star network, and choose the cheapest one. There are only $n = |N|$ such networks and we can compute the cost of each one in $\Theta(n)$ time by using the link dimensioning algorithm above, giving a $\Theta(n^2)$ time algorithm to find the cheapest star network.

For the example instance with flat traffic limits in Table 3.1, link costs γ are given in Table 3.2. The costs are the Euclidean distance between the switches, where the

Table 3.2: Link cost coefficients $\gamma(u, v)$

	A	B	C	D	Position
A	0	250	472	500	(100,500)
B		0	255	336	(250,300)
C			0	472	(500,250)
D				0	(100,0)

positions are given in the table. When we compute network costs, we consider these numbers to be the cost in dollars per 150 Mb/s link. The cheapest star network is the one with center switch B. It has link capacities as shown in the example network of Figure 2.1, and cost $579600/150 = \$3864$.

Why is a heuristic that only finds the cheapest star network worth considering? We show later that if the cheapest star network is not the cheapest among all nonblocking networks, then at least it is very close.

3.3. Computational complexity

In this section we prove that the following network configuration problem is MAX SNP-hard [57]. By the results of Arora et al. [5], this implies that if P $\neq$ NP, then there is no polynomial time approximation scheme (PTAS) for this problem. A PTAS is a family of algorithms parameterized by $\epsilon > 0$. Each member of the family runs in polynomial time, and is guaranteed to produce a solution that costs at most $1 + \epsilon$ times more than an optimal solution.

Thus, there exists some $\epsilon > 0$ for which no polynomial time algorithm is guaranteed to find a solution with cost at most $1 + \epsilon$ over optimal, unless P = NP. Note that a MAX SNP-hard problem is also NP-hard.

Nonblocking network configuration with simple flat traffic limits

INSTANCE: A set of switches N. For each ordered pair of distinct switches u, v, a cost coefficient $\gamma(u, v)$. These coefficients satisfy the triangle inequality (2.1). Traffic limits $\mathcal{T} = (\alpha, \omega)$.

SOLUTION: A capacity $cap(u, v)$ and a path $\pi(u, v)$ for each switch pair u, v. This assignment of capacity should make the network nonblocking for traffic $\mathcal{T} = (\alpha, \omega)$ when the fixed path routing algorithm with paths π is used.

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

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

OBJECT: Find a solution with minimum cost.

We refer to this problem as NETWORK CONFIGURATION. We prove that it is MAX SNP-hard by giving a special kind of transformation, called an *L-reduction*, from the following problem to the NETWORK CONFIGURATION problem.

Steiner(1,2)

INSTANCE: An undirected complete graph $G = (V, E)$. Every edge e has a weight $w(e) \in \{1, 2\}$. A set of terminal vertices $S \subseteq V$.

SOLUTION: A connected subgraph $G' = (V', E')$ of G that contains all vertices in S.

SOLUTION COST: The cost of the subgraph is the sum of the edge weights:

$$\sum_{e \in E'} w(e)$$

OBJECT: Find a solution with minimum cost.

Steiner(1,2) was proved to be MAX SNP-hard by Bern and Plassman [10]. The following definition of an L-reduction is due to Papadimitriou and Yannakakis [57].

Definition 3.2 *(Papadimitriou, Yannakakis) Let* Π *and* Π' *be two optimization (maximization or minimization) problems. We say that* Π L-reduces *to* Π' *if there are two polynomial time algorithms* f, g, *and constants* $a, b > 0$ *such that for each instance* I *of* Π*:*

(a) Algorithm f *produces an instance* $I' = f(I)$ *of* Π'*, such that the optima of* I *and* I', $OPT(I)$ *and* $OPT(I')$, *respectively, satisfy* $OPT(I') \leq a \cdot OPT(I)$.

(b) Given any solution of I' *with cost* c', *algorithm* g *produces a solution of* I *with cost* c *such that* $|c - OPT(I)| \leq b\,|c' - OPT(I')|$.

The basic idea of this definition is that if problem Π L-reduces to problem Π', and if there is an approximation algorithm for problem Π' that is guaranteed to produce a solution with cost at most $1 + \epsilon$ times more than optimal, then we can construct a polynomial time approximation algorithm for problem Π that is guaranteed to produce a solution with cost at most $1 + ab\epsilon$ times more than optimal.

Theorem 3.3 *The problem* NETWORK CONFIGURATION *is* MAX SNP*-hard.*

Proof: The algorithm f of our L-reduction from Steiner(1,2) to NETWORK CONFIGURATION is as follows. Let $N = V$ and let $\gamma(u, v) = w(e)$ for all $u, v \in N$, where e is the edge between vertices u and v. Note that these γ values satisfy the triangle inequality, because G is a complete graph and all edge weights are either 1 or 2. Pick an arbitrary "root" vertex $r \in S$, and let $\alpha(r) = 1$ and $\omega(r) = 0$. For all $u \in S - \{r\}$, let $\alpha(u) = 0$ and $\omega(u) = 1$. For all $u \in N - S$, let $\alpha(u) = \omega(u) = 0$.

We show that this algorithm f and an algorithm g described below satisfy the definition of an L-reduction, where $a = b = 1$. Thus this reduction can be used to turn any approximation algorithm for NETWORK CONFIGURATION into an approximation

algorithm for Steiner(1,2) with the same performance guarantee. In order to show this, we consider the set of all feasible solutions of the problem instances.

For any instance I of Steiner(1,2), the set of all feasible solutions $FEAS(I)$ is the set of connected graphs containing all vertices in S. The *good* solutions $GOOD(I)$ of instance I are defined to be the set of all tree solutions such that every leaf vertex of the tree is in S. Note that the set of optimal solutions $BEST(I)$ is a subset of $GOOD(I)$.

For any transformed instance $I' = f(I)$ of NETWORK CONFIGURATION, the set of all feasible solutions $FEAS(I')$ can be described by the link capacities $cap(u, v)$ for every $u, v \in N$, and the collection of directed paths π from the root vertex r to every other vertex in S. The other paths will never be used, due to the traffic limits $\mathcal{T} = (\alpha, \omega)$. Every link in some path $\pi(r, u)$, where $u \in S - \{r\}$, must have capacity at least 1 in a feasible solution, because otherwise the network could block when a rate 1 connection request is made from r to u.

The good solutions $GOOD(I')$ for I' are restricted in two ways. First, the link capacities are restricted. Any link not in the set $\Pi(r, S-\{r\}) = \bigcup_{u \in S-\{r\}} \pi(r, u)$ is never used, and should have a capacity of 0. Any link in the set $\Pi(r, S - \{r\})$ must have a capacity at least 1 in any feasible solution, but it need not have a capacity any larger than 1. This is because 1 is the maximum total rate of all connections that may exist simultaneously, since r is the only vertex with α larger than 0.

The second restriction is that there may be at most one path from r to any other vertex in the set of edges $\Pi(r, S - \{r\})$. Equivalently, the in-degree of any vertex is at most 1; also, the set of edges forms a directed tree with root r and all paths directed away from r.

Note that any feasible but non-good solution may be converted to a good solution with less cost in polynomial time. The capacities may be reduced easily, if necessary,

and if $\Pi(r, S-\{r\})$ contains more than one path from r to some other vertex u, then all but one of those paths (chosen arbitrarily) can be removed, and the paths $\pi(r, v)$ that used links in those paths can be rerouted along the remaining path to u. Note that the set of optimal solutions for I', $BEST(I')$, is a subset of $GOOD(I')$.

For every solution to a Steiner(1,2) instance I that is in $GOOD(I)$, there is a corresponding solution to instance $I' = f(I)$ of NETWORK CONFIGURATION that is in $GOOD(I')$, and it has the same cost. Simply take the tree solution to I with cost c, and let $\pi(r, u)$ be the directed path from r to u in the tree. When we assign a capacity of 1 to all links in $\Pi(r, S-\{r\})$ and capacity 0 to other links, the cost is exactly c.

Similarly, for every solution to an instance $I' = f(I)$ of NETWORK CONFIGURATION that is in $GOOD(I')$, there is a corresponding solution to instance I of Steiner(1,2) that is in $GOOD(I)$, and it has the same cost. Simply take the set of edges $\Pi(r, S-\{r\})$ and make them all undirected.

Therefore, $OPT(I') = OPT(I)$ for all instances I of Steiner(1,2). This satisfies part (a) of Definition 3.2, where $a = 1$.

Algorithm g of the L-reduction starts with an arbitrary feasible solution to the NETWORK CONFIGURATION instance $I' = f(I)$ with cost c'. If it is not a good solution, then g converts it to a good solution with cost $c_2 < c'$, otherwise it keeps the good solution. Finally, it converts the good solution of I' to a good solution of I with cost $c \leq c'$. Since $OPT(I') = OPT(I)$, we conclude that $c - OPT(I) \leq c' - OPT(I')$, and part (b) of Definition 3.2 is satisfied with $b = 1$. ■

Even though the general problem NETWORK CONFIGURATION is MAX SNP-hard, we conjecture that it is solvable in polynomial time when $\alpha(N) = \omega(N)$. For further evidence supporting this conjecture, see Section 3.7.

3.4. A lower bound

In this section we present an algorithm for computing a lower bound on the cost of *any* nonblocking network, using any routing algorithm, given only the traffic limits and the link cost coefficients γ. It works when all link cost functions are linear, and the link cost coefficients satisfy the triangle inequality (2.1).

Since we compute a lower bound, the method to be described can also be used when each link cost function is at least some linear function. For example, the step cost function $b\lceil cap/a \rceil$ described in Section 2.3 is at least $(b/a)cap$. Therefore a lower bound can be computed using the method below by assuming that the link cost function is $(b/a)cap$ for the given link. However, the quality of the lower bound may not be as good in such instances.

A lower bound is useful when we have an instance I of a network configuration problem and a nonblocking network $\mathcal{N}$ with cost $cost(\mathcal{N})$, but we do not know how close its cost is to the minimum possible. Suppose we have computed a lower bound $LB(I)$ on the cost of any nonblocking network for I. In particular, $LB(I)$ is a lower bound on the cost $OPT(I)$ of the cheapest solution. Therefore $LB(I) \leq OPT(I) \leq cost(\mathcal{N})$, which implies $\frac{cost(\mathcal{N})}{OPT(I)} \leq \frac{cost(\mathcal{N})}{LB(I)}$. For example, if $cost(\mathcal{N}) = 125$ and $LB(I) = 100$, then we are guaranteed that our solution costs at most 25% more than the minimum cost solution, and it may be closer.

We obtain an equivalent way of specifying compatible request sets by introducing a set of variables $\{x_{u,v} : u, v \in N, u \neq v\}$, where $x_{u,v}$ represents the total rate of all requests from u to v, $x_{u,v} = \sum_{(u,v,\rho)\in\mathcal{R}} \rho$. Therefore, the equations for α-*usage* (3.1) and ω-*usage* (3.2) may be written

$$\alpha\text{-}usage\,(u, \mathcal{R}) \;=\; \sum_{v \in N-\{u\}} x_{u,v} \tag{3.4}$$

$$\omega\text{-}usage(v, \mathcal{R}) = \sum_{u \in N - \{v\}} x_{u,v} \tag{3.5}$$

Hence we may rewrite the condition "$\mathcal{R}$ is compatible with $\mathcal{T} = (\alpha, \omega)$" (3.3) as

$$\begin{aligned} \sum_{v \in N - \{u\}} x_{u,v} &\leq \alpha(u) \quad \forall u \in N \\ \sum_{u \in N - \{v\}} x_{u,v} &\leq \omega(v) \quad \forall v \in N \\ x_{u,v} &\geq 0 \qquad \forall u, v \in N, u \neq v \end{aligned} \tag{3.6}$$

It should be clear that for every compatible request set $\mathcal{R}$, there is a unique x that satisfies system (3.6), and for every $x \neq 0$ satisfying system (3.6), there are many corresponding compatible request sets $\mathcal{R}$.

Suppose x, satisfying (3.6), is non-zero in only one variable $x_{u,v} = \rho$. This represents any set of compatible requests from u to v with total rate ρ. In any network that is nonblocking for $\mathcal{T}$, there must be a path from u to v such that all links in the path have at least ρ units of bandwidth. Given that the coefficients γ satisfy the triangle inequality, the cheapest network that can be built, to handle these requests only, is the one containing the single link (u, v) with capacity ρ. The cost of this network is $\gamma(u, v) \cdot \rho$, which is a lower bound on the cost of any network that is nonblocking for $\mathcal{T}$.

Similarly, for any x satisfying (3.6), the cheapest network that can handle a corresponding request set is the one that contains links directly between the switch pairs involved in requests. The cost of this network is

$$LB(I, x) = \sum_{u,v \in N, u \neq v} \gamma(u, v) x_{u,v} \tag{3.7}$$

The largest lower bound of this form can be obtained by maximizing $LB(I, x)$ subject to the inequalities (3.6).

This problem can be solved using a general linear programming algorithm. Two examples are Dantzig's simplex method [16], and Karmarkar's algorithm [4], which has

worst-case polynomial running time. It can also be solved by noting that this class of linear programs is equivalent to a restricted class of maximum cost flow problems [60, Section 8.4]. An instance of this problem consists of a directed graph $G = (V, E)$, where there is a distinguished source vertex $s \in V$, a sink vertex $t \in V$, and each edge $e \in E$ has a capacity $cap(e)$ and a cost per unit flow $cost(e)$. A feasible solution to this problem is a flow f on each edge satisfying the following constraints:

$$\begin{aligned} \sum_{(v,u)\in E} f(v,u) &= \sum_{(u,v)\in E} f(u,v) \quad \forall u \in V - \{s,t\} \\ f(u,v) &\leq cap(u,v) \qquad \forall (u,v) \in E \end{aligned} \tag{3.8}$$

The first constraint restricts the flow into a vertex to be equal to the flow out of the vertex, for all vertices other than s and t. The second constraint restricts the flow on each edge to be no larger than the capacity of the edge. The object is to find a feasible flow f with maximum cost $\sum_{(u,v)\in E} cost(u,v) f(u,v)$.

The lower bound graph $\mathcal{L} = (V, E, cap, cost)$ is a maximum cost flow instance with source s and sink t. It is defined below in terms of the network switches N, the traffic limits $\mathcal{T} = (\alpha, \omega)$, and the link cost coefficients γ. The edges are given in the form (u, v, cap, c), where the edge is from vertex u to vertex v and has capacity cap and cost c.

$$\begin{aligned} V &= \{s,t\} \cup \{u_s, u_d : u \in N\} \\ E &= \{(s, u_s, \alpha(u), 0),\ (u_d, t, \omega(u), 0) : u \in N\} \cup \\ &\quad \{(u_s, v_d, \infty, \gamma(u,v)) : u, v \in N, u \neq v\} \end{aligned} \tag{3.9}$$

The subscript s is short for "source", and the subscript d is short for "destination".

The lower bound graph for the traffic limits of Table 3.1 and the γ values of Table 3.2 is shown in Figure 3.1. In the literature, this problem is often called the minimum cost flow problem or the minimum cost circulation problem. The most efficient algorithms for this problem known to the author are cited by Goldberg and Tarjan [38].

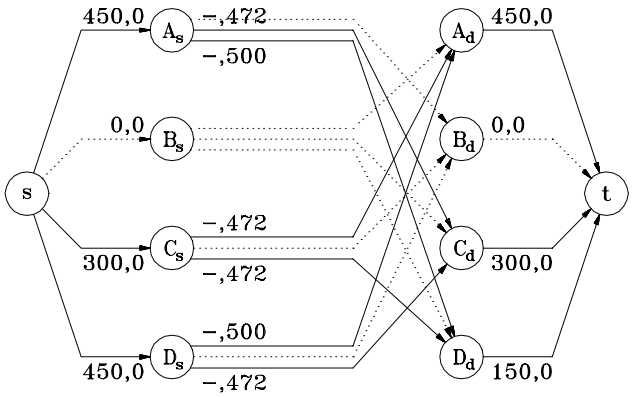

Figure 3.1: Lower bound graph for instance in Tables 3.1 and 3.2. Edges are labeled with their capacity and cost, in that order. An infinite capacity is shown as a dash (—). The dotted edges either have capacity 0, or they have flow 0 in every feasible flow. Their labels are not shown to avoid cluttering the figure.

To see why the linear program is equivalent to the maximum cost flow problem, let $x_{u,v} = f(u_s, v_d)$ for all $u, v \in N, u \neq v$. By the flow conservation constraints of (3.8), we must have $f(s, u_s) = \sum_{v \in N - \{u\}} x_{u,v}$. The system of linear inequalities (3.6) restricts this value to be at most $\alpha(u)$, which is exactly the capacity of edge (s, u_s) in $\mathcal{L}$. Similarly, edges of the form (v_d, t) have flow $\sum_{u \in N - \{v\}} x_{u,v}$, and the edge capacities limit this quantity to be at most $\omega(v)$. Finally, the cost of the flow is $\sum_{u,v \in N, u \neq v} cost(u_s, v_d) f(u_s, v_d)$, which is exactly the objective of the linear program, Equation (3.7).

A maximum cost flow for the lower bound graph in Figure 3.1 is the flow with $f(A_s, C_d) = 300$, $f(A_s, D_d) = 150$, and $f(D_s, A_d) = 450$, with a cost of $441600/150 =$ \$2944. Thus the star network with center B and cost \$3864 is at most 31.2% more expensive than optimal.

We close this section with an example that shows that the algorithm for computing a lower bound does not always work when the cost coefficients γ do not satisfy the triangle inequality, i.e., the value computed by the lower bound algorithm is larger than the cost of a nonblocking network. In practice, all network configuration instances will satisfy the triangle inequality, because if it is more expensive to install a link on a direct

path between two switches, we may simply install it on a cheapest indirect path, and use that cost in the problem instance. Such costs satisfy the triangle inequality.

The example instance has three switches A, B, and C. All α and ω values equal 1, and the cost coefficients are $\gamma(A,B) = \gamma(B,A) = 1$, $\gamma(B,C) = \gamma(C,B) = 2$, and $\gamma(A,C) = \gamma(C,A) = 4$. These costs violate the triangle inequality because $\gamma(A,C) > \gamma(A,B) + \gamma(B,C)$. The lower bound has value 8, as demonstrated by the most costly request set $\{(A,C,1),(C,A,1)\}$. A network that is nonblocking for these traffic limits contains the links (A,B), (B,A), (B,C), and (C,B), all with capacity 1. The network has cost 6.

This occurs because the lower bound is made high by using the large cost $\gamma(A,C) = 4$ directly between A and C, but the nonblocking network can get between A and C more cheaply by going through B. This cannot happen when the γ values satisfy the triangle inequality.

3.5. Experimental results

In this section, we compare the value of the lower bound to the cost of the cheapest star network. This is done for randomly generated problem instances.

In the following, whenever we say that a value is generated randomly in some interval, we mean that it is generated from a uniform distribution on the interval. Similarly, when we place a point randomly in some rectangle, we mean that its location is generated randomly with a uniform distribution on the area.

A single experiment consists of choosing a number of switches n, and a range of termination limit values $[\alpha_{\mathrm{lo}}, \alpha_{\mathrm{hi}}]$. Generate a random instance as follows. All switches are placed randomly in a unit square. Link cost coefficients $\gamma(u,v)$ are set equal to

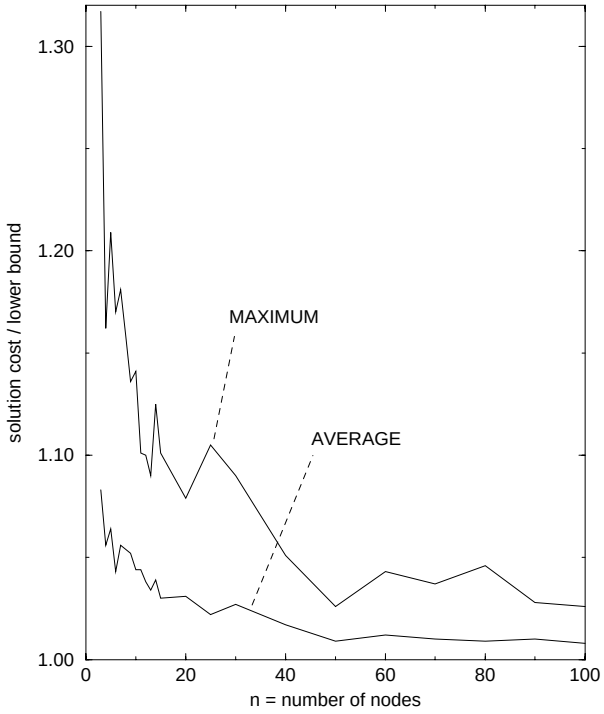

Figure 3.2: Experimental results for flat traffic

the Euclidean distance between u and v. For each switch u, choose the integer $\alpha(u)$ randomly in the interval $[\alpha_{\rm lo}, \alpha_{\rm hi}]$, and set $\omega(u) = \alpha(u)$.

After the instance has been generated, the cheapest star network is computed as described in Section 3.2, and a maximum cost flow in the lower bound graph is found. The *performance ratio*, which is a real value no less than 1, is equal to the cost of the star network divided by the lower bound.

In Figure 3.2, each data point on the lower curve is the average of the performance ratios of 50 randomly generated instances, all generated with the same values of n, $\alpha_{\rm lo} = 10$, $\alpha_{\rm hi} = 20$. Experiments were done for values of n ranging over the set $\{3, 4, 5, 6, \ldots, 14, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100\}$. The same experiments were performed with α randomly drawn from the interval $[1, 30]$, and with all α values equal to 10. The resulting plots are not significantly different than those in Figure 3.2, so they have not been shown.

Note that even at the worst (highest) part of the curve for small n, the average performance ratio is no more than about 1.08. This shows that the cheapest stars are within 8% of optimal on average for small n, and even closer for large n. The maximum (not the average) of all 50 of the individual performance ratios for $n = 3$ is 1.317. For $n \leq 7$, we have exhaustively enumerated all n^{n-2} tree networks of the switches, not just star networks, to find the one that gives the cheapest nonblocking network. This enumeration was done using an algorithm due to Gabow and Myers [27]. In every case, a star network was among the cheapest solutions. This led to the conjecture that the minimum cost star network is the cheapest among all tree networks. We have proved this conjecture for $\alpha(N) = \omega(N)$, and the proof is given in Section 3.7.

When the performance ratio is large, it means that either the lower bound is far below the optimal cost, the cheapest star network is far above the optimal cost, or both. We conjecture that the lower bound is far below the optimal cost for small n, and the minimum cost star network is optimal. More precisely, we conjecture that when $\alpha(N) = \omega(N)$, link cost functions are linear with capacity, and link cost coefficients satisfy the triangle inequality, the cheapest star network is the minimum cost nonblocking network among all nonblocking networks. The proof mentioned above proves that the minimum cost star is cheapest among all tree networks, but not necessarily among all solutions.

When n gets large, we see that the performance ratio gets closer to 1. This means that both the lower bound and the minimum cost star network are getting closer to the optimal value. This prompted the search for a proof that the curve does approach 1 as n gets large. In Section 3.8, we show that the probability the performance ratio is at most $1 + \epsilon$ goes to 1 as n gets large, for any $\epsilon > 0$. This result holds for other methods of randomly generating instances than that used in this section.

3.6. Star networks require least total capacity among all trees

Before proving the result in the next section, we first examine the special case when all γ values are the same. This case has practical significance for network configuration instances in which the cost of cable termination is the majority of the cost of installing links.

In this special case, the cheapest tree network is the star network with center C, where C is any switch with the largest value of $\alpha(C) + \omega(C)$ among all switches. Such a center C may be found in $O(n)$ time, and nonblocking link capacities may also be found in $O(n)$ time by the algorithm in Section 3.2.

Further, we conjecture that for an instance of this restricted type, the star network with center C is cheapest among all nonblocking networks, not just among tree networks.

Theorem 3.4 *When all γ values are equal and traffic limits are given by $\mathcal{T} = (\alpha, \omega)$, every tree network costs at least as much as the star network with center switch C, where C is any switch with the maximum value of $\alpha(C) + \omega(C)$ among all switches.*

Proof: Let $h(u) = \alpha(u) + \omega(u)$ for all $u \in N$, and let $h(X) = \sum_{u \in X} h(u)$ for any set of switches $X \subseteq N$. Let C be a switch with maximum value of $h(C)$ among all switches. In a star network with center C, the total capacity of the links (u, C) and (C, u) is

$$
\begin{aligned}
& cap(u,C) + cap(C,u) \\
&\quad = \min\{\alpha(u),\, \omega(N - \{u\})\} \;+\; \min\{\alpha(N - \{u\}),\, \omega(u)\} && \{\text{Lemma 3.1}\} \\
&\quad = \min\{\alpha(u),\, \omega(N) - \omega(u)\} \;+\; \min\{\alpha(N) - \alpha(u),\, \omega(u)\} \\
&\quad = \min\{\alpha(u) + \omega(u),\, \alpha(N) + \omega(N) - (\alpha(u) + \omega(u)),\, \alpha(N),\, \omega(N)\} \\
& && \{\text{properties of min}\} \\
&\quad = \min\{h(u),\, h(N) - h(u),\, \alpha(N),\, \omega(N)\} && \{\text{defn. of } h\} \\
&\quad = \min\{h(u),\, \alpha(N),\, \omega(N)\} && \{\text{see below}\}
\end{aligned}
\tag{3.10}
$$

The last step is justified by the observation

$$\begin{aligned} h(u) &\leq h(C) && \{ \text{ by choice of } C \ \} \\ &\leq \sum_{v \in N - \{u\}} h(v) && \{C \in N - \{u\}\} \\ &= h(N) - h(u) \end{aligned}$$

Now consider any tree network $\mathcal{N}$ for the switches N. Pick the same switch C used before and consider $\mathcal{N}$ as a rooted tree with C as the root and all other switches descendants of C. Let $S_C(u)$ be the set of descendants of u in this rooted tree (including u), for any $u \in N$. The total capacity necessary on the links $(u, p(u))$ and $(p(u), u)$ between a switch $u \neq C$ and its parent switch $p(u)$ is

$$\begin{aligned} & cap(u, p(u)) + cap(p(u), u) \\ &= \min\{\alpha(S_C(u)),\, \omega(N - S_C(u))\} \,+\, \min\{\alpha(N - S_C(u)),\, \omega(S_C(u))\} \\ & \qquad\qquad \{\text{Lemma 3.1}\} \\ &= \min\{\alpha(S_C(u)),\, \omega(N) - \omega(S_C(u))\} \,+\, \min\{\alpha(N) - \alpha(S_C(u)),\, \omega(S_C(u))\} \\ &= \min\{h(S_C(u)),\, h(N) - h(S_C(u)),\, \alpha(N),\, \omega(N)\} \\ & \qquad\qquad \{\text{properties of min, defn. of } h\} \end{aligned} \tag{3.11}$$

Now if we can show that (3.11) is always at least as large as (3.10) then the total capacity of the tree $\mathcal{N}$, $\sum_{u \in N - \{C\}} (cap(u, p(u)) + cap(p(u), u))$, is at least as large as the total capacity of the star network with center C, $\sum_{u \in N - \{C\}} (cap(u, C) + cap(C, u))$.

$h(S_C(u)) \geq h(u)$ follows immediately from $u \in S_C(u)$. Also

$$\begin{aligned} h(N) - h(S_C(u)) &= h(N - S_C(u)) \\ &\geq h(C) && \{C \in N - S_C(u)\} \\ &\geq h(u) && \{\text{choice of } C\} \end{aligned}$$

Therefore $\min\{h(S_C(u)), h(N) - h(S_C(u))\} \geq h(u)$. Now $x \geq y$ implies that $\min\{x, z\} \geq \min\{y, z\}$. With $x = \min\{h(S_C(u)), h(N) - h(S_C(u))\}$, $y = h(u)$, and $z = \min\{\alpha(N), \omega(N)\}$, we see that (3.11) $\geq$ (3.10) for all $u \in N - \{C\}$, and we have proved that all tree networks have at least as much total capacity as the star network with center switch C. ■

3.7. Star networks are almost cheapest among tree networks

Theorem 3.5 *Let traffic limits α, ω be given for a set of switches N, and let all link cost functions be linear with coefficients $\gamma(u,v)$ satisfying the triangle inequality. If $1+\epsilon = \max\left\{\frac{\alpha(N)}{\omega(N)}, \frac{\omega(N)}{\alpha(N)}\right\}$, and $1+\delta = \max_{u,v\in N}\left\{\frac{\gamma(u,v)}{\gamma(v,u)}\right\}$, then a cheapest star network costs at most $1+(1+\delta)\epsilon/2$ times more than the cheapest tree network.*

Note that for practical instances of the problem, the link costs are symmetric, so $\delta = 0$. For such instances, the upper bound is $1+\epsilon/2$.

We prove this theorem by showing that for any tree network, there is always a "centroid" switch c with the following property. The cost of the star network with center c is at most $1+(1+\delta)\epsilon/2$ times the cost of the original tree. Note that this does not say that the star with center c is the cheapest star network. However, if we find the cheapest star, then it must cost no more than $1+(1+\delta)\epsilon/2$ times the cheapest tree.

An interesting corollary of Theorem 3.5 results when we restrict $\alpha(N) = \omega(N)$.

Corollary 3.6 *Let traffic limits α, ω be given for a set of switches N, where $\alpha(N) = \omega(N)$, and let all link cost functions be linear with coefficients $\gamma(u,v)$ satisfying the triangle inequality. Then a star network is cheapest among all tree networks.*

Proof: Follows immediately from Theorem 3.5 since $\alpha(N) = \omega(N)$ implies $\epsilon = 0$. ■

Let $T = (V, E)$ be an undirected tree where each vertex v has a real weight $h(v) \geq 0$. Define $h(X)$ to be the sum of the weights of vertices in X, where X is either a subset of V or a subgraph of T. For any two distinct vertices u, v in T, let $f(u,v)$ be the first edge on the unique path from u to v in T, and define the *subtree of v with respect to u*, $S_u(v)$, to be the connected component of $T - f(v,u)$ that contains v. A vertex $c \in V$ is called a *centroid* if $h(S_c(v)) \leq w(T)/2$ for all vertices v adjacent to c.

Harary's definition of centroids [42] is equivalent to ours, except that his definition only includes trees in which all vertex weights are 1. Various versions of the following lemma have been proved as far back as 1869 [49]. This proof is original to the author, and is included here for completeness. See Bodlaender et al. [12] and Gabow [25] for other uses of centroids and some generalizations.

Lemma 3.7 *Let $T = (V, E)$ be an undirected tree, where each vertex $v \in V$ has a nonnegative weight $h(v)$. Then there exists a centroid $c \in V$.*

Proof: For any edge $e \in E$, define the *imbalance* of e, $I(e)$, to be $|h(S_u(v)) - h(S_v(u))|$, where $e = \{u, v\}$.

Let $e^* = \{u^*, v^*\}$ be an edge with minimum imbalance. Note that

$$(\forall u, v) \;\; \{u, v\} \in E \;\Rightarrow\; h(S_u(v)) + h(S_v(u)) = h(T) \tag{3.12}$$

If $I(e^*) = 0$, then $h(S_{u^*}(v^*)) = h(S_{v^*}(u^*)) = h(T)/2$, and both u^* and v^* are centroids.

If $I(e^*) > 0$, then suppose without loss of generality that $h(S_{v^*}(u^*)) < h(S_{u^*}(v^*))$, and so

$$I(e^*) = h(S_{u^*}(v^*)) - h(S_{v^*}(u^*)) \tag{3.13}$$

From Equation (3.12) it also follows that $h(S_{v^*}(u^*)) < h(T)/2 < h(S_{u^*}(v^*))$.

If u^* is the only vertex adjacent to v^*, then v^* is the only centroid of T. Call this Case 1.

Otherwise, let $v_1, \ldots, v_k$ be the neighbors of v^* other than u^*, and let $e_i = \{v^*, v_i\}$ for all i, $1 \le i \le k$. See Figure 3.3 for a diagram of the tree structure. Then we have the following for all i:

$$\begin{aligned} I(e_i) &= |h(S_{v_i}(v^*)) - h(S_{v^*}(v_i))| && \{\text{Defn. of } I\ \} \\ &= |h(T) - 2h(S_{v^*}(v_i))| && \{\text{Equation (3.12)}\ \} \end{aligned}$$

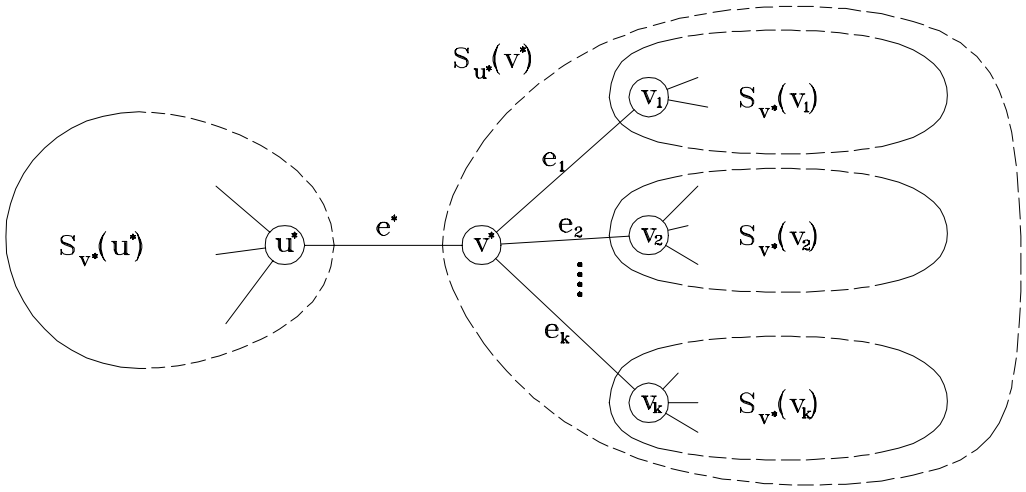

Figure 3.3: Tree structure in proof of Lemma 3.7

By the choice of e^*, we know that $(\forall i) I(e_i) \geq I(e^*)$. $|x| \geq y$ if and only if ($x \geq y$ or $-x \geq y$). Therefore $(\forall i) I(e_i) \geq I(e^*)$ is true if and only if

$$(\forall i) \;\; (h(T) - 2h(S_{v_*}(v_i)) \geq I(e^*)) \text{ or } \;\; (2h(S_{v_*}(v_i)) - h(T) \geq I(e^*))$$

Using Equations (3.12) and (3.13) we derive the equivalent condition

$$(\forall i) \;\; h(S_{v^*}(v_i)) \leq h(S_{v^*}(u^*)) \text{ or } \;\; h(S_{v^*}(v_i)) \geq h(S_{u^*}(v^*))$$

There are now two cases two consider. Case 2a is when $(\forall i) h(S_{v^*}(v_i)) \leq h(S_{v^*}(u^*))$. Recall that $h(S_{v^*}(u^*)) < h(T)/2$. Therefore vertex v^* is a centroid.

Case 2b is when there exists a vertex v_i such that $h(S_{v^*}(v_i)) \geq h(S_{u^*}(v^*))$. Since $S_{v^*}(v_i)$ is a subgraph of $S_{u^*}(v^*)$ and all vertex weights are nonnegative, we must have $h(S_{v^*}(v_i)) = h(S_{u^*}(v^*))$. This implies that $h(z) = 0$ for all vertices z in $S_{u^*}(v^*) - S_{v^*}(v_i)$. Now edge $\{v^*, v_i\}$ is also an edge of minimum imbalance, and we can "repeat the proof" on this edge. Since the graph is finite, this repetition eventually halts with either Case 1 or Case 2a. ■

Now we proceed with the proof of Theorem 3.5.

Proof: Suppose we are given a tree network $\mathcal{N}$ with underlying undirected tree T. Let $\mathcal{N}$ have the minimum necessary link capacities cap in order to be nonblocking. Let c be a centroid for T using the weights $h(u) = \alpha(u) + \omega(u)$. For all $u \in N - \{c\}$, define $p(u)$ to be the first switch on the path from u to c in $\mathcal{N}$ (possibly c itself). The cost of $\mathcal{N}$ is

$$cost(\mathcal{N}) = \sum_{x \in N-\{c\}} [cap(x, p(x))\gamma(x, p(x)) + cap(p(x), x)\gamma(p(x), x)] \tag{3.14}$$

Let $\mathcal{N}_c^*$ be the star network with switches N, center c, and minimum necessary link capacities cap^*. We write the cost of $\mathcal{N}_c^*$ in a form similar to (3.14) to compare the two costs more easily. The cost of $\mathcal{N}_c^*$ is

$$\begin{aligned} & cost(\mathcal{N}_c^*) \\ &= \sum_{u \in N-\{c\}} [cap^*(u, c)\gamma(u, c) + cap^*(c, u)\gamma(c, u)] \\ &\leq \sum_{u \in N-\{c\}} \left[cap^*(u, c) \sum_{(x,y) \in \pi(u,c)} \gamma(x, y) + cap^*(c, u) \sum_{(x,y) \in \pi(c,u)} \gamma(x, y) \right] \end{aligned} \tag{3.15}$$

Recall that $\pi(u, v)$ is the path in $\mathcal{N}$ from u to v. The inequality holds because γ satisfies the triangle inequality, Condition (2.1), and hence $\gamma(u, v)$ is at most the sum of γ values on any path from u to v, for any $u, v \in N$. Consider the sum of the first term of (3.15):

$$\sum_{u \in N-\{c\}} \left[cap^*(u, c) \sum_{(x,y) \in \pi(u,c)} \gamma(x, y) \right] \tag{3.16}$$

Let x be any vertex in $N - \{c\}$. In this sum, we are adding the quantity $cap^*(u, c)\gamma(x, p(x))$ once for each switch u in $S_c(x)$. Let U_x be the set of switches in $S_c(x)$, C_x the set of switches in $S_x(c)$, and $B_x = N - U_x - C_x$. Note that these sets are a partition of N. Thus (3.16) can be rewritten

$$\sum_{x \in N-\{c\}} \left[\left(\sum_{u \in U_x} cap^*(u, c) \right) \gamma(x, p(x)) \right]$$

Similarly for the second term of (3.15). Thus the entire expression can be rewritten as

$$\begin{aligned}
&\sum_{x\in N-\{c\}}\left[\left(\sum_{u\in U_x} cap^*(u,c)\right)\gamma(x,p(x)) + \left(\sum_{u\in U_x} cap^*(c,u)\right)\gamma(p(x),x)\right] \\
&\quad = \sum_{x\in N-\{c\}} [cap'(x,p(x))\gamma(x,p(x)) + cap'(p(x),x)\gamma(p(x),x)] \qquad (3.17)
\end{aligned}$$

where $cap'(x,p(x)) = \sum_{u\in U_x} cap^*(u,c)$ and $cap'(p(x),x) = \sum_{u\in U_x} cap^*(c,u)$.

We now have the two costs in a similar form. The only difference is that in the expression for $cost(\mathcal{N})$, (3.14), the capacities cap are those that make the network $\mathcal{N}$ nonblocking, but in the upper bound for $cost(\mathcal{N}_c^*)$, (3.17), the capacities are given by cap'. Next we show the values of cap and cap' are close by using c's properties as a centroid, and the condition $1+\epsilon = \max\left\{\frac{\alpha(N)}{\omega(N)}, \frac{\omega(N)}{\alpha(N)}\right\}$.

Suppose, without loss of generality, that $\alpha(N) \geq \omega(N)$ (the proof is very similar for the case $\alpha(N) \leq \omega(N)$). Then

$$\omega(N) \leq \alpha(N) = (1+\epsilon)\omega(N) \qquad (3.18)$$

Since c is a centroid for the tree with weights h, we know for all switches b adjacent to c that $h(S_c(b)) \leq h(N)/2$. If x is any switch in the tree, and b is the last switch on the path from x to c (possibly x itself), then

$$\begin{aligned}
h(S_c(b)) &= h(U_x \cup B_x) && \{\text{defn. of } S_c(b), U_x, B_x\} \\
&= \alpha(U_x \cup B_x) + \omega(U_x \cup B_x) && \{\text{defn. of } h = \alpha + \omega\} \\
&\leq h(N)/2 && \{c \text{ is a centroid}\} \\
&= (\alpha(N) + \omega(N))/2 && \{\text{defn. of } h\}
\end{aligned}$$

From the second and fourth lines of the above derivation, we may conclude that

$$\begin{aligned}
\alpha(U_x \cup B_x) &\leq (\alpha(N) + \omega(N))/2 - \omega(U_x \cup B_x) && \{\text{derivation above}\} \\
&= ((1+\epsilon)\omega(N) + \omega(N))/2 - \omega(U_x \cup B_x) && \{\text{Equation (3.18)}\} \\
&= \omega(N) - \omega(U_x \cup B_x) + (\epsilon/2)\omega(N) \\
&= \omega(C_x) + (\epsilon/2)\omega(N) && \{N = U_x \cup B_x \cup C_x\}
\end{aligned} \qquad (3.19)$$

Similarly

$$\begin{aligned}
\omega(U_x \cup B_x) &\leq (\alpha(N) + \omega(N))/2 - \alpha(U_x \cup B_x) && \{\text{derivation above}\} \\
&\leq \alpha(N) - \alpha(U_x \cup B_x) && \{\text{Inequality (3.18)}\} \\
&= \alpha(C_x) && \{N = U_x \cup B_x \cup C_x\} \qquad (3.20)
\end{aligned}$$

Now we are ready to compare cap and cap' values. Let x be chosen arbitrarily in $N - \{c\}$. For the links directed away from c:

$$\begin{aligned}
cap'(p(x), x) &= \sum_{u \in U_x} cap^*(c, u) && \{\text{defn. of } cap'\} \\
&= \sum_{u \in U_x} \min\{\alpha(N - \{u\}), \omega(u)\} && \{\text{Lemma 3.1}\} \\
&= \sum_{u \in U_x} \omega(u) && \{\text{Inequality (3.20)}, u \in U_x\} \\
&= \omega(U_x) \\
&= \min\{\alpha(B_x \cup C_x), \omega(U_x)\} && \{\text{Inequality (3.20)}\} \\
&= cap(p(x), x) && \{\text{Lemma 3.1}\} \qquad (3.21)
\end{aligned}$$

That is, all links directed away from c have unchanged capacities cap'. For the links directed towards c:

$$\begin{aligned}
cap'(x, p(x)) &= \sum_{u \in U_x} cap^*(u, c) && \{\text{defn. of } cap'\} \\
&= \sum_{u \in U_x} \min\{\alpha(u), \omega(N - \{u\})\} && \{\text{Lemma 3.1}\} \\
&\leq \sum_{u \in U_x} \alpha(u) \\
&= \alpha(U_x) \\
&= \min\{\alpha(U_x), \omega(B_x \cup C_x)\} \\
&\quad + [\alpha(U_x) - \min\{\alpha(U_x), \omega(B_x \cup C_x)\}] \\
&= cap(x, p(x)) + \max\{0, \alpha(U_x) - \omega(B_x \cup C_x)\} \\
&\qquad \{\text{Lem. 3.1}, -\min\{x, y\} = \max\{-x, -y\}\ \} \\
&\qquad (3.22)
\end{aligned}$$

That is, some links directed toward c can have larger capacities cap'. Define $N' = \{x : x \in N - \{c\}, \alpha(U_x) > \omega(B_x \cup C_x)\}$. Only the links $(x, p(x))$ such that $x \in N'$ can have

$cap'(x,p(x)) > cap(x,p(x))$. Now we find a bound on how much larger such a link's capacity can be. The following lower bound on $cost(\mathcal{N})$ is useful in the last part of the analysis.

$$\begin{aligned}
cost(\mathcal{N}) &\geq \sum_{x\in N'} [cap(x,p(x))\gamma(x,p(x)) + cap(p(x),x)\gamma(p(x),x)] \\
&\qquad \{\text{Eqn. (3.14)}, N' \subseteq N - \{c\}\} \\
&\geq \sum_{x\in N'} \left[cap(x,p(x))\gamma(x,p(x)) + cap(p(x),x)\left(\frac{1}{1+\delta}\right)\gamma(x,p(x))\right] \\
&\qquad \{\text{defn. of } \delta\} \\
&\geq \frac{1}{1+\delta}\sum_{x\in N'} [(cap(x,p(x)) + cap(p(x),x))\gamma(x,p(x))] \\
&\qquad \{\delta \geq 0, \text{ algebra}\} \\
&\geq \frac{1}{1+\delta}\sum_{x\in N'} [(\min\{\alpha(U_x), \omega(B_x \cup C_x)\} + \omega(U_x))\gamma(x,p(x))] \\
&\qquad \{\text{ Lem. 3.1, derivation of Eqn. (3.21) }\} \\
&\geq \frac{1}{1+\delta}\sum_{x\in N'} [\omega(N)\gamma(x,p(x))] \\
&\qquad \{\text{defn. of } N',\ N = U_x \cup B_x \cup C_x\}
\end{aligned} \tag{3.23}$$

We can now use the equations and inequalities above to prove the theorem.

$$\begin{aligned}
\frac{cost(\mathcal{N}_c^*)}{cost(\mathcal{N})} &= 1 + \frac{cost(\mathcal{N}_c^*) - cost(\mathcal{N})}{cost(\mathcal{N})} \\
&\leq 1 + \frac{\sum_{x\in N-\{c\}} [(cap'(x,p(x)) - cap(x,p(x)))\gamma(x,p(x))]}{cost(\mathcal{N})} \\
&\qquad \{cost(\mathcal{N}_c^*) \leq \text{(3.17)}, \text{Eqns. (3.14), (3.21)}\} \\
&\leq 1 + \frac{\sum_{x\in N-\{c\}} [\max\{0, \alpha(U_x) - \omega(B_x \cup C_x)\}\gamma(x,p(x))]}{cost(\mathcal{N})} \quad \{\text{Ineq. (3.22)}\} \\
&= 1 + \frac{\sum_{x\in N'} [(\alpha(U_x) - \omega(B_x \cup C_x))\gamma(x,p(x))]}{cost(\mathcal{N})} \quad \{\text{defn. of } N'\} \\
&\leq 1 + \frac{\sum_{x\in N'} [(\frac{\epsilon}{2}\omega(N))\gamma(x,p(x))]}{\frac{1}{1+\delta}\sum_{x\in N'} [\omega(N)\gamma(x,p(x))]} \quad \{\text{Ineqs. (3.19), (3.23)}\} \\
&\leq 1 + (1+\delta)\frac{\epsilon}{2}
\end{aligned}$$

■

Now we show that Theorem 3.5 is almost as good as possible, because there is a class of instances for which the cheapest star network approaches $1+\epsilon/2$ times the cost of a tree network that is not a star. Thus the theorem is tight for instances with $\delta = 0$, but it may be possible to improve either the theorem or the class of instances when $\delta \neq 0$. If $\delta = 0$, then the link costs γ are symmetric, and this is the case for most realistic instances of the network configuration problem.

Let $\epsilon, \delta \geq 0$ be any real numbers. We wish to find an instance $I = (N, \alpha, \omega, \gamma)$ satisfying the conditions of Theorem 3.5 such that the cheapest star costs as much more than a non-star $\mathcal{N}$ as possible.

Let $\Lambda \geq 0$ be chosen later, and let $\Delta = (1+\Lambda)\epsilon/2$. The switches in the following instance are given as triples $(u, \alpha(u), \omega(u))$.

$$\begin{aligned} N &= U \cup V \\ U &= \{(u_i, (1+\Delta)/n_u, \Lambda/n_u) : 1 \leq i \leq n_u\} \\ V &= \{(v_i, (\Lambda+\Delta)/n_v, 1/n_v) : 1 \leq i \leq n_v\} \\ n_u &\geq 1+\Delta \\ n_v &\geq 1+\Lambda/\Delta \\ \gamma(x,y) &= \begin{cases} 1+\delta & \text{if } x \in U, y \in V \\ 1 & \text{if } x \in V, y \in U \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.24}$$

It is tedious, but not difficult, to check that this instance satisfies the conditions of the theorem. The cost of any star network with center $u \in U$ is $cost(\mathcal{N}^*_u) = (1+\Lambda+\Delta+\delta)$, and the cost of any star network with center $v \in V$ is $cost(\mathcal{N}^*_v) = (1+\Delta)(1+\delta)+\Lambda$. Cheaper than any of these star networks is the tree network with links $(u,v),(v,u)$ for some $u \in U, v \in V$, and all other switches $x \in U$ are attached to u, and all other $y \in V$

are attached to v. This tree has cost $cost(\mathcal{N}) = (1 + \Lambda + \delta)$. Thus

$$\begin{aligned} \frac{cost(\mathcal{N}_v^*)}{cost(\mathcal{N})} &= \frac{1+\Lambda+\Delta+\delta}{1+\Lambda+\delta} \\ &= 1 + \frac{(1+\Lambda)\epsilon/2}{1+\Lambda+\delta} \quad \{\text{defn. of } \Delta\} \\ &= 1 + \frac{1}{1+(\frac{\delta}{1+\Lambda})}\frac{\epsilon}{2} \end{aligned} \tag{3.25}$$

This quantity is in the interval $[1, 1+\epsilon/2)$ for all $\Lambda \geq 0$, but it approaches $1+\epsilon/2$ as Λ goes to infinity.

For the other star networks, we have

$$\begin{aligned} \frac{cost(\mathcal{N}_v^*)}{cost(\mathcal{N})} &= \frac{(1+\Delta)(1+\delta)+\Lambda}{1+\Lambda+\delta} \\ &= 1 + \frac{1+\delta}{1+(\frac{\delta}{1+\Lambda})}\frac{\epsilon}{2} \quad \{\text{defn. of } \Delta, \text{ algebra}\} \end{aligned} \tag{3.26}$$

This quantity is in the interval $[1+\epsilon/2, 1+(1+\delta)\epsilon/2]$ for all $\Lambda \geq 0$, and it approaches $1+(1+\delta)\epsilon/2$ as Λ goes to infinity. Thus these star networks cost more than the others. It is also worthwhile to note that any of the switches $v \in V$ are centroids for the non-star network $\mathcal{N}$. Thus, if the theorem can be improved, it must be done by explicitly taking into account that a switch that is not a centroid might be a cheaper center for a star network than any centroid.

3.8. A probabilistic result

Suppose that instances I with flat traffic limits are randomly generated by the method given in Section 3.5, except that now the locations are randomly chosen in the interior of a unit circle on the plane, i.e., a unit disk. Also, $\alpha(u)$ and $\omega(u)$ values may be chosen randomly with any distribution desired, even with different distributions for different switches, with the restriction that the expected values of all such distributions equal $\overline{\alpha}$, and it is impossible to generate values outside of the interval $[\alpha_{\text{lo}}, \alpha_{\text{hi}}]$.

Let $A(I)$ be the cost of the cheapest star network for instance I, and $LB(I)$ be the value of the lower bound for instance I.

Theorem 3.8 *Let $\delta > 0$ be given. When instances of the nonblocking network design problem are generated as described above, then*

$$\lim_{n\to\infty} \Pr\left\{ \frac{A(I)}{LB(I)} \leq 1+\delta \right\} = 1$$

This theorem can also be proven when switch locations are uniformly distributed in the unit square, or many shapes that have their areas symmetrically arranged around their center points. Using the unit disk just makes some mathematical expressions in the proof simpler.

The proof is done in two parts. Lemma 3.11 states that a randomly generated instance I is "balanced" with probability approaching 1 as n goes to infinity. Lemma 3.14 then shows that $A(I)/LB(I)$ is close to 1 for all balanced instances I.

For the first lemma, we use a theorem of Hoeffding [43] and another due to Angluin and Valiant [3, 41].

Theorem 3.9 *(Hoeffding) Let X_i, $1 \leq i \leq n$, be independent random variables, each having mean μ and $\Pr\{a \leq X_i \leq b\} = 1$. Then for any real t, $0 < t < b - \mu$, we have*

$$\begin{aligned} \Pr\left\{ \sum_{i=1}^{n} X_i \geq n\mu + nt \right\} &\leq e^{-2nt^2/(b-a)^2} \\ \Pr\left\{ \sum_{i=1}^{n} X_i \leq n\mu - nt \right\} &\leq e^{-2nt^2/(b-a)^2} \end{aligned}$$

Theorem 3.10 *(Angluin, Valiant) Let X_i, $1 \leq i \leq n$, be independent random variables, each of which has probability p of being 1 and probability $1-p$ of being 0. Then for any t, $0 \leq t \leq 1$*

$$\begin{aligned} \Pr\left\{ \sum_{i=1}^{n} X_i \geq (1+t)np \right\} &\leq e^{-t^2 np/3} \\ \Pr\left\{ \sum_{i=1}^{n} X_i \leq (1-t)np \right\} &\leq e^{-t^2 np/2} \end{aligned}$$

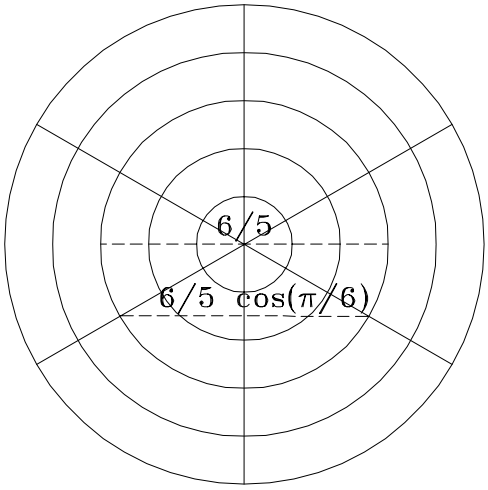

Figure 3.4: The 5-track 6-sector partitioning of the unit disk, $R_{5,3}$

In the proofs below, it is useful to subdivide the unit disk into smaller regions, and then reason about how many switches will be placed into each of the regions. The track and sector subdivision of the unit disk is as follows. Let T, S be positive integers. A *T-track $2S$-sector* division of the unit radius disk is obtained by drawing T concentric circles, where circle i, $1 \leq i \leq T$, has radius i/T, and then drawing S straight lines through the center of these circles, where each successive line makes an angle of π/S radians with the previous line. Figure 3.4 shows a 5-track 6-sector division of the unit disk. Track i, $0 \leq i \leq T-1$, is the region between the circle of radius i/T and the circle of radius $(i+1)/T$.

Let $R_{T,S}$ be the set of regions created by the T-track $2S$-sector partitioning of the unit disk. Define $|r|$ to be the area of the region r, and let $|R|$ be the total area of all regions (π for the unit disk). For an instance I, let N_r be the set of all switches that are located in region r.

For any region r, the probability that any one of the n switches is placed in the region is $\frac{|r|}{|R|}$, because the switches are placed with a uniform distribution on the unit

disk. The expected number of switches in r is $n\frac{|r|}{|R|}$, and the expected total values of $\alpha(N_r)$ and $\omega(N_r)$ are both $n\frac{|r|}{|R|}\overline{\alpha}$.

We define an instance I to be ϵ, d-*balanced*, with respect to a set of regions R, if

$$\begin{aligned}
&(\forall r \in R)\,[(1-\epsilon)\mathrm{E}(\alpha(N_r)) \;\le\; \alpha(N_r) \;\le\; (1+\epsilon)\mathrm{E}(\alpha(N_r))] \;\wedge \\
&(\forall r \in R)\,[(1-\epsilon)\mathrm{E}(\omega(N_r)) \;\le\; \omega(N_r) \;\le\; (1+\epsilon)\mathrm{E}(\omega(N_r))] \;\wedge \\
&\qquad (\exists v \in V)\left[\mathrm{dist}(v,C) \le n^{-d}\right]
\end{aligned} \tag{3.27}$$

where C is the center of the unit disk, and $\mathrm{dist}(v,C)$ denotes the Euclidean distance between the switch v and the center.

Lemma 3.11 *For instances I generated as described earlier, and all $0 < \epsilon < 1$, $0 < d < 1/2$, $T, S \ge 1$*

$$\lim_{n\to\infty} \Pr\{I \text{ is not } \epsilon, d\text{-balanced w.r.t. } R_{T,S}\} = 0$$

Proof:

$$\begin{aligned}
&\Pr\{I \text{ is not } \epsilon, d\text{-balanced w.r.t. } R_{T,S}\} \\
&= \Pr\Big\{(\exists r \in R_{T,S})\,[\alpha(N_r) < (1-\epsilon)\mathrm{E}(\alpha(N_r)) \;\vee\; \alpha(N_r) > (1+\epsilon)\mathrm{E}(\alpha(N_r))] \;\vee \\
&\qquad (\exists r \in R_{T,S})\,[\omega(N_r) < (1-\epsilon)\mathrm{E}(\omega(N_r)) \;\vee\; \omega(N_r) > (1+\epsilon)\mathrm{E}(\omega(N_r))] \;\vee \\
&\qquad (\forall v \in V)\left[\mathrm{dist}(v,C) > n^{-d}\right]\Big\} \\
&\le \sum_{r\in R_{T,S}} \Bigg[\Pr\left\{\alpha(N_r) < (1-\epsilon)n\frac{|r|}{|R|}\overline{\alpha}\right\} + \Pr\left\{\alpha(N_r) > (1+\epsilon)n\frac{|r|}{|R|}\overline{\alpha}\right\} \\
&\qquad\qquad + \Pr\left\{\omega(N_r) < (1-\epsilon)n\frac{|r|}{|R|}\overline{\alpha}\right\} + \Pr\left\{\omega(N_r) > (1+\epsilon)n\frac{|r|}{|R|}\overline{\alpha}\right\}\Bigg] \\
&\qquad + \left[1 - \frac{\pi n^{-2d}}{\pi}\right]^n
\end{aligned} \tag{3.28}$$

The equality holds by the definition of ϵ, d-balanced. The inequality holds because $\Pr\{A \vee B\} \le \Pr\{A\} + \Pr\{B\}$, even if A and B are dependent events, as some pairs of events above happen to be.

Now we find an upper bound for expressions of the form $\Pr\{A\}$, where A is either $\alpha(N_r) \leq (1-\epsilon)np\overline{\alpha}$ or $\alpha(N_r) \geq (1+\epsilon)np\overline{\alpha}$. We temporarily substitute p for $\frac{|r|}{|R|}$ for readability.

$$\begin{aligned}
\Pr\{A\} &= \sum_{k=0}^{n} \Pr\{|N_r| = k \wedge A\} && (3.29)\\
&\leq \sum_{k=0}^{\lfloor (1-\frac{\epsilon}{2})np \rfloor} \Pr\{|N_r| = k\} + \sum_{k=\lceil (1+\frac{\epsilon}{2})np \rceil}^{n} \Pr\{|N_r| = k\} \\
&\quad + \sum_{k=\lfloor (1-\frac{\epsilon}{2})np \rfloor + 1}^{\lceil (1+\frac{\epsilon}{2})np \rceil - 1} \Pr\{|N_r| = k \wedge A\} && (3.30)\\
&\leq \Pr\left\{|N_r| \leq (1-\frac{\epsilon}{2})np\right\} + \Pr\left\{|N_r| \geq (1+\frac{\epsilon}{2})np\right\} \\
&\quad + \epsilon np \max_k \Pr\{|N_r| = k \wedge A\} && (3.31)
\end{aligned}$$

where the maximization in the last line is over the range $\lfloor (1-\frac{\epsilon}{2})np \rfloor + 1 \leq k \leq \lceil (1+\frac{\epsilon}{2})np \rceil - 1$. Step (3.29) holds because we are simply partitioning the event A into $n+1$ separate events. The inequality in step (3.30) holds because $\Pr\{|N_r| = k \wedge A\} \leq \Pr\{|N_r| = k\}$. The first line of (3.31) is just a rewritten form of the first line of (3.30), and the inequality follows from the maximization in the second line. The factor of ϵnp is an upper bound on the number of terms in the summation on the second line of (3.30).

By applying Theorem 3.10 with $t = \epsilon/2$, we see that the first line of (3.31) is at most

$$\begin{aligned}
&\exp(-\epsilon^2 np/8) + \exp(-\epsilon^2 np/12) \\
&\leq 2\exp(-\epsilon^2 np/12) && (3.32)
\end{aligned}$$

where the inequality follows simply because the second term is the larger of the two.

Now we find an upper bound for the maximization term of (3.31), where A is $\alpha(N_r) \leq (1-\epsilon)np\overline{\alpha}$. If $|N_r| = k$, then let $N_r = \{v_1, \ldots v_k\}$

$$\begin{aligned}
&\Pr\{|N_r| = k \wedge \alpha(N_r) \leq (1-\epsilon)np\overline{\alpha}\} \\
&= \Pr\left\{\sum_{i=1}^{k} \alpha(v_i) \leq k\overline{\alpha} - k\overline{\alpha}\left(1-(1-\epsilon)\frac{np}{k}\right)\right\} && (3.33)\\
&\leq \exp\left(-2k\left(1-(1-\epsilon)\frac{np}{k}\right)^2 \overline{\alpha}^2/(\alpha_{\mathrm{hi}}-\alpha_{\mathrm{lo}})^2\right) && (3.34)\\
&\leq \exp\left(-2(1-\epsilon/2)np\left(1-\frac{1-\epsilon}{1-\epsilon/2}\right)^2 \overline{\alpha}^2/(\alpha_{\mathrm{hi}}-\alpha_{\mathrm{lo}})^2\right) && (3.35)\\
&= \exp\left(-np\left(\frac{\epsilon^2}{2-\epsilon}\right)\overline{\alpha}^2/(\alpha_{\mathrm{hi}}-\alpha_{\mathrm{lo}})^2\right) && (3.36)\\
&= \exp\left(-npf(\epsilon)\right) && (3.37)
\end{aligned}$$

Step (3.33) follows from rewriting and algebra. In step (3.34), we make use of Theorem 3.9 with $t = \overline{\alpha}(1-(1-\epsilon)np/k)$, which is larger than 0 because $k \geq (1-\epsilon/2)np$. Step (3.35) follows using the same lower bound on k, and step (3.36) follows from algebra. Step (3.37) is just a shorter form, where the function f has the obvious definition.

Using a similar derivation we may conclude

$$\begin{aligned}
&\Pr\{|N_r| = k \wedge \alpha(N_r) \geq (1+\epsilon)np\overline{\alpha}\} \\
&\leq \exp\left(-np\frac{\epsilon^2(2-\epsilon)}{(2+\epsilon)^2}\overline{\alpha}^2/(\alpha_{\mathrm{hi}}-\alpha_{\mathrm{lo}})^2\right) \\
&= \exp\left(-npg(\epsilon)\right) && (3.38)
\end{aligned}$$

We must use the upper bound on k, $k \leq (1+\epsilon/2)np$, for this derivation. Again, step (3.38) is for easier reading, and g is defined in the obvious way.

Now we may derive an upper bound on (3.28) by using (3.31), (3.32), (3.37), and (3.38).

$$\sum_{r \in R_{T,S}} \left[8\exp\left(-\epsilon^2 n\frac{|r|}{12|R|}\right) + 2\epsilon n\frac{|r|}{|R|}\left(\exp\left(-nf(\epsilon)\frac{|r|}{|R|}\right) + \exp\left(-ng(\epsilon)\frac{|r|}{|R|}\right)\right)\right]$$

$$
\begin{aligned}
& + \left[1 - n^{-2d}\right]^n \\
\leq \; & 2ST \left[8 \exp\left(-\epsilon^2 n/24ST^2\right)\right. \\
& \left. +2\epsilon n \frac{2T-1}{2ST^2} \left(\exp\left(-nf(\epsilon)/2ST^2\right) + \exp\left(-ng(\epsilon)/2ST^2\right)\right)\right] \\
& + \exp\left(-n^{1-2d}\right) \qquad (3.39) \\
= \; & c_1 \exp(-c_2 n) + c_3 n \exp(-c_4 n) + c_3 n \exp(-c_5 n) + \exp\left(-n^{1-2d}\right) \qquad (3.40)
\end{aligned}
$$

Step (3.39) follows because $\frac{1}{2ST^2} \leq \frac{|r|}{|R|} \leq \frac{2T-1}{2ST^2}$, which is true by the definition of the track and sector partition of the unit disk. The change in the last term is justified by the inequality $1 + x \leq e^x$ for all real x. Step (3.40) is just a rewritten form of (3.39), where all c_i are positive real constants for any permissible values of ϵ, S, T, and any $\overline{\alpha} > 0$.

Since $0 < d < 1/2$, we can now easily see that this function's limit is 0 as n goes to infinity. ■

Lemma 3.12 *Let I be an instance that is ϵ, d-balanced with respect to the partition $R_{T,S}$. Then the lower bound $LB(I)$ for this instance satisfies*

$$LB(I) \geq 2(1-\epsilon) n \overline{\alpha} \cos\left(\frac{\pi}{2S}\right) \frac{2(T+1/4)(T-1)}{3T^2} \qquad (3.41)$$

Proof: The lower bound is equal to the cost of a maximum cost flow in the lower bound graph. We prove the lemma by constructing a flow, not necessarily of maximum cost, that costs at least as much as the right hand side of Inequality (3.41). From this it follows that $LB(I)$ also satisfies the inequality.

In the track and sector partitioning $R_{T,S}$, let r and r' be two regions that are in the same track i, but in "opposite" sectors (i.e., going around track i from r to r' in either direction, we encounter $S-1$ other sectors before reaching r').

In the lower bound graph $\mathcal{L}$ for this instance, there are vertices u_s for each $u \in N_r$ and vertices u'_d for each $u' \in N_{r'}$. Because the traffic is α,ω-bounded, we can send $\min\{\alpha(N_r), \omega(N_{r'})\}$ flow from all of the source vertices to all of the destination vertices. Since I is ϵ, d-balanced, this quantity is at least

$$\begin{aligned}(1-\epsilon)\mathrm{E}(\alpha(N_r)) &= (1-\epsilon)n\frac{|r|}{|R|}\overline{\alpha} && (3.42)\\ &= (1-\epsilon)n\frac{2i+1}{2ST^2}\overline{\alpha} && (3.43)\end{aligned}$$

Each of the arcs in the lower bound graph has a cost equal to the distance between the switches. The distance between any switch in r and any switch in r' is at least $(2i/T)\cos(\pi/2S)$. See Figure 3.4 for a visual example of why the distance can be smaller than $2i/T$.

Therefore, we can make a flow from the vertices u_s to the vertices u'_d that costs at least

$$\left((1-\epsilon)n\frac{2i+1}{2ST^2}\overline{\alpha}\right)\left(\frac{2i}{T}\cos\left(\frac{\pi}{2S}\right)\right) \qquad (3.44)$$

A similar flow can be set up between switches in all pairs of opposite sectors. The total cost of such a flow is obtained by summing Equation (3.44) over all sectors.

$$\begin{aligned}&\sum_{i=0}^{T-1} 2S(1-\epsilon)n\frac{2i+1}{2ST^2}\overline{\alpha}\frac{2i}{T}\cos\left(\frac{\pi}{2S}\right)\\ &= 2(1-\epsilon)n\overline{\alpha}\cos\left(\frac{\pi}{2S}\right)\frac{1}{T^3}\sum_{i=0}^{T-1} i(2i+1)\\ &= 2(1-\epsilon)n\overline{\alpha}\cos\left(\frac{\pi}{2S}\right)\frac{2(T+1/4)(T-1)}{3T^2}\end{aligned}$$

where the equalities follow from algebraic manipulation and the identities $\sum_{i=0}^{n} i = n(n-1)/2$ and $\sum_{i=0}^{n} i^2 = n(n-1/2)(n-1)/3$. ■

Lemma 3.13 *Let I be an instance that is ϵ, d-balanced with respect to the partition $R_{T,S}$. Then the cost $A(I)$ of the minimum cost star network satisfies*

$$A(I) \leq 2(1+\epsilon)n\overline{\alpha}\left[\frac{2(T-1/4)(T+1)}{3T^2} + n^{-d}\right] \tag{3.45}$$

Proof: The proof is done by constructing a star network, not necessarily of minimum cost, that costs no more than the expression on the right hand side of Inequality (3.45). From this it follows that $A(I)$ satisfies the inequality.

Since I is ϵ, d-balanced, there is a switch C that is no further than n^{-d} from the center of the disk.

Consider a sector r in track i of $R_{T,S}$. For each switch $u \in N_r$, we can build a link (u, C) with capacity $\alpha(u)$ and a link (C, u) with capacity $\omega(u)$, and the resulting tree network is nonblocking. The distance from C to u is at most $(i+1)/T + n^{-d}$. Therefore the contribution to the total network cost from switches in r is at most

$$\begin{aligned}
&(\alpha(N_r) + \omega(N_r))\left(\frac{i+1}{T} + n^{-d}\right) \\
&\quad\leq\quad 2(1+\epsilon)\mathrm{E}(\alpha(N_r))\left(\frac{i+1}{T} + n^{-d}\right) \\
&\quad=\quad 2(1+\epsilon)n\frac{2i+1}{2ST^2}\overline{\alpha}\left(\frac{i+1}{T} + n^{-d}\right) \qquad (3.46)
\end{aligned}$$

because I is ϵ, d-balanced. Summing Equation (3.46) over all sectors we get

$$\begin{aligned}
&\sum_{i=0}^{T-1} 2S \cdot 2(1+\epsilon)n\frac{2i+1}{2ST^2}\overline{\alpha}\left(\frac{i+1}{T} + n^{-d}\right) \\
&\quad=\quad 2(1+\epsilon)n\overline{\alpha}\frac{1}{T^3}\sum_{i=0}^{T-1}(2i+1)(i+1+Tn^{-d}) \\
&\quad=\quad 2(1+\epsilon)n\overline{\alpha}\left[\frac{2(T-1/4)(T+1)}{3T^2} + n^{-d}\right]
\end{aligned}$$

where the equalities follow from algebraic manipulation and the same identities as used in the proof of Lemma 3.12. ■

Lemma 3.14 *Let I be an instance that is ϵ, d-balanced with respect to the partition $R_{T,S}$. Then for any $\delta > 0$,*

$$\frac{A(I)}{LB(I)} \leq 1 + \delta \tag{3.47}$$

if $\epsilon \leq \frac{x-1}{x+1}$, $S \geq \frac{\pi}{2\cos^{-1}(1/x)}$, $T \geq \max\{2, \frac{3(x+3)}{4(x-1)} + \frac{x-1}{2}\}$, and $n \geq \left[\frac{6}{x-1}\right]^{1/d}$, where $x = (1+\delta)^{1/3}$.

Proof: Since I is ϵ, d-balanced, then by Lemmas 3.12 and 3.13 and algebra we have

$$\frac{A(I)}{LB(I)} \leq \left(\frac{1+\epsilon}{1-\epsilon}\right) \frac{1}{\cos(\pi/2S)} \left[\frac{(T-1/4)(T+1)}{(T+1/4)(T-1)} + \frac{3T^2}{2(T+1/4)(T-1)} n^{-d}\right] \tag{3.48}$$

It is easy to verify that $\epsilon \leq \frac{x-1}{x+1}$ implies $\frac{1+\epsilon}{1-\epsilon} \leq x$. Similarly, $S \geq \frac{\pi}{2\cos^{-1}(1/x)}$ implies $\frac{1}{\cos(\pi/2S)} \leq x$.

$T \geq \frac{3(x+3)}{4(x-1)} + \frac{x-1}{2}$ implies $\frac{(T-1/4)(T+1)}{(T+1/4)(T-1)} \leq (x+1)/2$, and $T \geq 2$ and $n \geq \left[\frac{6}{x-1}\right]^{1/d}$ imply $\frac{3T^2}{2(T+1/4)(T-1)} n^{-d} \leq (x-1)/2$.

All together, these conditions imply that the right hand side of Equation (3.48) is at most

$$\begin{aligned} x \cdot x \cdot [(x+1)/2 + (x-1)/2] &= x^3 \\ &= 1 + \delta \end{aligned}$$

■

4. GENERAL FLAT TRAFFIC

4.1. Definition

An additional way to restrict the offered traffic is *point-to-point limits*. They are specified by giving a value $\mu(u,v)$ for each switch pair $u,v \in N, u \neq v$. The value $\mu(u,v)$ is the maximum total rate of all connections that may exist from u to v simultaneously.

Given a set of switches N and a collection of connection requests $\mathcal{R}$, define the *point-to-point usage* from switch u to switch v under requests $\mathcal{R}$ as

$$\mu\text{-}usage\,(u,v,\mathcal{R}) \;=\; \sum_{(u,v,\rho)\in\mathcal{R}} \rho \tag{4.1}$$

Let flat traffic limits $\mathcal{T} = (\alpha,\omega,\mu)$ be given. We say that the set of requests $\mathcal{R}$ is compatible with traffic limits $\mathcal{T}$ if

$$\begin{aligned} (\forall u \in N)\;\; & \Big(\alpha\text{-}usage\,(u,\mathcal{R}) \le \alpha(u) \;\wedge \\ & \qquad \omega\text{-}usage\,(u,\mathcal{R}) \le \omega(u)\Big) \wedge \\ (\forall u,v \in N)\;\; & (\mu\text{-}usage\,(u,v,\mathcal{R}) \le \mu(u,v)) \end{aligned} \tag{4.2}$$

This is similar to condition (3.3), but now we also require that no pair of switches is involved in more requests than their point-to-point limit allows.

An example of flat traffic limits is given in Table 4.1. The α,ω values are repeated from the previous chapter.

In the previous chapter, the set of requests $\mathcal{R} = \{(C,D,35),(A,D,90),(C,A,100),(C,A,125)\}$ was compatible with the traffic limits $\mathcal{T} = (\alpha,\omega)$. However, it is not compatible with the traffic limits $\mathcal{T} = (\alpha,\omega,\mu)$ in Table 4.1, because $\mu\text{-}usage\,(C,A,\mathcal{R}) =$

Table 4.1: An example of flat traffic limits

		v				
	$\mu(u,v)$	A	B	C	D	$\alpha(u)$
u	A	–	0	∞	∞	450
	B	0	–	0	0	0
	C	200	0	–	∞	300
	D	300	0	250	–	450
	$\omega(v)$	450	0	300	150	

$225 > \mu(C,A)$. The set of requests $\mathcal{R} = \{(C,D,35),(A,D,90),(C,A,100)\}$ is compatible with $\mathcal{T}$, however.

Note that $\mu\text{-}usage\,(u,v,\mathcal{R})$ is simply the value $x_{u,v}$ defined in the previous section. Thus the definition of a compatible request set (4.2) can be written by adding constraints to the system (3.6), giving the modified system:

$$\begin{aligned}
\sum_{v \in N-\{u\}} x_{u,v} &\leq \alpha(u) \qquad \forall u \in N \\
\sum_{u \in N-\{v\}} x_{u,v} &\leq \omega(v) \qquad \forall v \in N \\
x_{u,v} &\leq \mu(u,v) \quad \forall u,v \in N, u \neq v \\
x_{u,v} &\geq 0 \qquad\quad \forall u,v \in N, u \neq v
\end{aligned} \tag{4.3}$$

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

$$(\forall u,v \in N) \quad (\mu(u,v) \geq \min\{\alpha(u),\omega(v)\}) \tag{4.4}$$

It can be proved that when this condition holds, the first two lines of condition (4.2) imply the third line. Therefore the compatible sets of requests depend only upon the values of α and ω. When condition (4.4) holds, we call the traffic limits *termination limit bounded*, or α,ω*-bounded*. Chapter 3 dealt solely with α,ω-bounded traffic limits.

The second restriction is

$$(\forall u \in N) \quad \left(\sum_{v \in N} \mu(u,v) \leq \alpha(u) \quad \wedge \quad \sum_{v \in N} \mu(v,u) \leq \omega(u) \right) \tag{4.5}$$

When this condition holds, the third line of condition (4.2) implies the first and second lines. Therefore the compatible sets of requests depend only upon the values of μ. When condition (4.5) holds, we call the traffic limits *point-to-point bounded*, or *μ-bounded*.

Note that when the traffic limits are μ-bounded, there is one solution x^* to the system (4.3), $x^*_{u,v} = \mu(u,v)$ for all $u, v \in N$, that is the "largest". That is, all other solutions x' satisfy $x'_{u,v} \leq x^*_{u,v}$ for all $u, v \in N$. When traffic limits are not μ-bounded, there is no single solution x^* with this property.

Configuring a nonblocking network with μ-bounded traffic limits is very similar to other work done on designing networks that can handle traffic specified by a traffic matrix [34, 44, 45, 50, 53, 55]. In fact, the optimal solution to our network configuration problem with arbitrary γ values (not necessarily satisfying the triangle inequality) is a network in which the fixed path used from u to v is the shortest path from u to v in the complete directed graph with edge lengths given by γ [44, p. 198]. When γ values satisfy the triangle inequality, the direct link is always a shortest path, so the optimal solution is a complete network.

If the solution is so simple, why is there so much other research on the problem? The other work either allows additional nodes to be added, leading to a problem similar to the Steiner tree problem [34], restricts solutions to tree networks [45], or considers more complex link cost functions, ones that are not linear with the capacity of the links [50, 53, 55].

When the traffic is neither α,ω-bounded nor μ-bounded, we call it the *general* case of nonblocking traffic limits. The traffic limits in Table 4.1 are neither α,ω-bounded nor μ-bounded.

4.2. Lower bound

We may find a lower bound on the cost of any nonblocking network in nearly the same way as in Section 3.4. We still want to maximize the objective function (3.7), but now we want the solutions constrained by the new system of inequalities (4.3).

Again, this may be done by using general algorithms to solve linear programs, and again this class of linear programs is equivalent to a restricted class of maximum cost flow problems.

The new lower bound graph $\mathcal{L} = (V, E, cap, cost)$ is nearly the same as before. The only difference from (3.9) is that now the edges of the form (u_s, v_d) have capacities given by the μ values.

$$\begin{aligned} V &= \{s, t\} \cup \{u_s, u_d : u \in N\} \\ E &= \{(s, u_s, \alpha(u), 0),\ (u_d, t, \omega(u), 0) : u \in N\} \cup \\ &\quad \{(u_s, v_d, \mu(u, v), \gamma(u, v)) : u, v \in N, u \neq v\} \end{aligned} \tag{4.6}$$

The edge costs are the same as in Section 3.4.

The maximum cost flow in the lower bound graph for the general traffic limits of Table 4.1 and the γ values of Table 3.2 given previously, has a lower cost than the lower bound graph for the α, ω-bounded traffic limits in Chapter 3. This is because the point-to-point limits do not allow a traffic pattern that is as expensive. A maximum cost flow is the flow with $f(A_s, C_d) = 300$, $f(A_s, D_d) = 150$, $f(c_s, A_d) = 150$, and $f(D_s, A_d) = 300$, with a cost of $437400/150 = \$2916$, compared to \$2944 without the point-to-point limits. Thus the star network with center B with cost \$3864 is at most 32.5% more than optimal.

4.3. Link dimensioning

Fixed path routing has a property that makes it much easier to analyze than most other routing algorithms. No matter what sequence of requests to add and drop connections came before, *the state of the network is a function of the current active set of requests.*

Assume for the moment that all links that are used in some fixed path have a very large capacity that could not be exceeded even if every connection used it, e.g., $\min\{\alpha(N), \omega(N)\}$. Let h_π be the function that maps request sets to states (this function depends only upon the table of fixed paths π). Given any set of requests $\mathcal{R}$, we can determine the state of the network $h_\pi(\mathcal{R})$, and therefore the usage of each link l, $\lambda(l, h_\pi(\mathcal{R}))$. Let $ALL\mathcal{R}_\mathcal{T} = \{\mathcal{R} : \mathcal{R} \text{ is compatible with } \mathcal{T}\}$. We can determine the maximum possible usage of any given link l:

$$\lambda^*(l) = \max_{\mathcal{R} \in ALL\mathcal{R}_\mathcal{T}} \lambda(l, h_\pi(\mathcal{R})) \tag{4.7}$$

Note that this value is independent of any other link capacity. It depends only upon π, $\mathcal{T}$, and l.

If we dimension the network links with capacities cap that satisfy $cap(l) \geq \lambda^*(l)$ for all links l, then the network is nonblocking. If any link l has capacity less than $\lambda^*(l)$, the network blocks when the requests in a set $\mathcal{R}^*$ are made in any order, where $\lambda(l, h_\pi(\mathcal{R}^*)) = \lambda^*(l)$. Therefore, when the link cost functions are any nondecreasing functions of capacity, the capacities $cap(l) = \lambda^*(l)$, for all $l = (u, v)$, $u, v \in N$, produce the cheapest solution to the link dimensioning problem. This is true for any fixed paths and traffic limits, even traffic limits specified by an arbitrary system of linear inequalities. Therefore the link dimensioning problem has been reduced to computing the function λ^*.

Let P_l be the set of ordered switch pairs that must use link l to communicate, i.e., $P_l = \{(u,v) : l \in \pi(u,v)\}$. Let $\mathcal{R}$ be any compatible request set, and let x be the corresponding feasible solution to the system (4.3). Then $\lambda(l, h_\pi(\mathcal{R}))$ is just the sum of rates of those connections that use link l,

$$\sum_{(u,v)\in P_l} x_{u,v} \tag{4.8}$$

Therefore, we can find $\lambda^*(l)$ by maximizing (4.8) subject to the system of constraints (4.3). This class of linear programs is equivalent to a restricted class of maximum flow problems.

An instance of the maximum flow problem is similar to an instance of the maximum cost flow problem, except that there are no edge costs, only capacities. The objective is to find a flow with maximum value, where the value of a flow is the sum of flows on edges out of the source vertex s.

The linear program for the set of switch pairs P_l is equivalent to the maximum flow problem on the directed graph $\mathcal{L}_l = (V, E, cap)$ that is the same as $\mathcal{L}$, except there are no edge costs, and all edges of the form (u_s, v_d) such that $(u,v) \notin P_l$ have capacity 0. Alternately, such edges may be removed from $\mathcal{L}_l$.

Many efficient algorithms have been designed for the maximum flow problem. King, Rao, and Tarjan [51] have designed an algorithm with a worst-case running time of $O(mn + n^{2+\epsilon})$ for any $\epsilon > 0$, where m is the number of edges in the network, and n is the number of vertices. This algorithm has the best asymptotic efficiency known to the author. Goldberg and Tarjan [37, Section 4] describe an algorithm, which is quite easy to implement, with a worst-case running time of $O(n^3)$.

In a star network with center switch C, we may find the maximum usage of links $(u,C), (C,u)$ by finding a maximum flow in an appropriate graph $\mathcal{L}_l$. Note that for

link (u, C), we have $P_l = \{(u, v) : v \in N - \{u\}\}$. These are the only edges of the form (u_s, v_d) in $\mathcal{L}_l$ that might have non-zero capacity. The only edges that can have non-zero flow in any feasible flow are (s, u), and all edges of the form (u_s, v_d) and (v_d, t), where $v \in N - \{u\}$. In this kind of maximum flow instance, a maximum flow can be found in $O(n)$ time, and many maximum flow algorithms for general instances finish in $O(n)$ time when given such a restricted instance. Similarly, the maximum usage of link (C, u) may be found in $O(n)$ time.

Note that the maximum usage of links $(u, C), (C, u)$ is the same no matter which switch $C \neq u$ is the center. Therefore, to find the cheapest star network, we compute the capacities for links $(u, C), (C, u)$ once at the beginning, and then find the cost of each star network in $O(n)$ time per network. Here is an algorithm that finds the cheapest star network:

For each $u \in N$ do

Compute $out(u)$:= capacity of link (u, C) in any star with center $C \neq u$

Compute $in(u)$:= capacity of link (C, u) in any star

End for

For each $C \in N$ do

Comment: Compute the cost of the star network with center switch C

$cost(C) := \sum_{u \in N - \{C\}} (out(u)\gamma(u, C) + in(u)\gamma(C, u))$

End for

Find the minimum $cost(C)$

Each loop takes $O(n)$ time per iteration, so the total running time of this algorithm is $O(n^2)$.

The cheapest star network for the traffic limits of Table 4.1 and the link costs of Table 3.2 is still the one with center switch B, and all of the link capacities are the same

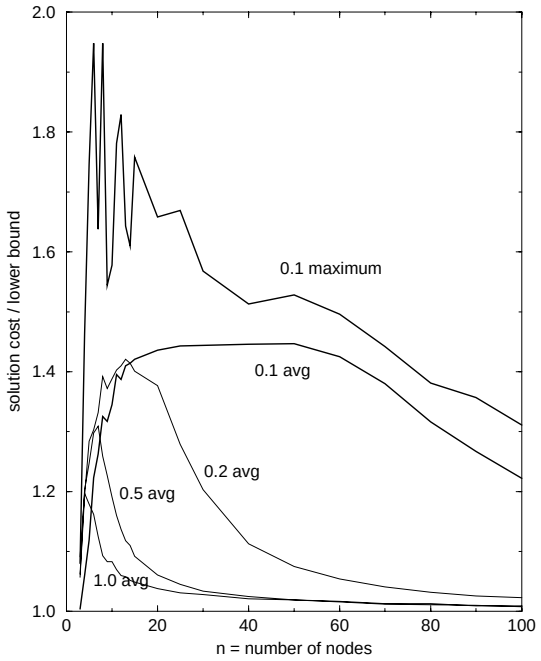

Figure 4.1: Experimental results for flat general traffic

as before. The μ values must be quite small relative to α, ω before they can reduce the link capacities of a star network.

4.4. Experimental results

A random instance is generated in much the same way as in Section 3.5. However, now we also specify a range of real numbers $[\mu_{lo}, \mu_{hi}]$, which is a sub-interval of $[0, 1]$. Place the switches randomly and generate α, ω values as before. Then for each switch pair u, v, choose a real value x randomly in the interval $[\mu_{lo}, \mu_{hi}]$ and set $\mu(u, v) = x \cdot \min\{\alpha(u), \omega(v)\}$. Note that if $\mu_{lo} = \mu_{hi} = 1$, then the traffic limits $\mathcal{T} = (\alpha, \omega, \mu)$ are α, ω-bounded.

Figure 4.1 shows several curves. All instances generated used the parameters $\alpha_{\rm lo} = 10$, $\alpha_{\rm hi} = 20$ as before, but now we set $\mu_{lo} = 0$ and each curve represents a different value of μ_{hi}, as labeled. These are general traffic limits, i.e., neither α, ω-bounded nor μ-bounded, although they may happen to be μ-bounded for small n and small values

of μ_{hi}. Fifty random instances were generated for each data point, and the same set of values of n was used as before.

The performance ratios are noticeably higher on average now. For smaller values of μ_{hi}, the performance ratios stay high for a longer time. This can be explained by examining the kinds of maximum cost flows that we get in the lower bound graph, and the capacities of star network links.

Suppose that $\mu(u,v)$ is so large that it does not restrict the traffic, i.e., $\mu(u,v) \geq \min\{\alpha(u), \omega(v)\} = U$. Then up to U units of flow are free to go from u_s to v_d in the lower bound graph. If traffic is α,ω-bounded, then $\mu(u,v) \geq \min\{\alpha(u), \omega(v)\}$ for all $u, v \in N$, and flow is free to travel between pairs of switches that are furthest apart, making the lower bound value large. When $\mu_{hi} = 0.1$, however, then the expected value of $\mu(u,v)$ is $U(\mu_{lo} + \mu_{hi})/2 = U/20$, and now flow from u must be split among 20 other switches, on average. When there are few switches, not many of them will be far away, and so flow must be sent to close switches, yielding a smaller lower bound. As n grows, it is more likely that there will be 20 switches that are far away, increasing the lower bound to a value close to that for unrestrictive μ values.

A similar examination of the capacities required on the star network links show that one would not expect them to be much smaller than when traffic is α,ω-bounded. Therefore, the minimum star network costs do not decrease as much as the lower bound does, and the performance ratio increases.

However, all curves eventually start to approach 1. This gives empirical evidence that we should be able to extend the probabilistic result of Section 3.8 to more general kinds of flat traffic limits.

When μ values are small, the traffic limits are more likely to be "close" to being μ-bounded. The closer the traffic is to satisfying this condition, then the more likely

it is that a complete network, with direct links between every switch pair, is cheaper than any tree solution. When the traffic limits are μ-bounded and γ values satisfy the triangle inequality, the complete network is the optimal solution [44, p. 198].

All of the curves except the top one are average performance ratios, averaged over 50 instances. The top curve is the maximum performance ratio among all 50 instances for $\mu_{hi} = 0.1$. Curves showing the maximum performance ratio for the other values of μ_{hi} would only clutter the plot. They are typically about twice as high above 1 as the corresponding average curve. The worst performance ratio over all was 2.051 for a four switch instance with $\mu_{hi} = 0.2$.

5. HIERARCHICAL TRAFFIC

5.1. Definition

The traffic limits defined in Chapters 3 and 4 give a good way of specifying the traffic for a "flat" network, with no additional structure. This method can be extended to specify traffic in a hierarchical way, where there are "clusters" of switches that may have high traffic among themselves, but less traffic between switches in the cluster and switches outside of the cluster. These clusters may come about because the network owners charge the users more to establish connections outside of their cluster, because there are groups of users who communicate more frequently among themselves than with others on the network, or some combination of these causes.

As an example, Figure 5.1 shows the hierarchical structure of an instance with 11 switches, A through K, which are organized into 3 clusters, 1, 2, and 3. We denote the set of clusters by $\mathcal{C}$.

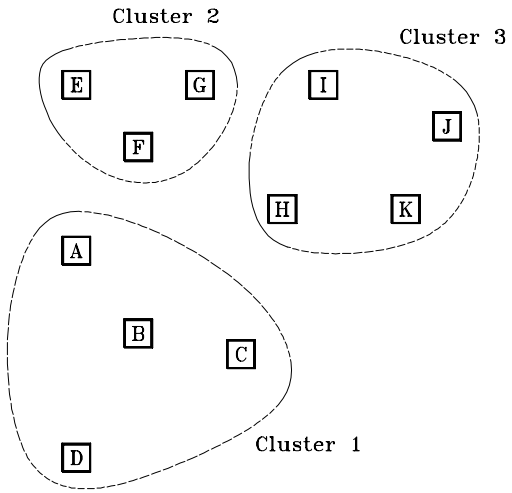

Figure 5.1: An example hierarchical problem instance

Table 5.1: Traffic limits for "root" cluster

		v			
	$\mu(u,v)$	1	2	3	$\alpha(u)$
u	1	–	∞	∞	600
	2	∞	–	∞	450
	3	∞	∞	–	300
	$\omega(v)$	450	450	300	

If the traffic were flat, then there could simultaneously exist connections with total rate up to $\alpha(\{A,B,C,D\}) = 450 + 0 + 300 + 450 = 1200$ Mb/s that have their source as one of A, B, C, D and their destination as one of $E \ldots K$. However, suppose we know that there will never be such connections with total rate more than 600 Mb/s at any given moment. This can be specified by saying that the source termination limit of cluster 1, $\alpha(1)$, is 600 Mb/s. Similarly, let the maximum total rate of connections with source outside of 1 and destination in 1 be 450 Mb/s, and call this the destination termination limit of cluster 1, $\omega(1)$. Similar values for clusters 2 and 3 are given in Table 5.1.

The example given is a two-level hierarchy. One can think of flat traffic as a one-level hierarchy, where there is only one cluster containing all of the switches. This method of specifying traffic can be used for more general hierarchical structures. For example, we could have a three level hierarchy in which clusters of switches were further grouped into "super clusters". In general, each cluster or "super cluster" has its own α and ω values, and arbitrary pairs of switches or clusters can have point-to-point limits between them. We denote hierarchical traffic limits $\mathcal{T}$ by the tuple $(\mathcal{H}, \alpha, \omega, \mu)$, where $\mathcal{H}$ represents the hierarchical structuring of the switches and clusters.

$\mathcal{H}$ may be represented as a rooted tree in which the switches in N are leaves, the clusters in $\mathcal{C}$ are internal vertices, and there is a root vertex R, which is just a place

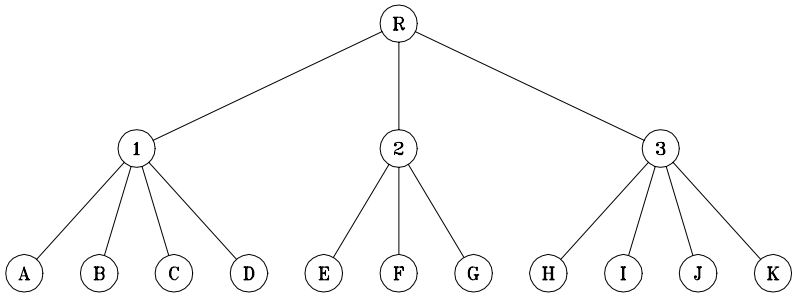

Figure 5.2: The hierarchy tree $\mathcal{H}$ for the instance in Figure 5.1

holder. Figure 5.2 shows the hierarchical structure $\mathcal{H}$ of the instance shown in Figure 5.1. We say that a switch/cluster u is in the cluster c if u is a descendant of c in the hierarchy tree $\mathcal{H}$.

Let a set of switches N with a hierarchical structure $\mathcal{H}$ and a collection of connection requests $\mathcal{R}$ be given. We can extend the definitions of the source usage, destination usage, and point-to-point usage given in Sections 3.1 and 4.1, to include clusters as well as switches. In the following definitions, let $s(c)$ be the set of all switches in N that are contained in the cluster c, or $s(c) = \{c\}$ if $c \in N$.

$$\alpha\text{-usage}\,(c, \mathcal{R}) = \sum_{\substack{(u,v,\rho)\in\mathcal{R},\\ u\in s(c),\, v\notin s(c)}} \rho \tag{5.1}$$

$$\omega\text{-usage}\,(c, \mathcal{R}) = \sum_{\substack{(u,v,\rho)\in\mathcal{R},\\ u\notin s(c),\, v\in s(c)}} \rho \tag{5.2}$$

$$\mu\text{-usage}\,(c_1, c_2, \mathcal{R}) = \sum_{\substack{(u,v,\rho)\in\mathcal{R},\\ u\in s(c_1),\, v\in s(c_2)}} \rho \tag{5.3}$$

We say that the set of requests $\mathcal{R}$ is compatible with the hierarchical traffic limits $\mathcal{T} = (\mathcal{H}, \alpha, \omega, \mu)$ if

$$\begin{gathered}(\forall u \in N \cup \mathcal{C})\ \ (\alpha\text{-usage}\,(u, \mathcal{R}) \le \alpha(u)\ \wedge\ \omega\text{-usage}\,(u, \mathcal{R}) \le \omega(u)) \wedge \\ (\forall u, v \in N \cup \mathcal{C})\ \ (\mu\text{-usage}\,(u, v, \mathcal{R}) \le \mu(u, v))\end{gathered} \tag{5.4}$$

Table 5.2: Traffic limits for cluster 2

		v			
	$\mu(u,v)$	E	F	G	$\alpha(u)$
u	E	–	∞	∞	450
	F	∞	–	∞	300
	G	∞	∞	–	600
	$\omega(v)$	450	300	600	

That is, no switch or cluster is involved in more requests than its termination limits allow, and no pair of switches/clusters is involved in more requests than their point-to-point limit allows.

As before, it is useful to write these conditions as linear inequalities on the x variables. The system of inequalities is:

$$\begin{aligned}
\sum_{u \in s(c), v \notin s(c)} x_{u,v} &\leq \alpha(c) && \forall c \in N \cup \mathcal{C} \\
\sum_{u \notin s(c), v \in s(c)} x_{u,v} &\leq \omega(c) && \forall c \in N \cup \mathcal{C} \\
\sum_{u \in s(c_1), v \in s(c_2)} x_{u,v} &\leq \mu(c_1, c_2) && \forall c_1, c_2 \in N \cup \mathcal{C} \\
x_{u,v} &\geq 0 && \forall u, v \in N, u \neq v
\end{aligned} \tag{5.5}$$

Tables 5.2 and 5.3 give traffic limits for the switches in clusters 2 and 3. All traffic limits for cluster 1, switches A through D, are the same as in Table 4.1 of the previous chapter. Point-to-point limits are only given between switches or clusters that are siblings in the hierarchy tree $\mathcal{H}$. See Section 5.2 for an explanation. The link cost coefficients γ are given by the entries of Table 5.4. The entries are symmetrical, and equal to the Euclidean distance between the switches (rounded up), where the position of each switch is given in the table.

Table 5.3: Traffic limits for cluster 3

		v				
	$\mu(u,v)$	H	I	J	K	$\alpha(u)$
u	H	–	∞	∞	∞	450
	I	∞	–	∞	∞	300
	J	∞	∞	–	∞	300
	K	∞	∞	∞	–	0
	$\omega(v)$	450	300	300	0	

Table 5.4: Link costs $\gamma(u,v)$ for hierarchical example

	A	B	C	D	E	F	G	H	I	J	K	Position
A	0	250	472	500	400	292	500	510	722	949	825	(100,500)
B		0	255	336	619	450	619	461	750	902	764	(250,300)
C			0	472	764	560	658	365	681	744	603	(500,250)
D				0	900	765	949	782	1082	1205	1064	(100,0)
E					0	213	300	584	600	906	825	(100,900)
F						0	213	381	475	752	652	(250,750)
G							0	361	300	609	539	(400,900)
H								0	317	448	317	(600,600)
I									0	317	283	(700,900)
J										0	142	(1000,800)
K											0	(900,600)

5.2. Lower bound

Once again, a lower bound can be found by maximizing the objective function (3.7) subject to our set of constraints (5.5). Unlike before, however, we know of no equivalent maximum cost flow problems for hierarchies with at least two levels, and believe that none exist for "most" such instances. However, we have found a class of maximum cost flow instances that have optimum solution values that are no larger than those of the linear programs. Unless improved algorithms are devised for solving general linear programs, or at least our restricted class of linear programs, solving a maximum cost flow problem will remain a faster way to compute a lower bound, although of lower quality.

Now we generalize the definition of the lower bound graph $\mathcal{L} = (V, E, cap, cost)$ for hierarchical traffic limits. This generalization works when point-to-point limits are specified only between pairs of switches or clusters that are both immediately contained within the same larger cluster, i.e., they are siblings in the hierarchy tree $\mathcal{H}$.

$$
\begin{aligned}
V &= \{s, t\} \cup \{u_{s1}, u_{d1} : u \in N\} \cup \{u_{s1}, u_{s2}, u_{d1}, u_{d2} : u \in \mathcal{C}\} \\
E &= \{(s, u_{s1}, \alpha(u), 0),\ (u_{d1}, t, \omega(u), 0) : u \in N\} \cup \\
&\quad \{(u_{s2}, u_{s1}, \alpha(u), 0),\ (u_{d1}, u_{d2}, \omega(u), 0) : u \in \mathcal{C}\} \cup \\
&\quad \{(u_{s1}, v_{s2}, \infty, 0),\ (v_{d2}, u_{d1}, \infty, 0) : u \text{ a child of } v \text{ in } \mathcal{H},\ v \neq R\} \cup \\
&\quad \{(u_{s1}, v_{d1}, \mu(u, v), cost(u, v)) : u, v \text{ are siblings in } \mathcal{H}\}
\end{aligned}
$$

The costs $cost(u, v)$ are defined below.

The lower bound graph for the hierarchical traffic limits of Tables 4.1, 5.1, 5.2, and 5.3, and link cost coefficients of Table 5.4, is shown in Figure 5.3.

How should edge costs in $\mathcal{L}$ be assigned? Consider the example lower bound graph. It should be clear from our previous experience that we should define the costs of all

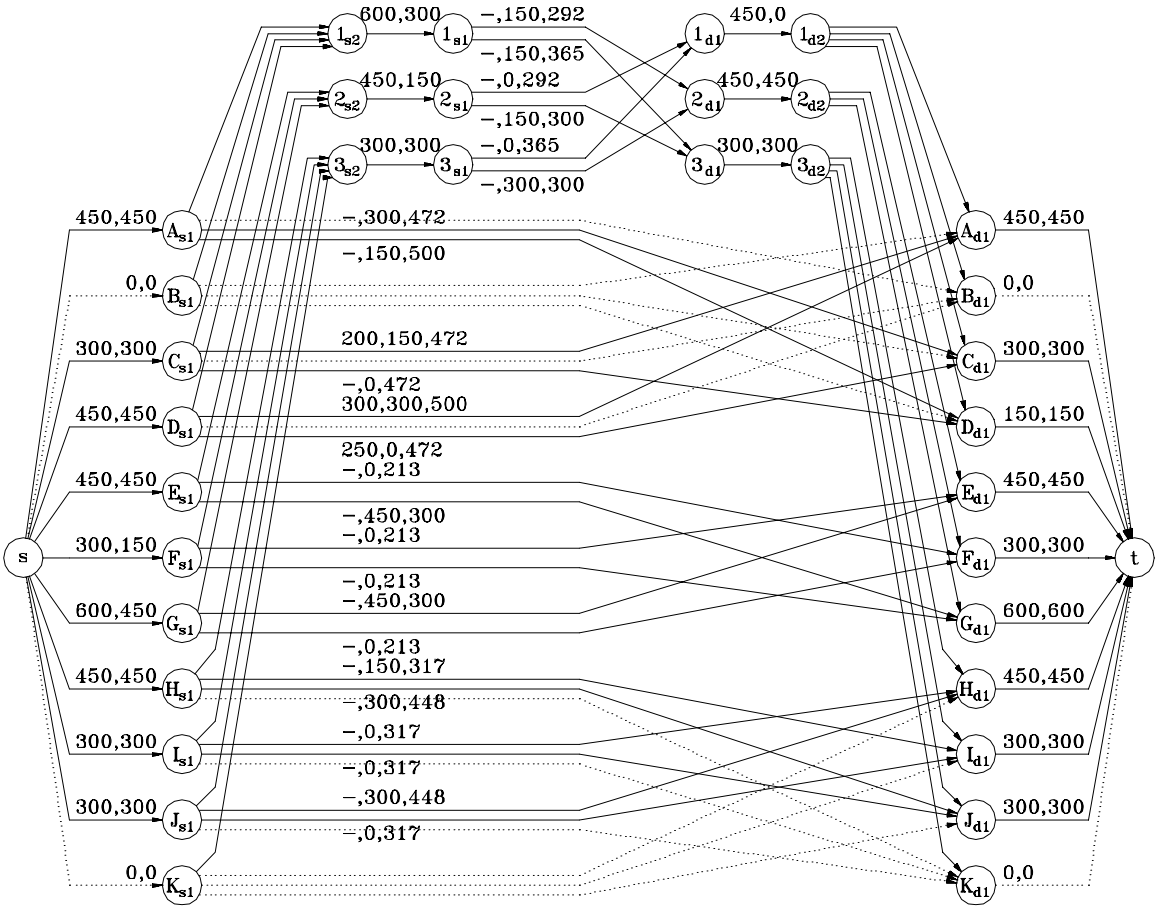

Figure 5.3: Lower bound graph for example hierarchical traffic limits. Edges are labeled with their capacity, an example flow, and cost, in that order. Edges with no cost label have 0 cost, and edges with no capacity label, or a dash (—), have infinite capacity. The dotted edges either have capacity 0, or they have flow 0 in every feasible flow. Their labels are not shown to avoid cluttering the figure.

edges of the form (s, u_{s1}) and (v_{d1}, t) to be 0, and the costs of all edges of the form (u_{s1}, v_{d1}) to be $\gamma(u, v)$, where u, v are both switches in N (e.g., edges (A_{s1}, B_{d1}) and (F_{s1}, E_{d1}), but not $(1_{s1}, 2_{d1})$). However, how should we define the other edge costs?

One property that should hold for our assignment of edge costs is: for any switch pair u, v, the total cost of all edges on any path from u_{s1} to v_{d1} should be at most $\gamma(u, v)$. We call this the *valid cost* property, and any path violating it is called *invalid*. If the valid cost property is violated, we are not guaranteed that the cost of a flow is a

lower bound for the cost of any nonblocking network. Any flow on an invalid path may cost more than is required to build the necessary links.

Given the restriction of keeping the costs valid, we want to define the edge costs so that the maximum cost flow costs as much as possible. It appears that computing these optimum edge costs exactly would require solving a nonlinear optimization problem. Even if this task could be done efficiently, we believe there are instances of hierarchical network design problems for which the costs between switches in different clusters cannot be accurately modeled in the lower bound graph. However, there is a simple way to define edge costs that usually seems to do fairly well, according to the experimental results in Section 5.4, and these costs can be determined very easily.

The basic idea is to examine some of the edges of the lower bound graph in a given order. On each step, increase the cost of the current edge as much as possible while still maintaining the validity of the edge costs. The approach used in obtaining the experimental results is to consider the edges (u_{s1}, v_{d1}), where u, v range over all sibling pairs of switches in the hierarchy tree. If only these edges are considered, then the order of their consideration does not matter. The resulting edge costs are the same in any case.

Valid edge costs are given by the third number labeling each edge in Figure 5.3. If there is no edge label, or only two numbers labeling the edge, then the cost is 0. One can see that the cost of edge $(1_{s1}, 2_{d1})$ is 292, which is the smallest cost from any switch in cluster 1 to any switch in cluster 2, $cost(A, F)$. This is the largest valid cost that edge $(1_{s1}, 2_{d1})$ may have. Therefore traffic between some pairs of switches in those clusters is not "charged" as much as possible. The linear program to compute the lower bound does not have this problem.

Once a maximum cost flow f^* in $\mathcal{L}$ has been computed, the cost of f^* is a lower bound on the cost of any nonblocking network. In Figure 5.3, a maximum cost flow is given by the second number labeling each edge. Its cost is $1257300/150 = \$8382$.

Even better, we may find any solution x that corresponds to f^*, and a lower bound is $LB(I, x)$ (3.7). This value is at least as large as the cost of f^*, because the edge costs must be valid. In most cases it is larger than the cost of f^*, and this method was used to compute the lower bounds for the experimental results in Section 5.4. For example, one solution x that corresponds to the maximum cost flow of Figure 5.3 is $x_{A,C} = 300, x_{A,D} = 150, x_{C,F} = 150, x_{C,A} = 150, x_{D,H} = 150, x_{D,A} = 300, x_{E,G} = 450, x_{F,I} = 150, x_{G,E} = 450, x_{H,I} = 150, x_{H,J} = 300, x_{I,F} = 150, x_{I,G} = 150, x_{J,H} = 300$. The cost implied by this set of requests is $1412550/150 = \$9417$, which is about 12.3% higher than the cost of the flow itself. This extra quality in the lower bound can be obtained in $O(n)$ time after the maximum cost flow has been found, much less time than it takes to find the flow itself.

An optimal solution to the linear program is $x_{A,C} = 250, x_{A,J} = 50, x_{C,A} = 200, x_{C,E} = 100, x_{D,G} = 350, x_{D,I} = 100, x_{E,C} = 50, x_{E,G} = 250, x_{E,H} = 150, x_{F,E} = 150, x_{G,D} = 100, x_{G,E} = 200, x_{G,F} = 300, x_{H,I} = 200, x_{H,J} = 250, x_{I,D} = 50, x_{I,H} = 250, x_{J,A} = 250, x_{J,H} = 50$, giving a lower bound of $1796550/150 = \$11,977$. This is 27.2% higher than the lower bound \$9417 found using the maximum cost flow method. For the randomly generated instances of Section 5.4, the average difference between these two lower bounds is often much less than this.

5.3. Link dimensioning, and hierarchical star networks

Link dimensioning in a tree network can be done using the same method as in section 4.3, based on solving the linear program of maximizing (4.8), but now subject to our new system of constraints (5.5).

Unlike before, we know of no maximum flow problems that are equivalent, at least for arbitrary sets of switch pairs P_l. However, when P_l is of the form $\{(u,v) : u \in X, v \in Y\}$ for some sets of switches $X, Y \subseteq N$, the linear program is equivalent to a maximum flow problem on a graph $\mathcal{L}_l = (V, E, cap)$ that is the same as $\mathcal{L}$ of the previous section, except that we reduce the capacities of all edges in the sets $\{(s, u_{s1}) : u \notin X\}$ and $\{(v_{d1}, t) : v \notin Y\}$ to 0. This special case is useful, because P_l is of the proper form when solving the link dimensioning problem in a tree network.

For hierarchical traffic, we generalize the notion of a star network to a hierarchical star network. We go through the hierarchy tree of the instance in a bottom-up fashion, computing the cheapest star subnetwork for each of the lowest level clusters. Then, treating each of the lowest level clusters as a single switch located at the center of the chosen star subnetwork, find the cheapest star subnetwork for the next to lowest level clusters.

As before, we may precompute the minimum necessary link capacities in and out of each switch, instead of performing a redundant calculation for each star subnetwork tried. We may also precompute the minimum necessary link capacities in and out of each cluster, super-cluster, etc., when finding the cheapest star networks connecting lower level clusters.

For example, consider again the instance with hierarchical traffic limits given by Tables 4.1, 5.1, 5.2, and 5.3, and link cost coefficients given in Table 5.4. For cluster 1,

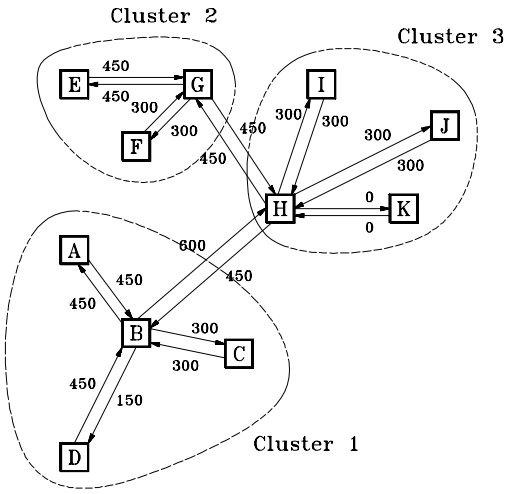

Figure 5.4: A nonblocking network for the example hierarchical instance

the cheapest star subnetwork is the one with center switch B, exactly as we found at the end of Section 4.3. For cluster 2, the cheapest choice of center is G, and for cluster 3, the cheapest center is H. Now we find the cheapest star network connecting the subnetworks, where we treat all switches in 1 as if they are located at B, all switches in 2 at G, and all switches in 3 at H. This network is the one with center H, so we add links $\{(B,H),(H,B),(G,H),(H,G)\}$ to the star subnetworks already found. This network is shown in Figure 5.4. The cost of this network is $2245350/150 = \$14969$, which is 25.0% over the linear programming lower bound of \$11977, and 59.0% over the maximum cost flow lower bound of \$9417. Therefore this network costs at most 25% over optimal.

This procedure can be implemented to run using $2(|N|+|\mathcal{C}|)$ maximum flow computations, each requiring only $O(n)$ time due to their special structure. The best centers for each cluster can be determined in time $\sum_{c\in\mathcal{C}\cup\{R\}}|c|^2$, where $|c|$ is the number of switches and sub-clusters that are children of c in $\mathcal{H}$.

5.4. Experimental results

In this section we present experimental results comparing the cost of inexpensive hierarchical star networks, computed as described in Section 5.3, to the lower bounds. Here the lower bounds are computed in two ways. First there is the faster but less accurate method of using the lower bound graph with valid edge costs, described in Section 5.2, and then the slower but more accurate method of solving the linear program LP, described later in the same section.

All instances generated are two-level hierarchies, although the methods for instance generation and all algorithms work for arbitrary hierarchical structures.

To generate random two-level hierarchies, we are given a number of clusters c and a number of switches m to place within each cluster, giving a total of $n = cm$ switches. In addition to the the parameters given for flat instances in Section 4.4, we are given a range of real numbers $[\alpha^c_{lo}, \alpha^c_{hi}]$, which is a sub-interval of $[0, 1]$.

When placing switches, we assume that not only do clusters have higher traffic among the switches within, but the switches are also geographically clustered together. For each cluster, we generate a random width w in the interval $[.1, .25]$ and a height h in the same interval. A bounding rectangle for the cluster with these dimensions is then randomly placed in the unit square, and all m switches within the cluster are randomly placed within this bounding rectangle. Bounding rectangles for different clusters are allowed to overlap. Link cost coefficients $\gamma(u, v)$ are set equal to the Euclidean distance between u and v. The α values for switches (but not clusters) are generated as before. For each cluster U containing switches $u_1, \ldots u_m$, generate a random real value x in the interval $[\alpha^c_{lo}, \alpha^c_{hi}]$ and set $\alpha(U) = \omega(U) = x \cdot \sum_{i=1}^{m} \alpha(u_i)$. If this interval is $[1, 1]$ then the clusters do not constrain source and termination capacity any more than that of the constituent switches (although μ values for the clusters may constrain the traffic more

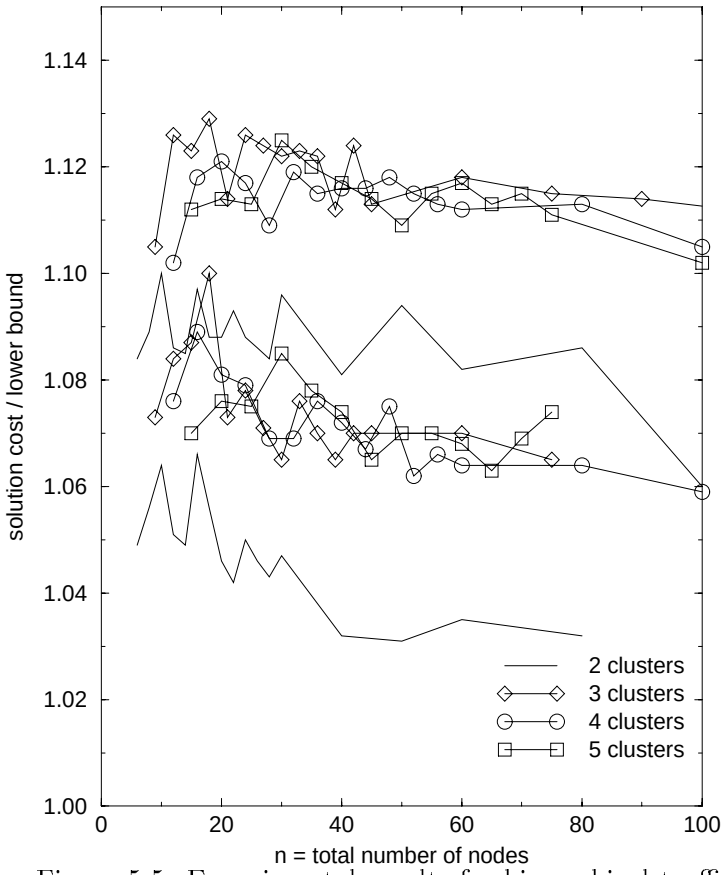

Figure 5.5: Experimental results for hierarchical traffic

than if there were no clusters). All experiments reported here use the interval $[.1, .3]$, so that 10% to 30% of a cluster's traffic may be between switches within it and switches in other clusters.

The top four curves of Figure 5.5 show the average ratio of the hierarchical star network cost over the lower bound obtained by finding a maximum cost flow in the lower bound graph, and then finding the cost of a solution x corresponding to this flow. Each curve represents a different value of c, the number of clusters, as shown. The number of switches per cluster m ranges over the same values as n did in the previous section, as long as $n = cm \leq 100$. Each data point is the average performance ratio of 80 randomly generated instances. Experiments were also run for up to $c = 10$ clusters,

but the curves for more than 5 clusters do not differ significantly from those with 3 to 5 clusters, so they are not shown.

It seems that there may be a trend towards 1, but if so, then it occurs much more slowly than before. The performance ratios are significantly better when there are two clusters, as opposed to three or more. If we had flat instances with two switches, the lower bound and the minimum cost solution have exactly the same value. It is only for three or more switches that they can differ. Similarly, when we have two level hierarchical instances with two clusters, the portion of the lower bound caused by the inter-cluster traffic is close to the cost of the links between the clusters. When there are three or more clusters, these values can differ significantly. We expect that curves for large numbers of clusters (e.g. 50) would be closer to the $c = 2$ curves than to the curves for 3,4, or 5 clusters.

The lower four curves of Figure 5.5 show the average ratio of the hierarchical star network cost over the lower bound computed by the linear program. Each data point is the average ratio of 30 randomly generated instances, which are identical to 30 of the instances used in generating the upper four curves. The linear programming lower bound was not computed for the larger instances, because of the time and memory required to compute them. The largest instances for which the linear program was solved required up to 3 hours of CPU time on a Sun 4/110 workstation, and approximately 15 megabytes of virtual memory. They were solved using an unsophisticated (but free) implementation of the simplex method, and both the time and memory required could probably be reduced by a factor of 5 to 10 by using more careful implementations that take advantage of the sparsity of non-zero coefficients. Again, experiments were run for up to 10 clusters, but the curves for more than 5 clusters are very similar to those for 3 to 5 clusters.

The range of values is notably lower now, from about 1.03 up to 1.10. There seems to be more of a tendency to approach 1 as n increases. Thus we should also be able to extend the probabilistic result of Section 3.8 to some kinds of hierarchical traffic limits as well.

6. MULTIPOINT CONNECTIONS

Thus far we have only explicitly considered point-to-point requests and connections. In this chapter we show how the fixed path routing algorithm can be extended in a straightforward way to handle multipoint connections, and we prove that if a network is nonblocking for point-to-point requests when using fixed path routing, then it is also nonblocking for multipoint requests.

6.1. Definition

A multipoint connection request is a triple $r = (src, dest, \rho)$. Instead of having only a single source and a single destination, there may be multiple sources and multiple destinations. Switches $src(r) \subseteq N$ are the sources of information flow, and switches $dest(r) \subseteq N$ are the destinations of flow. The desired rate of the connection is $\rho(r)$, as before.

We have studied three kinds of multipoint requests. The first is a *one to many* request in which $|src(r)| = 1$ and $|dest(r)| > 1$. Such a request can be realized by a connection $c = (\Pi, \rho)$ where $\Pi(c)$ is a directed rooted tree in which the single switch $src(r)$ is the root, all links are directed away from the root, and all switches in $dest(r)$ are either leaves or internal switches of the tree. It is wasteful of link bandwidth to allow leaves in the connection tree that are not switches in $dest(r)$. Every switch that is an internal switch of the connection receives data at rate $\rho(c) = \rho(r)$ from its upstream neighbor (i.e., its parent in the tree), and sends copies of the data to all of its downstream neighbors (its children in the tree). If the switch is in $dest(r)$, then a

copy of the data is also sent to the directly attached terminals that wish to receive it. We assume that the switches operate in a nonblocking fashion with respect to new one to many connection requests.

The second kind of request is a *many to one* request in which $|src(r)| > 1$ and $|dest(r)| = 1$. Such a request can be realized by a connection $c = (\Pi, \rho)$ where $\Pi(c)$ is a directed rooted tree in which the single switch $dest(r)$ is the root, all links are directed towards the root, and all switches in $src(r)$ are either leaves or internal switches of the tree. Every switch that is an internal switch of the connection receives data with a rate at most $\rho(c) = \rho(r)$ from each of its upstream neighbors (its children in the tree), and multiplexes and sends it to its downstream neighbor (its parent). If the switch is in $src(r)$, then at least one of the sources being multiplexed is a directly attached terminal.

Note that since multiplexing of sources is being performed, we really can't expect to use all of the $\rho(c)$ bandwidth that we have reserved on each link in $\Pi(c)$ for sustained periods of time. If all senders want to send at full rate constantly, then one should set up multiple point-to-point connections, not a many to one connection. However, we assume that bandwidth $\rho(c)$ is reserved for connection c on all links in the connection, and may not be used by other connections.

The third kind of request is a *many to many* request in which $src(r) = dest(r)$, the sources and destinations are the same. A many to many request can be realized by a set of links $\Pi(c)$ that forms a tree network containing switches $src(r)$, and possibly others. Every switch u receives data with a rate at most $\rho(c) = \rho(r)$ from each of its neighbors in the connection. Its neighbors in the connection include the adjacent switches in the connection tree $\Pi(c)$, and, if u is in $src(r)$, some attached terminals. Data received from a particular neighbor is copied and sent to all other neighbors except the source of the data.

For the last time, here are the extended definitions of α-*usage*, ω-*usage*, and μ-*usage*. These definitions include all of the previous ones as special cases. Let a set of switches N and hierarchical traffic limits $\mathcal{T} = (\mathcal{H}, \alpha, \omega, \mu)$ be given. Let $s(c)$ be the set of all switches in N that are contained in the cluster $c \in \mathcal{C}$, or just $\{c\}$ if $c \in N$ is a switch.

$$\alpha\text{-}usage\,(c, \mathcal{R}) = \sum_{\substack{r \in \mathcal{R}, \\ src\,(r) \cap s(c) \neq \emptyset \\ dest\,(r) \cap \overline{s(c)} \neq \emptyset}} \rho(r) \tag{6.1}$$

$$\omega\text{-}usage\,(c, \mathcal{R}) = \sum_{\substack{r \in \mathcal{R}, \\ src\,(r) \cap \overline{s(c)} \neq \emptyset \\ dest\,(r) \cap s(c) \neq \emptyset}} \rho(r) \tag{6.2}$$

$$\mu\text{-}usage\,(c_1, c_2, \mathcal{R}) = \sum_{\substack{r \in \mathcal{R}, \\ src\,(r) \cap s(c_1) \neq \emptyset, \\ dest\,(r) \cap s(c_2) \neq \emptyset}} \rho(r) \tag{6.3}$$

The definition of a compatible request set does not change.

6.2. Point-to-point traffic is the worst

In every previous section on link dimensioning, we have only explicitly considered point-to-point requests and connections. If allowing multipoint connections could increase the maximum usage of link l, $\lambda^*(l)$, then we would want to find a way to compute these larger values. In this section, we show that $\lambda^*(l)$ remains the same when multipoint connections are allowed. This means that the worst-case traffic for any link can be achieved by point-to-point connections only.

For now, we assume that the fixed path routing algorithm routes a one to many request (u, V, ρ) by using all of the links $\Pi(u, V) = \bigcup_{v \in V} \pi(u, v)$. In a real network, the connection would form a rooted tree whose links are a subset of $\Pi(u, V)$, but our result holds even when all of these links are used. We assume that a many to one request uses

all links in $\Pi(U,v) = \bigcup_{u \in U} \pi(u,v)$. We examine many to many requests in the context of a tree network, where we know that the nodes and links in the connection must also form a tree network.

Here is how we show that $\lambda^*(l)$ does not increase when multipoint connections are allowed. Let l be a link with an arbitrary set of switch pairs P_l that must use l to communicate, $P_l = \{(u,v) : l \in \pi(u,v)\}$. Given any compatible set of requests $\mathcal{R}$ containing multipoint requests, we show how to replace any multipoint request r with a point-to-point request r' that induces the same traffic on l, but adds no more than r does to α-*usage*, ω-*usage*, and μ-*usage*. Thus the set $\mathcal{R}'$ resulting from replacing all multipoint requests with point-to-point requests must also be compatible. We have already considered all compatible point-to-point request sets when finding the maximum usage of l.

The replacement is quite simple. If $r = (u, V, \rho)$ is a one to many multipoint request, and there is some $v \in V$ for which $(u,v) \in P_l$, then the connection will use link l. If we replace r with $r' = (u, v, \rho)$, then the connection for r' will use the same capacity ρ on link l that r did, but r adds ρ to ω-*usage*(x) and to μ-*usage*(u,x) for all $x \in V - \{v\}$, while r' does not. If $r = (u, V, \rho)$, and there is no $v \in V$ for which $(u,v) \in P_l$, then the connection satisfying r does not use link l, and so may be removed from the request set.

A many to one multipoint request $r = (U, v, \rho)$ can be handled similarly. If there is a switch $u \in U$ such that $(u,v) \in P_l$, replace r with $r' = (u, v, \rho)$, otherwise remove r from $\mathcal{R}$.

For a many to many multipoint request $r = (U, U, \rho)$ in a tree network, link l is used in the connection if and only if there is some $u_1 \in U \cap X_l$, where X_l is the set of switches on the source side of l, and some $u_2 \in U \cap (N - X_l)$. If this holds, replace r with $r' = (u_1, u_2, \rho)$, otherwise remove r from $\mathcal{R}$.

Thus, the maximum usage of any link can be achieved by point-to-point requests only, and hence any network that is nonblocking for point-to-point requests is nonblocking for multipoint requests as well. This result holds when fixed path routing is used, and multipoint requests are realized by connections that perform the copying as late as possible.

The intuitive idea for why this works is that the termination limits of the switches are "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.

6.3. Dynamic multipoint connections

The results of the previous section also imply that we may dynamically add switches to and remove switches from multipoint connections, without removing the entire connection and replacing it with a new one. This requires that the switches are also capable of doing so. This is a useful property for a network to satisfy, since network users would desire such an ability for multipoint applications like teleconferencing. However, some rearrangement of the connection may be necessary when removing a switch, depending on the structure of the fixed paths.

In this section, the discussion focuses on one to many multipoint connections, but it applies just as well to many to one connections. Most if it does not apply to many to many connections, because we have only proved that they are nonblocking when the connection is always a tree.

As an example of the rearrangement required, consider the network of Figure 6.1, where $\pi(A, D) = ACD$ and $\pi(A, E) = ABCE$. Suppose there is first a connection request $(A, D, 1)$, which can be satisfied by the path ACD. Now suppose that we wish to add the destination E to this connection. One way would be to remove the existing

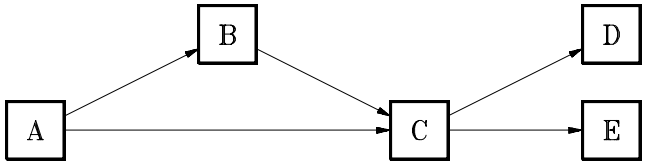

Figure 6.1: Simple network to demonstrate dynamic multipoint connections

connection and then build a connection for the request $(A, \{D, E\}, 1)$. Another would be to add links to the existing connection so that the result is a tree that satisfies the connection request $(A, \{D, E\}, 1)$.

The following way makes as small a change as possible to the existing connection. Step backwards along the path $\pi(A, E)$ until a switch is reached that is already in the connection. In this example, add the link (C, E) and then stop, because C is in the connection. This method works 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 $\Pi(u, V)$.

However, suppose that we now wish to remove D 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 that must remain in the connection (because of other remaining destinations). If we tried that in the example, we would remove the link (C, D) from the tree, ending up with the path $ACE \neq ABCE = \pi(A, E)$.

As far as the existing connection requests are concerned, there is only $(A, E, 1)$. However, the link (A, C) is used in the current state, and so future (A, D) 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 $\Pi(u, V)$ 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 $(A, C), (C, E)$ and add links $(A, B), (B, C)$.

Note that this problem does not occur in a tree network, and removing destinations can always be done by removing links but not adding any.

7. EXTENSIONS

7.1. Blocking due to link fragmentation

A requirement in the ATM standard is that all cells in a virtual circuit must be delivered in the same order they are sent. While all ATM switch architectures known to the author maintain cell order for virtual circuit cell streams arriving on a single input port and leaving on a single output port, few architectures are designed to maintain cell order for cell streams arriving on multiple input ports and leaving on multiple output ports. Maintaining cell order on multiple parallel high speed links is also a challenging task. The Aurora test bed network was designed to do this [17]. See Saidi et al. [58, 59] for one example of a switch architecture designed to do this, and citations to several others.

For the rest of this section, we assume that the switches used are incapable of maintaining cell order for streams on multiple ports. This means that if there are multiple physical links between a pair of switches, all cells within a virtual circuit must go on the same physical link.

In all of the previous chapters except the introduction, we have represented a group of physical links between a pair of switches as a single directed edge that has a capacity equal to the total of all of the physical links. This made it easier to think about the problem, but such a model does not accurately represent a real ATM network. A network for which we have dimensioned the links to be nonblocking will block when the aggregate capacity links are implemented with multiple physical links with smaller capacity.

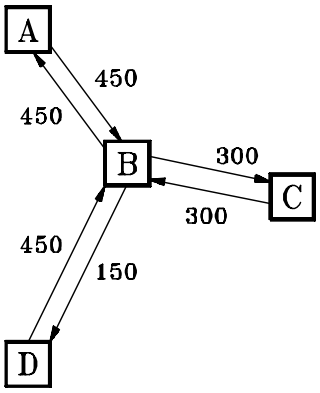

Figure 7.1: Repeat of nonblocking network of Figure 2.1

Table 7.1: Repeat of α, ω-bounded traffic limits in Table 3.1

u	$\alpha(u)$	$\omega(u)$
A	450	450
B	0	0
C	300	300
D	450	150

For example, recall the first example network given in Chapter 2, shown here again as Figure 7.1. We saw in Section 3.2 that this network was nonblocking for the flat α, ω-bounded traffic limits given there, shown again in Table 7.1. However, when the network is implemented with 150 Mb/s physical links, as shown in Figure 7.2, we see that the following compatible sequence of requests blocks. The request $(C, A, 100)$ can be satisfied by using either of the physical links from C to B, but the next request $(C, A, 125)$ can only be satisfied by using the other one. It is now compatible with the traffic limits to make the request $(C, A, 75)$. We see that there is a total of 75 Mb/s available on the two links from C to B, but it is not all available on a single link. Therefore the request blocks.

The main cause of this problem is allowing requests with rates that are close to the rates of the physical links. In these cases, it is possible to enter states where each

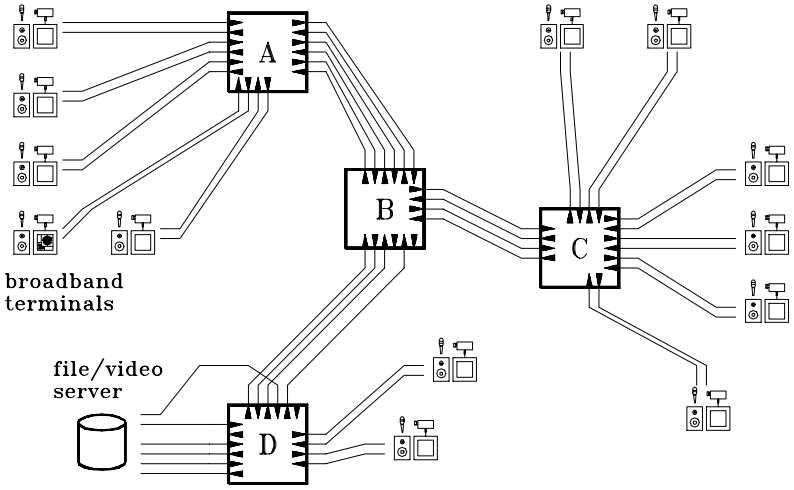

Figure 7.2: Physical switches and links that implement network in Figure 7.1

one of a group of physical links can be used a little bit, and then a single large rate request is made that cannot fit on any of them. If we restrict the maximum connection rate allowed to be a fraction of the physical link rate, this problem can be solved by installing physical links with a total rate larger than the aggregate rate needed for a nonblocking network.

Suppose that all physical links installed between a particular switch pair u, v have the same capacity R. Let ρ^* be the maximum connection rate allowed. Let $x = \lambda^*(u, v)$ be the maximum traffic from u to v. If we wish to add a new connection with rate ρ that uses a link in the group from u to v, then this request will block if and only if the utilization of all links is at least $R - \rho + \epsilon$, for any $\epsilon > 0$. A new request with rate ρ is compatible with the traffic limits only if the current total usage is at most $x - \rho$. With this traffic, the number of links that can be utilized at least $R - \rho + \epsilon$ is $\left\lfloor \frac{x-\rho}{R-\rho+\epsilon} \right\rfloor$. The maximum of this expression over all $\epsilon > 0$ is $y = \left\lceil \frac{x-\rho}{R-\rho} \right\rceil - 1$. This is the maximum number of links that are too full to handle the new connection. Thus if we have at least $y + 1 = \left\lceil \frac{x-\rho}{R-\rho} \right\rceil$ links from u to v, no request with rate ρ will ever block.

We desire that no compatible request blocks, so we should maximize $\left\lceil \frac{x-\rho}{R-\rho} \right\rceil$ over all values of ρ, $0 < \rho \leq \min\{\rho^*, x\}$. This maximum is

$$\max_{0<\rho\leq\min\{\rho^*,x\}} \left\lceil \frac{x-\rho}{R-\rho} \right\rceil = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{if } 0 < x \leq R \\ \left\lceil \frac{x-\rho^*}{R-\rho^*} \right\rceil & \text{otherwise} \end{cases} \tag{7.1}$$

which is achieved (for $x > R$) when $\rho = \rho^*$. If we let $\rho' = \rho^*/R$, then this is

$$n(x,\rho') = \begin{cases} 0 & \text{if } x/R = 0 \\ 1 & \text{if } 0 < x/R \leq 1 \\ \left\lceil \frac{x/R-\rho'}{1-\rho'} \right\rceil & \text{otherwise} \end{cases} \tag{7.2}$$

If there are at least $n(x,\rho')$ links from u to v, then no request will block because of insufficient capacity there.

Note that if the cells of a single connection could be split up among multiple physical links, then only $\lceil x/R \rceil = n(x,0)$ links would be needed. $n(x,\rho')$ is roughly $1/(1-\rho')$ times larger than $n(x,0)$. For any value of ρ', the total instantaneous traffic on all of the links is x, so the $n(x,\rho')$ links have an average utilization that is no more than about $(1-\rho')$.

Let us take these results and apply them to our example. Let $R = 150Mb/s$ and $\rho^* = 75Mb/s$, giving $\rho' = 1/2$. The maximum traffic from A to B is $450Mb/s$, so the minimum number of links needed from A to B is $\left\lceil \frac{450/150-1/2}{1-1/2} \right\rceil = 5$. Similarly for the opposite direction. Repeating this calculation for all switch pairs, we get the network in Figure 7.3.

Note that switch A serves no useful purpose in this network that cannot be performed by switch B just as well, because A has 5 links in each direction between its terminals and itself, and 5 links in each direction between itself and B. Therefore, it can be removed, and all terminals that were previously attached to A can now be attached to B, as in Figure 7.4.

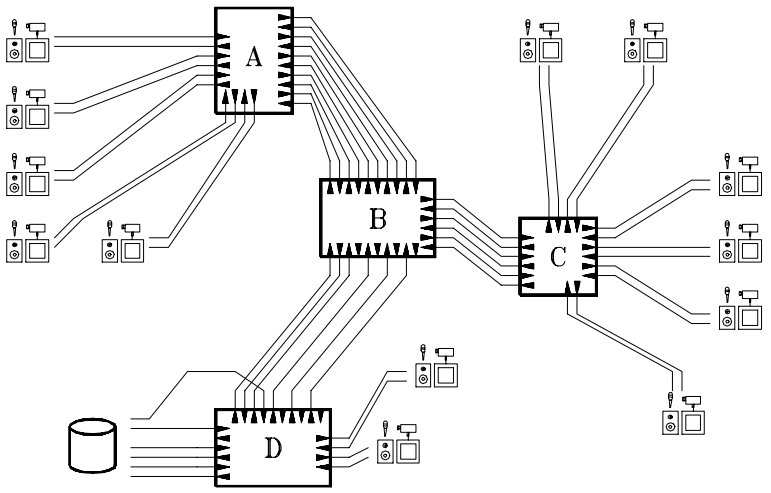

Figure 7.3: Network that is nonblocking even with link fragmentation

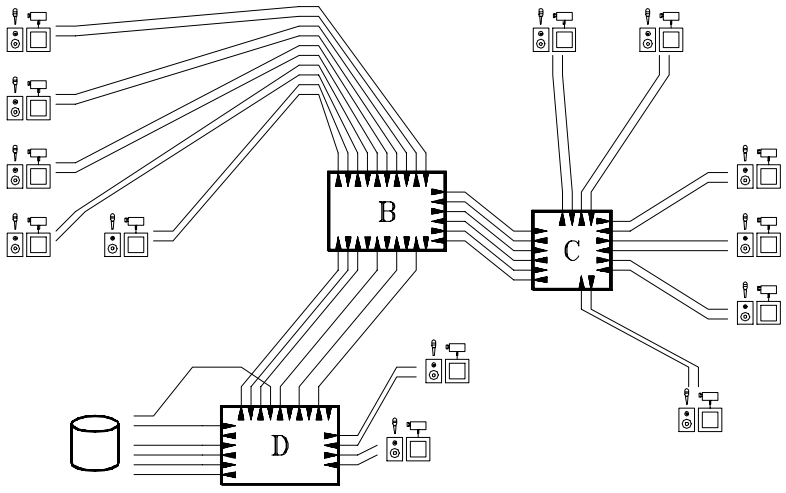

Figure 7.4: Network with unnecessary switch A removed

Note that this substitution of $n(x,0)$ physical links with $n(x,\rho')$ links increases the cost of the network by a factor of about $1/(1-\rho')$. There may be a way to increase the value of the lower bound in a similar way, by replacing the objective function (3.7) with

$$LB'(I,x) \quad = \quad \sum_{u,v \in N, u \neq v} \gamma(u,v) n(x_{u,v}, \rho') \tag{7.3}$$

Unfortunately, this no longer leads to a linear program because of the shape of the $n(x,\rho')$ function. We can use a linear function of x that is always no more than $n(x,\rho')$, but the only such function is x. We would prefer something like $x/(1-\rho')$, to keep the cost of the network and the value of the lower bound close.

7.2. Handling physical constraints on network installation

When a network managers decide to install a network, it is not automatically apparent how to formulate their problem as a network configuration problem defined in Section 2.3. They know (or must find out) what the physical layout of their site is, i.e., where terminals are located, where switches may be placed, where cables may be installed, or have already been installed, building codes, etc. They also have a variety of switch types they may buy with different features and costs, different types of cables to choose from, and various host-network interface cards that may be installed in the workstations.

In this section we outline some of the details that must be considered by the network manager. A manager's dream software package would take all of these details into account and assist the manager in installing and upgrading the network as cheaply as possible over its lifetime. Some of these details can be taken into account by the research that has been done.

Several structures arise when specifying an instance of the network configuration problem and when discussing solutions. A *physical constraint graph* describes physical constraints for where equipment may be placed. *Equipment descriptions* specify the types and properties of physical objects that may be used in constructing the network. A *physical network* describes a way of connecting equipment together to form the network, an *embedding* specifies where to place equipment, and a *logical network* ignores the exact equipment used in a physical network and concentrates on the properties of the equipment related to network performance. *Traffic requirements* specify how we expect to use the network.

We have already discussed traffic limits earlier in this dissertation. They are a way of specifying traffic requirements for a network. Physical networks are graphs like the ones in Figures 7.3 and 7.4, and logical networks are simply the networks we have been using throughout the dissertation. More detailed descriptions of physical constraint graphs, equipment descriptions, and embeddings are contained in the following subsections. The last subsection discusses which parts of this very general network configuration problem that we know how to account for using our work.

7.2.1. Physical constraint graphs

A *physical constraint graph*, or simply *constraint graph*, represents the constraints placed on the location of equipment by the site (buildings, walls, etc.) in which the network will be installed. It may certainly be possible to alter such restrictions (tear down buildings, put up walls, etc.), but for our purposes, any such changes are considered either impossible or too costly.

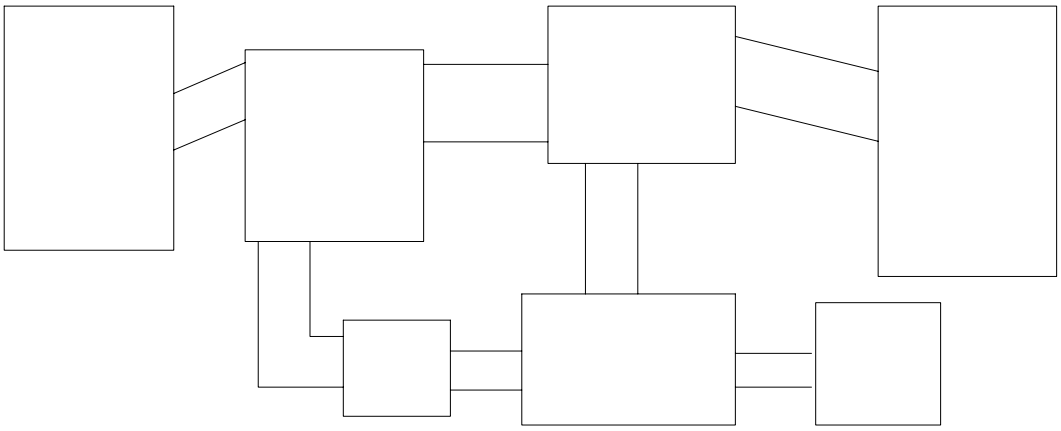

Figure 7.5: An example physical constraint graph

A constraint graph is an undirected multigraph $\mathcal{P} = (V, E)$. Each vertex $v \in V$ represents a *place* where switching equipment could be installed (e.g. a communication closet). Each place has several parameters that are important for the design problem:

- Volume available.
- Physical conditions (e.g., humidity, temperature).
- Cost of making the room suitable for switch installation.

Each edge $e \in E$ represents a *conduit* where one or more transmission cables may be placed, e.g., a duct or chase in a building, a tunnel between buildings. Each conduit has several parameters:

- Length
- Cross-sectional area available.
- Physical conditions (e.g., electromagnetic interference).
- Cost for laying cable on this conduit.

An example constraint graph is given in Figure 7.5. Rectangles represent places, and "fat lines" represent conduits. These things have been drawn with different sizes to suggest their respective volumes, cross-sectional areas, and lengths.

7.2.2. Equipment descriptions

When installing a network, one must have in mind what sort of equipment has been or may be bought, including its cost and performance characteristics.

There may be several types of switches available for installation. Each type is parameterized by:

- Volume occupied.
- Number and type of links that may be connected to the switch.
- Total switching capacity, e.g., aggregate throughput of 2.4 Gb/s, or, the switch is nonblocking for all connection assignments that do not overload the ports.
- Multicast capability.
- Cost, possibly broken down into an initial cost and a continuing maintenance cost.
- Need for a controlled environment.

There can be several types of cables (e.g., twisted pair, coaxial, optical fiber) that can be installed. Each type is parameterized by:

- Cross-sectional area occupied.
- Maximum length over which a signal may be effectively carried.
- Bandwidth measured in bits/sec.
- Cost per unit length.
- Termination cost. This is the cost of connecting a cable to a switch, and may depend on the type of both.
- Shielding, i.e., resistance to electromagnetic interference.

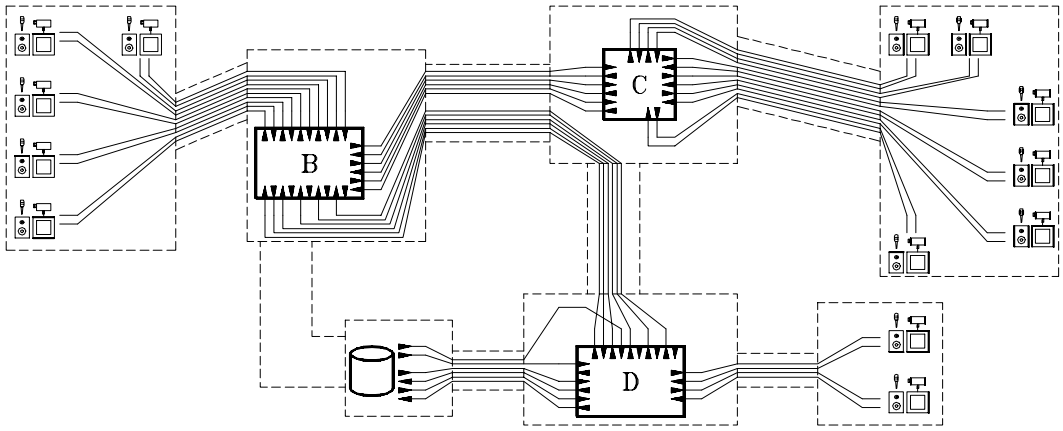

Figure 7.6: An example embedding

7.2.3. Embeddings

An *embedding* is a mapping from a network $\mathcal{N}$ to a physical constraint graph $\mathcal{P}$. It specifies where each switch and physical link is located. Switches in $\mathcal{N}$ are mapped to vertices in $\mathcal{P}$ (places). Links in $\mathcal{N}$ are mapped to paths in $\mathcal{P}$, because a cable may pass through zero or more conduits before connecting to a switch.

An embedding is *feasible* if it satisfies constraints such as volume available in places, cross-sectional area available in conduits, and environment constraints. Taken together, a physical constraint graph, equipment descriptions, a network, and an embedding determine the cost of the network.

An example embedding is given in Figure 7.6. It is the network of Figure 7.4 embedded into the physical constraint graph of Figure 7.5. It is drawn to suggest the embedding used. The physical constraint graph is drawn with dashed lines.

7.2.4. Physical constraints that can be accounted for with our results

The network configuration problem of Section 2.3 specifies a set of switches and the costs of installing links between them. This models an instance of the more general

problem in which some switches have already been installed in particular places and connected to nearby terminals, and they may not be moved. The only choices left are where to install links. Section 7.3 describes how we can solve an instance in which there are switches already given, as before, but additional switches may be installed in any place.

The linear link costs can model the cost of the cable itself, the termination costs, and, partially, the cost of the switches. For example, if a particular type of fiber optic cable contains 12 fibers capable of carrying 150 Mb/s each costs \$1.50 per foot, transceivers for converting electrical and optical signals cost \$500 per pair, and 16-port switches cost \$25,000, then we can set the cost of installing links between a pair of switches 1000 feet apart at $1000 \times (1.50/12) + 500 + 25{,}000/16 = \$2187.50$ per 150 Mb/s link. This is not exact, but it does give a fairly good approximation when many cables are installed and most switch ports are used.

7.3. Network configuration in which additional switches are allowed

In this section, we examine a network configuration problem in which there are several switches with given locations and traffic limits, but additional switches other than those given may be added to the solution.

7.3.1. Discrete choices for location of additional switches

Suppose that we are given an instance of the network configuration problem with a physical constraint graph as defined in Section 7.2.1, some switches N that are required to be in specific places, flat α,ω-bounded traffic limits with $\alpha(N) = \omega(N)$, and costs of installing links between each pair of places, satisfying the triangle inequality. The

objective is to find a nonblocking network containing the given switches, and possibly more, with minimum cost.

Augment the given set of switches N by adding nodes N'. N' contains one switch at each place in the physical constraint graph that does not already have one. Let $\alpha(u) = \omega(u) = 0$ for all $u \in N'$.

From Corollary 3.6, we know that the cheapest tree network for the switches $N \cup N'$ is a star network. Assume for the moment that a star network is cheapest among all networks. Let C be the center switch chosen for this star.

If $C \in N'$, note that all of the other switches in $N' - \{C\}$ are attached to C by links with capacity 0. They did not add anything to the cost of the network, and need not actually be installed since they will never be used to carry connections. This star network with the additional switch C as the center is no more expensive than any star that contains only the given switches N, and may be cheaper. Hence the additional switch may have helped reduce the cost of the network. For example, the cheapest star for our original instance in Chapter 3 had switch B, with $\alpha(B) = \omega(B) = 0$ as the center. The cheapest star using only switches A, C, and D has center A and cost $583200/150 = \$3888$, compared to \$3864 for the star with center B. This represents a savings of only 0.62%, but it may be significantly larger.

If $C \in N$, then all switches in N' are connected to C by links with capacity 0. None of them need to be added to the network, and furthermore, a cheaper network cannot be built by adding them to the network in any way. For example, the cheapest star subnetwork for cluster 3 in the hierarchical instance of Chapter 5 was H, not K.

Thus we see that when a star network is cheapest among all networks, it never helps to add more than one additional switch to those given. In addition, if several discrete choices for the location of an additional switch are given, we only need to evaluate each

one of them as a candidate for the center of the star network to determine if the location helps reduce the network cost.

Note that when the cost of switches is more accurately modeled, the star with an additional switch may actually cost more because it has one more switch than the cheapest star using the given switches N only. One may simply compare the cost of the cheapest star with the additional switch to the cost of the cheapest star without it, and use the cheaper one.

The same technique may be used repeatedly when configuring hierarchical star networks, once for each star subnetwork. All of the lower bounds described previously are also lower bounds for these solutions.

7.3.2. No restriction on location of additional switches in the Euclidean plane

The method in the previous section works if there is a finite set of possible locations to place an additional switch. If we are given a set of switches in the Euclidean plane, and additional switches may be placed anywhere, then there are an infinite number of choices and that method does not work. However, it is still the case that at most one additional switch can help reduce the cost of the star network.

Let the cost of installing links be any function $cost(cap) \cdot \text{dist}$ that is linear with the Euclidean distance dist between the switches. It does not matter what the shape of the function is with respect to link capacity cap. Then the cost of a star network with center switch C is

$$\sum_{u \in N} (cost(\alpha(u)) + cost(\omega(u))) \cdot \text{dist}(u, C) \tag{7.4}$$

because there must be a link (u, C) with capacity $\alpha(u)$ and a link (C, u) with capacity $\omega(u)$ between every node $u \in N$ and C.

To find the best location for the additional switch C, we may think of this problem as a mechanical system in which the links between C and switches in N are springs, C is a mass that may move about, and the switches in N are fixed anchor points for the springs. The springs pull the mass C towards the spring's anchor point with a constant force $(cost(\alpha(u)) + cost(\omega(u)))$, no matter how far it is stretched.

The potential energy function for this mechanical system is exactly the cost of the network (7.4). When the potential energy function of a mechanical system is at a local minimum, the mechanical system is in a state of stable equilibrium. Thus, we may simulate the mechanical system, possibly with some "friction" added to damp oscillation, and the position of C will eventually settle to a point of stable equilibrium. If any of the cost functions *cost* have a discontinuous "jump" in the cost from 0 capacity to positive capacity, then the discrete locations of all switches N should also be tried as centers.

This interpretation of the optimization problem as a mechanical system was introduced to the author through the work of Gilbert and Pollak [34, 35], who examined a similar physical system for finding minimum cost Steiner trees in the Euclidean plane.

We conjecture that the cost of the network with one additional switch costs at least $\pi/4 \approx 0.7854$ times as much as the cheapest star when no additional switches are allowed. This ratio is approached as n grows for instances where n switches are equally spaced around a unit circle.

7.4. Expanding an already installed network

After a network has been installed, the traffic offered to the network often changes over relatively long periods of time (e.g., months). A network manager may react to these changes by modifying the installed network to handle the new traffic. The goal is then

not to modify the existing network to become the cheapest network for the new traffic, but to modify the network as cheaply as possible to become a network that can handle the new traffic.

Such a problem has been studied by Jack, Kai, Shulman, and Bienstock in the context of expanding the cable plant of a local access telephone network [11, 46]. It has been studied by Balakrishnan et al. under a more abstract setting [7]. In all of this work, the offered traffic is essentially what we would call flat α, ω-bounded traffic limits. We believe that this work can be either used directly for or adapted to the problem of expanding a nonblocking connection-oriented network with this kind of traffic limits.

For flat μ-bounded traffic limits, Yaged [63] and Zadeh [64] have studied a problem in which there is a sequence of traffic forecasts for several time periods in the future, and the objective is to find an installation plan that minimizes the present value of all future installations, discounted according to some given interest rate.

7.5. Improving the lower bound for more general link cost functions

Sometimes a more realistic cost of installing links between switches u and v must take into account a *fixed cost* $\Gamma_{\{u,v\}}$. This represents a cost of installing links between u and v, in either direction, that is independent of the number of such links to be installed. For example, this could be the cost of digging a trench and installing an underground conduit where none had existed before.

In this case, the cost of the network can be written

$$\sum_{u,v \in N} \Gamma_{\{u,v\}} \left(cap(u,v) + cap(v,u) > 0 \right) + \sum_{u,v \in N} \gamma(u,v) cap(u,v) \tag{7.5}$$

where the notation $(cap(u,v) + cap(v,u) > 0)$ is a conditional expression. It is equal to one if the condition is true, and 0 if the condition is false.

We have already studied how to find a lower bound for the second term of (7.5). We can independently find a lower bound for the first term and add these two bounds together to get a lower bound for the total cost. The computational problem of finding the largest such lower bound possible is:

Fixed charge lower bound

INSTANCE: A set of switches N. For each unordered pair of distinct switches u, v, a fixed cost $\Gamma_{\{u,v\}}$. Traffic limits $\mathcal{T}$.

SOLUTION: A set of undirected edges T between switches N. If the traffic limits $\mathcal{T}$ allow connections from u to v, then there must be a path from u to v in the graph $G = (N, T)$.

SOLUTION COST:

$$\sum_{\{u,v\} \in T} \Gamma_{\{u,v\}}$$

OBJECT: Find a solution with minimum cost.

This is a lower bound on the fixed charge portion of the cost, because any nonblocking network must have a path with positive capacity links between switches that can have connections to one another.

Note that for many problems, all pairs of switches can have connections to one another, so the restriction on T is equivalent to "$G = (N, T)$ is connected", and the problem above is that of finding a minimum spanning tree.

For instances in which not all switch pairs may have connections to one another, define an undirected graph $C = (N, E)$ such that $\{u, v\} \in E$ if and only if either u can connect to v, or v can connect to u. Let the sets $N_1, \ldots, N_k$ be the switches of the connected components of C. Note that sets N_i that contain a single switch u are

isolated switches in C. Thus they may not connect to any other switch, and so need not be part of any nonblocking network, i.e., they are tandem switches.

If the sets $N_1, \ldots, N_k$ contain only one set, say N_1, with more than one switch, and the Γ values satisfy the triangle inequality, then the solution to the computational problem is to find a minimum spanning tree of the switches in N_1, and ignore the rest. If the Γ values do not satisfy the triangle inequality, then the solution is to find a minimum Steiner tree for the switches in N_1.

Since finding a minimum Steiner tree is NP-hard, we may use any approximation algorithm that always finds a solution that costs at most R times more than optimal. Then a lower bound for the fixed charge portion of our network configuration problem is the cost of the tree found divided by R. The algorithm with the lowest value of R known to the author is due to Berman and Ramaiyer [9]. They give a family of polynomial time approximation algorithms, where smaller values of R give larger running times. $R = 16/9$ can be achieved in low degree polynomial time.

In the most general case, the sets $N_1, \ldots, N_k$ are an arbitrary partition of the switches N. Then our lower bound problem is that of finding a minimum generalized Steiner tree [2, 36]. Agrawal, Klein, and Ravi found the first approximation algorithm for this problem with $R = 2$.

For a problem in the Euclidean plane where additional switches are allowed, the Γ values are proportional to the Euclidean distance between the nodes, and all of the sets $N_1, \ldots, N_k$ except N_1 are singleton sets, the lower bound is the cost of a minimum Euclidean Steiner tree on the points N_1. In this case, the minimum spanning tree algorithm has $R = 2/\sqrt{3}$ [19, 20].

This method of splitting up the terms in the cost function and finding a lower bound for them separately is similar to a lower bound found by Gilbert [34].

8. OTHER ROUTING ALGORITHMS

8.1. Alternate path routing

8.1.1. Definition

The alternate path routing algorithm AP is a straightforward generalization of fixed path routing. For each ordered pair of switches u, v, there are two (possibly identical) paths from u to v, a primary path $\pi_1(u, v)$ and a secondary path $\pi_2(u, v)$. When a point-to-point request (u, v, ρ) is made, the path $\pi_1(u, v)$ is checked to see if every link has at least ρ available bandwidth. If so, then that path is used to connect the switches. If not, then the path $\pi_2(u, v)$ is checked. If it has enough bandwidth available, then it is used. Otherwise, the connection request blocks.

8.1.2. Link dimensioning with flat traffic limits

In this section we show that the link dimensioning problem can be solved in exponential time under the following conditions. Path tables π_1 and π_2 are given and satisfy certain restrictions (condition (8.1) below), link costs are nondecreasing and concave functions of capacity, traffic limits are flat, and all connection requests must be be point-to-point with rate 1. Our results show that under these conditions, the cheapest solutions are those for which every switch pair u, v either always uses $\pi_1(u, v)$ but never $\pi_2(u, v)$, or vice versa. Thus an alternate path routing solution always costs at least as much as a fixed path routing solution.

The restriction on the primary and secondary paths we consider is:

$$(\forall u, v \in N) \text{ (no link in } \pi_1(u,v) - \pi_2(u,v) \text{ is contained in any path except } \pi_1(u,v)) \quad (8.1)$$

A simple path is a path that visits each switch at most once. It is not a good idea for a path to be anything other than simple, as it would only waste network resources. If $\pi_1(u,v) \subseteq \pi_2(u,v)$ or $\pi_2(u,v) \subseteq \pi_1(u,v)$, then either they are the same, or at least one of them is not a simple path. Thus, if all paths are simple, then either $\pi_1(u,v) = \pi_2(u,v)$, or both paths contain links that the other does not.

When condition (8.1) holds, it is useful to define:

$$\begin{aligned} L' &= \bigcup_{u,v \in N} (\pi_1(u,v) \cup \pi_2(u,v)) \\ p(u,v) &= \pi_1(u,v) - \pi_2(u,v) \\ s(u,v) &= \pi_2(u,v) - \pi_1(u,v) \\ \mathcal{P} &= \bigcup_{u,v \in N} p(u,v) \\ \overline{\mathcal{P}} &= L' - \mathcal{P} \end{aligned}$$

L' is the set of all links used in some path, so any link not in L' should have 0 capacity. $p(u,v)$ is the set of "primary only" links from u to v, and $s(u,v)$ is the set of "secondary only" links.

The algorithm found for the link dimensioning problem has been proven to work when link costs are nondecreasing and concave functions of capacity. 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 the straight line lies on or below f. Let *maxflow*(n,m) be the time required to solve a maximum flow problem on a graph with n vertices and m edges.

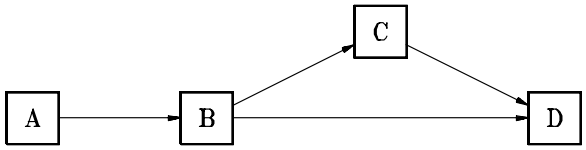

Figure 8.1: A simple instance with AP routing

Theorem 8.1 *The link dimensioning problem with restrictions described above can be solved in* $O(2^d \cdot m \cdot \text{maxflow}(n, U))$ *time, where* $m = \left|\overline{\mathcal{P}}\right|$, n *is the number of switches,* U *is a parameter depending on the paths that is at most* $n^2 + n$, *and* d *is a parameter depending on the paths that is at most the number of* u, v *pairs such that* $p(u, v) \neq \emptyset$.

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

First, a couple of trivial results. If we restrict $\pi_1(u, v) = \pi_2(u, v)$ for all $u, v \in N$, then the alternate path routing algorithm AP behaves exactly like fixed path routing, and all of the results found for fixed path routing apply.

If we apply the link dimensioning algorithm in Section 4.3 with fixed paths given by π_1, then we find link capacities for which the network is nonblocking 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 always will. This idea also works for any routing algorithm like AP that first checks a single path to see if it has enough capacity, and returns it if so. However, this may not be the cheapest solution to the link dimensioning problem for such routing algorithms.

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 8.1, 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

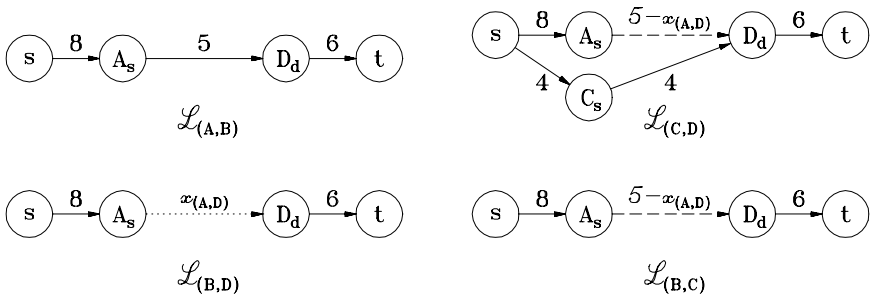

Figure 8.2: $\mathcal{L}_l$ graphs for the simple AP routing instance

D can be the destination of connections. Let $\pi_1(A,D) = ABD$, $\pi_2(A,D) = ABCD$, and $\pi_1(C,D) = \pi_2(C,D) = CD$. The other paths do not matter because they can not be used.

Let us generalize the definition of $\mathcal{L}_l$, originally given in Section 4.3, for alternate path routing. Let $\mathcal{L}_l = (V, E_1 \cup E_2 \cup E_{12}, cap)$, where

$$\begin{aligned} V &= \{s,t\} \cup \{u_s, u_d : u \in N\} \\ E_{12} &= \{(s, u_s, \alpha(u)),\ (u_d, t, \omega(u)) : u \in N\} \cup \\ &\quad \{(u_s, v_d, \mu(u,v)) : l \in \pi_1(u,v) \cap \pi_2(u,v)\} \\ E_1 &= \{(u_s, v_d, x_{u,v}) : l \in p(u,v)\} \\ E_2 &= \{(u_s, v_d, \mu(u,v) - x_{u,v}) : l \in s(u,v)\} \end{aligned} \tag{8.2}$$

The graphs $\mathcal{L}_{(A,B)}$, $\mathcal{L}_{(B,C)}$, $\mathcal{L}_{(B,D)}$, and $\mathcal{L}_{(C,D)}$ are shown in Figure 8.2. All edges that have 0 flow in all feasible flows have been omitted. Edges in E_1 are shown as dotted lines, edges in E_2 are shown as dashed lines, and edges in E_{12} are shown as solid lines. The dotted edge (A_s, D_d) in $\mathcal{L}_{(B,D)}$ signifies that link (B,D) is in the primary path from A to D, but not in the secondary path. The dashed edge (A_s, D_d) in $\mathcal{L}_{(B,C)}$ and $\mathcal{L}_{(C,D)}$ 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 edge (A_s, D_d) in $\mathcal{L}_{(A,B)}$ signifies that link (A,B) is in both the primary and secondary path from A to D. Similarly for the solid edge (C_s, D_d) in $\mathcal{L}_{(C,D)}$.

The edge capacities require some explaining. For solid edges $(u_s, v_d) \in \mathcal{L}_{(x,y)}$, we know that whatever path the routing algorithm AP finds, that path *must* contain the link (x, y). Therefore, if such a graph $\mathcal{L}_{(x,y)}$ contains all solid edges, as $\mathcal{L}_{(A,B)}$ does in this example, we may find the maximum flow on $\mathcal{L}_{(x,y)}$ to obtain the maximum usage of link (x, y). All $\mathcal{L}_l$ graphs are like this when we use fixed path routing.

When a graph $\mathcal{L}_{(x,y)}$ contains dotted or dashed edges, however, it introduces dependencies between link capacities. In this simple instance, assign a capacity $x_{(A,D)}$, $0 \leq x_{(A,D)} \leq \mu(A, D)$, to link (B, D), and then 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 rate 1 connections from A to D. As long as no more than $x_{(A,D)}$ requests are using link (B, D), the routing algorithm AP will find that there is enough capacity available on the primary path ABD, and the connections will follow that path. However, as soon as $x_{(A,D)}$ requests are using link (B, D) and an additional request comes in (this can only occur if $x_{(A,D)} < \mu(A, D) = 5$), AP will find that ABD cannot be used, and so it returns the secondary path $ABCD$ instead. We say that the request *overflows* onto its secondary path in such a case.

The quantity $over(x_{(A,D)})$ 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 $x_{(A,D)}$. When all requests have rate 1, it should be easy to see that $over(x_{(A,D)}) = \mu(A, D) - x_{(A,D)}$.

In $\mathcal{L}_{(B,D)}$, we assign the edge (A_s, D_d) the capacity $x_{(A,D)}$ 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 edge capacity "means the same thing" as it did in the fixed path case, but now $x_{(A,D)}$ can be any quantity at most $\mu(A, D)$. Similarly, we assign the edge (A_s, D_d) in $\mathcal{L}_{(B,C)}$ the capacity $\mu(A, D) - x_{(A,D)} = 5 - x_{(A,D)}$ to denote

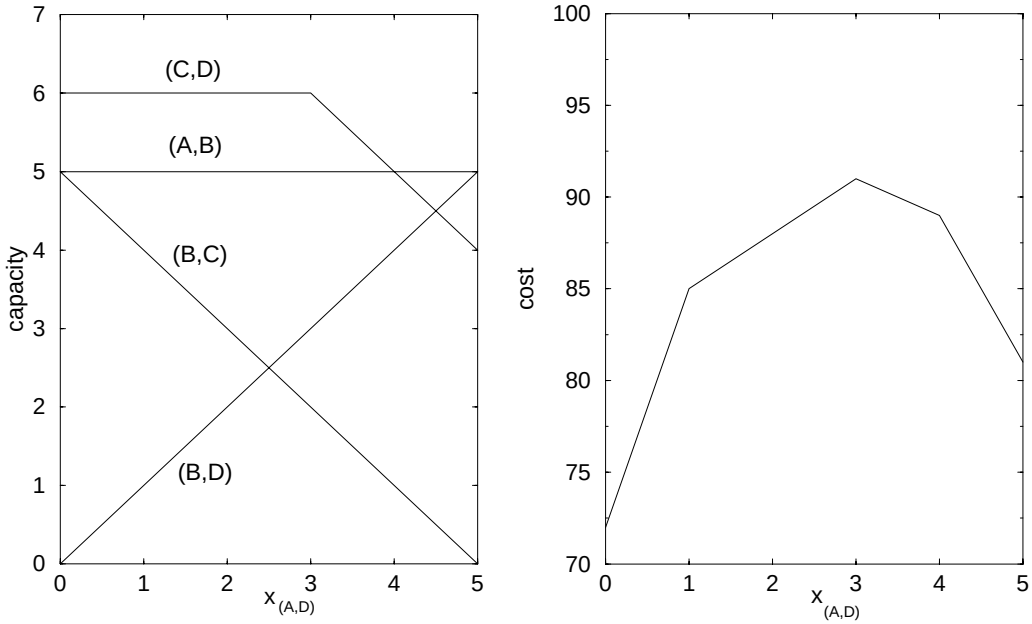

Figure 8.3: Maximum link usages and total network cost as a function of $x_{(A,D)} = cap(B,D)$

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 $x_{(A,D)}$, where $0 \leq x_{(A,D)} \leq \mu(A,D)$, 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 $x_{(A,D)}$.

For example, let the cost functions of each link in the example be $\Gamma(u,v)+\gamma(u,v)\cdot cap$ if the capacity $cap > 0$, and 0 otherwise (these functions are nondecreasing and concave when $\Gamma(u,v), \gamma(u,v) \geq 0$). Let $\Gamma(A,B) = 0$, $\gamma(A,B) = 3$, $\Gamma(B,C) = 6$, $\gamma(B,C) = 4$, $\Gamma(B,D) = 10$, $\gamma(B,D) = 7$, $\Gamma(C,D) = 1$, and $\gamma(C,D) = 5$. Then the link capacities are given as functions of $x_{(A,D)}$ in the left plot of Figure 8.3, and the total cost of the network is given as a function of $x_{(A,D)}$ in the right plot of Figure 8.3.

Note that this function is minimized when $x_{(A,D)} = 0$, one of the extreme values that $x_{(A,D)}$ may take. The other extreme value is 5. $x_{(A,D)}$ may be larger than 5, but if

so, any capacity over 5 will never be used. We show that under the conditions given at the beginning of this section, the minimum cost network is always found by assigning extreme values to the links in $\mathcal{P}$.

Lemma 8.2 *Suppose we have primary and secondary paths that satisfy condition (8.1). For all links $l \in \mathcal{P}$, $\mathcal{L}_l$ contains exactly one edge of the form (u_s, v_d), and it is in E_1 (i.e., a dotted edge in the figures). For all other links $l \in \overline{\mathcal{P}}$, $\mathcal{L}_l$ has $E_1 = \emptyset$.*

Proof: Follows directly from the modified definition of $\mathcal{L}_l$ (8.2) given earlier in this section. ■

Note that it is possible for $|p(u,v)| > 1$ to hold for some $u, v \in N$. 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 that 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 $\mathcal{D} = \{(u,v) : p(u,v) \neq \emptyset\}$.

Lemma 8.3 *For all $(u,v) \in \mathcal{D}$, let every link in $p(u,v)$ have capacity $x_{(u,v)}$. Then for every link $l \in \overline{\mathcal{P}}$, $\lambda^*(l)$ can be found in $O(\text{maxflow}(n,U))$ 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 link dimensioning problem when fixed path routing is used. The only difference is that some of the point-to-point limits μ have been reduced for links in $\overline{\mathcal{P}}$. Thus $\lambda^*(l)$ is equal to the value of a maximum flow in $\mathcal{L}_l$. ■

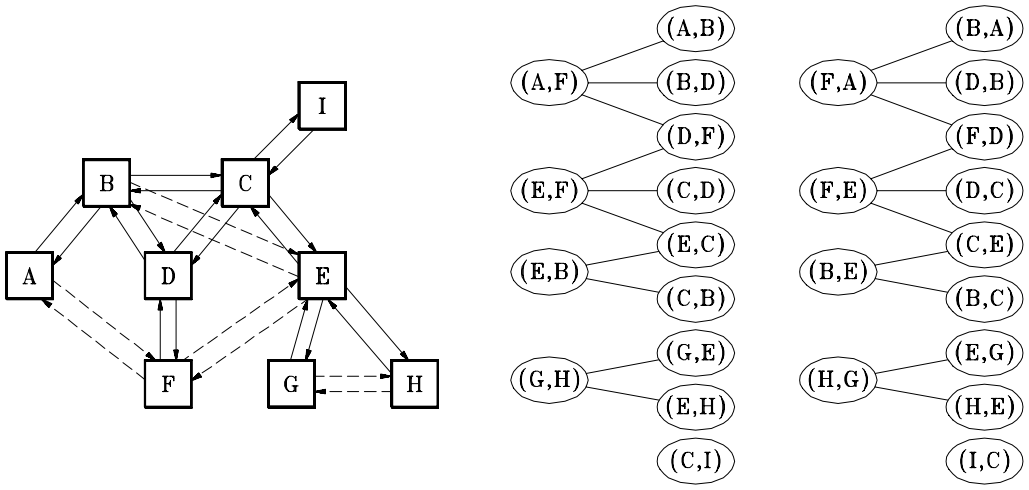

Figure 8.4: Alternate path network instance and its dependency graph

Lemma 8.3 tells us that $\lambda^*(l)$ of each link $l \in \overline{\mathcal{P}}$ is functionally dependent on a subset of $X = \{x_{(u,v)} : (u,v) \in \mathcal{D}\}$. For each $l \in \overline{\mathcal{P}}$, denote this subset of X by X_l, and make the dependence of $\lambda^*(l)$ on X_l explicit by writing it $\lambda^*(l, X_l)$.

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

$$\sum_{(u,v)\in\mathcal{D}} \sum_{l\in p(u,v)} cost_l(x_{(u,v)}) + \sum_{l\in\overline{\mathcal{P}}} cost_l(\lambda^*(l, X_l)) \tag{8.3}$$

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 represents the functional dependencies between link capacities. The graph is bipartite, with one partition containing a vertex for each $(u,v) \in \mathcal{D}$, and the other containing a vertex for each link $l \in \overline{\mathcal{P}}$. There is an edge between two vertices (u,v) and l if $\lambda^*(l, X_l)$ depends on $x_{(u,v)}$, i.e., $l \in s(u,v)$.

For example, in the network of Figure 8.4, the links drawn with dashed lines are those in the set $\mathcal{P}$. Let $\pi_1(A,F) = AF$, $\pi_2(A,F) = ABDF$, $\pi_1(A,E) = ABE$, $\pi_2(A,E) = ABCE$, $\pi_1(F,E) = FE$, $\pi_2(F,E) = FDCE$, $\pi_1(G,H) = GH$, and $\pi_2(G,H) = GEH$.

For the opposite directions, reverse the paths above (e.g., $\pi_1(F, A) = FA$ and $\pi_2(F, A) = FDBA$). For all other switch pairs $u, v \in N$, $\pi_1(u, v) = \pi_2(u, v)$ and all links in these paths use only solid links. These paths satisfy condition (8.1). The dependency graph for this instance is shown in Figure 8.4. For example, the capacities of links (E, C) and (C, B) depend on $x_{(E,A)}$ because $s(E, A) = \{(E, C), (C, B)\}$.

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

$$\sum_{(u,v)\in\mathcal{D}'} \sum_{l\in p(u,v)} cost_l(x_{(u,v)}) + \sum_{l\in\overline{\mathcal{P}}'} cost_l(\lambda^*(l, X_l)) \tag{8.4}$$

can be minimized by choosing values for the variables $\{x_{(u,v)} : (u, v) \in \mathcal{D}'\}$ 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 8.4, we may minimize the total cost of links (G, H), (G, E), and (E, H) 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 (C, I) and (I, C) depend on no other link capacities, and their maximum usages may be determined exactly as in Section 4.3 because the graphs $\mathcal{L}_{(C,I)}$ and $\mathcal{L}_{(I,C)}$ contain all solid edges. In this example, this technique of finding independent subproblems reduced the exponent d from $|\mathcal{D}| = 8$ to 3, the size of the largest set $\mathcal{D}'$ induced by the components of the dependency graph.

Lemma 8.4 *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 edge with a capacity given by the variable x.*

All other edges $e \in E$ have a fixed capacity cap(e). 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 [21] 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 edges $(u, v) \in E$ for which $u \in X$ and $v \in \overline{X}$. The capacity of a cut is the sum of the capacities of the edges within it. Let $\mathcal{C}$ be the set of all $s - t$ cuts.

Some cuts contain the edge 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 that do not contain the edge 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 8.3 are of the form $m(5 - x)$.

Lemma 8.5 *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 8.4, then cost$(m(x))$ is a nondecreasing and concave function of x.*

Proof: Let $m(x)$ have the form described in Lemma 8.4 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 8.6 *$cost(m(\mu - x))$ is a nonincreasing and concave function of x on the interval $[0, \mu]$.*

Fact 8.7 *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 8.8 *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 [8, Thm. 3.4.6, p. 99], for example.

Lemma 8.9 *Let $\mathcal{D}'$ and $\overline{\mathcal{P}}'$ be all of the vertices 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 $O(2^{|\mathcal{D}'|} \cdot \left|\overline{\mathcal{P}}'\right| \cdot \text{maxflow}(n, U))$ time, where n is the number of switches in the network and $U \le n^2 + n$.*

Proof: Let l be any link in $\overline{\mathcal{P}}'$. For all $(u, v) \in \mathcal{D}'$, either $\mathcal{L}_l$ has a dashed edge (u_s, v_d) with capacity $\mu(u, v) - x_{(u,v)}$, or it has no edge (u_s, v_d). Any other (u_s, v_d) edges in $\mathcal{L}_l$ must be solid (with a fixed capacity). If $(u, v) \in \mathcal{D}'$ and (u_s, v_d) is in $\mathcal{L}_l$, and if we let $x_{(u,v)}$ change while leaving the other x values fixed, Corollary 8.6 tell us that the cost of link l as a function of $x_{(u,v)}$ is nonincreasing and concave on $[0, \mu(u, v)]$. If $(u, v) \in \mathcal{D}'$ and (u_s, v_d) is not in $\mathcal{L}_l$, then the maximum usage of l 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 8.7 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 8.8 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 8.3, 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 8.1 follows directly from Lemma 8.9 by adding the running times for each component of the dependency graph. This theorem implies that alternate path routing with these constraints gives no lower cost than fixed path routing.

8.2. Shortest available path routing

The fixed and alternate path routing algorithms share the following property: they can block even when a path from the source to the destination with enough available capacity exists. In this section we study a routing algorithm that always find a path if one exists, but gives preference to those with fewer links. We call it shortest available path routing, denoted *SAP*.

Given a request $r = (u,v,\rho)$ and a network state $\mathcal{S}$, *SAP* returns a path with ρ available capacity that has the fewest number of links among all such paths. If there

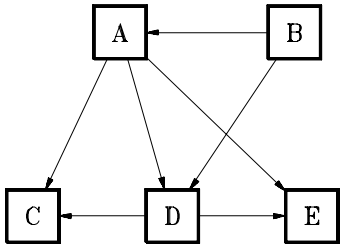

Figure 8.5: Network to demonstrate *SAP* routing algorithm

are none, then the request blocks. To make *SAP*'s choice of path deterministic, we can think of it as examining all paths with the same length in some fixed order, and returning the first one with enough available capacity. For example, if all switches are given unique numbers 1 to n, and a path is represented by the sequence of switches that it visits, then the paths may be examined in lexicographic order.

As a demonstration of the *SAP* routing algorithm, consider the network of Figure 8.5. All links have capacity 1, and all requests and connections in this demonstration have rate 1. Initially, there are no connections in the state of the network. The request $(A, C, 1)$ uses the length 1 path AC. If the same request is made again, it will use the length 2 path ADC this time. For the request $(B, E, 1)$, there are two shortest paths, BAE and BDE. If we use the lexicographic ordering rule to decide between them, *SAP* returns the path BAE. If the same request is made again, BDE must be used this time. Any further request blocks, because all links are now fully used.

Now we show that the network analysis problem with flat α, ω-bounded traffic limits using *SAP* routing is NP-hard in the strong sense [28, Section 4.2]. That is, it is NP-hard even when the magnitudes of numbers in the problem instances (i.e., the link capacities and traffic limits) are "small", bounded by a polynomial in the number of switches and links. We assume that requests may only have rates that are a multiple of some minimum rate $b = 1/b'$, for some positive integer b'. The definition of the network

analysis problem is given again here, stated more specifically than it was in Section 2.3. For brevity, we call this problem BLOCKING NETWORK.

BLOCKING NETWORK

INSTANCE: A network $\mathcal{N} = (N, L, cap)$. Flat α,ω-bounded traffic limits $\mathcal{T} = (\alpha, \omega)$.

QUESTION: Is there a sequence of point-to-point requests compatible with $\mathcal{T}$, each with a rate that is a multiple of b, such that at least one of the requests blocks?

It is not clear to us whether this problem is in NP, because the length of a sequence of requests that block is not obviously bounded by a polynomial in the length of the instance, even if we allow multiple consecutive requests between the same pair of switches to be written as a single request with the same total rate.

Theorem 8.10 *BLOCKING NETWORK is* NP*-hard in the strong sense.*

Proof: The proof is done by giving a pseudo-polynomial transformation from the problem 3-PARTITION, which is NP-complete in the strong sense.

3-PARTITION

INSTANCE: A collection of $3m$ positive integers $A = \{a_1, \ldots, a_{3m}\}$, and a bound B, such that each integer a_i satisfies $B/4 < a_i < B/2$ and such that $\sum_{i=1}^{3m} a_i = mB$.

QUESTION: Is there a partition of the collection into disjoint sets $A_1, \ldots, A_m$ such that the total of the values in each set is exactly B? (Note that the restrictions on the values in A imply that every A_i must contain *exactly* three elements.)

The transformed instance of the network analysis problem is a network $\mathcal{N} = (N, L, cap)$ with traffic limits $\mathcal{T} = (\alpha, \omega)$. The set of switches and links are given below, where a switch u with source termination limit $\alpha(u)$ and destination termination limit $\omega(u)$ is given by the triple $(u, \alpha(u), \omega(u))$, and a link from switch u to v with capacity cap is given by the triple (u, v, cap).

$$\begin{aligned}
N &= \{(x, b, 0), (y, 0, b)\} \\
&\quad \cup \{(s_i, B, 0) : 1 \le i \le m\} \\
&\quad \cup \{(u_{i,j}, 0, 0) : 0 \le i \le m, 1 \le j \le 3m\} \\
&\quad \cup \{(t_{i,j}, 0, a_j) : 1 \le i \le m, 1 \le j \le 3m\} \\
L &= \{(x, u_{0,j}, 1) : 1 \le j \le 3m\} \\
&\quad \cup \{(u_{i-1,j}, u_{i,j}, a_j) : 1 \le i \le m, 1 \le j \le 3m\} \\
&\quad \cup \{(u_{m,j}, y, 1) : 1 \le j \le 3m\} \\
&\quad \cup \{(s_i, u_{i-1,j}, a_j) : 1 \le i \le m, 1 \le j \le 3m\} \\
&\quad \cup \{(u_{i,j}, t_{i,j}, a_j) : 1 \le i \le m, 1 \le j \le 3m\} \\
&\quad \cup \{(s_i, t_{l,j}, a_j) : 1 \le i \le m, 1 \le j \le 3m, 1 \le l \le m, l \ne i\} \\
&\quad \cup \{(s_i, y, 1) : 1 \le i \le m\} \\
&\quad \cup \{(x, t_{i,j}, 1) : 1 \le i \le m, 1 \le j \le 3m\}
\end{aligned} \tag{8.5}$$

It should be clear that this transformation can be performed in polynomial time.

As an example of this transformation, consider the 3-PARTITION instance $A = \{11, 11, 12, 13, 14, 19\}$ with $m = 2$ and $B = 40$. This instance has no solution, and neither should the transformed instance of BLOCKING NETWORK, shown in Figure 8.6. All links from the last three lines of L's definition have not been shown, to avoid cluttering the figure.

Given a solution $A_1, \ldots, A_m$ to the 3-PARTITION instance, we can construct a sequence of requests that blocks in the transformed BLOCKING NETWORK instance. For each $a_j \in A_i$, make a sequence of requests from s_i to $t_{i,j}$ with total rate a_j. This completely uses the links $(u_{i-1,j}, u_{i,j})$ for all three $a_j \in A_i$. The final request is (x, y, b).

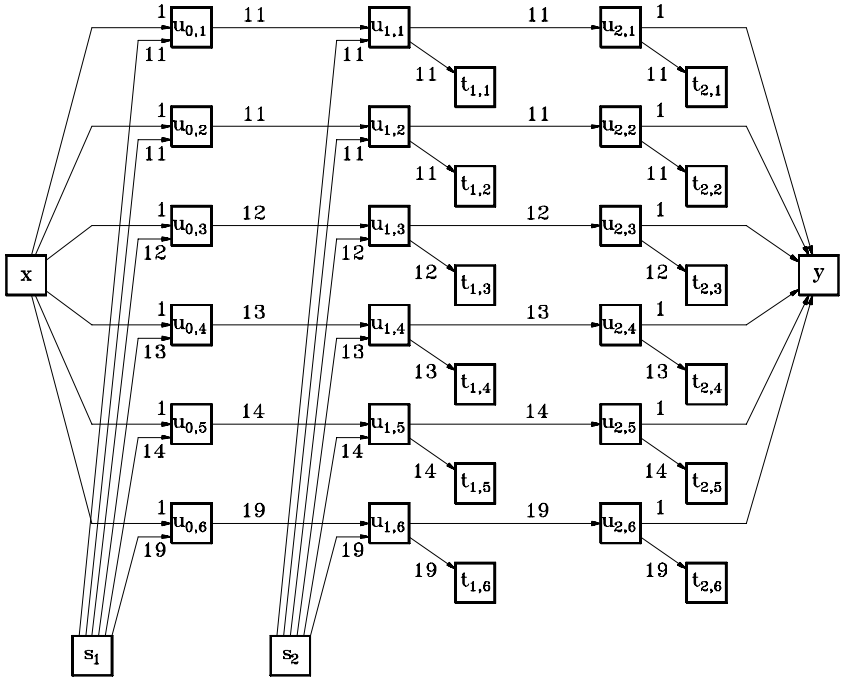

Figure 8.6: An example of the transformation from 3-PARTITION to BLOCKING NETWORK

Table 8.1: Possible requests and connections in a BLOCKING NETWORK instance

Request	Possible connections	"Important" links used
$(s_i, t_{l,j}, \rho), l \neq i$	$s_i t_{l,j}$	—
(s_i, y, ρ)	$s_i y$	—
$(x, t_{l,j}, \rho)$	$x t_{l,j}$	—
$(s_i, t_{i,j}, \rho)$	$s_i u_{i-1,j} u_{i,j} t_{i,j}$	$(u_{i-1,j}, u_{i,j})$
(x, y, ρ)	$x u_{0,j} u_{1,j} \ldots u_{m,j} y$ for some $j, 1 \leq j \leq 3m$	all in the path

This request blocks, because all paths from x to y have at least one completely used link. It is easy to verify that this sequence of requests is compatible with the traffic limits.

Given a sequence of requests $r_1, \ldots, r_k$ that blocks, we must now show that there is a solution $A_1, \ldots, A_m$ to the 3-PARTITION instance. It is helpful to consider every possible connection request and the paths that will be used to satisfy them. These are given in Table 8.1. The first three lines of the table describe connection requests that are always satisfied by a direct link, because the link between the switches u, v has capacity at least $\min\{\alpha(u), \omega(v)\}$.

The request $(s_i, t_{i,j}, \rho)$ in the fourth line of the table has only one possible path that can satisfy it. Such a request can never block, because every link on the path has capacity at least $\min\{\alpha(s_i), \omega(t_{i,j})\} = a_j$.

The only request that can possibly block is (x, y, ρ). Because $\alpha(x) = b$, the minimum rate of any request, we must have $\rho = b$. Since all request rates are multiples of b, and 1 is an integer multiple of b, the only way for a request (x, y, b) to block is for each path from x to y to have at least one completely used link. (If a link is not completely used, then it has at least b capacity available.)

The "important" links in Table 8.1 are the links on paths from x to y. Only requests that use important links can lead to blocking.

Let $r_1, \ldots, r_k$ be a sequence of requests to add and drop connections such that the first request that blocks is r_k. We have learned that only a request (x, y, b) can block, so $r_k = (x, y, b)$. $\alpha(x) = \omega(y) = b$, the minimum connection rate, so any previous request to add a connection involving x or y must have been dropped later in the sequence. We ignore these requests. We also ignore any requests of the form $(s_i, t_{l,j}, \rho)$, where $l \neq i$, because such requests do not use any important links. It is acceptable to ignore all of these connection requests, because none of them affect future routing decisions.

Thus all of the requests in the sequence $r_1, \ldots r_{k-1}$ may as well be add requests of the form $(s_i, t_{i,j}, \rho)$. Let $L_i = \{j : \lambda((u_{i-1,j}, u_{i,j}), \mathcal{S}) = a_j\}$, where $\mathcal{S}$ is the state of the network immediately before request $r_k = (x, y, b)$ arrives. That is, L_i is the set of j values such that the links $(u_{i-1,j}, u_{i,j})$ are completely used when the request that blocks arrives.

Because $\alpha(s_i) = B$ for all i and $B/4 < a_j$ for all j, each L_i must contain at most three elements. Since the request (x, y, b) blocks in state $\mathcal{S}$, it must be that every path from x to y has at least one completely used link. This is equivalent to saying that the union of the sets L_i contains $\{1, \ldots, 3m\}$. Since each L_i contains at most three elements, the L_i must be disjoint sets that partition $\{1, \ldots, 3m\}$.

Since $\alpha(s_i) = B$ for all i, it must be that $\sum_{j \in L_i} a_j \leq B$ for all i. The total of all a_j values is mB, so we must have $\sum_{j \in L_i} a_j = B$ for all i.

Therefore a solution to the original instance of 3-PARTITION is $A_i = \{a_j : j \in L_i\}$.

■

This NP-hardness result for the network analysis problem with *SAP* routing does not necessarily imply that the link dimensioning or network configuration problems with

SAP routing are NP-hard. In fact, if a star network is optimum among all solutions for flat α, ω-bounded traffic limits with $\alpha(N) = \omega(N)$ and for all routing algorithms (or at least for *SAP* routing), then the network configuration problem can be solved in $O(n^2)$ time by the algorithm at the end of Section 3.2. However, this NP-hardness result does show that there are some cases when it is difficult to determine whether a given network is nonblocking.

A similar result can be shown for the routing algorithm *hops at most* K, in which all paths with at most K links are tried in some fixed order.

Note that network analysis in a tree network is easily solved for any routing algorithm, because any routing algorithm reduces to fixed path routing in a tree.

9. CONCLUSIONS

In Chapter 3, we presented termination limit bounded, or α, ω-bounded, traffic limits. We believe that these are a reasonable model of the offered traffic for small local area networks, and for larger networks in which the traffic patterns are highly variable. They are also easy for a network manager to understand and estimate. We showed that the network configuration problem with such traffic limits is MAX SNP-hard, leading us to consider algorithms that find approximate solutions. We presented the perhaps surprising result that a star network is very close to optimal, if not exactly optimal. This result was verified through experiments on randomly generated instances of the network configuration problem, and through analytical results. We proved that star networks are cheapest among all tree networks when $\alpha(N) = \omega(N)$, and we believe that they are cheapest among all networks in that case. If so, then the network configuration problem can be efficiently and exactly solved for such instances.

We proved that star networks are never much more expensive than the cheapest tree when $\alpha(N)$ is close to $\omega(N)$, and gave a class of instances that demonstrate that a non-star tree network can be slightly cheaper than any star network. We believe that many realistic network configuration problems would have $\alpha(N)$ close to $\omega(N)$. A network that could have $\omega(N)$ much larger than $\alpha(N)$ is a video distribution network, e.g., a cable television network with residential subscribers. It would be interesting to see if our results can be extended to such instances. For example, the lower bound is currently determined by finding a set of point-to-point connection requests that are expensive

to satisfy. In a network with $\omega(N) \gg \alpha(N)$, the lower bound could be improved by allowing multipoint connection requests in the set as well.

In Chapter 4, we further restricted the offered traffic by allowing point-to-point limits to be specified. Our experimental results suggest that a star network is still close to optimal unless the point-to-point limits are quite small relative to the termination limits. It requires significantly more data about network usage for a network manager to specify such restrictions accurately, and it may not help reduce the cost of a nonblocking network even then.

In Chapter 5, we extended the flat traffic limits to allow one to specify clusters of switches that have high traffic among themselves, but less to other parts of the network. We believe that these are useful for many large local area networks, and possibly larger networks as well. Like termination bounded limits, we believe that they are simple for a network manager to understand and estimate, and for networks with many managers, it decomposes the traffic specification in a natural way. We showed through experiments that hierarchical star networks are often within 10% of optimal for randomly generated instances.

In Chapter 6, we showed that if a network using fixed path routing is nonblocking for point-to-point connections, then it is also nonblocking for multipoint connections, given that the switches have multicasting capabilities. This result is useful not only in this work, but in any network design research that specifies traffic in a way that is a special case of our traffic limits. This includes all of the local access network design problems, and all of the problems that specify a worst case traffic matrix, cited in the introduction.

Chapter 7 presented some extensions to the previous results, including handling link fragmentation in ATM networks, allowing additional switches other than those specified

to be added, and improving the lower bound for more general link cost functions. The most interesting avenue of further work here involves creating the dream software package of all network managers. Such a package has been created at GTE for the design of local access telephone networks [47], and there are many existing packages for various kinds of data networks [62].

In Chapter 8, we presented some results for routing algorithms other than fixed path routing. We saw that the cheapest solution to a restricted link dimensioning problem using alternate path routing reduces to a solution using fixed path routing. Hence the additional flexibility of alternate path routing does not help reduce the cost of the network. We also saw that the additional flexibility of shortest available path routing greatly increases the complexity of simply determining whether a given network is nonblocking. We believe that fixed path routing produces the cheapest network configurations for a wide class of instances with the worst case model of traffic presented here.

One reason why a network manager would not want to install a tree network is lack of redundancy. There has been other network design research that focuses on making networks tolerant to link or node failures [1, 15, 26, 39, 40, 56], but all of the work known to us either ignores the exact link capacities needed, requiring only a two-connected network, or specifies point-to-point traffic with a traffic matrix. It would be interesting to extend such work for more general kinds of traffic.

10. ACKNOWLEDGMENTS

Thanks to Chuck Cranor for collecting and updating the LaTeX thesis style files, and to James Sterbenz for making the little workstations used in Figure 1.1 (and many others). Thanks to Greg Peterson and Rex Hill for many entertaining lunch discussions on topics that had absolutely nothing whatsoever to do with school. Thanks to Ellen Zegura and Dakang Wu for being great office mates who could tolerate my particular form of (dis)organization.

Thanks to the Computer Science office staff, particularly Jean Groethe for helping me use the copier to make slides at the last minute on more occasions than I care to admit, and Myrna Harbison for her sense of humor.

Thanks to my father, John Fingerhut, for buying an Apple][+ computer for the family in Christmas of 1981. I spent many happy hours with that machine, and will always remember it fondly.

Thanks to my mother, Shirley Fingerhut, for teaching me the rules of chess when I was a boy, and then letting me beat her at it. Success is such a great way to build confidence.

Thanks to my brother Matt for saving my life once. One day he may regret that he did not force me to agree to give him 50% of all my future earnings.

Thanks to my sister Amy for being brutally honest at all times.

Thanks to my wife Tonya for understanding that one may not see one's spouse as often as one wishes during the last months before the dissertation is complete.

Thanks to George Varghese for all of our stimulating discussions, and for showing me lots of great ideas on how to teach a course on network protocols.

But most of all, I would like to thank my advisor Jonathan Turner, to whom this work is dedicated. You were patient with me, and helped me to start my research when I did not know how. Research and proofs seemed like magic being performed behind a curtain. You showed me that there was no magic, but plenty of trial, error, and practice. While others have helped, you have been the best teacher – by example.

A. BIBLIOGRAPHY

[1] Yogesh K. Agarwal. An algorithm for designing survivable networks. *AT&T Technical Journal*, 68(3):64–76, May/June 1989.

[2] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. In *Proc. 23rd ACM Symp. Theory of Computing*, pages 134–144, 1991.

[3] Dana Angluin and Leslie G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *Journal of Computer and System Sciences*, 18:155–193, 1979.

[4] Ami Arbel. *Exploring Interior-Point Linear Programming: Algorithms and Software*. Foundations of Computing. MIT Press, November 1993.

[5] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. In *Proc. 33rd and Symp. Foundations of Comp. Sci.*, pages 14–23. IEEE, 1992.

[6] G. R. Ash, Richard H. Cardwell, and R. P. Murray. Design and optimization of networks with dynamic routing. *Bell System Technical Journal*, 60(8):1787–1820, October 1981.

[7] Anantaram Balakrishnan, Thomas L. Magnanti, Alexander Shulman, and Richard T. Wong. Models for planning capacity expansion in local access telecommunication networks. In J. MacGregor Smith and Pawel Winter, editors, *Topological Network Design (Annals of Operations Research 33)*, pages 239–284. J. C. Baltzer AG, 1991.

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

[9] 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.

[10] Marshall Bern and Paul Plassman. The Steiner problem with edge lengths 1 and 2. *Information Processing Letters*, 32:171–176, September 1989.

[11] Daniel Bienstock. Computational experience with an effective heuristic for some capacity expansion problems in local access networks. *Telecommunication Systems*, 1(4):379–400, May 1993.

[12] Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. In G. Schmidt and R. Berghammer, editors, *Graph-Theoretic Concepts in Computer Science (Lecture Notes in Computer Science 570)*, pages 1–12. Springer-Verlag, 1992.

[13] Robert R. Boorstyn and Howard Frank. Large-scale network topological optimization. *IEEE Transactions on Communications*, COM-25(1):29–47, January 1977.

[14] E. Brockmeyer, H. L. Hastrøm, and Arne Jensen. *The life and works of A. K. Erlang*, volume 6 of *Acta polytechnica Scandinavica. Mathematics and Computing Machinery Series*. Danish Academy of Technical Sciences, 2nd edition, 1960.

[15] Richard H. Cardwell, Clyde L. Monma, and Tsong-Ho Wu. Computer-aided design procedures for survivable fiber optic networks. *IEEE Journal on Selected Areas in Communications*, 7(8):1188–1197, October 1989.

[16] Vašek Chvátal. *Linear Programming.* W. H. Freeman and Company, 1980.

[17] D. D. Clark, B. S. Davie, D. J. Farber, I. S. Gopal, B. K. Kadaba, W. D. Sincoskie, J. M. Smith, and D. L. Tennenhouse. The AURORA gigabit testbed. *Computer Networks and ISDN Systems*, 25(6), January 1993.

[18] Martin de Prycker. *Asynchronous transfer mode: Solution for broadband ISDN.* Ellis Horwood, 2nd edition, 1993.

[19] D.-Z. Du and F. K. Hwang. An approach for proving lower bounds: Solution of Gilbert-Pollak's conjecture on Steiner ratio. In *Proc. 31st Symp. Foundations of Comp. Sci.*, pages 76–85, October 1990.

[20] Ding-Zhu Du and Frank K. Hwang. The state of art on Steiner ratio problems. In Ding-Zhu Du and Frank K. Hwang, editors, *Computing in Euclidean Geometry*, pages 163–191. World Scientific, 1992.

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

[22] Howard Frank and Wushow Chou. Topological optimization of computer networks. *Proceedings IEEE*, 60:1385–1397, 1972.

[23] Howard Frank, Ivan T. Frisch, and Wushow Chou. Topological considerations in the design of the ARPA computer network. In *AFIPS Conference Proceedings*, pages 581–587, Montvale, NJ, May 1970. AFIPS Press.

[24] L. Fratta, Mario Gerla, and Leonard Kleinrock. The flow deviation method: an approach to store-and-forward communication network design. *Networks*, 3:97–133, 1973.

[25] Harold N. Gabow. A representation for crossing set families with applications to submodular flow problems. In *4th and ACM-SIAM Symp. Discrete Algorithms*, pages 202–211, 1993.

[26] Harold N. Gabow, Michel X. Goemans, and David P. Williamson. An efficient approximation algorithm for the survivable network design problem. In *Proc. Third Conf. on Integer Programming and Combinatorial Optimization*, 1993. Erice, Italy, April 29 – May 1.

[27] Harold N. Gabow and Eugene W. Myers. Finding all spanning trees of directed and undirected graphs. *SIAM Journal on Computing*, 7(3):280–287, August 1978.

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

[29] Bezalel Gavish. Topological design of centralized computer networks – formulations and algorithms. *Networks*, 12:355–377, 1982.

[30] Bezalel Gavish. Formulations and algorithms for the capacitated minimal directed tree problem. *Journal of the ACM*, 30(1):118–132, January 1983.

[31] Bezalel Gavish. Augmented Lagrangean based algorithms for centralized network design. *IEEE Transactions on Communications*, COM-33(12):1247–1257, December 1985.

[32] Bezalel Gavish. Topological design of telecommunication networks - local access design methods. In J. MacGregor Smith and Pawel Winter, editors, *Topological Network Design (Annals of Operations Research 33)*, pages 17–71. J. C. Baltzer AG, 1991.

[33] Mario Gerla and Leonard Kleinrock. On the topological design of distributed computer networks. *IEEE Transactions on Communications*, COM-25(1):48–60, January 1977.

[34] E. N. Gilbert. Minimum cost communication networks. *Bell System Technical Journal*, pages 2209–2227, 1967.

[35] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[36] Michel X. Goemans and David P. Williamson. A general approximation technique for constrained forest problems. In *Proc. 3rd ACM-SIAM Symp. Discrete Algorithms*, pages 307–315, 1992. also submitted to SIAM Journal on Computing.

[37] 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.

[38] Andrew V. Goldberg and Robert Endre Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, August 1990.

[39] Martin Grötschel, Clyde L. Monma, and Mechthild Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40(2):309–330, March–April 1992.

[40] Martin Grötschel, Clyde L. Monma, and Mechthild Stoer. Design of survivable networks. In *Handbook in Operations Research and Management Science*. 1994. In preparation.

[41] Torben Hagerup and Christine Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33(6):305–308, 1990.

[42] Frank Harary. *Graph Theory.* Addison-Wesley Series in Mathematics. Addison-Wesley, 1969.

[43] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, March 1963.

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

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

[46] Carolyn Jack, Sheng-Roan Kai, and Alexander Shulman. Design and implementation of an interactive optimization system for telephone network planning. *Operations Research*, 40(1):14–25, January-February 1992.

[47] Carolyn Jack, Sheng-Roan Kai, and Alexander Shulman. NETCAP - an interactive optimization system for GTE telephone network planning. *Interfaces*, 22(1):72–89, January-February 1992.

[48] Amos E. Joel, Jr. *A History of Engineering and Science in the Bell System*, volume Switching Technology (1925–1975). Bell Telephone Laboratories, Inc., 1982. See especially Chapter 5, Part III.

[49] C. Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math.*, 70:185–190, 1869.

[50] 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.

[51] 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.

[52] AT&T Bell Laboratories. *Engineering and operations in the Bell System.* The Laboratories, 2nd edition, 1984. See especially Chapters 4 and 5 for network planning and traffic analysis.

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

[54] Patrick V. McGregor and Diana Shen. Network design: An algorithm for the access facility location problem. *IEEE Transactions on Communications*, COM-25(1):61–73, January 1977.

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

[56] Clyde L. Monma and David F. Shallcross. Methods for designing communication networks with certain two-connected survivability constraints. *Operations Research*, 37(4):531–541, 1989.

[57] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[58] Hossein Saidi. *Multi-Channel Switching in Broadband ATM Network.* PhD thesis, Dept. of Electrical Engineering, Washington University, May 1994.

[59] Hossein Saidi, Paul S. Min, and Manjunath V. Hegde. Guaranteed cell sequence in nonblocking multi-channel switching. In *INFOCOM*. IEEE, June 1994.

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

[61] C. J. Truitt. Traffic engineering techniques for determining trunk requirements in alternate routed networks. *Bell System Technical Journal*, 33(2):277–302, March 1954.

[62] Harrell J. Van Norman. *LAN/WAN optimization techniques*. Artech House, 1992.

[63] Bernard Yaged, Jr. Minimum cost routing for dynamic network models. *Networks*, 3:193–224, 1973.

[64] Norman Zadeh. On building minimum cost communication networks over time. *Networks*, 4:19–34, 1974.

[65] Ellen Witte Zegura. Architectures for ATM switching systems. *IEEE Communications Magazine*, 31(2):28–37, February 1993.

B. VITA

Bibliographical items on the author of the dissertation, J. Andrew Fingerhut.

- Born May 11, 1969 in Belleville, Illinois.
- Attended Southern Illinois University at Edwardsville, January 1981 to June 1987. Received the degree Bachelor of Arts with majors in Computer Science, Mathematics, and Physics.
- Attended Washington University from September 1987 to present. Received the degree Master of Science in Computer Science in May, 1990.
- Worked as a research assistant in the Computer and Communications Research Center from January 1990 to present.
- Member of ACM, SIAM, and ORSA.

May, 1994