Analysis of algorithms for online routing and scheduling in networks

Jessen Tait Havill
College of William & Mary - Arts & Sciences

ANALYSIS OF ALGORITHMS FOR ONLINE ROUTING AND SCHEDULING IN NETWORKS

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

by

Jessen Tait Havill

1998

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

[illegible]

Author

Approved, June 1998

Weizhen Mao

Weizhen Mao

[illegible]

J. Philip Kearns

[illegible]

Rahul Simha

[illegible]

Paul Stockmeyer

[illegible]

Rex Kincaid
Department of Mathematics

to beth

Table of Contents

ACKNOWLEDGEMENTS

I owe my thanks to many people for the various ways in which they have supported me over the past four years. First and foremost, I would like to thank Beth for her overwhelming love and support. I have, at times, been difficult to live with (to say the least): thank you for being there for me through it all.

I am indebted to my advisor, Dr. Weizhen Mao, for guiding me through this enormously educational process. I especially thank her for her patience and her constant attententiveness. By her example, I have developed a greater appreciation for this field. I would also like to thank the members of my committee — Dr. Rahul Simha, Dr. Phil Kearns, Dr. Paul Stockmeyer, and Dr. Rex Kincaid — for their time and their suggestions on how to improve my work.

I would like to thank all the members of my family for their support and patience while I completed this degree. I would like to thank my friends in Williamsburg for their company, their interesting conversation, and their interesting diversions.

Lastly, I would like to acknowledge the Virginia Space Grant Consortium for their financial support through a Virginia Space Grant Fellowship during my last year at William and Mary.

List of Tables

List of Figures

ABSTRACT

We study situations in which an algorithm must make decisions about how to best route and schedule data transfer requests in a communication network before each transfer leaves its source. For some situations, such as those requiring quality of service guarantees, this is essential. For other situations, doing work in advance can simplify decisions in transit and increase the speed of the network. In order to reflect realistic scenarios, we require that our algorithms be online, or make their decisions without knowing future requests. We measure the efficiency of an online algorithm by its competitive ratio, which is the maximum ratio, over all request sequences, of the cost of the online algorithm's solution to that of an optimal solution constructed by knowing all the requests in advance.

We identify and study two distinct variations of this general problem. In the first, data transfer requests are permanent virtual circuit requests in a circuit-switched network and the goal is to minimize the network congestion caused by the route assignment. In the second variation, data transfer requests are packets in a packet-switched network and the goal is to minimize the makespan of the schedule, or the time that the last packet reaches its destination. We present new lower bounds on the competitive ratio of any online algorithm with respect to both network congestion and makespan.

We consider two greedy online algorithms for permanent virtual circuit routing on arbitrary networks with unit capacity links, and prove both lower and upper bounds on their competitive ratios. While these greedy algorithms are not optimal, they can be expected to perform well in many circumstances and require less time to make a decision, when compared to a previously discovered asymptotically optimal online algorithm. For the online packet routing and scheduling problem, we consider an algorithm which simply assigns to each packet a priority based upon its arrival time. No packet is delayed by another packet with a lower priority. We analyze the competitive ratio of this algorithm on linear array, tree, and ring networks.

ANALYSIS OF ALGORITHMS FOR ONLINE ROUTING AND SCHEDULING IN NETWORKS

Chapter 1

Introduction

1.1 Problem Statement

Routing and scheduling are fundamental problems in the design of a networking protocol. To route a data transfer[1] is to choose the path in the network that the data will follow to reach its destination. To schedule a data transfer is to decide when it should be transferred across each link in its path.

Traditional store-and-forward protocols transfer large amounts of data by breaking the data into many unit size packets and handling each packet individually. Packets are routed in a piecemeal fashion; when a packet arrives at a network node, the node decides where to next forward the packet by consulting a local routing table. Routing tables may be updated periodically based on statistical measures of network traffic. Scheduling is accomplished by means of a local queuing algorithm at each node which, at each time step, chooses to forward

[1]We will use the term *data transfer* to refer to any transfer of data, regardless of the type of network protocol used. When we talk about packet-switched networks, we will refer specifically to *file transfers* or *packets*. When we discuss circuit-switched networks, we will refer to *virtual circuits*.

a subset of packets currently in its queue.

The emergence of high speed networks and modern network applications has initiated a reappraisal of how to best handle network data transfers. One alternative to local routing and scheduling is to make routing and/or scheduling decisions in advance at the source, before any data is sent. This design decision makes sense in a variety of situations. For example, many multimedia streams require guarantees of both low latency and low data loss. The latter problem is a result of buffer overflow which can arise from even small delays in high speed networks. Such guarantees are met by allocating a *virtual circuit* to the stream in advance. A virtual circuit consists of guaranteed bandwidth on a fixed route for a specified duration. Once the virtual circuit is allocated, all data sent along the circuit is conceptually considered to be a FIFO stream (or flow). Virtual circuit service is an important component of emerging Asynchronous Transfer Mode (ATM) networks in which several different types of traffic, each with its own bandwidth requirement, vie for the same network resources.

Source routing and scheduling may also be advantageous in packet switched networks when file transfers are very large (e.g., multimedia, network backups, distributed database updates). By using available global information, or actually gathering such information, one may be able to prevent congestion and more efficiently transfer the file. When files are large, the time required to globally determine a good route and schedule for a file transfer is likely to be small compared to the time required for the actual transfer. Furthermore, transferring files efficiently in light of current traffic conditions is more important for large files than for small ones. Consider the case of two files whose assigned paths intersect at some link in the network. If the files are small, the delay resulting from such a "collision" is small. However, if the files are quite large, the delay may be substantial. Traditional packet switching cannot (for lack of global information) prevent such delays, whereas a centralized routing and scheduling method can carefully spread out file transfers to reduce overall completion times. Furthermore, the scheduling of file transfers will impact available bandwidth for other, perhaps real time, transmissions.

Making routing and/or scheduling decisions in advance also decreases the required complexity, and increases the speed, of network switches, an important issue in high speed networks. If preassigned routes (or ATM virtual circuit/path identifiers) are encoded in packet (or ATM cell) headers then switches simply forward packets without any significant time loss. In packet switched networks, encoded schedules, which may be as simple as fixed priorities, can also reduce the control overhead.

We consider *online* source routing and scheduling problems in which data transfer (file transfer or virtual circuit) requests become known to an algorithm one at a time and both the routing and scheduling decisions must be made for each request before any future requests are known. This approach is in contrast to an *offline* approach in which it is assumed that all requests are known *a priori*. While there always exists an optimal offline algorithm for a problem, an online algorithm is only able to give a suboptimal solution based on its limited knowledge. In order to measure the efficiency of an online algorithm, we compare its performance, with respect to some objective function, to that of an optimal offline algorithm. For a minimization problem, which is the type of problem we consider, the maximum ratio of the performance of an online algorithm to that of an optimal offline algorithm is called the *competitive ratio.*

In its most general form, the online routing and scheduling problem is formally defined as follows.[2] We are given a weighted directed graph $G = (V, E, c, q)$ representing a communication network. The set V of nodes represents a set of machines (or switches) which are capable of sending and receiving data. The set E of directed edges represents a set of unidirectional communication links between machines. We assume that each machine can participate in as many simultaneous sends and receives as there are incident outgoing and incoming edges, respectively.

The function $c: E \rightarrow \{1, 2, 3, \dots\}$ describes the capacity of the network links. In a packet switching model, we assume that a file transfer consists of a number of fixed-size packets and define $c(e)$ to be the number of packets that can be transmitted over link e in unit time. In a virtual circuit model, $c(e)$ represents the total bandwidth available for virtual circuit requests on link e. We will assume that, for all $e \in E$, $c(e)$ is equal to a constant w (which may differ from network to network). w may be called the *width* of the network links.

In a packet switching model, each machine v contains an intermediate queue which may be used for temporarily storing packets as they are forwarded to their destinations in the network. The function $q: V \rightarrow \{0, 1, 2, \dots\}$ describes the *queue length* or *memory constraint* associated with machine v. In particular, $q(v)$ is the number of forwarded packets that can be stored at machine v in an intermediate queue at any given time. We assume that each machine v also possesses an initial queue and a final queue of arbitrarily large capacity. Files sent from v are initially stored in its initial queue and packets with destination v are immediately taken from the intermediate queue at v and placed in the final queue. In a

[2] A list of the following notation, and other notation used in this dissertation, is included in Appendix B for reference.

virtual circuit model, these memory constraints can be ignored.

The input to our problem is a sequence of file transfer (or virtual circuit) requests $\sigma = f_1, f_2, \ldots, f_k$. Each $f_j \in \sigma$ is represented by a tuple (s_j, t_j, l_j, a_j). The nodes $s_j, t_j \in V$ are the source and destination of the request, respectively. The value $l_j \in \{1, 2, 3, \ldots\}$ is the size of the request, the length of the file (in fixed-size packets) or the required bandwidth of the virtual circuit, and $a_j \in \{0, 1, 2, \ldots\}$ is the request's arrival time. ($a_i \leq a_j$ if and only if $i < j$.)

When a request $f_j \in \sigma$ arrives, a centralized algorithm in a packet-switched environment must immediately make both a routing and scheduling decision based solely on the decisions it made for requests $f_1, f_2, \ldots, f_{j-1}$. In a circuit-switched environment, an algorithm need only make an online routing decision. We will consider the problem of routing *permanent* virtual circuit requests (vs. *switched* virtual circuits) which are assumed to exist indefinitely. Equivalently, one may think of this model as one in which the finishing times of all requests are the same.

An algorithm routes a request f_j by assigning to f_j a path $P_j \in \mathcal{P}_j$, where $\mathcal{P}_j$ is the set of paths between machines s_j and t_j in the network. (We assume that $|\mathcal{P}_j| > 0$ for all $j = 1, 2, \ldots, k$.) An algorithm schedules a file transfer by specifying the times at which it will cross each link in P_j. If a file transfer is not crossing a link and has not yet reached its destination, then it must be assigned to a queue at the head of the next link on its path. At all times, no link e may be assigned more than $c(e) = w$ packets and no node v may be assigned more than $q(v)$ packets.

We will seek to minimize two objective functions, one for each network model we consider. In the virtual circuit model, we will minimize the *network congestion*, defined to be the maximum ratio, over all links in the network, of the bandwidth assigned to the link to the link's capacity. Minimizing congestion attempts to avoid hot spots or bottlenecks by spreading bandwidth as evenly as possible over the network links. Minimizing congestion also attempts to increase the network bandwidth available for other users and applications.

Formally, we define

$$\mu_j(e) = \sum_{i<j: e \in P_i} \frac{l_i}{c(e)} \tag{1.1}$$

to be the *congestion on link* e after an online algorithm has routed requests $f_1, f_2, \ldots, f_{j-1}$. (The particular online algorithm will always be clear in the current context.) For simplicity, let $\mu(e) = \mu_{k+1}(e)$, the congestion on link e after an on-line algorithm has routed all k requests.

The *congestion on path* P after an online algorithm has routed requests $f_1, f_2, \ldots, f_{j-1}$ is defined to be

$$\mu_j(P) = \max_{e \in P} \mu_j(e).$$

Let $\mu(P) = \mu_{k+1}(P)$.

Finally, the *network congestion* after an online algorithm has routed requests $f_1, f_2, \ldots, f_{j-1}$ is defined to be

$$\mu_j = \max_{e \in E} \mu_j(e).$$

The total congestion incurred by an online algorithm is denoted $\mu = \mu_{k+1}$. Clearly, since we do not allow requests to be rejected, the congestion of a link may exceed one. If an optimal offline algorithm can service the request sequence while maintaining congestion at most one, and an online algorithm incurs congestion $c > 1$, this is an indication that the algorithm requires that the network capacity be increased by a factor of c in order to service the sequence. Alternatively, $c > 1$ may be interpreted as a degree of slowdown in service. From a theoretical point of view, the factor c is also an indication of how well the online algorithm can find suitable routes.

The second objective function we consider, applicable in the packet switching model, is the *makespan*, defined to be the time the last packet reaches its destination. Formally, we define C_j to be the *completion time* of file transfer f_j. The makespan of a schedule is defined to be

$$C = \max_{1 \leq j \leq k} C_j.$$

1.2 Competitive Analysis

An online algorithm for a minimization problem is traditionally called *competitive* if it always finds a solution whose cost, or objective function value, is within a small factor of the cost incurred by an optimal offline algorithm. An online algorithm for a maximization problem is competitive if it always finds a solution whose profit (again, the objective function value) is a sufficiently large fraction of the optimal profit. This technique is known as *competitive analysis* [48]. Intuitively, competitive analysis measures the degree to which the performance of an online algorithm suffers, due to its lack of knowledge about the future, compared to that of an optimal offline algorithm.

Competitive analysis was introduced by Sleator and Tarjan [89], who studied online algorithms for accessing the elements of a list. Their techniques were further refined by Borodin, Linial, and Saks [24, 25] in the context of metrical task systems; Karlin, Manasse, Rudolph, and Sleator [47] in the context of snoopy caching; and Manasse, McGeoch, and Sleator [65, 66] in their classical work on the k-server problem. Competitive analysis has since been applied to many common resource allocation problems including single processor scheduling [71], file allocation [21, 11], load balancing (e.g., [17, 18]), multiprocessor scheduling [85], and virtual circuit routing [80].

As indicated above, the competitive ratio is defined differently for minimization and maximization problems. First, consider an online algorithm A for a minimization problem. Let $\text{cost}(\sigma)$ be the cost of the solution obtained by A given request sequence σ. Let $\text{cost}^*(\sigma)$ be the cost obtained by an optimal offline algorithm given the same request sequence.

Definition 1.1

Online algorithm A is c competitive for a minimization problem if and only if, for all request sequences σ, $\text{cost}(\sigma) \leq c \cdot \text{cost}^*(\sigma) + a$, where a is a constant.

Definition 1.2

The competitive ratio of online algorithm A for a minimization problem is defined to be

$$\sup_{\sigma} \frac{\text{cost}(\sigma)}{\text{cost}^*(\sigma)}.$$

We note that very often c is not a constant, but rather a function of some input parameters. For a maximization problem, let $\text{profit}(\sigma)$ denote the profit or benefit of the solution obtained by A given request sequence σ. Let $\text{profit}^*(\sigma)$ be the profit gained by an optimal offline algorithm.

Definition 1.3

Online algorithm A is c competitive for a maximization problem if and only if, for all request sequences σ, $\text{profit}(\sigma) \geq \frac{1}{c} \cdot \text{profit}^*(\sigma) + a$, where a is a constant.

Definition 1.4

The competitive ratio of online algorithm A for a maximization problem is defined to be

$$\sup_{\sigma} \frac{\text{profit}^*(\sigma)}{\text{profit}(\sigma)}.$$

Definition 1.5

Let $c(A)$ denote the competitive ratio of an online algorithm A for a problem P. The competitive ratio of P is defined to be $\inf_A c(A)$.

In order to prove a strong result about the competitive ratio of a online algorithm, it is necessary to bound the competitive ratio from both below and above. If a value (or a function) c is an upper bound on the competitive ratio of an online algorithm then, for all request sequences, the algorithm is guaranteed to produce a solution whose cost is at most c times the cost of an optimal solution (plus perhaps a constant). If c is a lower bound on the competitive ratio of an online algorithm, then the competitive ratio of that algorithm cannot be better than c. If c is a lower bound on the competitive ratio of a problem, then no online algorithm for the problem can achieve a competitive ratio better than c. If a known lower bound for a problem matches the upper bound for an online algorithm, then we know that the online algorithm is the best possible. A lower bound proof is often framed as a game between an online algorithm and a cruel adversary. The adversary issues a request sequence with the goal of forcing the online algorithm to perform badly. If the adversary can generate a request sequence that forces the ratio of the online algorithm's cost to its cost to be c, then c must be a lower bound on the competitive ratio of the online algorithm. If this strategy works for any online algorithm, then c is a lower bound on the competitive ratio for the problem.

One can also use competitive analysis to measure the performance of a randomized online algorithm. A randomized algorithm is defined to be a probability distribution over a set of deterministic algorithms. The competitiveness of a randomized online algorithm is measured with respect to its expected performance, where the expectation is taken over its random choices. The competitive analysis of randomized algorithms is more subtle because the competitiveness of an algorithm depends on how much an offline adversary knows about the algorithm's random choices. In contrast, a deterministic algorithm is completely predictable and easier for an adversary to "outsmart". Ben-David, et al. [22, 23] introduced three types of adversaries against which one may compare a randomized online algorithm. An *oblivious adversary* chooses its request sequence in advance without knowing the algorithm's random choices and then responds to the sequence optimally. An *adaptive online adversary* chooses each request based on the algorithm's responses to previous requests but must serve each request itself before future responses of the algorithm are known. Lastly, an *adaptive offline adversary* chooses each request in the same way as the previous adversary but may respond to the entire sequence at the end. Adaptive adversaries are stronger than an oblivious adversary. An adaptive offline adversary is so strong that if there exists a randomized online algorithm for a problem that is c competitive against an adaptive offline adversary, then there exists a c competitive deterministic online algorithm as well. [22, 23]

Competitive analysis allows us to analyze online algorithms without having to make any assumptions about the input. Such assumptions, especially those characterizing arrivals and service times in terms of probability distributions, are sometimes unnatural and poor approximations to real systems, especially in the case of packet routing. Competitive analysis provides a robust worst case analysis of an online algorithm; if an online algorithm can be shown to have a small competitive ratio then this is a strong result. Since the optimal performance is fixed, this method also gives an algorithm designer the benefit of being able to compare the performance of two online algorithms by using the constant optimal performance as a benchmark.

1.3 Related Research

There are at least four related threads of research in the current literature that relate to the online routing and scheduling problems we consider. In this section, we will review these results. In Section 1.4, we will describe how our research relates to these results and give an outline of the dissertation.

1.3.1 Offline File Transfer Routing and Scheduling

The offline version of the file transfer routing and scheduling problem on packet-switched networks was introduced and studied independently by two groups — Mao and Simha [73, 72] and Rivera-Vega, Varadarajan, and Navathe [82, 83, 95].

Essentially two special cases of the offline problem have been studied in the literature. In both versions, unless specified otherwise, network links have unit capacity, nodes contain arbitrarily large queues, and file transfers have unit length. In the first version of the problem, network links are switched on a file by file basis and the goal is to minimize the makespan of a schedule. In the second version of the problem, network links can accommodate several file transfers simultaneously with multiplexing and the goal is to minimize the network congestion of the route assignment. (This version is equivalent to a permanent virtual circuit routing problem.) Mao and Simha [72] called these two versions the serial model and the shared model, respectively.

Complexity Results

Rivera-Vega, Varadarajan, and Navathe [82] proved that the makespan minimization problem on general networks is NP-hard and can be formulated as a 0-1 integer programming problem. They were able to limit the number of variables in the integer program by proving that the length of an optimal schedule is tightly bounded from below by L and from above by $L+k-1$, where L is the length of the longest shortest path over all file transfers and k is the number of file transfers. If the length of the shortest path of every file is different, then a schedule of length L is always possible by routing the files over shortest paths and giving priority to files with the longest shortest path. Rivera-Vega, Varadarajan, and Navathe [83] proved that the same version of the problem is NP-hard on fully connected networks, even if the routes are allowed to contain at most two edges. However, they designed a $O(|V|+k)$ time algorithm that finds an optimal two-edge schedule in fully connected networks when there is only one source or one destination.

Mao and Simha [73, 72] considered both the serial and shared link models, under the constraint that no node acts as both a sender and a receiver of a file. They proved that both versions are NP-hard when file transfers can have arbitrary lengths, even if the network is bipartite. However, both versions can be solved in polynomial time on bipartite networks if files have unit lengths, there is only one file sent between any pair of nodes, or the network graph has only single links between any pair of nodes. Define an *L-layer* digraph (or a *layered* digraph with L layers) to be one in which the nodes can be partitioned into $L+1$ sets $V_0, V_1, \ldots, V_L$ such that, for all edges $(u,v) \in E$, $u \in V_i$ and $v \in V_{i+1}$ for some $0 \leq i \leq L-1$. (An N input butterfly network is an example of a layered network with $\log N - 1$ layers[3].) Mao and Simha proved that for *2-layer* digraphs with only single, unit capacity edges, both versions of the problem are NP-hard, even if only one unit length file transfer is sent between any pair of nodes. Mao and Simha also showed that if the network is a ring with arbitrary link speeds and every node acts as a sender and receiver of an arbitrary length file, then the congestion minimization problem is NP-hard.

Approximation Algorithms

Given that so many simple versions of the problem are likely intractable, it makes sense to examine polynomial time approximation algorithms. Rivera-Vega, Varadarajan, and Navathe [82] proposed using an *adaptive path scheduling* heuristic to approximately schedule file transfers in the makespan minimization problem. Files are assigned a fixed priority and considered in this order. The heuristic assigns to each file the path that requires the least time based on conflicts with higher priority files. No analysis is given in [82] for this algorithm, but the authors conjectured that it constructs schedules with makespan bounded by the theoretical upper bound of $L+k-1$ if a suitable priority scheme is chosen.

Rivera-Vega, Varadarajan, and Navathe [83] conjectured that two-edge schedules are optimal on fully connected networks and proposed a two phase algorithm to construct such a schedule. In the first phase, a set of two-edge routes are found for the file transfers. The au-

[3]We use log to denote $\log_2$.

thors gave an optimal integer programming formulation and two approximation algorithms for this phase. The first approximation algorithm produces schedules with makespan within a factor of four of optimal. The second approximation algorithm assigns all but an initial subset of the files one at a time to a two-edge route with minimum congestion. Simulations suggest that the performance guarantee of the second approximation algorithm is roughly twice optimal. In the second phase of the two phase algorithm, a schedule is computed for the set of routes found in the first phase. The authors gave an algorithm for the second phase that is optimal with respect to a given set of two-edge routes. Varadarajan and Rivera-Vega [95] later designed an approximation algorithm with a performance guarantee of two which computes two-edge routes and a schedule simultaneously.

Mao and Simha [72] studied three simple online list scheduling (LS) algorithms for assigning routes to file transfers in general networks. The file transfers are scheduled so that a file stays at an intermediate node only when the next link in its route is busy transferring a file with a smaller index. The three online routing algorithms are described as follows:

Algorithm LS1

For file transfer request f_j, assign any route $P \in \mathcal{P}_j$.

Algorithm LS2

For file transfer request f_j, assign a route $P \in \mathcal{P}_j$ with the fewest links that have been used before.

Algorithm LS3

For file transfer request f_j, assign any route $P \in \mathcal{P}_j$ which minimizes

$$\max_{e \in E} \begin{cases} \mu_j(e), & e \notin P \\ \mu_j(e) + \frac{l_j}{c(e)}, & e \in P \end{cases} .$$

Mao and Simha showed that the competitive ratios of algorithms LS1 and LS2 are $\Omega(k)$, and the competitive ratio of algorithm LS3 is $\Omega\left(\sqrt{k}\right)$, with respect to both congestion and makespan, even on 2-layer digraphs with unit speed links. Simulation experiments showed that in practice both LS2 and LS3 outperform LS1 by a large margin while LS3 constructs schedules just a little better than LS2 does.

File Transfer Scheduling

The offline file transfer routing and scheduling problem is related to the File Transfer Scheduling problem of Coffman, Garey, Johnson, and LaPaugh [32]. An instance of this problem is described by a multigraph called a *file transfer graph*. Each graph vertex represents a transceiver which is capable of sending a file to or receiving a file from any other transceiver via an underlying fully connected network. Each edge represents a file to be transferred between the transceivers represented by its endpoints. The length of an edge represents the time required to transfer the file. Associated with each vertex is a *port constraint* which specifies the maximum number of file transfers in which the vertex may be engaged at any time. The goal is to assign start times to the file transfers, subject to port constraints, so that makespan is minimized. Coffman, et al. [32] proved that the general problem, as well as several special cases, are NP-hard. Other special cases can be solved optimally by polynomial time algorithms. [32, 27]

For those cases of the problem that are NP-hard, Coffman, et al. [32] studied the List Scheduling (LS) approximation algorithm and variants thereof. At time 0 and every time thereafter that marks the completion of a file transfer, LS scans a priority list for file transfers that have not yet started and for which there are available ports at both endpoints, and starts each such file transfer immediately. If all file transfers have equal length, the performance ratio of LS is strictly less than two. If the file transfers have arbitrary lengths and port constraints are unit then the performance ratio of LS is at most two. If the files have arbitrary lengths and the port capacities are arbitrary then the performance ratio of LS is 2.5 if the priority list is maintained in decreasing order of transfer times. For many special graphs, better performance ratios are possible with both LS and other algorithms. [32, 27, 75, 76, 74]

Coffman, et al. [32] also studied the problem in a distributed environment in which each machine makes scheduling decisions for the file transfers that originate with it, and no machine has accurate knowledge of the global system state. The authors analyzed two distributed algorithms for situations when Byzantine failures are not possible, and communication delays are small compared to file transfer times.

Several variations of the original problem were studied subsequently. Mao [69] examined the special case in which the file transfer graph is a directed multigraph. Several authors studied versions allowing preemption [41, 40, 77, 29, 26, 28, 38]. Whitehead [97] studied a file transfer scheduling problem that allows file forwarding.

1.3.2 Online Virtual Circuit Routing

Several variations of the online virtual circuit routing problem have been studied recently [80]. Variants of recently discovered algorithms, based on techniques developed to approximately solve offline multicommodity flow problems [84, 60, 52], are reportedly being implemented in AT&T's ATM switches, and the research was recently recognized by the SIGACT Long Range Planning Committee as a major contribution of theoretical computer science [3]. The general problem (more general than we described earlier) is composed of two parts: *admission control* and *routing*. The admission control component is to decide whether to accept a virtual circuit request given limited resources. The routing component is to decide which path to use for an accepted request.

A virtual circuit request sequence is denoted $\beta_1, \beta_2, \ldots, \beta_k$ where each β_j is represented by a tuple $(s_j, t_j, r_j(\tau), a_j, b_j)$. Nodes s_j and t_j are the source and destination of the request, respectively. a_j and b_j are the requested start and finishing times for the virtual circuit. When $b_j = \infty$ we call β_j a *permanent* virtual circuit. Otherwise, β_j is a *switched* virtual circuit. There may be situations when the finishing time is finite, but becomes known only after the virtual circuit is finished. In these cases, we modify the model slightly to include both initiation and termination requests. When the finishing time is known, the duration is denoted $T(j) = b_j - a_j$. $r_j(\tau)$ describes the bandwidth requirement for virtual circuit β_j as a function of time. For permanent virtual circuits, the rate is assumed to be constant with time and is denoted r_j.

Two general versions of the virtual circuit routing problem have been studied. In the first, an algorithm must accept all requests and the goal is to minimize the network congestion incurred by those requests. (This is the version we study.) In the second version of the problem, some requests may be rejected and the goal is to maximize network throughput while obeying capacity constraints.

Algorithms for Minimizing Congestion

Aspnes, et al. [4, 5] studied permanent virtual circuit routing in the congestion minimization model. They designed an elegant online algorithm which assigns to each request β_j a shortest path with respect to the following exponential cost function:

$$c_e(j) = a^{\mu_j(e)+\frac{r_j}{c(e)}} - a^{\mu_j(e)}$$

where $a = 1+\gamma$ for any $0 < \gamma < 1$. The quantity $\mu_j(e)$ is defined as in equation (1.1). with $l_j = r_j$. This algorithm, which we will call Exp_Route, is $O(\log n)$ competitive. where $n = |V|$, on arbitrary networks.[4] Furthermore, Aspnes, et al. proved that Exp_Route is optimal (up to a constant factor) by showing that no algorithm can have a competitive ratio better than $\frac{\log n - 1}{2}$.

A $O(\log(nT))$ competitive algorithm, where T is the maximum duration. can be derived for switched virtual circuits by concurrently using the algorithm above to solve one instance of permanent virtual circuit routing for each time step [16]. No matching lower bound is currently known. Awerbuch, et al. [10] noted that the competitive ratio of any online algorithm for routing switched virtual circuits with unknown durations must be $\Omega(\sqrt[4]{n})$. However. they designed a $O(\log n)$ competitive algorithm that reroutes a request $O(\log n)$ times. A

[4]This is technically only true if the optimal congestion is 1. For the more general case, the algorithm must be modified slightly, with only a factor of 4 increase in the competitive ratio.

rerouted circuit is simply reassigned to another path between its source and destination during its duration. Rerouting causes a brief disruption in service, but ultimately allows the transmission to continue to completion.

For a model with Poisson arrivals, Kamath, Palmon, and Plotkin [45] presented an algorithm that is similar to Exp_Route in that circuits are routed along shortest paths with respect to an exponential cost function. However, the maximum congestion of the algorithm is guaranteed to stay within an additive $\sqrt{r \log n}$ factor of the optimal congestion, where r is the maximum fraction of bandwidth that can be requested by a single virtual circuit on any link, as long as r is relatively small.

Awerbuch, Azar, and Fiat [8] applied a generalization of Exp_Route to an online packet routing and scheduling problem. Their problem differs most notably from ours in that they allow queue and link capacities to be exceeded while minimizing average packet delay $\frac{1}{k}\sum_j (C_j - a_j)$. Their algorithm is simultaneously $\mathrm{O}\left(\log\left(T\sum_{e\in E} c(e)\right)\right)$ competitive with respect to congestion, where T is the maximum packet delay, and $\frac{2}{1-\gamma}$ competitive with respect to average packet delay.

Algorithms for Maximizing Throughput

In the throughput maximization version of the virtual circuit routing problem, a profit ρ_j is specified for each request β_j and the goal is to maximize the total profit of the accepted requests. Among other things, the total profit can be defined to be throughput or the number of accepted circuits. When durations are unknown, no competitive algorithm exists for the problem, even on a single link and even if preemption is allowed [34]. However, Awerbuch, Azar, and Plotkin [9] designed an admission control and routing algorithm for virtual circuits with known duration. As with Exp_Route, the algorithm first assigns to each link a cost that is an exponential function of the current congestion. In this case, $a = 2LTF + 1$ where L is the maximum number of hops taken by a virtual circuit, T is the maximum duration of a virtual circuit, and F is a constant. The cost of a link e at time τ, just before request β_j arrives, is defined to be

$$c_e(j,\tau) = c(e)\left(a^{\mu_j(e,\tau)} - 1\right)$$

where $\mu_j(e,\tau)$ is defined as it was for the congestion minimization problem, with an added time parameter. Then a request is accepted only if there exists a route whose total cost is small relative to the profit to be gained by accepting it. Specifically, a request β_j is accepted and assigned to any path P_j satisfying

$$\sum_{e\in P_j}\sum_{\tau}\left(\frac{r_j(\tau)}{c(e)}\,c_e(j,\tau)\right) \leq \rho_j$$

if such a path exists. Otherwise, the request is rejected. It is interesting to note that the algorithm does not necessarily choose a unique route for a request. Rather, a request is routed on *any* path that fulfills the requirement.

If the following two assumptions hold on the input, then the algorithm is O(log(LTF)) competitive on general networks. By removing the element of time, an O(log(LF)) competitive algorithm for permanent virtual circuits can be derived from this result [80]. The first aforementioned assumption requires that the profit of a request be proportional to its throughput. Specifically, for any request β_j,

$$1 \leq \frac{1}{L}\cdot\frac{\rho_j}{r_j(\tau)T(j)} \leq F.$$

Secondly, the bandwidth requirement of each request must be small relative to the link capacities. Specifically, no online algorithm can have a polylogarithmic competitive ratio on arbitrary networks unless

$$r_j(\tau) \leq \frac{\min_{e \in E} c(e)}{\log a}. \tag{1.2}$$

Without these assumptions, randomized online algorithms with polylogarithmic competitive ratios have been designed for trees, meshes, trees of meshes, and hypercubes [12, 13, 61].

Variants of the exponential routing algorithm have been evaluated with simulations on an existing commercial network topology consisting of 25 nodes and 61 uniform capacity links. One variant of the algorithm [36], using greedy admission control for permanent virtual circuits, was shown to route 6.5% to 15% more bandwidth[5] than an algorithm choosing the shortest path (`min-hop`), up to 25% more bandwidth than an algorithm which chooses the path with minimum maximum congestion, and 60% more bandwidth than an algorithm which chooses routes randomly. Another variant of the exponential algorithm [37], using the non-greedy admission control method of [9] for switched virtual circuits and improved using results from stochastic analysis, was shown to outperform `min-hop` with greedy admission control in a number of scenarios. These results support a theoretical result showing that any algorithm using greedy admission control is $\Theta(n)$ competitive with respect to throughput [36].

Other authors have studied variants of the routing and admission control problem. Competitive online algorithms that allow requests to be preempted have been designed for a single link (which models a virtual path) and for linear arrays [34, 35, 62, 19]. Kamath,

[5]The routed bandwidth is measured when, for the first time, more than 50% of a set of requests is rejected.

Palmon, and Plotkin [45] studied a model with Poisson distributed arrival times and exponentially distributed durations, and gave an algorithm that achieves an expected rejection ratio within an additive constant of optimal if a granularity condition similar to (1.2) is satisfied. Feldmann, et al. [33] considered a model that allows delay before a route assignment. They showed that if the maximum allowed delay is bounded, then no improvement is possible: the competitive ratio of any algorithm is $\Omega(\log n)$ with respect to the admission ratio (the ratio of the profit gained by admitted requests to the total profit of all requests), matching a previous lower bound on a linear array with no delay allowed [12, 62]. However, if the maximum delay is unbounded and all requests (with unknown duration) must be routed, they showed that a batch-style algorithm is $\Theta(\log n)$ competitive with respect to makespan on binary trees with unit capacities and $O(\sqrt{n})$ competitive on bounded degree planar graphs.[6] In contrast to the $\Omega(\log n)$ lower bound for the bounded delay case, the authors could only provide a lower bound of $\Omega\left(\frac{\log\log n}{\log\log\log n}\right)$ on the competitive ratio of any algorithm for the unbounded delay case on a full binary tree. For linear arrays, they presented algorithms with constant competitive ratios.

Distributed Virtual Circuit Routing

Awerbuch and Azar [7] studied a distributed resource allocation problem of which distributed virtual circuit routing is a special case. The problem is described by a set of clients $\mathcal{X}$ and a set of servers E. Associated with each server $e \in E$ is a capacity $c(e)$, and associated with each client $j \in \mathcal{X}$ is a demand r_j and a set $\mathcal{P}_j$ of feasible subsets of E. A client j

[6]Shmoys, Wein, and Williamson [85, 86] show how any batch-style scheduling algorithm can be made into a fully online algorithm (allowing delay) with competitive ratio increased by at most a factor of two. Feldmann, et al. [33] refine this result for their algorithms by demonstrating the existence of fully online versions with competitive ratio increased by only an additive one.

may be satisfied by simultaneously assigning its entire demand to each server in some set $P_j \in \mathcal{P}_j$. In terms of virtual circuit routing, each client j corresponds to a virtual circuit request with bandwidth requirement r_j and feasible routes $\mathcal{P}_j$. Two versions of the problem were studied. In the first, the goal is to maximize throughput and in the second, the goal is to minimize the makespan.

In the first problem, the goal is to maximize the throughput by accepting a maximal subset of clients that can be scheduled concurrently without exceeding capacity constraints. The authors relaxed the problem to a fractional version where a client may distribute its load over all the feasible subsets. An algorithm for the fractional version may be converted into a randomized algorithm for the integral version as long as a granularity condition is satisfied with respect to the maximum demand. Without this assumption, no approximation algorithm exists unless P=NP. Awerbuch and Azar designed an algorithm for the fractional problem that is $O(\log n)$ competitive with respect to throughput and $O(\log n)$ competitive with respect to the time needed to complete the assignment, provided the number of routes for each request is polynomial in n, where $n = \max\left\{|\mathcal{X}|, |E|, \frac{\max_e c(e)}{\min_e c(e)}\right\}$.

In the makespan minimization case, the goal is assign to each client j a feasible subset $P_j \in \mathcal{P}_j$ and a time $T(j)$ so that the clients complete in the least amount of time and do not exceed capacity constraints. This version is relaxed in two ways. First, it is not required that all edges in a feasible set be scheduled simultaneously. If a granularity condition holds, then a randomized algorithm for the minimum makespan case can be constructed from an algorithm for this relaxed load (congestion) minimization version. The load minimization problem is relaxed to a fractional version as in the maximum throughput case. They gave an algorithm for the fractional version of the problem which is $O(\log n)$ competitive with respect to load and O($\log^3 n$) competitive with respect to time. Simply stated, the algorithms work in phases, progressively assigning the demand to the servers in progressively larger pieces, until either the demand is satisfied or an exponential weight is exceeded.

1.3.3 Online Load Balancing

The congestion minimization version of the online virtual circuit routing problem is a generalization of an online load balancing problem in which each job in a sequence must be assigned to one of n independent, parallel machines. Each machine m_i has a *speed* $s(i)$ which reflects its processing power. In general, job j is specified by a tuple $(\vec{p}(j), a_j, b_j)$ where $\vec{p}_j$ is the job's load vector, a_j is the job's arrival time, and b_j is its finishing time. Each component $p_i(j)$ of the load vector is the weight of the job if assigned to machine i. In the *identical machines* case, the weight of every job is the same for all machines and is denoted w_j. In the *identical machines with assignment restriction* case, each $p_i(j)$ is either equal to a constant w_j or ∞. In the *related machines* case, $p_i(j) = \frac{w_j}{s(i)}$. When a job arrives at the scheduler it must be assigned immediately to exactly one machine, whose load is increased by exactly $p_i(j)$ for the duration of the job. In non-preemptive load balancing, a job, once assigned, may not be reassigned to another machine. On the other hand, if preemption is allowed, jobs may be reassigned during their execution. The goal of any algorithm is to minimize the maximum load on the machines at any given time.

We point out that the literature on online load balancing distinguishes itself from *online machine scheduling* [85, 86]. In the latter problem, each job is specified by an arrival time and a load. The existence of a job becomes known upon arrival, but its load is not known until the job is completed. The key difference between this model and online load balancing of jobs with unknown durations is that when a job need not be scheduled immediately upon arrival. The goal is to minimize the makespan of the schedule. If preemption is allowed, jobs may be canceled and completed at a later time, possibly on another machine, without penalty. In a non-preemptive model, free reassignments are not allowed, but jobs may be canceled and restarted at a later time.

Permanent Jobs

Permanent jobs are assumed to execute forever. When machines are identical, the load balancing problem with permanent jobs is the classical machine scheduling problem for which Graham [39] proved that the greedy algorithm which assigns each job to the machine with the least load is $(2 - \frac{1}{n})$ competitive. Recently, new algorithms with constant (for all n) competitive ratios strictly less than two were designed by Bartal, Fiat, Karloff, and Vohra [20] and Karger, Phillips, and Torng [46]. For the identical machines with assignment restriction case, Azar, Naor, and Rom [17, 18] proved that the greedy algorithm is $\lceil \log n \rceil + 1$ competitive. They further showed that this bound is tight, up to an additive 1, by proving a lower bound of $\lceil \log(n+1) \rceil$ for any algorithm. They also designed a randomized algorithm which is $\ln n$ competitive against an oblivious adversary, and proved this result is tight as well. For the related machines case, the greedy algorithm is also $\Theta(\log n)$ competitive, but Aspnes, et al. [4] showed that a non-greedy algorithm is 8 competitive. For the most general case where the load vector is unrestricted, Aspnes, et al. [4] designed a $O(\log n)$ competitive algorithm which is essentially the virtual circuit routing algorithm EXP_ROUTE applied to load balancing. The greedy algorithm in this case is $\Theta(n)$ competitive.

Jobs with Unknown Duration

Jobs with unknown duration arrive and depart the system at arbitrary times. The departure time of a job is not known to an algorithm when the job arrives and becomes known only after the job departs. For the identical machines case, the greedy algorithm of Graham [39] is still $\left(2-\frac{1}{n}\right)$ competitive. Azar and Epstein [15] proved that this algorithm is optimal. However, for identical machines with assignment restriction, Azar, Broder, and Karlin [14] showed that any algorithm, deterministic or randomized, must be $\Omega\left(\sqrt{n}\right)$ competitive, and that the greedy algorithm is $O\left(n^{2/3}\right)$ competitive. A $\Theta\left(\sqrt{n}\right)$ competitive algorithm was later discovered by Azar, et al. [16]. Phillips and Westbrook [79] designed two algorithms that use preemption to surpass the $\Omega\left(\sqrt{n}\right)$ lower bound for the special case in which each $p_i(j)$ is either 1 or ∞. The first algorithm performs ρ amortized reassignments per job and is $O\left(\frac{\log n}{\rho}\right)$ competitive. The second algorithm improves the total running time of the first algorithm at the expense of the load. Awerbuch, Azar, Plotkin, and Waarts [10] designed an algorithm which is 16 competitive and performs at most $O(\log n)$ reassignments per job, if the optimal load is at least $\log n$. Westbrook [96] later extended the algorithm to remove this requirement. For the related machines case, Azar, et al. [16] designed a 20-competitive deterministic algorithm. There is no known $o(n)$ competitive algorithm for the unrelated machines case. However, if preemption is allowed, the online virtual circuit routing algorithm of Awerbuch, et al. [10] can be applied to achieve a $O(\log n)$ competitive ratio with $O(\log n)$ reassignments per job.

Jobs with Known Duration

Nothing is known about whether knowing the durations helps in the cases of identical machines with assignment restriction or related machines. However, for the unrelated machines case, Azar, et al. [16] designed a $O(\log nT)$ competitive algorithm, where T, known in advance, is the ratio of the maximum to minimum duration. This algorithm is essentially a special case of the algorithm for routing switched virtual circuits with known durations.

1.3.4 Packet Routing and Scheduling

The problem of efficiently moving packets along paths in a network is important both in general purpose store-and-forward networks and in large-scale general-purpose parallel computers, where typically a large proportion of time is spent moving data among processors. There are many variations of the packet routing problem; the one we adopt here is taken from Leighton, et al. [59] and subsequent literature.

At each network node there are two queues, an *initial queue* and a *final queue*. In a static routing problem, all packets initially (at time 0) reside in initial queues at various nodes of the network. The most common type of static problem is a k-k routing problem in which each node initially contains k packets which are sent to arbitrary destinations. At the end of the routing, each node contains k packets. The $1-1$ routing problem is also called a permutation routing problem. On the other hand, in a dynamic problem, packets may arrive in initial queues over time. In either the static or dynamic case, the packets must be moved toward their destinations by using a store-and-forward routing scheme. Every edge in the network has at its head an *edge queue* which is used to temporarily store packets being forwarded to another node. At each time step, an algorithm residing at a node must decide whether each packet in an edge queue should wait in the queue or be forwarded to the edge queue at the head of the next edge on its path. In some versions of the problem, routes and/or schedules (packet forwarding times) may be decided at the source and simply read from packet headers by the nodes. At most one packet (or, in general, w packets) may traverse any edge at any time. The goal of an algorithm is to deliver all packets to the final queues at their destinations quickly and using small edge queues. The packet routing problem has been studied extensively in many forms [55, 54]; we will describe only a few results here that most closely related to our work.

Most work on packet routing deals with specific network topologies that are common in large-scale parallel architectures (e.g., arrays, trees, hypercubes, butterfly networks). However, in recent papers [59, 57, 56, 58], the problem is considered on more general networks and the work is divided into two distinct stages. In the first stage, a path is chosen for each packet connecting its source and destination. The goal of this stage is to find a set of paths with small congestion μ and dilation d. For some networks including meshes, butterflies, and shuffle-exchange networks, stage one can be accomplished easily using a method described by Valiant [93] in which a packet is first routed to a random intermediate destination before it is routed to its final destination. This method gives a set of paths with values of μ and d that are within a small constant factor of the diameter of the network. In the second stage, a schedule is constructed describing the movement of packets along their predetermined paths. The goal of this stage is to both minimize the length of the schedule and the maximum queue size in terms of the congestion and dilation of the paths.

General Networks

Leighton, Maggs, and Rao [59, 57] proved that a schedule exists with length $O(\mu + d)$ for any static routing problem on an arbitrary network, using constant size queues. Leighton and Maggs [58] later showed how to find such a schedule offline in $O(k|E| + |E| \log^{\epsilon}(k|E|))$ time for any fixed $\epsilon > 0$, where k is the number of packets. Leighton, Maggs, and Rao also described a randomized local control algorithm for arbitrary networks that produces a schedule of length $O(\mu + d \log(kd))$ using queues of size $O(\log(kd))$ with high probability. An improved local control algorithm was given by Rabani and Tardos [81] which routes all packets in $O(\mu) + (\log^* k)^{O(\log^* k)} d + \log^6 k)$ steps with high probability. One year later, Ostrovsky and Rabani [78] presented an algorithm that delivers all packets to their destinations in just $O(\mu + d + \log^{1+\epsilon} k)$ steps with high probability.

Layered Networks

Leighton, Maggs, Ranade, and Rao [59, 56] studied the static problem on the class of bounded degree layered networks. For an arbitrary L-layer network, they designed a randomized algorithm that constructs a schedule of length $O(\mu + L + \log k)$ with high probability and requires only constant size queues. They were able to apply their result to meshes, butterflies, shuffle-exchange networks, multidimensional arrays and hypercubes, and fat-trees by reducing the problem of routing on the original network to the problem of routing on a similar layered network.

Linear Arrays and Rings

The static k-k routing problem on a linear array can be solved optimally by sending each packet toward its destination immediately and giving priority to packets with the farthest distance yet to travel. This strategy produces a schedule with makespan $\max_{i<j}\{j-i+\mu(i,j)\}-1$ in the worst case, where $\mu(i,j)$ is the number of packets passing through nodes i and j [49]. Kaufmann and Sibeyn [51] showed that *any* priority scheme for a static problem will result in a packet being delayed at most μ times.

Makedon and Symvonis [64] studied a special case of the k-k packet routing problem on ring networks in which all k packets originating at a node have the same destination. Their algorithms are ***oblivious*** in the sense that they choose the route for each packet at the source and then forward packets locally. The goal of an algorithm is to come close to achieving the worst case lower bound with respect to makespan (a lower bound on the makespan of a worst case instance) for any instance. The worst case lower bound for a ring is at least $\max\{\frac{n-1}{2},\frac{nk}{4}\}$, where n is the number of nodes. (The first term in the bound is the maximum path length. The second term results from an instance in which all $\frac{n}{2}$ nodes on one half of a ring send their k packets to nodes on the other half of the ring. Then all $\frac{nk}{2}$ packets must cross one of two links, resulting in congestion at least $\frac{nk}{4}$ on one of the links.) For the case where $k=2$, Makedon and Symvonis proved a new lower bound of $\frac{2n}{3}$ and designed an algorithm which matches the bound. For the case where $k>2$, they designed an algorithm that constructs a schedule with makespan at most $\frac{kn}{4}+\frac{5n}{2}$. The algorithm divides the packets initially at a node between the packets' two routes depending on the shortest distance the packets must travel. Packets are forwarded using a FIFO policy. On the other hand, the greedy algorithm that assigns every packet to its shortest route requires at least $k\lfloor\frac{n}{2}\rfloor$ steps in the worst case. (If every node sends k packets to the node $\lfloor\frac{n}{2}\rfloor$ nodes to the right, then all packets are sent in the same direction and a bisection argument shows that at least $k\lfloor\frac{n}{2}\rfloor$ packets must cross any link.)

Kaufmann and Sibeyn [50] designed deterministic oblivious algorithms for the general k-k routing problem on a ring. In each of their algorithms, packets that have the furthest yet to travel are given priority. In addition to proving several new lower bounds for algorithms possessing varying amounts of knowledge, they designed and analyzed five algorithms with varying degrees of optimality for different values of k. The main result was an algorithm that requires at most $\frac{kn}{4} + \sqrt{n}$ steps, beating the algorithm of Makedon and Symvonis [64]. The algorithm was simplified in [88] and better algorithms were given for the cases where $k = 2, 3$. Sibeyn [88] also considered the dynamic problem and designed an algorithm that requires at most $\frac{kn}{3} + \frac{n}{3}$ steps in the worst case to route a k-k distribution. The algorithm sends each packet along its shortest route if the length of that route is at most $\frac{n}{3}$. Other packets are alternately directed clockwise and counter-clockwise toward their destinations. To our knowledge, no lower bound on makespan is known for the dynamic problem.

Trees

Symvonis [91] designed two offline algorithms for the permutation routing problem on trees. The model specifies that each queue has capacity one and that during any time step all edges can carry at most one packet in each direction. The first algorithm requires $O(n^2)$ time and $O(n^2)$ space to construct a schedule with length at most $n - 1$ (which is worst case optimal). The second algorithm requires $O(n^3)$ time and $O(n \log n)$ space to construct a schedule with length at most $n-1$. This second algorithm constructs *direct routes*: once a packet leaves its source, it moves one link closer to its destination with each step. Symvonis [90] also designed an online, local control algorithm for trees by adapting the greedy online algorithm to use less buffer space and still complete in $n-1$ steps. (The greedy algorithm can require queues as large as $\frac{n}{2}-2$.) Alstrup, et al. [2] gave an offline algorithm for direct routing on trees that improves upon the algorithm given in [91]. Their algorithm requires $O(n \log n \log\log n)$ time and $O(n \log n)$ space to construct a schedule with length at most $n-1$.

Greedy Packet Scheduling

Some interesting results have been shown about the efficiency of simple greedy queuing policies. A queuing policy is greedy if it always forwards a packet when there is an available link on which to send it. Cidon, Kutten, Mansour, and Peleg [30] showed that if the set of paths is layered, then any greedy policy guarantees that each packet p_j arrives at its destination within $a_j + d_j + k - 1$ steps, where a_j is the packet's arrival time, d_j is the number of links packet p_j traverses, and k is the total number of packets. This result can be extended to the case of ***unique subpaths*** where, for every pair of nodes, whenever two different routes contain a segment joining the pair, the two segments are identical. Cidon, Kutten, Mansour, and Peleg [30] proved that when each packet is routed over a ***shortest path***, the specific greedy strategy that forwards the packet which has traveled the least distance so far guarantees that every packet arrives within $d_j + k - 1$ time units. Mansour and Patt-Shamir [68] generalized this result to ***any*** greedy strategy. They showed that a greedy schedule is guaranteed to deliver packet j in no more than $d_j + \lfloor \frac{k-1}{w} \rfloor$ time units where w is the maximum number of packets that can traverse a link simultaneously. Greedy policies do not guarantee such bounds on an arbitrary set of paths; Cidon, et al. [30] proved that some greedy strategies may produce schedules with length $\Omega\left(d\sqrt{k}+k\right)$.

1.3.5 Multi-Operation Job Scheduling

Mao [70] studied a related multi-operation job scheduling problem. A job shop contains l different types of identical machines $T_1, T_2, \ldots, T_l$. Each job must be executed first by a machine of type T_1, then by a machine of type T_2, etc., and finally by a machine of type T_l. However, not all machine orders are possible for all jobs; a problem instance includes a set of *feasible machine orders* that apply to all jobs, and (possibly infinite) processing times for each job/machine pair. Conceptually, the machines in this problem are analogous to links in a routing problem on a layered network with l layers, and the feasible machine order and processing times together are analogous to a path of length l. However, we note that this problem formulation is not the equivalent to the routing problem for two reasons. First, the goal of the problem is to minimize makespan rather than congestion. Second, there are instances of job scheduling that do not correspond to any network routing problem.

1.4 Outline

1.4.1 Online Permanent Virtual Circuit Routing

As discussed in the introduction, we are interested in the online permanent virtual circuit routing problem in which all network links have the same capacity. We do not believe that the unit capacity assumption causes significant loss of generality in real situations, especially if we consider backbone networks. Recall that Awerbuch, et al. [4, 5] proved that the competitive ratio of any online virtual circuit routing algorithm for this case is at least $\frac{\log n-1}{2}$. This lower bound also applies to randomized online algorithms against an oblivious adversary. In Chapter 2, we offer an alternative lower bound proof that uses a different technique to show that any online algorithm is at least $\frac{\log n+3}{2}$ competitive, slightly improving the previous lower bound by an additive factor of two. Our construction requires a network with fewer nodes and links, and approximately half as many requests to force any online algorithm to incur the same congestion. Our lower bound result, in terms of the number of requests, is tight in the sense that there exists a class of instances whose competitive ratios are at most within an additive 1 of this lower bound. Our lower bound applies to randomized online algorithms against an adaptive online adversary rather than against an oblivious adversary.

In Chapters 3 and 4, we analyze the competitive ratio of two greedy virtual circuit routing algorithms. We are motivated primarily by the results of Mao and Simha [72], Aspnes, et al. [4, 5], and Azar, Naor, and Rom [17, 18]. As discussed previously, Mao and Simha considered greedy online algorithms for their shared model of file transfer routing and scheduling, which is essentially identical to the congestion minimization model of permanent virtual circuit routing. They showed that their best algorithm, LS3, is $\Omega\left(\sqrt{k}\right)$ competitive (and, implicitly, $\Omega\left(\sqrt{m}\right)$ competitive where m is the number of network links), leaving open the upper bound and the question of whether there exist better greedy algorithms. Aspnes, et al. [4, 5] showed that the non-greedy (in the strict sense) algorithm Exp_Route is asymptotically optimal (with competitive ratio $O(\log n)$) for permanent virtual circuit routing. Exp_Route is a generalization of an online algorithm for load balancing permanent jobs in the unrelated machines case. Since the greedy algorithm for this load balancing problem is $\Theta(n)$ competitive, so is any greedy algorithm for the most general case of virtual circuit routing in which the load applied to a link by a request is independent of the link's capacity. However, the greedy load balancing algorithm is $\Theta(\log n)$ competitive in the identical machines with assignment restriction and related machines cases [17, 18, 4, 5]. This raises a natural question: is a generalization of the greedy algorithm similarly competitive for virtual circuit generalizations of these load balancing cases (which are those with practical applications)? The problem we consider is a generalization of the problem of load balancing permanent jobs on identical machines with assignment restriction, studied by Azar, Naor, and Rom [17, 18].

The real motivation behind the greedy algorithms is the possibility of reducing the control overhead involved in virtual circuit routing. EXP_ROUTE is capable of making excellent routing decisions but requires the computation of many exponential functions with real bases and exponents for each decision. We wonder whether algorithms that require fewer computational resources can also demonstrate sufficient efficiency in at least some non-trivial cases. These issues and tradeoffs between computational and routing efficiency will be discussed in more detail in the sequel.

In Chapter 3, we will improve the lower bound given by Mao and Simha [72] for LS3 (calling it GREEDY_ROUTE1) and provide an upper bound with respect to congestion that is tight in many cases, including the ones considered in [72]. Specifically, we will prove that, when the bandwidth requirements of requests are approximately equal (or when various other conditions hold), the competitive ratio of GREEDY_ROUTE1 is $\Theta\left(\sqrt{\mathcal{D}m}\right)$ on arbitrary networks, where $\mathcal{D}$ is the maximum over all requests of the ratio of the lengths of the longest to shortest path for the request and m is the number of network links. We also provide a lower bound for layered networks (in which case $\mathcal{D} = 1$) which shows that the competitive ratio is $\Theta(\sqrt{m})$ in this case. When bandwidth requirements are arbitrary, the upper bounds are increased by a factor of $O\left(\sqrt{\mathcal{L}}\right)$, where $\mathcal{L}$ is the ratio of the maximum to minimum bandwidth requirement, although we suspect that this term does not belong in the true competitive ratio.

In Chapter 4, we analyze a better greedy algorithm called GREEDY_ROUTE2, and show that it is (approximately) $O(d \log n)$ competitive, where d is the length of the longest path assigned to a request.[7] Therefore, for networks with relatively short paths relative to n, this algorithm performs well; if $d = O(\log n)$, the competitive ratio is polylogarithmic and if $d = O(1)$, the competitive ratio is asymptotically optimal. We also prove a lower bound of $\Omega(d + \log(n - d))$ for arbitrary networks and $\Omega\left(d + \log\left(\frac{n}{d} - d\right)\right)$ for layered networks. We stress that the upper bounds for both GREEDY_ROUTE1 and GREEDY_ROUTE2 apply to *arbitrary* topology networks, i.e., GREEDY_ROUTE2 is polylogarithmically competitive for *any* network with $d = O(\log n)$ rather than just those with a specific topology.

The analyses of GREEDY_ROUTE1 and GREEDY_ROUTE2 answer an interesting question about the generalization of the greedy algorithm to routing. On the one hand, the greedy load balancing algorithm, when applied to routing, is no longer optimal, having a competitive ratio in $\omega(\log n)$ for general networks. On the other hand, it may perform well in some cases. The primary reason that the greedy algorithms can perform poorly is that they can choose routes that are unnecessarily long, thereby increasing the congestion on too many

[7]Our upper bound holds if the optimal route assignment has a small degree of overlap. We discuss this further in Chapter 4.

links. Taking this into account, the factor of d in the competitive ratio of GREEDY_ROUTE2 makes intuitive sense.

We can apply our virtual circuit routing algorithms, with identical results, to a variant of the multi-operation job scheduling problem [70] in which each processing time is either unit or infinite (an infinite processing time means that we cannot assign that job to that machine), and the goal is to minimize the maximum load. (This is true because our analyses do not depend on the fact that routes are paths in the network; they could be arbitrary sets of network links.) In this problem, each job is assigned to each machine in a feasible machine order permanently. For this problem, GREEDY_ROUTE2 is $O(l \log n)$ where l is the number of machine types. If the number of machine types is constant (or small relative to the total number of machines), then this is a good bound.

We should mention that GREEDY_ROUTE2 is identical to the min-max algorithm simulated in [36], and compared to the min-hop algorithm[8] and a variant of EXP_ROUTE. In these simulations the goal was to maximize throughput rather than minimize congestion. GREEDY_ROUTE2 was shown to route up to 25% less throughput than the exponential algorithm. But the simulated network contains many routes with lengths close to n. (By inspection, it is easy to see that, for most source/destination pairs, there is a simple path that contains at least 75% of the network nodes.) We suspect that the simulation results would be different in a network with shorter paths.

[8]As an aside, we point out that the competitive ratio of the min-hop algorithm can be arbitrarily bad with respect to congestion. An adversary can supply an arbitrarily long sequence of requests between a pair of directly connected nodes for which there are many longer alternative routes. While the min-hop algorithm will always assign every request to the direct route, an optimal algorithm will appropriately spread out the load over all the routes. This idea still holds for an algorithm that minimizes congestion over all shortest paths.

1.4.2 Online Packet Routing and Scheduling

As indicated earlier, there have been several different packet routing and scheduling models studied in the literature. The models can essentially be categorized based upon four characteristics:

1. whether the route is assigned at the source or piecemeal during the packet's lifetime
2. whether the schedule is assigned at the source or piecemeal during the packet's lifetime
3. whether a routing and scheduling algorithm (at the source or distributed among the nodes) has only local knowledge or has (or can obtain) global knowledge
4. whether a routing and scheduling algorithm has knowledge of future requests (offline) or not (online/dynamic)

In the following discussion, we will refer to a model with the notation `1/2/3/4`. `1` and `2` will be replaced by either `S` or `P`, depending upon whether the routing and scheduling decisions are made at the source or in a piecemeal fashion, respectively. In some models, routes are assumed to be given and decoupled from the rest of the problem, or scheduling is not relevant (as in a circuit-switched model); in these cases, we will write `-` in place of `1` or `2`, as appropriate. `3` will be replaced by either `L` or `G`, depending upon whether the model uses local or global information. `4` will replaced by `Off` or `On`, depending upon whether the algorithms in the model have knowledge of future requests.

Different models are appropriate for different situations. For instance, piecemeal routing and scheduling decisions are often good because decisions are deferred until the last minute and such algorithms usually require very little overhead at the source. On the other hand, such schemes cannot really prevent congestion; they can only deal with it when it occurs so that packets are not lost. Global algorithms can prevent collisions and congestion, and decrease switch complexity. However, these algorithms require more overhead at the source and the global information may be hard to gather.

At one end of this spectrum are the `S/S/G/Off` model studied by Mao and Simha [72] and Rivera-Vega, et al. [82, 83, 95] and the `-/S/G/Off` model studied on general networks by Leighton, Maggs, and Rao [59, 57]. Of the literature cited earlier, some of the work on trees also falls into the latter category [91, 2]. At the other extreme are the traditional distributed permutation routing problems (`P/P/L/Off`) and dynamic routing problems (`P/P/L/On`). There are also distributed models that decouple routing from the scheduling (`-/P/L/On`) [56, 30, 68]. The oblivious algorithms for k-k routing on ring networks fall into the `S/P/L/Off` and `S/P/G/Off` models [64, 88]. The online virtual circuit routing problems can be written as `S/-/G/On` [4, 9] or `S/-/L/On` [7].

We are interested in the `S/S/G/On` model. We would like to be able to assign source routes and schedules to arbitrary packets arriving at arbitrary times to minimize makespan. This model was inherent in the algorithms studied by Mao and Simha [72] which scheduled files according to a *greedy, fixed priority schedule*. A file transfer's priority is its index in the file transfer sequence. The schedule is greedy in the sense that a file transfer is only held at a node if it is blocked by files with higher priorities (smaller indices). This was also the idea in the so-called adaptive path scheduling heuristic of Rivera-Vega, et al. [82].

There are few reasons why this model is interesting, besides the extension of the work cited above. First, from an engineering point of view, a method such as this helps to reduce the complexity of network nodes. Reducing overhead is important and is probably why many practical parallel computers use simple non-optimal routing algorithms [92]. Second, most packet routing algorithms are either studied with respect to static permutation routing problems or stochastic modeling. Dynamic problems with arbitrary arrivals are less common and pose different challenges with respect to algorithm analysis. For example, one cannot assume packets are evenly distributed throughout the network as one can with a permutation problem. Sibeyn [88, 87], for one, mentioned that future research on the dynamic problem on rings is important.

The canonical efficiency measure of a packet routing and scheduling algorithm in the literature is how close it comes to being able to schedule every instance in worst case optimal time. The worst case optimal makespan for a problem is the number of steps required by an optimal algorithm to move every packet in a worst case instance to its destination. A stronger result would be one analogous to a competitive ratio: if an algorithm has competitive ratio (or performance ratio) c then it can schedule *every* instance within c times the optimal number of steps for that instance (not just the worst case instance). This sentiment was voiced by Kaufmann and Sibeyn [50], who expressed interest in showing that their algorithms were not only worst case optimal, but also optimal for each particular instance.

In Chapter 5, we will revisit the greedy, fixed priority scheduling algorithm (GFP) and prove a few results with respect to competitive ratio on some simple networks with unbounded queues. Specifically, we analyze linear arrays, trees, and ring networks. For linear arrays (and other networks whose nodes have indegree one) we show that the competitive ratio of GFP is $2-\epsilon$. We generalize the proof of this result to directed trees to show that the competitive ratio of GFP is $\Theta(\log n)$. Lastly, we consider full-duplex ring networks on which each request can follow one of exactly two routes. In this case, we show that GFP scheduling combined with minimum hop routing gives an algorithm that is at most $3 - \epsilon$ competitive. If all packets arrive at the same time and originate at the same node, then this algorithm is at 2 competitive. We show that the competitive ratio of any algorithm which always assigns a request to its shortest path if the length of the shortest path is at most βn for some $0 < \beta \leq \frac{1}{2}$ is at least $2 - \epsilon$. This includes the dynamic algorithm designed by Sibeyn [88]. This result implies that if there is an algorithm that is better than $2 - \epsilon$ competitive on ring networks, then it must be adaptive in some sense. But such an algorithm must also take into account path lengths; an algorithm that does not, such as GREEDY_ROUTE1 or GREEDY_ROUTE2, can be $\Omega(n)$ competitive.

Chapter 2

Lower Bounds

2.1 Introduction

In this chapter, we will provide lower bounds on the competitive ratio of any online routing and scheduling algorithm, with respect to both network congestion and makespan.[1] The lower bounds can be expressed in terms of the number of network nodes n, the number of network links m, or the number of requests k. The bound in terms of k is less useful since it only holds on sufficiently large networks. For this reason, in the sequel we will state upper bounds for specific online algorithms in terms of the number of nodes n and the number of links m.

The main result of this chapter can be formally stated as follows:

[1]A preliminary version of the results in this chapter appeared in [44].

Theorem 2.1

The competitive ratio of any deterministic online routing and scheduling algorithm is at least

(a) $\frac{\log n+3-\epsilon}{2}$ for arbitrarily small positive $\epsilon \leq \log 3$, with respect to congestion;

(b) $\frac{\log n+5-\epsilon}{4}$ for arbitrarily small positive $\epsilon \leq \log 3$, with respect to makespan;

(c) $\frac{\log m+2-\epsilon}{2}$ for arbitrarily small positive $\epsilon \leq \log \frac{3}{2}$, with respect to congestion;

(d) $\frac{\log m+4-\epsilon}{4}$ for arbitrarily small positive $\epsilon \leq \log \frac{3}{2}$, with respect to makespan;

(e) $\log k + 1$ with respect to congestion; and

(f) $\frac{\log k}{2} + 1$ with respect to makespan.

Furthermore, these results hold even when all requests have unit length[2] and share one destination[3], and the network is a simple 2-layer digraph with unit capacity links.

To prove Theorem 2.1, we will construct a network and a sequence of requests that force any online algorithm to incur a sufficiently high cost (congestion or makespan). In the proof, we will imagine a game between an arbitrary online algorithm and a malicious adversary. The adversary will give a network and then issue a corresponding sequence of requests, which must be answered by both the online algorithm and the adversary. The adversary's goal is to make the online algorithm perform as badly as possible. Our adversary will be *adaptive* and *online*: adaptive in the sense that it will issue each request f_j only after it has watched the online algorithm make its decision for request f_{j-1}, and online in the sense that it must make its own decision for request f_{j-1} before it issues request f_j. Before we present

[2]In this chapter, we will refer to requests as file transfers with unit length, rather than virtual circuit requests with unit bandwidth requirements.
[3]We can alternatively construct a 2-layer network with one source.

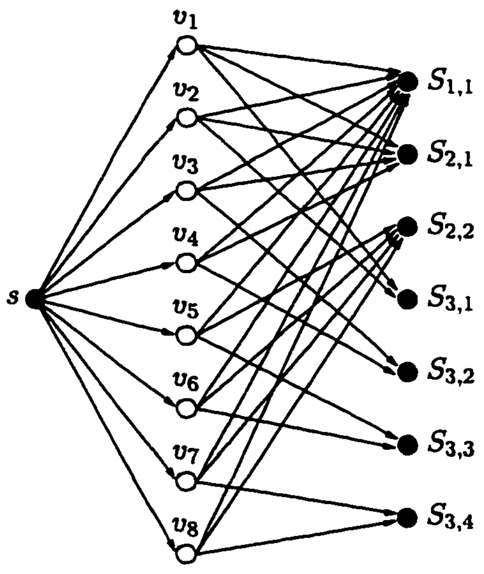


Figure 2.1: The network graph of Aspnes, et al. with $N = 8$.

this proof, however, we will discuss our results in the context of previous lower bounds for online routing.

2.2 Relationship to Previous Work

Aspnes, et al. [4] were the first to show that the competitive ratio of any online routing algorithm must be $\Omega(\log n)$ with respect to network congestion. For any value of N, where N is a power of 2, the lower bound instance of Aspnes, et al. consists of a directed 2-layer network with $n = 2N$ nodes — one source node, N intermediate nodes, and $N - 1$ destination nodes — and $m = N \log N$ links connecting source nodes to intermediate nodes and intermediate nodes to destination nodes. (An example of this network, with $N = 8$, is displayed in Figure 2.1.) The request sequence corresponding to this network contains $k = N - 1$ unit length requests, which force any online algorithm to incur congestion at

	Aspnes, et al. Construction	Our Construction
n	$2^{2\mu+1}$	$2^{2\mu-3}+3\cdot 2^{\mu-2}+1$
m	$2^{2\mu}(2\mu+1)$	$2^{2\mu-2}+2^{\mu-1}$
k	$2^{2\mu}-1$	2^{μ}

Table 2.1: A comparison of our lower bound construction and that of Aspnes, et al.

least $\frac{\log N}{2}$. On the other hand, an optimal algorithm can always achieve unit congestion. Therefore, their construction shows that the competitive ratio of any online algorithm must be at least $\frac{\log n-1}{2}$ and $\frac{\log(k+1)}{2}$ with respect to congestion.

From the statement of Theorem 2.1 above, we see that our lower bounds are better by $\frac{4-\epsilon}{2}$ in the first case and slightly more than a factor of two in the second. In other words, our construction requires a slightly smaller network and half as many requests to force any online algorithm to incur the same congestion. See Table 2.1 for a comparison of the values of n, m, and k in the two constructions, in terms of μ, the forced competitive ratio with respect to congestion.

Part (e) of Theorem 2.1 is tight (up to an additive one) for the class of permanent virtual circuit instances that correspond to permanent job load balancing instances on identical machines with assignment restriction [17, 18]. Each such load balancing instance can be transformed into a virtual circuit instance on a network similar to the one used in our proof of the lower bound.

In the context of randomized online algorithms, the result of Aspnes, et al. holds for randomized online algorithms against a weak oblivious adversary while our proof holds only against a stronger adaptive online adversary. (This is because their adversary provides a request sequence that is independent of the decisions of the online algorithm.) So while our result says nothing about randomized online algorithms against an oblivious adversary, it does show how the lower bounds for randomized online algorithms can be increased against an adaptive adversary.

2.3 Lower Bounds for Deterministic Online Algorithms

As discussed earlier, the proof of Theorem 2.1 will be framed as a game between an arbitrary online algorithm and a malicious adversary who supplies requests one at a time in response to previous decisions of the online algorithm. In the next two subsections, we will carefully define the network graph and the adaptive request sequences used to prove our result. We will follow these constructions with the formal proof of the theorem.

2.3.1 The Network

The network graph we use in the lower bound proof can be defined in two equivalent ways; we will describe both. The first definition defines the nodes and edges directly, while the second defines them recursively. While the first definition may be easier to visualize, the second is more convenient for our proof.

For any integer $i > 0$, we will define a network graph G_i. Every link in G_i will have unit capacity and every node will have arbitrarily large queues. Network graphs G_2 and G_3 are displayed in Figures 2.2 and 2.3, respectively.

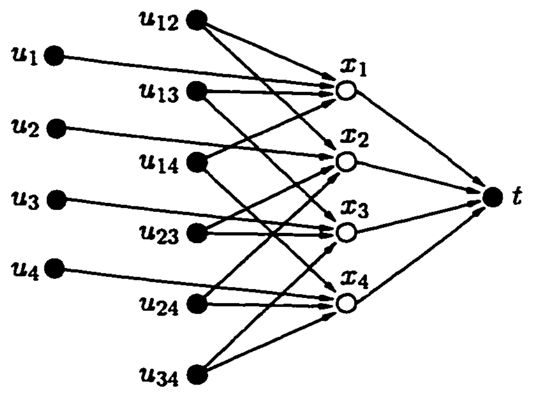


Figure 2.2: The network graph G_2.

A Direct Definition

To simplify notation, let $N = 2^i$. The vertex set of G_i is defined to be $S_i \cup X_i \cup T_i$ where

$$S_i = \{u_{ab}\colon 1 \leq a < b \leq N\} \cup \{u_a\colon 1 \leq a \leq N\}$$

is the set of source nodes,

$$X_i = \{x_1, x_2, \dots, x_N\}$$

is the set of intermediate nodes, and $T_i = \{t\}$ is the set containing a single destination node. Clearly, the number of nodes in G_i is

$$\begin{aligned} n &= |S_i \cup X_i \cup T_i| \\ &= \frac{N(N-1)}{2} + 2N + 1 \\ &= \frac{N^2}{2} + \frac{3N}{2} + 1. \end{aligned} \tag{2.1}$$

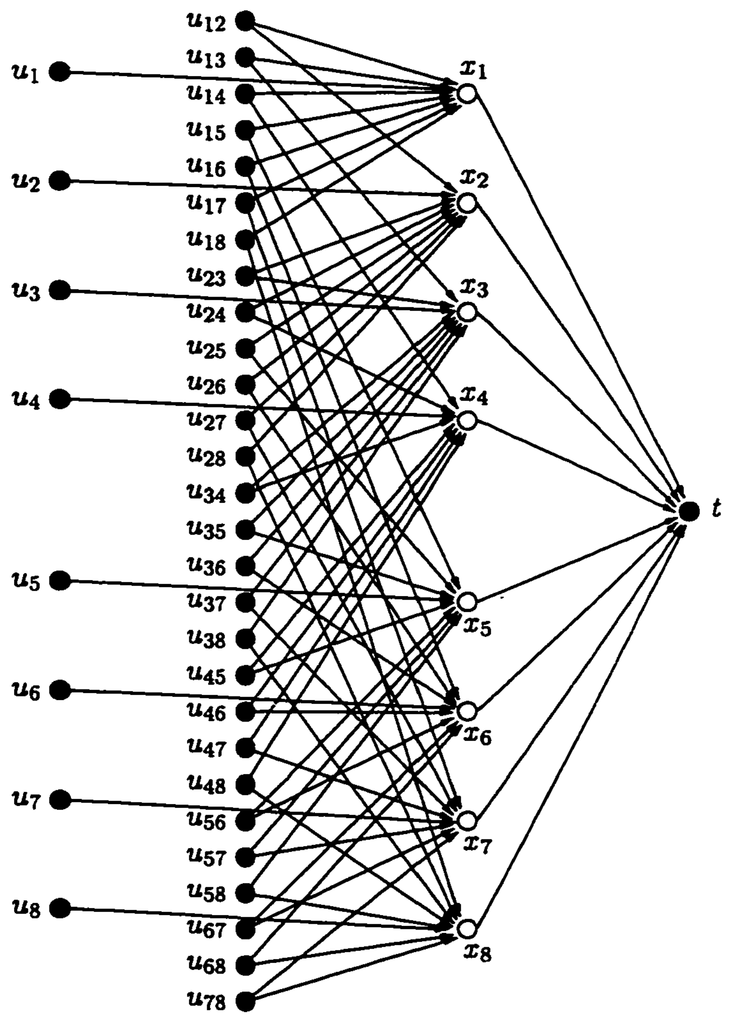


Figure 2.3: The network graph G_3.

The set of directed edges of G_i is defined to be the union $E_i^1 \cup E_i^2$ where

$$E_i^1 = \{(u_{ab}, x_a), (u_{ab}, x_b) \colon 1 \le a < b \le N\} \cup \{(u_a, x_a) \colon 1 \le a \le N\}$$

is the set of edges between the nodes in S_i and X_i, and

$$E_i^2 = \{(x_a, t) \colon 1 \le a \le N\}$$

is the set of edges between the nodes in X_i and T_i. Clearly, the number of edges in G_i is

$$\begin{aligned} m &= \left|E_i^1 \cup E_i^2\right| \\ &= N(N-1) + 2N \\ &= N^2 + N. \end{aligned} \tag{2.2}$$

A Recursive Definition

For the recursive definition, we will begin by defining G_0 to be the directed graph with three nodes — a source node u_1, an intermediate node x_1, and a destination node t — and two directed edges (u_1, x_1) and (x_1, t). For any integer $i > 0$, G_i is constructed recursively as follows:

1. Make two copies of G_{i-1}.
2. For any two intermediate nodes x_a and x_b, one in each G_{i-1}, add a source node u and edges (u, x_a) and (u, x_b).
3. Contract the two destination nodes in the two copies of G_{i-1} into one destination node t.

4. Relabel the intermediate nodes $x_1, x_2, \ldots, x_{2^i}$.

5. Relabel a source node u_{ab}, with $a < b$, if it is connected to intermediate nodes x_a and x_b. Relabel a source node u_a if it is only connected to intermediate node x_a.

Figure 2.4 illustrates the construction of network graphs G_0, G_1 and G_2.

2.3.2 The Adversary's Request Sequence

We observe that in any network graph G_i there are two possible routes that an algorithm can assign to a request between source u_{ab} and destination t: the route passing through intermediate node x_a or the route passing through intermediate node x_b. Which of the two is chosen depends entirely on a particular online algorithm A.

As discussed earlier, we will design an adaptive online adversary that will issue its requests based on earlier choices made by A. In order to define this adaptive request sequence, we will first define, for any online algorithm A, an operator $|_A$ as follows: $a|_A b$ returns a if A selects a route passing through intermediate node x_a when the adversary gives A the choice between a route passing through x_a and a route passing through another intermediate node x_b. Otherwise, $a|_A b$ returns b.[4] Hence, for a request between nodes u_{ab} and t on some network G_i, online algorithm A will assign route $\left\langle u_{ab}, x_{a|_A b}, t\right\rangle$. Note that $a|_A b$ is different from "a or b" in that once the algorithm makes its decision, the value returned from $a|_A b$ is fixed. In the sequel, $a|_A b$ will be denoted simply by $a|b$ when the specific online algorithm is clear in the current context.

[4]For any values of a and b, the request sequence in which the choice between x_a and x_b is given will be clear from the context, and the adversary will give A such a choice at most once in any request sequence. Therefore, no ambiguity will arise as a result of not specifying information about the memory of the algorithm as input to $|_A$.

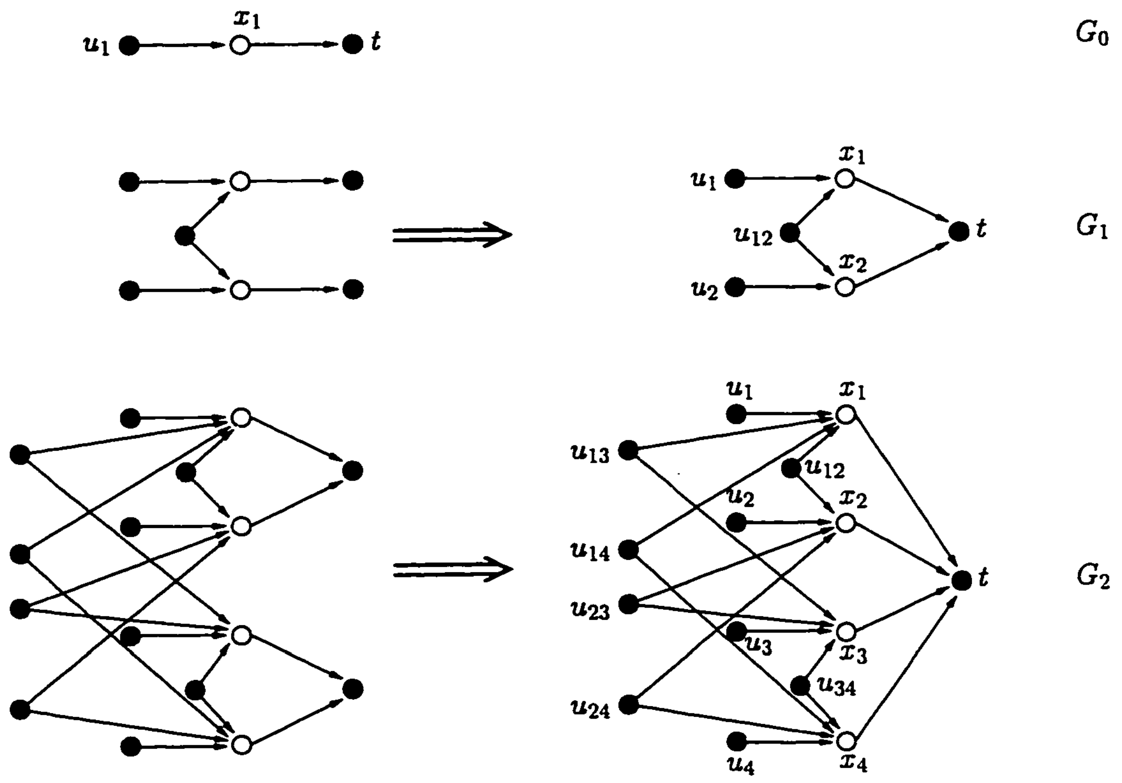


Figure 2.4: The network graphs G_0, G_1 and G_2, constructed recursively.

Each request f_j issued by the adversary will have length $l_j = 1$ and arrival time $a_j = 0$. (Even though the requests all arrive at the same time, we still require that they be assigned routes in the order they appear in the sequence. An alternative would be to assign arrival times that differ by an arbitrarily small amount. Given these model simplifications, in what follows, we will represent f_j simply by the pair (s_j, t_j).

For network graph G_i, the request sequence issued by the adversary to online algorithm A will be denoted $\sigma_i'(A)$ and will consist of a recursively defined sequence, denoted $\sigma_i(A)$, followed by one last request. We begin by defining $\sigma_i(A)$. The base case, $\sigma_1(A)$, issued to online algorithm A on network graph G_1, trivially contains just one request, (u_{12}, t). In general, to define $\sigma_i(A)$ for network graph G_i and online algorithm A, we first concatenate two copies of $\sigma_{i-1}(A)$, one for each G_{i-1} used in the construction of G_i, with labels of the source nodes modified accordingly. Let u_{ab} and u_{cd} be the source nodes of the last request in each of the two $\sigma_{i-1}(A)$ sequences. Then the last request in $\sigma_i(A)$ is $\left(u_{(a|_A b)(c|_A d)}, t\right)$.

The actual request sequence that the adversary will provide to an online algorithm A is $\sigma_i'(A)$. Let u_{yz} be the source node of the last request in $\sigma_i(A)$. Then request sequence $\sigma_i'(A)$ is defined to be $\sigma_i(A)$ followed by the single request $\left(u_{y|_A z}, t\right)$. For example, refer to Table 2.2 for request sequence $\sigma_3'(A)$. Notice that

$$k = |\sigma_i'(A)| = 2^i. \tag{2.3}$$

Before continuing, we present this simple lemma which will be needed in later proofs. The proof of the lemma is obvious from the construction of $\sigma_i(A)$.

$\sigma'_3(A)$	Request	Route assigned by online algorithm A
f_1	(u_{12}, t)	$\langle u_{12}, x_{1\|2}, t\rangle$
f_2	(u_{34}, t)	$\langle u_{34}, x_{3\|4}, t\rangle$
f_3	$(u_{(1\|2)(3\|4)}, t)$	$\langle u_{(1\|2)(3\|4)}, x_{(1\|2)\|(3\|4)}, t\rangle$
f_4	(u_{56}, t)	$\langle u_{56}, x_{5\|6}, t\rangle$
f_5	(u_{78}, t)	$\langle u_{78}, x_{7\|8}, t\rangle$
f_6	$(u_{(5\|6)(7\|8)}, t)$	$\langle u_{(5\|6)(7\|8)}, x_{(5\|6)\|(7\|8)}, t\rangle$
f_7	$(u_{((1\|2)\|(3\|4))((5\|6)\|(7\|8))}, t)$	$\langle u_{((1\|2)\|(3\|4))((5\|6)\|(7\|8))}, x_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, t\rangle$
f_8	$(u_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, t)$	$\langle u_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, x_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, t\rangle$

Table 2.2: The definition of $\sigma'_3(A)$ and the routes assigned by online algorithm A.

Lemma 2.1

Let i be any positive integer and let A be any online algorithm. Then, for any two requests $(u_{ab}, t), (u_{cd}, t) \in \sigma_i(A)$, $a \neq c$ or $b \neq d$.

2.3.3 Proof of the Lower Bounds

We will now examine the congestion and makespan of a solution produced by an arbitrary online algorithm A, as well as those of an optimal solution, using $\sigma'_i(A)$ as the request sequence on network graph G_i. As a result, we establish a lower bound for our problem with respect to both congestion and makespan.

Let us first consider the example of routing and scheduling the requests in $\sigma'_3(A)$ on network G_3. Since $i = 3$, there are $k = 8$ requests to transfer. As shown in Table 2.2, f_1 and f_2 use the parallel routes $\left\langle u_{12}, x_{1|_A 2}, t\right\rangle$ and $\left\langle u_{34}, x_{3|_A 4}, t\right\rangle$, respectively, causing the makespan to be 2 (2 hops for the requests to reach destination node t) and the congestion to be 1. For request f_3, we observe that A must choose a route that passes through a previously used link. This forces the makespan to reach 3 (since a link can only be used by one request during any time unit), and the congestion to reach 2. The situation with f_4, f_5 and f_6, in the lower half of the network, is analogous to that of the first three requests and leaves the overall makespan and congestion unchanged. When f_7 arrives, because of the way $\sigma'_3(A)$ and G_3 are designed, A has no choice but to assign a route passing through one of the two most heavily used links, forcing the makespan and congestion to increase to 4 and 3, respectively. Lastly, when f_8 arrives, A must assign it to its only route, which passes over the most heavily used link in the network, causing the makespan and congestion to increase to 5 and 4, respectively.

In general, we have the following lemma regarding the congestion incurred by A when presented with $\sigma_i(A)$. Lemma 2.2 will later be used to prove Lemma 2.3 which deals with the congestion and makespan resulting from $\sigma'_i(A)$.

Lemma 2.2

For any positive integer i and any online algorithm A, let (u_{yz}, t) denote the last request in the request sequence $\sigma_i(A)$ (on network graph G_i). Then $\mu\left(\left(x_{y|_A z}, t\right)\right) = i$.

Proof

The proof is by induction on i.

Base case ($i = 1$). When $i = 1$, the lemma holds since there is only one request in $\sigma_1(A)$ and that request is assigned route $\langle u_{12}, x_{1|2}, t\rangle$, causing congestion 1 on edge $(x_{1|2}, t)$. (To simplify notation, we use $|$ as shorthand for $|_A$ throughout the proof.)

Induction step ($i > 1$). By definition, the edge set of the network graph G_i contains two disjoint edge sets from two copies of G_{i-1} (with nodes relabeled). Let $k = |\sigma_i(A)| = 2^i - 1$. Then, also by definition, the first $k-1$ requests in $\sigma_i(A)$ consist of the concatenation of two $\sigma_{i-1}(A)$ sequences (with nodes relabeled), each of which contains requests whose potential routes are contained entirely in one copy of G_{i-1}. Let (u_{ab}, t) denote the last request in one $\sigma_{i-1}(A)$ sequence and let (u_{cd}, t) denote the last request in the other $\sigma_{i-1}(A)$ sequence. By the induction hypothesis and the fact that any path assigned to a request in one $\sigma_{i-1}(A)$ is edge disjoint from any path assigned to a request in the other $\sigma_{i-1}(A)$,

$$\mu_k\left((x_{a|b}, t)\right) = \mu_k\left((x_{c|d}, t)\right) = i - 1. \tag{2.4}$$

The last request in $\sigma_i(A)$ is, by definition, $\left(u_{(a|b)(c|d)}, t\right)$, to which A will assign the route $\langle u_{(a|b)(c|d)}, x_{(a|b)|(c|d)}, t\rangle$. Since this route contains edge $\left(x_{(a|b)|(c|d)}, t\right)$, which is either $\left(x_{a|b}, t\right)$ or $\left(x_{c|d}, t\right)$, by (2.4),

$$\mu\left(\left(x_{(a|b)|(c|d)}, t\right)\right) = \mu_{k+1}\left(\left(x_{(a|b)|(c|d)}, t\right)\right) = i. \qquad \blacksquare$$

Lemma 2.3

Let i be any positive integer and let A be any online algorithm. When given network graph G_i and request sequence $\sigma_i'(A)$, A will construct a solution with congestion at least $i+1$ and makespan at least $i+2$.

Proof

By definition, request sequence $\sigma_i'(A)$ is the concatenation of $\sigma_i(A)$ and request $\left(u_{y|z}, t\right)$, where u_{yz} is the source node of the last request in $\sigma_i(A)$. (As in the proof of Lemma 2.2, we use $|$ as shorthand for $|_A$.) Let $k = |\sigma_i'(A)| = 2^i$. From Lemma 2.2, we know that $\mu_k\left((x_{y|z}, t)\right) = i$. Thus, since $\langle u_{y|z}, x_{y|z}, t\rangle$ is the only route in G_i that can be assigned to request $(u_{y|z}, t)$,

$$\mu\left((x_{y|z}, t)\right) = \mu_{k+1}\left((x_{y|z}, t)\right) = i + 1.$$

So the network congestion is $i+1$. This being the case, the last request to cross link $(x_{y|z}, t)$ will do so at least $i+2$ time units after it arrived — at least 1 unit to reach node $x_{y|z}$ plus at least another $i+1$ units to reach node t, causing the makespan to be $i+2$. ■

Next we show that, for any online algorithm A, there exists a corresponding offline algorithm $\bar{A}$ that can assign routes to the requests in $\sigma_i'(A)$ (on the network graph G_i) and achieve makespan equal to 2 and congestion equal to 1. For each online algorithm A, $\bar{A}$ is defined to be the algorithm that chooses, whenever possible, the route *not* chosen by A.

The following lemma will be used to prove this result:

Lemma 2.4

Let i be any positive integer and let A be any online algorithm. Suppose that, for some request $f_j = (u_{ab}, t) \in \sigma_i(A)$, algorithm $\bar{A}$ chooses the route $\langle u_{ab}, x_\gamma, t\rangle$, where $\gamma \in \{a, b\}$. Then, for all requests $f_h = (u_{yz}, t)$, where $h > j$, $y \neq \gamma$ and $z \neq \gamma$.

Proof

The lemma follows from the recursive construction of $\sigma_i(A)$. According to the construction, once algorithm A decides to not choose (and $\bar{A}$ decides to choose) a route containing an intermediate node with index γ for a request f_j, no future requests can possibly have a source node with an index containing γ. ■

Lemma 2.5

Let i be any positive integer and let A be any online algorithm. The set of routes assigned by $\bar{A}$ to the requests in $\sigma_i(A)$ are pairwise edge disjoint.

Proof

From Lemma 2.1 we know that no link between a source node and an intermediate node can be contained in more than one route assigned by algorithm $\bar{A}$. To finish proving the lemma, we must show the same about the links between intermediate nodes and destination nodes. Consider an arbitrary link $e = (x_\gamma, t)$ that is contained in a route assigned by algorithm $\bar{A}$ to request $f_j = (u_{ab}, t) \in \sigma_i(A)$. By Lemma 2.4, we know that e cannot be contained in a route assigned by $\bar{A}$ to a future request. By applying Lemma 2.4 to each f_h, $h < j$, we see that no earlier requests could have had source nodes whose indices contain γ. Thus, e cannot be contained in a route assigned by $\bar{A}$ to an earlier request either. Thus, e is contained in the route assigned by algorithm $\bar{A}$ to request f_j and no other. ■

We are now ready to prove Theorem 2.1, restated here for completeness:

Theorem 2.1

The competitive ratio of any deterministic online routing and scheduling algorithm is at least

(a) $\frac{\log n+3-\epsilon}{2}$ for arbitrarily small positive $\epsilon \leq \log 3$, with respect to congestion;

(b) $\frac{\log n+5-\epsilon}{4}$ for arbitrarily small positive $\epsilon \leq \log 3$, with respect to makespan;

(c) $\frac{\log m+2-\epsilon}{2}$ for arbitrarily small positive $\epsilon \leq \log \frac{3}{2}$, with respect to congestion;

(d) $\frac{\log m+4-\epsilon}{4}$ for arbitrarily small positive $\epsilon \leq \log \frac{3}{2}$, with respect to makespan;

(e) $\log k + 1$ with respect to congestion; and

(f) $\frac{\log k}{2} + 1$ with respect to makespan.

Proof

From Lemma 2.5, we know that the congestion of the route assignment constructed by $\bar{A}$ is 1 and that all requests can be sent to their common destination t in 2 time units. Combining this fact with Lemma 2.3, we see that, for any value of $i \geq 1$, there exists a network graph and a request sequence for which the competitive ratio of any online algorithm is at least $i+1$ with respect to congestion and $\frac{i+2}{2}$ with respect to makespan.

To prove the lower bound on the competitive ratio in each part of the theorem, we will simply rewrite these quantities in terms of the appropriate variables to show that there exist arbitrarily large networks and corresponding request sequences which force any online algorithm to incur the lower bound.

(a) Recall that $N = 2^i$. From (2.1), we see that, for all $N \geq 2$,

$$n = \frac{N^2}{2} + \frac{3N}{2} + 1 = \frac{\gamma}{2}N^2$$

for some $1 < \gamma \leq 3$. Therefore, since $N = 2^i$,

$$i = \frac{\log\left(\frac{2n}{\gamma}\right)}{2} = \frac{\log n + 1 - \epsilon}{2} \tag{2.5}$$

for some $0 < \epsilon = \log\gamma \leq \log 3$. Notice that, as $i \to \infty$, γ approaches 1 and ϵ approaches 0. Therefore, there exist arbitrarily large network graphs (and corresponding request sequences) for which the competitive ratio of any online algorithm, with respect to congestion, is at least

$$i + 1 = \frac{\log n + 3 - \epsilon}{2}$$

for arbitrarily small positive ϵ.

(b) From (2.5), we can conclude that there exist arbitrarily large networks (and corresponding request sequences) for which the competitive ratio of any online algorithm, with respect to makespan, is at least

$$\frac{i+2}{2} = \frac{\log n + 5 - \epsilon}{4}$$

for arbitrarily small positive ϵ.

(c) From (2.2), we see that, for all $N \geq 2$,

$$m = N^2 + N = \gamma N^2$$

for some $1 < \gamma \leq \frac{3}{2}$. Therefore, since $N = 2^i$,

$$i = \frac{\log\left(\frac{m}{\gamma}\right)}{2} = \frac{\log m - \epsilon}{2} \tag{2.6}$$

for some $0 < \epsilon = \log \gamma \leq \log \frac{3}{2}$. Notice that, as $i \to \infty$, γ approaches 1 and ϵ approaches 0. Therefore, there exist arbitrarily large network graphs (and corresponding request sequences) for which the competitive ratio of any online algorithm, with respect to congestion, is at least

$$i + 1 = \frac{\log m + 2 - \epsilon}{2}$$

for arbitrarily small positive ϵ.

(d) From (2.6), we can conclude that there exist arbitrarily large networks (and corresponding request sequences) for which the competitive ratio of any online algorithm, with respect to makespan, is at least

$$\frac{i+2}{2} = \frac{\log m + 4 - \epsilon}{4}$$

for arbitrarily small positive ϵ.

(e) From (2.3), we know that $k = |\sigma_i'| = 2^i$. Therefore, there exist arbitrarily large request sequences such that the competitive ratio of any online algorithm, with respect to congestion, is at least

$$i + 1 = \log k + 1.$$

(f) From (2.3), we see that there exist arbitrarily large request sequences such that the competitive ratio of any online algorithm, with respect to makespan, is at least

$$\frac{i+2}{2} = \frac{\log k}{2} + 1. \qquad \blacksquare$$

2.4 Lower Bounds for Randomized Online Algorithms

The lower bounds in Theorem 2.1 also hold for any randomized online algorithm against an adaptive online adversary. We saw earlier that our adversary is adaptive in the sense that it chooses each request based on the online algorithm's responses to previous requests. Our adversary, represented by the algorithm $\bar{A}$, is also online since $\bar{A}$ requires knowledge of the online algorithm's responses to previous and current requests, but not of its responses to future requests. Thus we also have the following corollary:

Corollary 2.1

The competitive ratio of any randomized online routing and scheduling algorithm against an adaptive online adversary is at least

(a) $\frac{\log n+3-\epsilon}{2}$ for arbitrarily small positive $\epsilon \leq \log 3$, with respect to congestion;

(b) $\frac{\log n+5-\epsilon}{4}$ for arbitrarily small positive $\epsilon \leq \log 3$, with respect to makespan;

(c) $\frac{\log m+2-\epsilon}{2}$ for arbitrarily small positive $\epsilon \leq \log \frac{3}{2}$, with respect to congestion;

(d) $\frac{\log m+4-\epsilon}{4}$ for arbitrarily small positive $\epsilon \leq \log \frac{3}{2}$, with respect to makespan;

(e) $\log k + 1$ with respect to congestion; and

(f) $\frac{\log k}{2} + 1$ with respect to makespan.

Chapter 3

Online Algorithm GREEDY_ROUTE1

3.1 Introduction

In this chapter, we study a simple greedy online algorithm, named GREEDY_ROUTE1, for routing permanent virtual circuit requests on networks with uniform link capacity w.[1] GREEDY_ROUTE1 was first proposed by Mao and Simha [72] as an approximation algorithm for the NP-hard offline file transfer routing and scheduling problem. Mao and Simha (implicitly) proved that the competitive ratio of GREEDY_ROUTE1 is $\Omega(\sqrt{m})$ with respect to network congestion, where m is the number of network links, even when all requests have unit length and the network is a simple 2-layer digraph.[2] We improve this lower bound result by considering more complex networks. We also present an upper bound for arbitrary networks that is asymptotically tight when the ratio of the maximum to minimum bandwidth requirement (or file length, in file transfer terms) is constant or when the optimal

[1]The contents of this chapter are also contained in [42].

[2]Mao and Simha [72] called this algorithm LS3 and showed that its competitive ratio is $\Omega\left(\sqrt{k}\right)$, where k is the number of requests. Their bound can also be expressed as $\Omega(\sqrt{m})$.

congestion is at most one. Otherwise, our lower and upper bounds differ asymptotically by a factor equal to the above ratio of bandwidth requirements. Previously, no upper bound was known for GREEDY_ROUTE1.

Although the competitive ratio of GREEDY_ROUTE1 is inferior to that of the exponential routing algorithm EXP_ROUTE in [4], GREEDY_ROUTE1 makes its routing decisions much more quickly, which may prove advantageous in certain situations. We will examine this tradeoff between the speed/time complexity of an algorithm and the congestion of its route assignment, and conclude that GREEDY_ROUTE1 should be acceptable, especially when fast decision making is critical, on restricted classes of small networks.

3.1.1 The Algorithm

We formally define GREEDY_ROUTE1 in the following way:

Algorithm GREEDY_ROUTE1

For request f_j, assign any route $P \in \mathcal{P}_j$ which minimizes

$$\max_{e\in E} \begin{cases} \mu_j(e), & e \notin P \\ \mu_j(e) + \frac{l_j}{w}, & e \in P \end{cases} .$$

Ties are broken arbitrarily.

In other words, GREEDY_ROUTE1 will choose any path that does not increase the current network congestion, unless such an increase is unavoidable. Consider, for example, the network in Figure 3.1. A value on a network link in the figure indicates the current congestion on that link at some time τ. If an online algorithm receives the request $(s_j, t_j, 1, \tau)$ in this example, it can assign any of the following six routes:

1. $\langle s_j, v_1, v_4, v_5, t_j \rangle$,

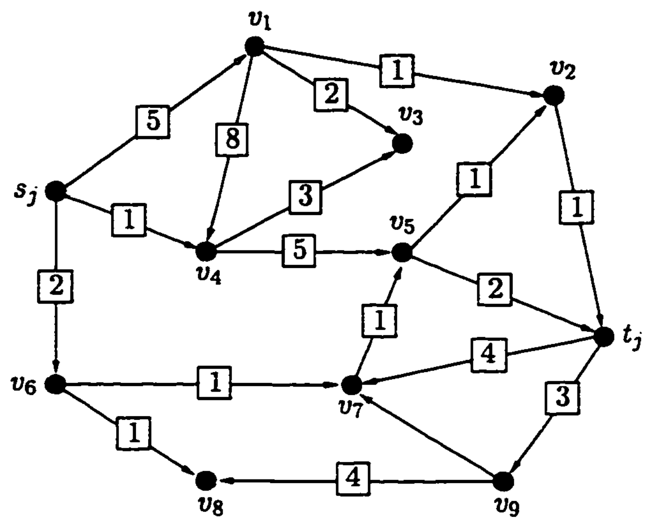


Figure 3.1: An example of a network graph.

2. $\langle s_j, v_1, v_4, v_5, v_2, t_j \rangle$,

3. $\langle s_j, v_1, v_2, t_j \rangle$,

4. $\langle s_j, v_4, v_5, v_2, t_j \rangle$,

5. $\langle s_j, v_4, v_5, t_j \rangle$, or

6. $\langle s_j, v_6, v_7, v_5, t_j \rangle$.

If an online algorithm used either of the first two routes, the resulting network congestion would be 9. On the other hand, if an online algorithm used any of the last four routes, the resulting network congestion would remain 8. Therefore, GREEDY_ROUTE1 could choose any route but the first two.

Before we can state our results formally, it will be necessary to define some notation. The following definitions hold for any particular problem instance, consisting of a request sequence $\sigma = f_1, f_2, \ldots, f_k$ and a network graph G.

- $d_j = \min_{P \in \mathcal{P}_j} |P|$ is the length of the shortest path between nodes s_j and t_j.
- $D_j = \max_{P \in \mathcal{P}_j} |P|$ is the length of the longest path between nodes s_j and t_j.
- $\mathcal{D} = \max_{1 \le j \le k} \frac{D_j}{d_j}$.[3]
- $\lambda = \min_{1 \le j \le k} l_j$ is the minimum bandwidth requirement among the requests in σ.
- $\Lambda = \max_{1 \le j \le k} l_j$ is the maximum bandwidth requirement among the requests in σ.
- $\mathcal{L} = \frac{\Lambda}{\lambda}$.

As we will discuss in Section 3.1.2, the advantage of GREEDY_ROUTE1 lies in its speed and simplicity. We will prove in Section 3.2 that the competitive ratio of GREEDY_ROUTE1 is $O\left(\sqrt{\mathcal{D}\mathcal{L}m}\right)$ on arbitrary networks, which is greater than the lower bound in [72] by a factor of $O\left(\sqrt{\mathcal{D}\mathcal{L}}\right)$. In Section 3.3, we will present an improved lower bound, showing that our upper bound is in fact off by at most a factor of $O\left(\sqrt{\mathcal{L}}\right)$ in general. The upper bound is asymptotically tight when the bandwidth requirements of the requests differ by a constant factor. We will show that the upper bound is tight when other conditions hold as well.

3.1.2 Tradeoffs Between Time Complexity and Routing Efficiency

If we compare GREEDY_ROUTE1 and EXP_ROUTE [4], we see a clear tradeoff between computational requirements and routing efficiency. GREEDY_ROUTE1 provides fast and simple decision making at the expense of the quality of the route assignment. On the other hand, the routes assigned by EXP_ROUTE are guaranteed to be asymptotically optimal, but at a heavier computational price.

[3]The value $\mathcal{D}$ can actually be defined to be $\max_{1 \le j \le k} \frac{|P_j|}{d_j}$ and the results in this chapter will still hold. But since this quantity depends on arbitrary decisions that GREEDY_ROUTE1 makes, it seems less desirable.

```
GREEDY_ROUTE1
    $\mu \leftarrow 0$
    for each $f_j \in \sigma$ do
        $P_j \leftarrow$ Assign_GREEDY_ROUTE1($j$, $\mu$)
    end for

function Assign_GREEDY_ROUTE1($j$, $\mu$)
    $\mu' \leftarrow \infty$
    $P' \leftarrow \emptyset$
    for all $P \in \mathcal{P}_j$ do
        temp $\leftarrow \max_{e \in P} \left\{ \mu_j(e) + \frac{l_i}{w} \right\}$
        if temp $\leq \mu$ then
            return $P$
        end if
        if temp < $\mu'$ then
            $\mu' \leftarrow$ temp
            $P' \leftarrow P$
        end if
    end for
    $\mu \leftarrow \mu'$
    return $P'$
```

Figure 3.2: A pseudocode representation of GREEDY_ROUTE1.

Consider the pseudocode for GREEDY_ROUTE1 in Figure 3.2. In this implementation, GREEDY_ROUTE1 requires $\sum_{P \in \mathcal{P}_j} (|P| + 1)$ comparisons and $\sum_{P \in \mathcal{P}_j} |P|$ additions in the worst case to assign a route to request f_j.[4] In the best case, GREEDY_ROUTE1 can require far fewer operations (as few as $|P_j|$ comparisons and additions) since it will exit as soon as it finds a suitable route. On the other hand, consider the straightforward implementation of EXP_ROUTE in Figure 3.3, which always requires the computation of $2 \sum_{P \in \mathcal{P}_j} |P|$ exponential functions in addition to $|\mathcal{P}_j|$ comparisons and $\sum_{P \in \mathcal{P}_j} (3|P| - 1)$ additions and subtractions.[5]

[4]If the algorithm has available to it O(m) bytes of working memory, then the values $\left(\mu_j(e) + \frac{l_i}{w}\right)$ can be computed once for each path, bounding the number of additions by O(m).

[5]This is a simplified version which assumes that the optimal congestion is 1 [80]. As noted earlier for GREEDY_ROUTE1, if the algorithm has O(m) bytes of working memory available to it, the number of arithmetic

```
EXP_ROUTE
   let $a = 1 + \gamma$ for some $0 < \gamma < 1$
   for each $f_j \in \sigma$ do
      $P_j \leftarrow$ Assign_EXP$(a, j)$
   end for

function Assign_EXP$(j, a)$
   min $\leftarrow \infty$
   $P' \leftarrow \emptyset$
   for all $P \in \mathcal{P}_j$ do
      temp $\leftarrow \sum_{e \in P} \left( a^{\mu_j(e) + \frac{l_j}{c(e)}} - a^{\mu_j(e)} \right)$
      if temp < min then
         min $\leftarrow$ temp
         $P' \leftarrow P$
      end if
   end for
   return $P'$
```

Figure 3.3: A pseudocode representation of EXP_ROUTE.

Clearly, GREEDY_ROUTE1 requires significantly less time to make each decision.

The obvious question now is whether the routes assigned by GREEDY_ROUTE1 are good enough to warrant its use in time critical situations. The answer really depends on the situation, but we can discuss the factors in the bound to get some ideas. There are three factors to consider: $\sqrt{\mathcal{D}}$, $\sqrt{\mathcal{L}}$, and $\sqrt{m}$. Consider for a moment instances where $\mathcal{D}$ and $\mathcal{L}$ are small. For these instances, the difference between the competitive ratio of GREEDY_ROUTE1 and that of EXP_ROUTE is obviously arbitrarily large if we consider arbitrarily large networks. However, if we consider real networks with less than a few hundred uniform speed links, the competitive ratios of the two algorithms do not differ significantly. As shown in Figure 3.4, when $m \leq 250$, $\sqrt{m}$ is at most twice $\log m$ and when $m \leq 850$, $\sqrt{m}$ is at most three times

operations can be bounded by $O(m)$.

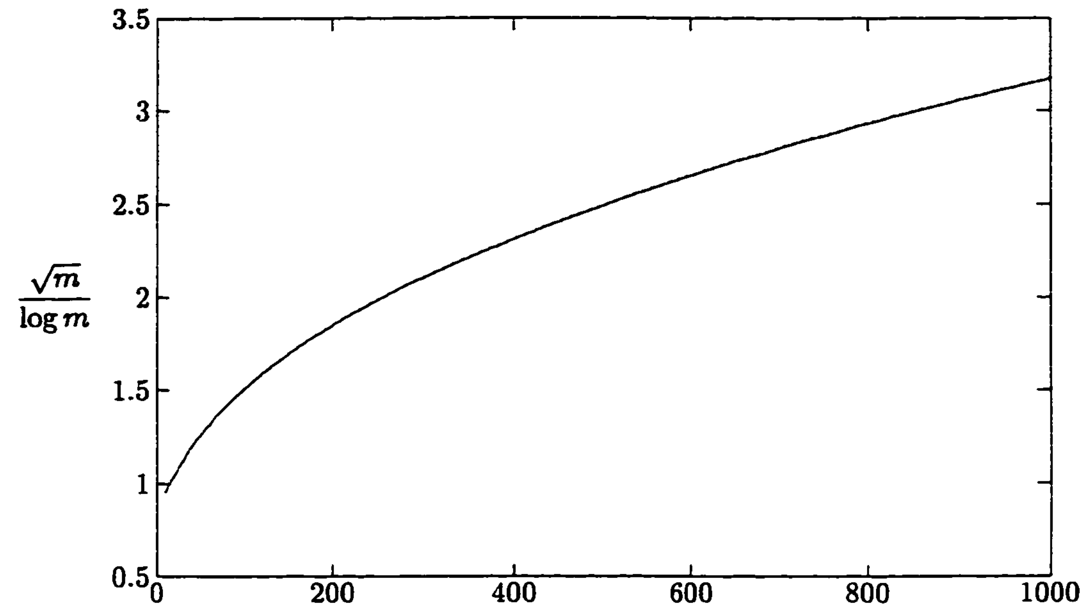


Figure 3.4: Relative difference between $\sqrt{m}$ and $\log m$ on small networks.

$\log m$. Even when $m = 2000$, $\sqrt{m}$ is just slightly more than four times $\log m$.[6]

Whereas the competitive ratio of EXP_ROUTE is dependent only upon the size of the network, our upper bound for GREEDY_ROUTE1 also depends on the two factors $\sqrt{\mathcal{D}}$ and $\sqrt{\mathcal{L}}$. The first factor is the maximum ratio, over all requests, of the length of the longest path between the request's source and destination and the shortest distance between the source and destination. The existence of this factor in the competitive ratio of GREEDY_ROUTE1 is intuitive: the algorithm does not prefer short paths to long ones (as EXP_ROUTE does) and therefore may add to the congestion of many more links than is necessary. For some networks though, including all layered networks, $\mathcal{D} = O(1)$, and GREEDY_ROUTE1 is guaranteed to be

[6]Since we are considering small values of m, it would be inappropriate to continue without addressing the constants hidden in the big-oh notation. For GREEDY_ROUTE1, the constant is only $\sqrt{2}$ and for EXP_ROUTE, the constant is some number strictly greater than 1 which depends on constant parameters chosen by the algorithm designer. When it is not assumed that the optimal congestion is 1, this constant increases by a factor of 4. Therefore, we can safely ignore the constants in our comparison.

$O\left(\sqrt{\mathcal{L}m}\right)$ competitive (but not much better — in Section 3.4 we show that the competitive ratio of GREEDY_ROUTE1 is still $\Omega\left(\sqrt{m}\right)$ on layered networks). Although it seems unlikely that $\mathcal{D}$ would be a large function of m in any network, if this were the case it might be reasonable to limit the paths from which GREEDY_ROUTE1 is allowed to choose in order to minimize the error.

The second factor unique to GREEDY_ROUTE1 is the ratio of the maximum to minimum bandwidth requirement. We conjecture that this factor does not belong in the true competitive ratio, but rather is a byproduct of our proof technique. However, if the competitive ratio of GREEDY_ROUTE1 does depend on $\sqrt{\mathcal{L}}$ then GREEDY_ROUTE1 is more efficient on instances consisting of requests with like bandwidth requirements. Such instances are common. For instance, in a network serving video or audio streams to customers, each request would require equal bandwidth.

3.2 An Upper Bound

In this section, we will prove our upper bound on the competitive ratio of GREEDY_ROUTE1:

Theorem 3.1

The competitive ratio of GREEDY_ROUTE1 is $O\left(\sqrt{\mathcal{D}\mathcal{L}m}\right)$.

In the proof of Theorem 3.1, we will use the following notation relating to an arbitrary problem instance.

- Let P_j^* denote the path assigned to request $f_j \in \sigma$ by an optimal offline algorithm.
- Let μ^* denote the optimal network congestion for the problem instance.

- Let $\widetilde{\Lambda} = \frac{\Lambda}{w}$, where Λ is the maximum bandwidth requirement of a request and w is the capacity of the network links. Notice that $\widetilde{\Lambda} \leq \mu^*$.
- Let $\kappa = \left\lfloor \mu/\widetilde{\Lambda} \right\rfloor$, where μ is the congestion of the routes assigned by GREEDY_ROUTE1. For each $i = 1, 2, \ldots, \kappa$, $f_{y_i} \in \sigma$ is the first request to cause a link to have congestion at least $i\widetilde{\Lambda}$ in the route assignment constructed by GREEDY_ROUTE1. In other words, for all y_i, $\mu_{y_i+1}(P_{y_i}) \geq i\widetilde{\Lambda}$, and $\mu_h(e) < i\widetilde{\Lambda}$ for all edges $e \in E$ and all $h = 1, 2, \ldots, y_i$.
- For each $i = 1, 2, \ldots, \kappa$, $e_{z_i} \in P_{y_i}$ is a link satisfying $\mu_{y_i+1}(e_{z_i}) = \mu_{y_i+1}(P_{y_i})$. Ties are broken in favor of the link with the smallest index.

The following three lemmas are fundamental to the proof of Theorem 3.1. The first two lemmas motivate our definitions of y_i and z_i. Lemma 3.1 states that no two indices in the set $\{y_1, y_2, \ldots, y_\kappa\}$ are the same and Lemma 3.2 quantifies the congestion on any path $P \in \mathcal{P}_{y_i}$ just before f_{y_i} arrives and is assigned a route. Lemma 3.3 is a technical detail used to bound the total congestion incurred by GREEDY_ROUTE1 in terms of μ^*.

Lemma 3.1

For all integers h and i, $1 \leq h < i \leq \kappa$, $y_h \neq y_i$.

Proof

Assume that there exist h and i, where $1 \leq h < i \leq \kappa$, such that $y_h = y_i$. Since, by definition, $\mu_{y_i+1}(e_{z_i}) \geq i\widetilde{\Lambda}$, we know that

$$\mu_{y_i}(e_{z_i}) \geq (i-1)\widetilde{\Lambda}. \tag{3.1}$$

We also know, by definition, that $\mu_{y_h}(e) < h\tilde{\Lambda}$, for all $e \in E$. In particular, we know that $\mu_{y_h}(e_{z_i}) < h\tilde{\Lambda}$. But since $y_h = y_i$,

$$\mu_{y_i}(e_{z_i}) = \mu_{y_h}(e_{z_i}) < h\tilde{\Lambda} \leq (i-1)\tilde{\Lambda},$$

which is a contradiction to (3.1). ■

Lemma 3.2

For all $i = 1, 2, \ldots, \kappa$, $\mu_{y_i}(P) \geq (i-1)\tilde{\Lambda}$ for all $P \in \mathcal{P}_{y_i}$.

Proof

Suppose that there exists a path $P \in \mathcal{P}_{y_i}$ such that $\mu_{y_i}(P) < (i-1)\tilde{\Lambda}$. Since, by definition, $\mu_{y_i+1}(P_{y_i}) \geq i\tilde{\Lambda}$, we know that $\mu_{y_i}(P_{y_i}) \geq (i-1)\tilde{\Lambda}$. But this means that GREEDY_ROUTE1 should have assigned f_{y_i} to path P rather than P_{y_i} ($\neq P$), since this decision would have resulted in a smaller network congestion. (Notice that the network congestion did increase with request f_{y_i} by the definition of y_i.) This is a contradiction to the selection of P_{y_i} by GREEDY_ROUTE1. ■

Lemma 3.3

$\mathcal{D}m\mu^* \geq \sum_{e \in E} \mu(e)$.

Proof

Since no algorithm can assign a request f_j to more than D_j links, and the optimal solution had to have assigned each request to at least d_j links, it must be the case that

$$\max_{1 \leq j \leq k} \frac{D_j}{d_j} \sum_{e \in E} \mu^*(e) \geq \sum_{e \in E} \mu(e).$$

The lemma follows from the fact that

$$\mathcal{D}m\mu^* \geq \max_{1\leq j\leq k} \frac{D_j}{d_j} \sum_{e\in E} \mu^*(e).$$

■

We can now use the preceding lemmas to prove the main result of this section.

Proof of Theorem 3.1

From Lemma 3.2, we know that, in particular, $\mu_{y_i}(P^*_{y_i}) \geq (i-1)\tilde{\Lambda}$ for all $i = 1, 2, \ldots, \kappa$. Thus,

$$\begin{aligned}\sum_{i=1}^{\kappa} \mu_{y_i}(P^*_{y_i}) &\geq \sum_{i=1}^{\kappa}(i-1)\tilde{\Lambda} \\ &= \frac{\kappa(\kappa-1)\tilde{\Lambda}}{2} \\ &= \frac{\left\lfloor \frac{\mu}{\Lambda} \right\rfloor \left(\left\lfloor \frac{\mu}{\Lambda} \right\rfloor - 1\right)\tilde{\Lambda}}{2} \\ &> \frac{\left(\frac{\mu}{\Lambda}-1\right)\left(\frac{\mu}{\Lambda}-2\right)\tilde{\Lambda}}{2} \\ &> \frac{\left(\frac{\mu}{\Lambda}-2\right)^2\tilde{\Lambda}}{2}. \end{aligned} \tag{3.2}$$

To clarify the presentation, let us assume for the moment that the paths in the set

$$\{P^*_{y_1}, P^*_{y_2}, \ldots, P^*_{y_\kappa}\}$$

are pairwise edge disjoint. From this assumption and the fact that $y_i \leq k$, we see that

$$\sum_{e\in E}\mu(e) > \sum_{e\in E}\mu_k(e) \geq \sum_{i=1}^{\kappa}\mu_{y_i}(P^*_{y_i}). \tag{3.3}$$

Combining this fact with (3.2) and Lemma 3.3, we see that

$$\begin{aligned}
\mathcal{D}m\mu^* &\geq \sum_{e \in E} \mu(e) && \text{by Lemma 3.3} \\
&\geq \sum_{i=1}^{\kappa} \mu_{y_i}(P^*_{y_i}) && \text{by (3.3)} \\
&> \frac{\left(\frac{\mu}{\tilde{\Lambda}} - 2\right)^2 \tilde{\Lambda}}{2} && \text{by (3.2).}
\end{aligned}$$

This implies that

$$\begin{aligned}
\mu &< \left(\sqrt{\frac{2\mathcal{D}m\mu^*}{\tilde{\Lambda}}} + 2\right)\tilde{\Lambda} \\
&= \sqrt{2\mathcal{D}m\mu^*\tilde{\Lambda}} + 2\tilde{\Lambda} \\
&\leq \sqrt{2\mathcal{D}m(\mu^*)^2} + 2\mu^* \\
&= \left(\sqrt{2\mathcal{D}m} + 2\right)\mu^*. \qquad (3.4)
\end{aligned}$$

So when the paths in the set $\{P^*_{y_1}, P^*_{y_2}, \dots, P^*_{y_\kappa}\}$ are pairwise edge disjoint, GREEDY_ROUTE1 is $O\left(\sqrt{\mathcal{D}m}\right)$ competitive. (See Corollary 3.1 below.) But to prove the theorem we must consider the more general case in which the optimal paths in the set $\{P^*_{y_1}, P^*_{y_2}, \dots, P^*_{y_\kappa}\}$ are not necessarily pairwise edge disjoint.

In general, let $I \geq 1$ be the maximum number of paths in the set $\{P^*_{y_1}, P^*_{y_2}, \dots, P^*_{y_\kappa}\}$ that intersect at an edge. Obviously, $\frac{I\lambda}{w} \leq \mu^*$. When $I > 1$, inequality (3.3) is not necessarily true because the congestion on some edges $e \in E$ may be counted up to I times in the summation on the right hand side. However, with the aid of Lemma 3.1, we can modify

(3.3) to say that

$$I \sum_{e \in E} \mu(e) > \sum_{i=1}^{\mathcal{K}} \mu_{y_i}(P^*_{y_i}). \tag{3.5}$$

Then, we see that

$$\begin{aligned}
\mathcal{D}mI\mu^* &\geq I \sum_{e \in E} \mu(e) && \text{by Lemma 3.3} \\
&\geq \sum_{i=1}^{\mathcal{K}} \mu_{y_i}(P^*_{y_i}) && \text{by (3.5)} \\
&> \frac{\left(\frac{\mu}{\tilde{\Lambda}} - 2\right)^2}{2} \cdot \tilde{\Lambda} && \text{by (3.2).}
\end{aligned}$$

This implies that

$$\begin{aligned}
\mu &< \left(\sqrt{\frac{2\mathcal{D}mI\mu^*}{\tilde{\Lambda}}} + 2\right)\tilde{\Lambda} \\
&= \sqrt{2\mathcal{D}\tilde{\Lambda}mI\mu^*} + 2\tilde{\Lambda} \\
&\leq \sqrt{2\mathcal{D}\frac{\Lambda}{\lambda}m(\mu^*)^2} + 2\mu^* \\
&= \left(\sqrt{2\mathcal{D}\mathcal{L}m} + 2\right)\mu^*.
\end{aligned} \tag{3.6}$$

Thus, GREEDY_ROUTE1 is $O\left(\sqrt{\mathcal{D}\mathcal{L}m}\right)$ competitive. ■

We note that there are several realistic conditions under which we can infer that the competitive ratio of GREEDY_ROUTE1 is $O\left(\sqrt{\mathcal{D}m}\right)$. We define a *feasible* request sequence to be one for which there exists a set of routes with congestion $\mu^* \leq 1$. We point out that this modified definition of *competitive*, which limits consideration to a subset of instances, namely feasible instances, has appeared previously in the literature [8].

Corollary 3.1

The competitive ratio of GREEDY_ROUTE1 is $O\left(\sqrt{\mathcal{D}m}\right)$ if any of the following are true:

(a) $\mathcal{L} = O(1)$;

(b) the set of optimal routes is pairwise edge disjoint;

(c) the request sequence is feasible and $w = O(1)$ or $w = O(\lambda)$; or

(d) $\Lambda \leq w$ and $w = O(1)$ or $w = O(\lambda)$.

Proof

(a) Obvious.

(b) Follows from (3.4) in the proof of Theorem 3.1.

(c) When $\mu^* \leq 1$, we know that $I\lambda \leq w$. Also, $\tilde{\Lambda} \leq \mu^* \leq 1$. Thus, by (3.6),

$$\begin{aligned}\mu &< \sqrt{2\mathcal{D}\tilde{\Lambda}mI\mu^*} + 2\tilde{\Lambda} \\ &\leq \sqrt{2\mathcal{D}m\frac{w}{\lambda}} + 2 \\ &= O\left(\sqrt{\mathcal{D}m}\right).\end{aligned}$$

(d) When $\Lambda \leq w$, it follows that $\tilde{\Lambda} \leq 1$. Thus, by (3.6),

$$\begin{aligned}\mu &< \sqrt{2\mathcal{D}\tilde{\Lambda}mI\mu^*} + 2\tilde{\Lambda} \\ &\leq \sqrt{2\mathcal{D}m\frac{w}{\lambda}(\mu^*)^2} + 2 \\ &\leq O\left(\sqrt{\mathcal{D}m}\right)\mu^*.\end{aligned}$$

■

Notice that the first condition in part (d) must hold for real virtual circuit request sequences. Otherwise, the request with the largest bandwidth requirement cannot be assigned to any route without exceeding link capacities.

3.3 A Lower Bound

In this section we show that the upper bound in Theorem 3.1 is tight up to a factor of $\sqrt{\mathcal{L}}$. This result is formally stated as follows:

Theorem 3.2

The competitive ratio of GREEDY_ROUTE1 is $\Omega\left(\sqrt{\mathcal{D}m}\right)$ for arbitrarily large values of $\mathcal{D}$ and m.

We note that Theorem 3.2 implies that the upper bound in Theorem 3.1 is tight up to a constant factor when at least one of the conditions in Corollary 3.1 is true.

To prove Theorem 3.2, we will construct an arbitrarily large network and corresponding request sequence, and then show that, if GREEDY_ROUTE1 makes bad decisions, it can incur the above network congestion on that instance. On the other hand, we will show that an optimal algorithm can always find a set of pairwise edge disjoint routes for the requests.

3.3.1 The Network

For any integers $h > 1$ and $i \geq 1$, the network is represented by a directed graph $G_{h,i}$ containing i connected components and having a longest path with length h. All links have unit capacity. As examples, $G_{2,i}$ and $G_{3,i}$ are displayed in Figure 3.5, and $G_{6,2}$ is displayed in Figure 3.6.

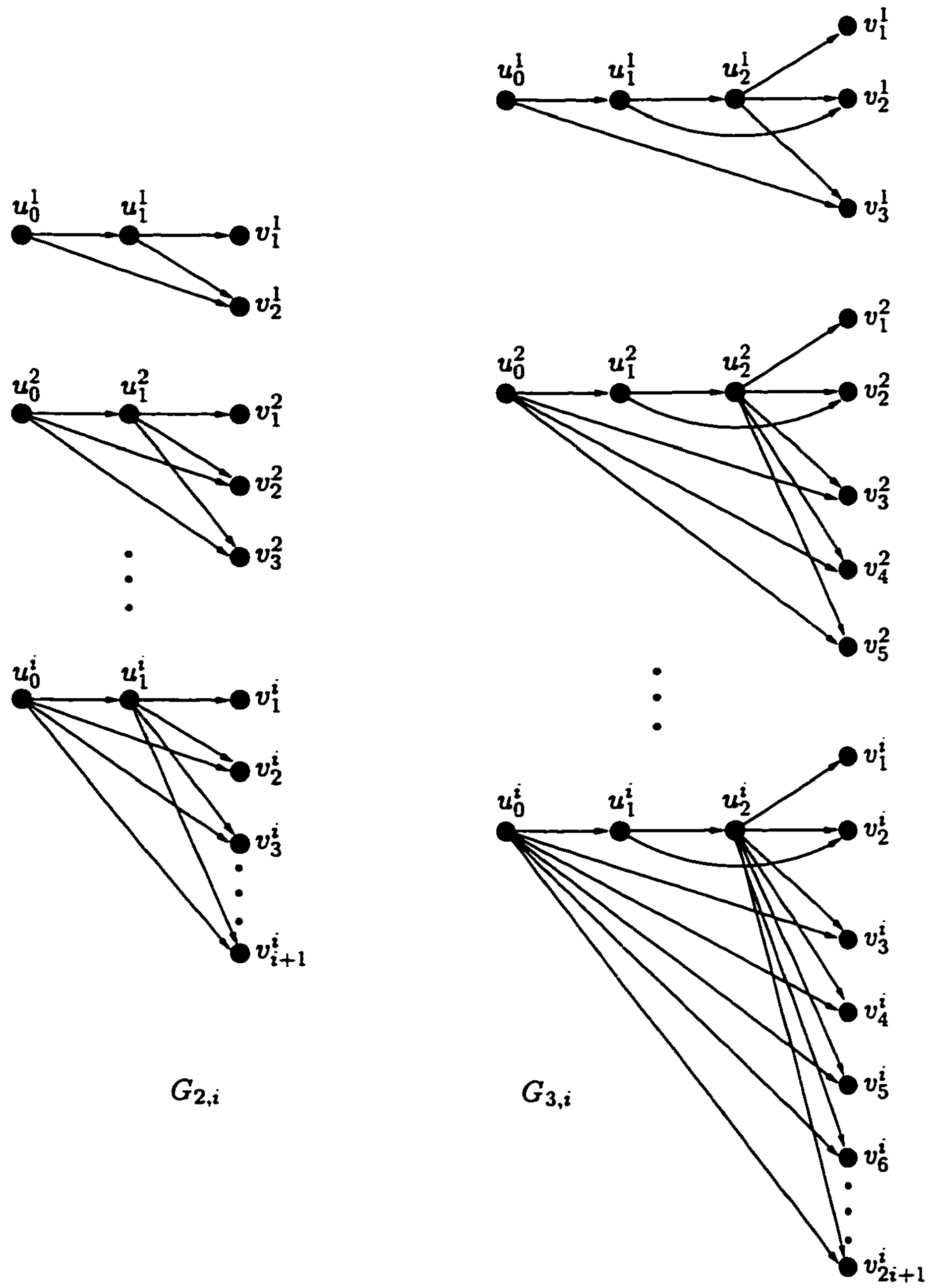


Figure 3.5: The networks $G_{2,i}$ and $G_{3,i}$.

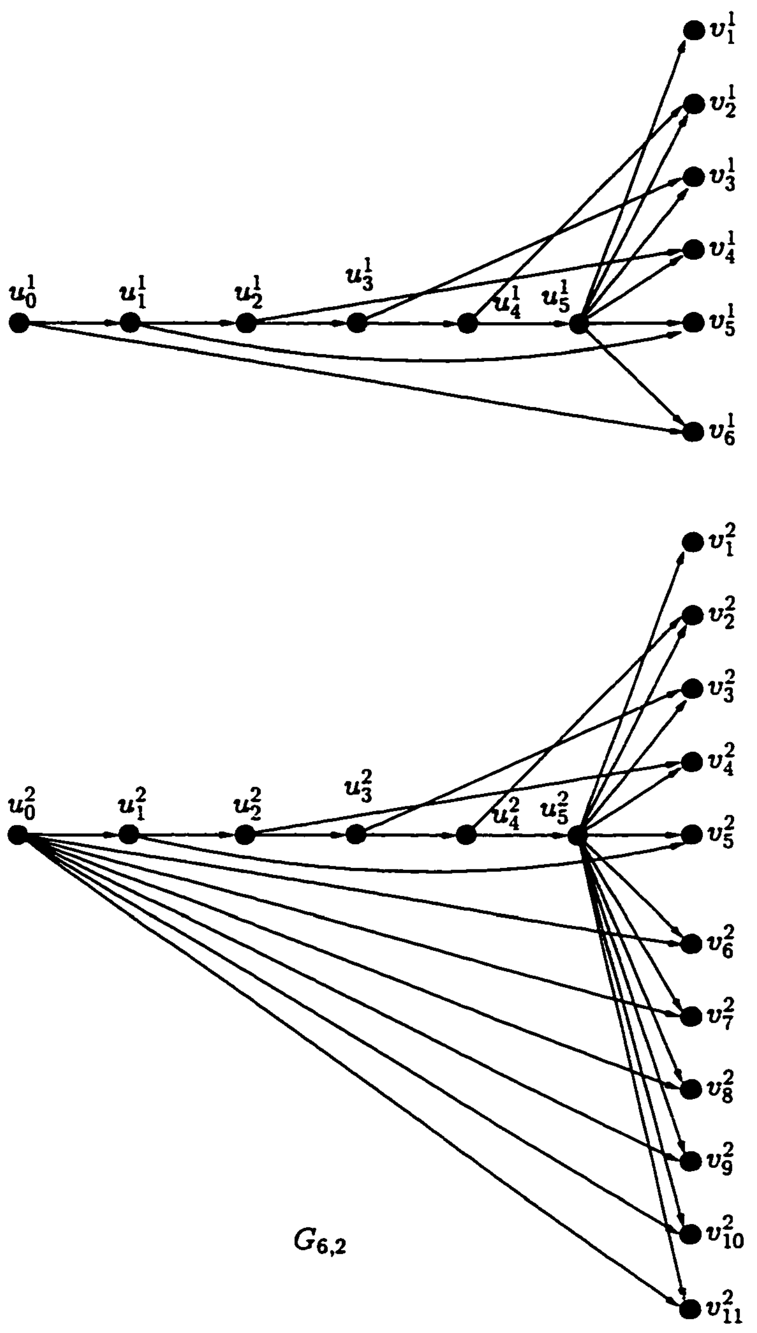


Figure 3.6: The network $G_{6,2}$.

The vertex set of $G_{h,i}$ is defined to be the union

$$\bigcup_{\iota=1}^{i} (U_{h,\iota} \cup V_{h,\iota})$$

where

- $U_{h,\iota} = \{u_0^\iota, u_1^\iota, \dots, u_{h-1}^\iota\}$ contains the set of source nodes, and
- $V_{h,\iota} = \left\{v_1^\iota, v_2^\iota, \dots, v_{(h-1)\iota+1}^\iota\right\}$ is the set of destination nodes.

Note that

$$\begin{aligned} n &= \left|\bigcup_{\iota=1}^{i} (U_{h,\iota} \cup V_{h,\iota})\right| \\ &= \left(\frac{h-1}{2}\right) i^2 + \left(\frac{3h+1}{2}\right) i. \end{aligned}$$

The arc set of $G_{h,i}$ is defined to be

$$\bigcup_{\iota=1}^{i} E_{h,\iota}$$

where

$$E_{h,\iota} = \bigcup_{\kappa=1}^{h-2} \left\{\left(u_\kappa^\iota, v_{h-\kappa}^\iota\right)\right\} \cup \bigcup_{\kappa=1}^{(h-1)\iota+1} \{(u_{h-1}^\iota, v_\kappa^\iota)\} \cup \bigcup_{\kappa=0}^{h-2} \{(u_\kappa^\iota, u_{\kappa+1}^\iota)\} \cup \bigcup_{\kappa=h}^{(h-1)\iota+1} \{(u_0^\iota, v_\kappa^\iota)\}.$$

Note that

$$\begin{aligned} m &= \left|\bigcup_{\iota=1}^{i} E_{h,\iota}\right| \\ &= (h-1)i^2 + (2h-1)i \\ &\le (3h-2)i^2. \end{aligned} \tag{3.7}$$

Notice that the i connected components of $G_{h,i}$ are the subgraphs

$$\{(U_{h,\iota} \cup V_{h,\iota}, E_{h,\iota}) : \iota = 1, 2, \ldots, i\}.$$

3.3.2 The Request Sequence

Let $\sigma_{h,i}$ denote the request sequence for the network $G_{h,i}$. As was the case in the proof of Theorem 2.1, each request will arrive at time 0 and will require unit bandwidth. Therefore, we will represent each request $f_j \in \sigma_{h,i}$ simply by its (s_j, t_j) pair.

The sequence $\sigma_{h,i}$ is formally defined to be the concatenation of i smaller subsequences:

$$\sigma_{h,i} = \sigma_h^1 \circ \sigma_h^2 \circ \cdots \circ \sigma_h^i.$$

All paths between the source and destination of each request in subsequence σ_h^ι, $\iota = 1, 2, \ldots, i$, are contained in the subgraph $(U_{h,\iota} \cup V_{h,\iota}, E_{h,\iota})$. Each subsequence σ_h^ι is further defined to be the concatenation of two subsequences A_h^ι and B_h^ι where

$$A_h^\iota = (u_0^\iota, v_h^\iota), (u_0^\iota, v_{h+1}^\iota), \ldots, \left(u_0^\iota, v_{(h-1)\iota+1}^\iota\right)$$

and

$$B_h^\iota = \left(u_0^\iota, v_{h-1}^\iota\right), \left(u_1^\iota, v_{h-2}^\iota\right), \ldots, \left(u_{h-2}^\iota, v_1^\iota\right).$$

Note that

$$\begin{aligned} k &= |\sigma_{h,i}| \\ &= \sum_{\iota=1}^{i} \left(|A_h^\iota| + |B_h^\iota|\right) \\ &= \left(\frac{h-1}{2}\right) i^2 + \left(\frac{h+1}{2}\right) i. \end{aligned}$$

	f_j	(s_j, t_j)	P_j	P^*_j
σ_3^1	f_1	(u_0^1, v_3^1)	$\langle u_0^1, u_1^1, u_2^1, v_3^1 \rangle$	$\langle u_0^1, v_3^1 \rangle$
	f_2	(u_0^1, v_2^1)	$\langle u_0^1, u_1^1, u_2^1, v_2^1 \rangle$	$\langle u_0^1, u_1^1, v_2^1 \rangle$
	f_3	(u_1^1, v_1^1)	$\langle u_1^1, u_2^1, v_1^1 \rangle$	$\langle u_1^1, u_2^1, v_1^1 \rangle$
σ_3^2	f_4	(u_0^2, v_3^2)	$\langle u_0^2, u_1^2, u_2^2, v_3^2 \rangle$	$\langle u_0^2, v_3^2 \rangle$
	f_5	(u_0^2, v_4^2)	$\langle u_0^2, u_1^2, u_2^2, v_4^2 \rangle$	$\langle u_0^2, v_4^2 \rangle$
	f_6	(u_0^2, v_5^2)	$\langle u_0^2, u_1^2, u_2^2, v_5^2 \rangle$	$\langle u_0^2, v_5^2 \rangle$
	f_7	(u_0^2, v_2^2)	$\langle u_0^2, u_1^2, u_2^2, v_2^2 \rangle$	$\langle u_0^2, u_1^2, v_2^2 \rangle$
	f_8	(u_1^2, v_1^2)	$\langle u_1^2, u_2^2, v_1^2 \rangle$	$\langle u_1^2, u_2^2, v_1^2 \rangle$
σ_3^3	f_9	(u_0^3, v_3^3)	$\langle u_0^3, u_1^3, u_2^3, v_3^3 \rangle$	$\langle u_0^3, v_3^3 \rangle$
	f_{10}	(u_0^3, v_4^3)	$\langle u_0^3, u_1^3, u_2^3, v_4^3 \rangle$	$\langle u_0^3, v_4^3 \rangle$
	f_{11}	(u_0^3, v_5^3)	$\langle u_0^3, u_1^3, u_2^3, v_5^3 \rangle$	$\langle u_0^3, v_5^3 \rangle$
	f_{12}	(u_0^3, v_6^3)	$\langle u_0^3, u_1^3, u_2^3, v_6^3 \rangle$	$\langle u_0^3, v_6^3 \rangle$
	f_{13}	(u_0^3, v_7^3)	$\langle u_0^3, u_1^3, u_2^3, v_7^3 \rangle$	$\langle u_0^3, v_7^3 \rangle$
	f_{14}	(u_0^3, v_2^3)	$\langle u_0^3, u_1^3, u_2^3, v_2^3 \rangle$	$\langle u_0^3, u_1^3, v_2^3 \rangle$
	f_{15}	(u_1^3, v_1^3)	$\langle u_1^3, u_2^3, v_1^3 \rangle$	$\langle u_1^3, u_2^3, v_1^3 \rangle$

Table 3.1: Request sequence $\sigma_{3,3}$ on the network $G_{3,3}$.

As an example of this construction, consider the first three columns of Table 3.1, which contain the requests in $\sigma_{3,3}$. (See Figure 3.5 for the corresponding network $G_{3,3}$.)

3.3.3 Proof of the Lower Bound

We will use the construction in the previous two subsections to prove Theorem 3.2. We first notice that an optimal algorithm can always find a pairwise edge disjoint set of routes for the requests in $\sigma_{h,i}$. Informally, this route assignment is the one in which each request in $\sigma_{h,i}$ is assigned to its shortest path in $G_{h,i}$. Formally, we have the following lemma:

Lemma 3.4

For all integers $h > 1$ and $i \geq 1$, the set of optimal routes for request sequence $\sigma_{h,i}$ on network $G_{h,i}$ is pairwise edge disjoint.

Proof

Notice that the set of paths for the requests in each subsequence σ_h^ι of $\sigma_{h,i}$ is contained in a different connected component of $G_{h,i}$, namely the subgraph $(U_{h,\iota} \cup V_{h,\iota}, E_{h,\iota})$. Thus, to prove the lemma, we can simply specify a pairwise edge disjoint set of paths for each subsequence σ_h^ι, $\iota = 1, 2, \ldots, i$.

First, we assign each request $(u_0^\iota, v_\kappa^\iota) \in A_h^\iota$, $\kappa = h, h+1, \ldots, (h-1)\iota + 1$, to the path $\langle u_0^\iota, v_\kappa^\iota \rangle$. Since the destinations of all the requests in A_h^ι are different, this set of paths is pairwise edge disjoint. To each request $\left(u_\kappa^\iota, v_{h-\kappa-1}^\iota\right) \in B_h^\iota$, $\kappa = 0, 1, \ldots, h-2$, we assign the path $\left\langle u_\kappa^\iota, u_{\kappa+1}^\iota, v_{h-\kappa-1}^\iota \right\rangle$. In this set of paths, each link $\left(u_\kappa^\iota, u_{\kappa+1}^\iota\right)$, $\kappa = 0, 1, \ldots, h-2$, is used exactly once and each link $\left(u_\kappa^\iota, v_{h-\kappa}^\iota\right)$, $\kappa = 1, 2, \ldots, h-1$ is used exactly once. ■

In the worst case, GREEDY_ROUTE1 can assign to each request its non-optimal route (with the exception of the requests $\left(u_{h-2}^\iota, v_1^\iota\right)$, $\iota = 1, 2, \ldots, i$, which each have only one route). For example, the fourth column of Table 3.1 contains the worst case routes assigned by GREEDY_ROUTE1, while the fifth column contains an optimal route assignment. In the example, GREEDY_ROUTE1 has incurred congestion 3 after routing the requests in σ_3^1, 5 after routing the requests in σ_3^2, and 7 after routing the requests in σ_3^3. In general, we have the following lemma:

Lemma 3.5

For all integers $h > 1$, $i \geq 1$, and $1 \leq \iota \leq i$, GREEDY_ROUTE1 can incur congestion $(h-1)\iota + 1$ after being issued request subsequence $\sigma_h^1 \circ \sigma_h^2 \circ \ldots \circ \sigma_h^\iota$ on network $G_{h,i}$.

Proof

The proof is by induction on ι.

Base case ($\iota = 1$). For the base case, we will consider how GREEDY_ROUTE1 might assign routes to the requests in the subsequence σ_h^1. The first request will be (u_0^1, v_h^1). Since GREEDY_ROUTE1 may assign either of the two possible routes to the request, suppose it assigns the route $\langle u_0^1, u_1^1, \ldots, u_{h-1}^1, v_h^1 \rangle$. The next request will be (u_0^1, v_{h-1}^1). Each of this request's two possible routes contains the link (u_0^1, u_1^1) and so assigning the request to either route will increase the maximum congestion to 2. Therefore GREEDY_ROUTE1 may assign the route $\langle u_0^1, u_1^1, \ldots, u_{h-1}^1, v_{h-1}^1 \rangle$. Now consider the next request, (u_1^1, v_{h-2}^1). As was the case previously, each of the routes for this request contains a common link, in this case (u_1^1, u_2^1), which has the maximum congestion. Thus, GREEDY_ROUTE1 may assign the route $\langle u_1^1, u_2^1, \ldots, u_{h-1}^1, v_{h-2}^1 \rangle$. The maximum congestion is now 3. In general, when request $(u_\kappa^1, v_{h-\kappa-1}^1)$, $\kappa = 0, 1, \ldots, h-2$, arrives, both routes will contain link $(u_\kappa^1, u_{\kappa+1}^1)$ which has the maximum congestion $\kappa + 1$. Thus GREEDY_ROUTE1 can assign the route $\langle u_\kappa^1, u_{\kappa+1}^1, \ldots, u_{h-1}^1, v_{h-\kappa-1}^1 \rangle$, causing the maximum congestion to reach $\kappa + 2$. When the last request in σ_h^1, (u_{h-2}^1, v_1^1), arrives, it is assigned the route $\langle u_{h-2}^1, u_{h-1}^1, v_1^1 \rangle$, which causes the maximum congestion to be h.

Induction step ($\iota > 1$). Let us now consider how GREEDY_ROUTE1 might assign the requests in $\sigma_h^1 \circ \sigma_h^2 \circ \ldots \circ \sigma_h^\iota$. By the induction hypothesis, after assigning the requests in $\sigma_h^1 \circ \sigma_h^2 \circ \ldots \circ \sigma_h^{\iota-1}$, GREEDY_ROUTE1 could have incurred congestion $(h-1)(\iota-1)+1$. Let us say that this has happened. Now we consider what can happen when the requests in σ_h^ι arrive. The paths assigned to these requests will all be contained in the subgraph $(U_{h,\iota} \cup V_{h,\iota}, E_{h,\iota})$. By construction, none of the links in $E_{h,\iota}$ has been traversed by a previous request. Therefore, none of the first $(h-1)(\iota-1)+1$ requests in σ_h^ι (the requests in A_h^ι)

will increase the maximum congestion. Thus, GREEDY_ROUTE1 may assign each such request $(u_0^\iota, v_\kappa^\iota) \in A_h^\iota$, $\kappa = h, h+1, \ldots, (h-1)\iota+1$ to an arbitrary route, say $\langle u_0^\iota, u_1^\iota, \ldots, u_{h-1}^\iota, v_\kappa^\iota\rangle$. As a result of this assignment, every link $(u_\nu^\iota, u_{\nu+1}^\iota)$, $\nu = 0, 1, \ldots, h-2$, will have congestion $(h-1)(\iota-1)+1$.

The next request to arrive is $(u_0^\iota, v_{h-1}^\iota)$. Each of this request's two possible routes contains the link (u_0^ι, u_1^ι), which currently has the maximum congestion. Thus, GREEDY_ROUTE1 may assign the route $\langle u_0^\iota, u_1^\iota, \ldots, u_{h-1}^\iota, v_{h-1}^\iota\rangle$. As a result, the maximum congestion is now $(h-1)(\iota-1)+2$. Now consider the next request, $(u_1^\iota, v_{h-2}^\iota)$. As was the case previously, each of the routes for this request contains a common link, in this case (u_1^ι, u_2^ι), which has the maximum congestion. Thus, GREEDY_ROUTE1 may assign the route $\langle u_1^\iota, u_2^\iota, \ldots, u_{h-1}^\iota, v_{h-2}^\iota\rangle$. The maximum congestion is now $(h-1)(\iota-1)+3$. In general, when request $(u_\kappa^\iota, v_{h-\kappa-1}^\iota)$, $\kappa = 0, 1, \ldots, h-2$, arrives, both routes will contain link $(u_\kappa^\iota, u_{\kappa+1}^\iota)$ which has the maximum congestion $(h-1)(\iota-1)+\kappa+1$. Thus GREEDY_ROUTE1 can assign the route

$$\langle u_\kappa^\iota, u_{\kappa+1}^\iota, \ldots, u_{h-1}^\iota, v_{h-\kappa-1}^\iota\rangle,$$

causing the maximum congestion to reach $(h-1)(\iota-1)+\kappa+2$. When the last request in σ_h^ι, $(u_{h-2}^\iota, v_1^\iota)$, arrives, it is assigned the route $\langle u_{h-2}^\iota, u_{h-1}^\iota, v_1^\iota\rangle$, which causes the maximum congestion to be $(h-1)(\iota-1)+h = (h-1)\iota+1$. ■

We will now use Lemmas 3.4 and 3.5 to prove Theorem 3.2.

Proof of Theorem 3.2

By Lemma 3.5, when GREEDY_ROUTE1 is issued the request sequence $\sigma_{h,i} = \sigma_h^1 \circ \sigma_h^2 \circ \cdots \circ \sigma_h^i$, it can incur congestion

$$\mu = (h-1)i + 1. \tag{3.8}$$

To rewrite μ in terms of m, we first notice from (3.7) that

$$i \geq \sqrt{\frac{m}{3h-2}}. \tag{3.9}$$

Substituting (3.9) into (3.8), we see that

$$\begin{aligned}\mu &\geq (h-1)\left(\sqrt{\frac{m}{3h-2}}\right) + 1\\ &\geq (h-1)\left(\sqrt{\frac{m}{4h-4}}\right) + 1 \qquad \text{(since } h > 1\text{)}\\ &= \frac{\sqrt{(h-1)m}}{2} + 1\\ &= \Omega\left(\sqrt{\mathcal{D}m}\right).\end{aligned}$$

The last equality follows from the fact that, for every request $f_j \in A_h^\iota$, $\iota = 1, 2, \ldots, i$, $d_j = 1$, $D_j = h$, and these are the minimum and maximum such values possible in the network $G_{h,i}$. Thus, since $\mu^* = 1$ by Lemma 3.4, the competitive ratio of GREEDY_ROUTE1 is $\Omega\left(\sqrt{\mathcal{D}m}\right)$. ■

3.4 A Lower Bound for Layered Networks

According to Theorem 3.1, GREEDY_ROUTE1 is $O\left(\sqrt{\mathcal{L}m}\right)$ competitive on layered networks since the length of every path for any request is the same, causing $\mathcal{D} = 1$. In this section, we show that Theorem 3.1 is tight up to a factor of $O\left(\sqrt{\mathcal{L}}\right)$ on $\mathcal{L}$-layered networks, even on an *end-to-end* instance in which every path has length L.

This result also answers a relevant question about the way GREEDY_ROUTE1 breaks ties. GREEDY_ROUTE1, as defined, breaks ties arbitrarily: it may choose any route that satisfies the requirements of the algorithm. Thus, the upper bound of Theorem 3.1 holds for any tie breaking scheme. In practice however it would be reasonable to choose the shortest route that satisfies the algorithm. In this way, the fewest links are affected by each decision and requests will presumably reach their destinations sooner. However, this result shows that if GREEDY_ROUTE1 uses this tie breaking scheme, its competitive ratio is still $\Omega(\sqrt{m})$. Notice that this result also holds for an algorithm which first narrows its consideration to the set of shortest routes and then uses GREEDY_ROUTE1 to choose from among those routes.

We will prove the following theorem:

Theorem 3.3

The competitive ratio of GREEDY_ROUTE1 is $\Omega(\sqrt{m})$ on layered networks for arbitrarily large m.

The proof will use the same format and notation as the proof of Theorem 3.2. We will first construct a layered network and a corresponding request sequence, and then use them to show that GREEDY_ROUTE1 can incur a congestion factor sufficient to prove the theorem.

3.4.1 The Network

For any integers $h > 1$ and $i \geq 1$, the network is represented by a directed graph $G_{h,i}$ with h layers and i connected components. (Note that we use the same notation as in Section 3.3, but with different definitions.) All network links have unit capacity. $G_{2,i}$ and $G_{3,i}$ are displayed in Figure 3.7 and $G_{6,2}$ is displayed in Figure 3.8.

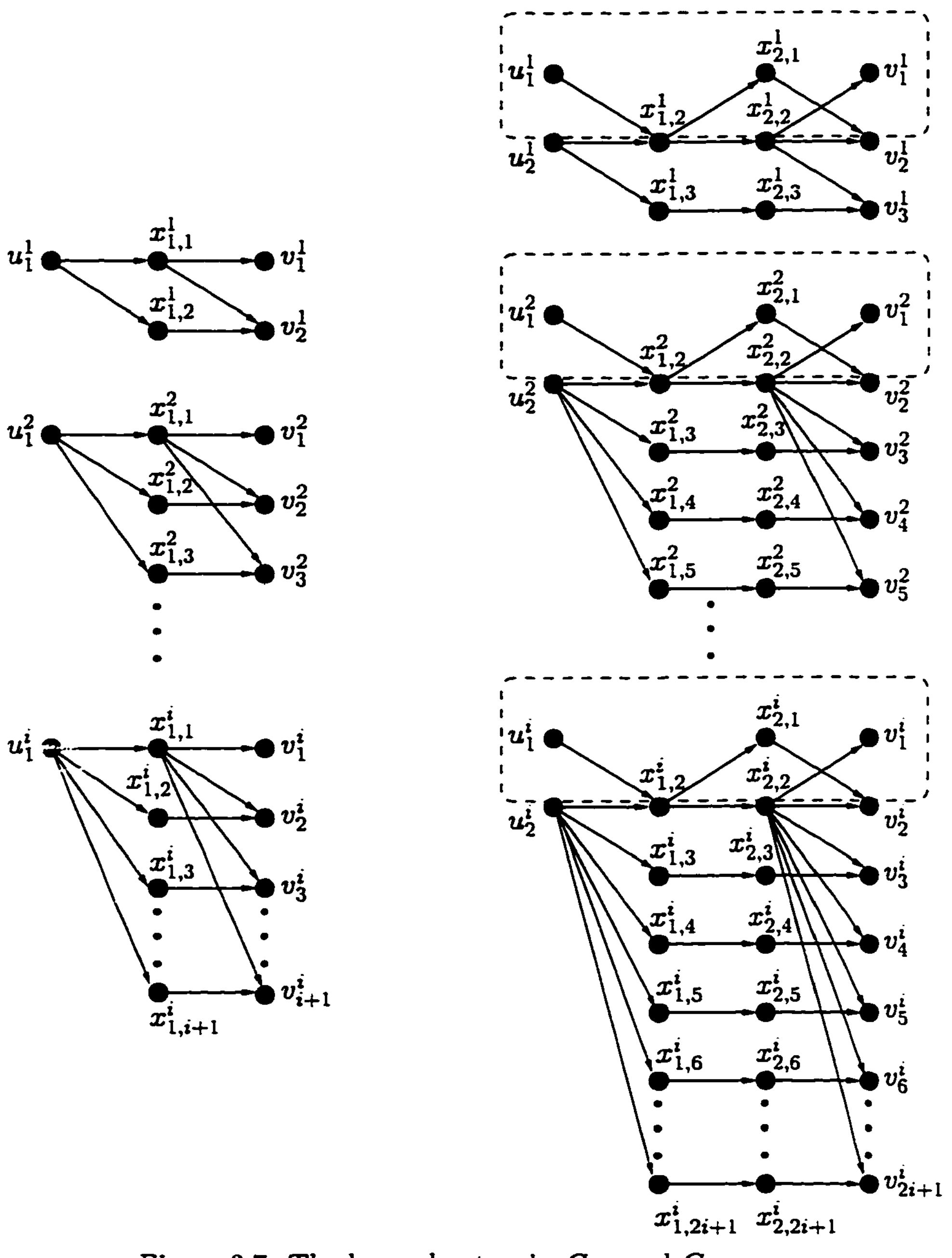
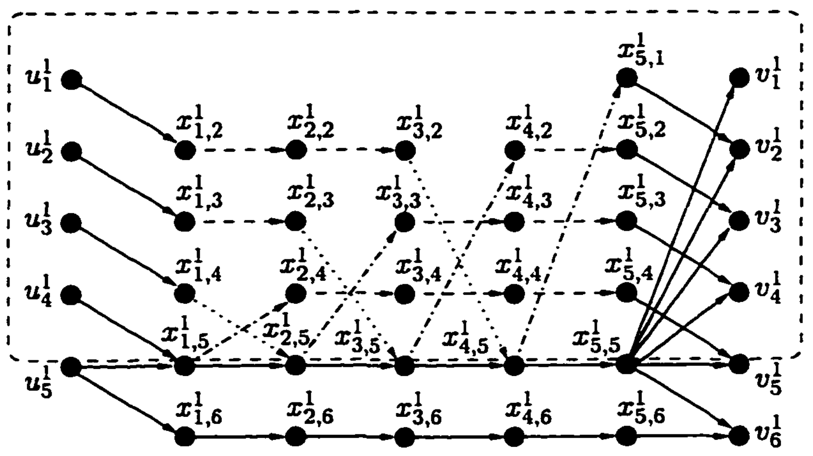


Figure 3.7: The layered networks $G_{2,i}$ and $G_{3,i}$.



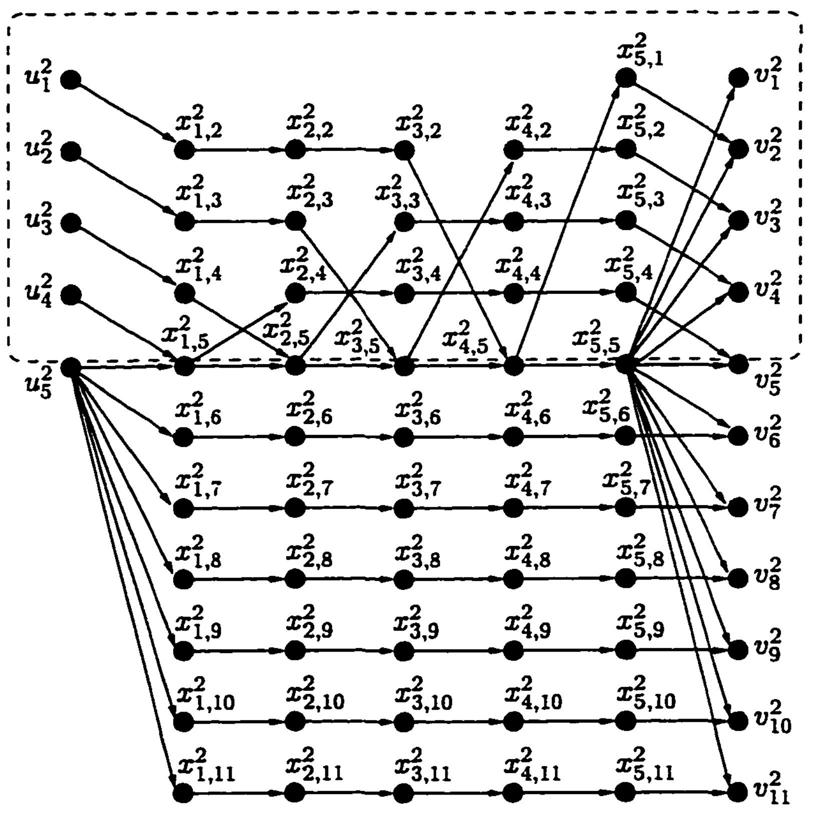


Figure 3.8: The layered network $G_{6,2}$.

The vertex set of $G_{h,i}$ is defined to be the union

$$\bigcup_{\iota=1}^{i} (U_{h,\iota} \cup V_{h,\iota} \cup X_{h,\iota})$$

where

- $U_{h,\iota} = \{u_1^\iota, u_2^\iota, \ldots, u_{h-1}^\iota\}$ is the set of source nodes,
- $V_{h,\iota} = \left\{v_1^\iota, v_2^\iota, \ldots, v_{(h-1)\iota+1}^\iota\right\}$ is the set of destination nodes, and
- $X_{h,\iota} = \bigcup_{\kappa=2}^{(h-1)\iota+1} \bigcup_{\nu=1}^{h-1} \{x_{\nu,\kappa}^\iota\} \cup \{x_{h-1,1}^\iota\}$ is the set of intermediate nodes.

Note that

$$\begin{aligned} n &= \left| \bigcup_{\iota=1}^{i} (U_{h,\iota} \cup V_{h,\iota} \cup X_{h,\iota}) \right| \\ &= \left(\frac{h(h-1)}{2}\right) i^2 + \left(\frac{h(h+1)+2}{2}\right) i. \end{aligned}$$

The arc set of $G_{h,i}$ is defined to be

$$\bigcup_{\iota=1}^{i} E_{h,\iota}.$$

Each set $E_{h,\iota}$, $\iota = 1, 2, \ldots, i$, is the arc set of a different connected component

$$(U_{h,\iota} \cup V_{h,\iota} \cup X_{h,\iota}, E_{h,\iota})$$

and is most clearly defined as the union of two sets: Upper($E_{h,\iota}$) and Lower($E_{h,\iota}$). The arcs in Upper($E_{h,\iota}$) are enclosed in a dashed box in Figures 3.7 and 3.8. Upper($E_{h,\iota}$) is composed of three logical parts:

- The "head" of Upper($E_{h,\iota}$) contains the arcs leaving the source nodes, and is defined to be

$$\bigcup_{\kappa=1}^{h-2} \left\{ \left(u_\kappa^\iota, x_{1,\kappa+1}^\iota\right) \right\}.$$

- The "tail" of Upper($E_{h,\iota}$) contains the arcs entering the destination nodes, and is defined to be

$$\bigcup_{\kappa=1}^{h-2} \left\{ \left(x_{h-1,\kappa}^\iota, v_{\kappa+1}^\iota\right), \left(x_{h-1,h-1}^i, v_\kappa^\iota\right) \right\}.$$

- The "body" of Upper($E_{h,\iota}$) is the union of three sets:

$$\bigcup_{\kappa=2}^{h-2} \bigcup_{\nu=1}^{h-2} \left\{ \left(x_{\nu,\kappa}^\iota, x_{\nu+1,\kappa}^\iota\right) \right\} \setminus \bigcup_{\kappa=2}^{h-2} \left\{ \left(x_{h-\kappa-1,\kappa}^\iota, x_{h-\kappa,\kappa}^\iota\right) \right\} \cup \tag{3.10}$$

$$\bigcup_{\kappa=2}^{h-2} \left\{ \left(x_{h-\kappa-1,\kappa}^\iota, x_{h-\kappa,h-1}^\iota\right) \right\} \cup \tag{3.11}$$

$$\bigcup_{\kappa=1}^{h-2} \left\{ \left(x_{h-\kappa-1,h-1}^\iota, x_{h-\kappa,\kappa}^\iota\right) \right\}. \tag{3.12}$$

The edges in the "body" of Upper($E_{h,\iota}$) are identified in the top component of $G_{6,2}$ in Figure 3.8. The edges marked (3.10) are dashed, the edges marked (3.11) are dotted, and the edges marked (3.12) are dot-dashed.

Lower($E_{h,\iota}$) (the edges not enclosed in a dashed box in Figures 3.7 and 3.8) is similarly composed of three logical parts:

- The "head" of Lower($E_{h,\iota}$) is

$$\bigcup_{\kappa=h-1}^{(h-1)\iota+1} \left\{ \left(u_{h-1}^\iota, x_{1,\kappa}^\iota\right) \right\}.$$

- The "tail" of Lower$(E_{h,\iota})$ is

$$\bigcup_{\kappa=h}^{(h-1)\iota+1} \left\{\left(x^{\iota}_{h-1,h-1}, v^{\iota}_{\kappa}\right), \left(x^{\iota}_{h-1,\kappa}, v^{\iota}_{\kappa}\right)\right\} \cup \left\{\left(x^{\iota}_{h-1,h-1}, v^{\iota}_{h-1}\right)\right\}.$$

- The "body" of Lower$(E_{h,\iota})$ is

$$\bigcup_{\kappa=h-1}^{(h-1)\iota+1} \bigcup_{\nu=1}^{h-2} \left\{\left(x^{\iota}_{\nu,\kappa}, x^{\iota}_{\nu+1,\kappa}\right)\right\}.$$

Note that

$$\begin{aligned} m &= \left|\bigcup_{\iota=1}^{i} (\text{Upper}(E_{h,\iota}) \cup \text{Lower}(E_{h,\iota}))\right| \\ &= \left(\frac{h^2-1}{2}\right) i^2 + \left(\frac{h^2-1}{2} + h\right) i \\ &\le (h^2+h-1)i^2. \end{aligned} \tag{3.13}$$

3.4.2 The Request Sequence

We now define the request sequence for the network $G_{h,i}$. As in the previous section, we will refer to each request simply as (s_j, t_j). The request sequence will be denoted $\sigma_{h,i}$ and is defined to be the concatenation of i subsequences:

$$\sigma_{h,i} = \sigma^1_h \circ \sigma^2_h \circ \cdots \circ \sigma^i_h.$$

Each σ^{ι}_h is further defined to be the concatenation of two subsequences A^{ι}_h and B^{ι}_h where

$$A^{\iota}_h = \left(u^{\iota}_{h-1}, v^{\iota}_h\right), \left(u^{\iota}_{h-1}, v^{\iota}_{h+1}\right), \ldots, \left(u^{\iota}_{h-1}, v^{\iota}_{(h-1)\iota+1}\right)$$

	f_j	(s_j, t_j)	P_j	P_j^*
σ_3^1	f_1	(u_2^1, v_3^1)	$\langle u_2^1, x_{1,2}^1, x_{2,2}^1, v_3^1 \rangle$	$\langle u_2^1, x_{1,3}^1, x_{2,3}^1, v_3^1 \rangle$
	f_2	(u_2^1, v_2^1)	$\langle u_2^1, x_{1,2}^1, x_{2,2}^1, v_2^1 \rangle$	$\langle u_2^1, x_{1,2}^1, x_{2,1}^1, v_2^1 \rangle$
	f_3	(u_1^1, v_1^1)	$\langle u_1^1, x_{1,2}^1, x_{2,2}^1, v_1^1 \rangle$	$\langle u_1^1, x_{1,2}^1, x_{2,2}^1, v_1^1 \rangle$
σ_3^2	f_4	(u_2^2, v_3^2)	$\langle u_2^2, x_{1,2}^2, x_{2,2}^2, v_3^2 \rangle$	$\langle u_2^2, x_{1,3}^2, x_{2,3}^2, v_3^2 \rangle$
	f_5	(u_2^2, v_4^2)	$\langle u_2^2, x_{1,2}^2, x_{2,2}^2, v_4^2 \rangle$	$\langle u_2^2, x_{1,4}^2, x_{2,4}^2, v_4^2 \rangle$
	f_6	(u_2^2, v_5^2)	$\langle u_2^2, x_{1,2}^2, x_{2,2}^2, v_5^2 \rangle$	$\langle u_2^2, x_{1,5}^2, x_{2,5}^2, v_5^2 \rangle$
	f_7	(u_2^2, v_2^2)	$\langle u_2^2, x_{1,2}^2, x_{2,2}^2, v_2^2 \rangle$	$\langle u_2^2, x_{1,2}^2, x_{2,1}^2, v_2^2 \rangle$
	f_8	(u_1^2, v_1^2)	$\langle u_1^2, x_{1,2}^2, x_{2,2}^2, v_1^2 \rangle$	$\langle u_1^2, x_{1,2}^2, x_{2,2}^2, v_1^2 \rangle$
σ_3^3	f_9	(u_2^3, v_3^3)	$\langle u_2^3, x_{1,2}^3, x_{2,2}^3, v_3^3 \rangle$	$\langle u_2^3, x_{1,3}^3, x_{2,3}^3, v_3^3 \rangle$
	f_{10}	(u_2^3, v_4^3)	$\langle u_2^3, x_{1,2}^3, x_{2,2}^3, v_4^3 \rangle$	$\langle u_2^3, x_{1,4}^3, x_{2,4}^3, v_4^3 \rangle$
	f_{11}	(u_2^3, v_5^3)	$\langle u_2^3, x_{1,2}^3, x_{2,2}^3, v_5^3 \rangle$	$\langle u_2^3, x_{1,5}^3, x_{2,5}^3, v_5^3 \rangle$
	f_{12}	(u_2^3, v_6^3)	$\langle u_2^3, x_{1,2}^3, x_{2,2}^3, v_6^3 \rangle$	$\langle u_2^3, x_{1,6}^3, x_{2,6}^3, v_6^3 \rangle$
	f_{13}	(u_2^3, v_7^3)	$\langle u_2^3, x_{1,2}^3, x_{2,2}^3, v_7^3 \rangle$	$\langle u_2^3, x_{1,7}^3, x_{2,7}^3, v_7^3 \rangle$
	f_{14}	(u_2^3, v_2^3)	$\langle u_2^3, x_{1,2}^3, x_{2,2}^3, v_2^3 \rangle$	$\langle u_2^3, x_{1,2}^3, x_{2,1}^3, v_2^3 \rangle$
	f_{15}	(u_1^3, v_1^3)	$\langle u_1^3, x_{1,2}^3, x_{2,2}^3, v_1^3 \rangle$	$\langle u_1^3, x_{1,2}^3, x_{2,2}^3, v_1^3 \rangle$

Table 3.2: Request sequence $\sigma_{3,3}$ on the layered network $G_{3,3}$.

and

$$B_h^\iota = \left(u_{h-1}^\iota, v_{h-1}^\iota\right), \left(u_{h-2}^\iota, v_{h-2}^\iota\right), \ldots, \left(u_1^\iota, v_1^\iota\right).$$

Note that

$$\begin{aligned} k &= |\sigma_{h,i}| \\ &= \sum_{\iota=1}^{i} \left(|A_h^\iota| + |B_h^\iota|\right) \\ &= \left(\frac{h-1}{2}\right) i^2 + \left(\frac{h+1}{2}\right) i. \end{aligned}$$

As an example of this construction, consider the first three columns of Table 3.2, which contain the requests in $\sigma_{3,3}$. (See Figure 3.7 for the corresponding network $G_{3,3}$.) The fourth column of Table 3.2 contains the worst case routes assigned by GREEDY_ROUTE1, while the fifth column contains optimal routes.

3.4.3 Proof of the Lower Bound

As in the previous section, the optimal route assignment for $\sigma_{h,i}$ is pairwise edge disjoint. Informally, in an optimal route assignment, each request in A_h^ι, for $\iota = 1, 2, \ldots, i$, is assigned its "lower" route (as the networks are drawn in Figures 3.7 and 3.8), while each request in B_h^ι, for $\iota = 1, 2, \ldots, i$, is assigned its "upper" route. In general, we have the following lemma:

Lemma 3.6

For all integers $h > 1$ and $i \geq 1$, the set of optimal routes for request sequence $\sigma_{h,i}$ on the layered network $G_{h,i}$ is pairwise edge disjoint.

Proof

Notice that the set of paths for the requests in each subsequence σ_h^ι of $\sigma_{h,i}$ is contained in a different connected component of $G_{h,i}$. Thus, to prove the lemma, we can simply specify a pairwise edge disjoint set of paths for each subsequence σ_h^ι, $\iota = 1, 2, \ldots, i$.

First, we assign each request $(u_{h-1}^\iota, v_\kappa^\iota) \in A_h^\iota$, $\kappa = h, h+1, \ldots, (h-1)\iota + 1$, to the path

$$\langle u_{h-1}^\iota, x_{1,\kappa}^\iota, x_{2,\kappa}^\iota, \ldots, x_{h-1,\kappa}^\iota, v_\kappa^\iota \rangle .$$

Since the value of κ for each request in A_h^ι is different, this set of paths is pairwise edge disjoint. To each request $(u_\kappa^\iota, v_\kappa^\iota) \in B_h^\iota$, $\kappa = 1, 2, \ldots, h-2$, we assign the path

$$\langle u_\kappa^\iota, x_{1,\kappa+1}^\iota, \ldots, x_{h-\kappa-2,\kappa+1}^\iota, x_{h-\kappa-1,h-1}^\iota, x_{h-\kappa,h-1}^\iota, x_{h-\kappa+1,\kappa-1}^\iota, \ldots, x_{h-1,\kappa-1}^\iota, v_\kappa^\iota \rangle .$$

To request $\left(u^{\iota}_{h-1}, v^{\iota}_{h-1}\right) \in B^{\iota}_{h}$, we assign the path

$$\left\langle u^{\iota}_{h-1}, x^{\iota}_{1,h-1}, x^{\iota}_{2,h-2}, \ldots, x^{\iota}_{h-1,h-2}, v^{\iota}_{h-1}\right\rangle . \qquad \blacksquare$$

In the worst case, GREEDY_ROUTE1 can assign to each request its non-optimal route (with the exception of the requests $(u^{\iota}_{1}, v^{\iota}_{1})$, $\iota = 1, 2, \ldots, i$, which each have only one route). In the example in Table 3.2, GREEDY_ROUTE1 has incurred congestion 3 after routing the requests in σ^{1}_{3}, 5 after routing the requests in σ^{2}_{3}, and 7 after routing the requests in σ^{3}_{3}.

In general, we have the following lemma:

Lemma 3.7

For all integers $h > 1$, $i \geq 1$, and $1 \leq \iota \leq i$, GREEDY_ROUTE1 can incur congestion $(h-1)\iota + 1$ after being issued request subsequence $\sigma^{1}_{h} \circ \sigma^{2}_{h} \circ \ldots \circ \sigma^{\iota}_{h}$ on layered network $G_{h,i}$.

Proof

The proof is by induction on ι.

Base case ($\iota = 1$). For the base case, we will consider how GREEDY_ROUTE1 might assign routes to the requests in subsequence σ^{1}_{h}. The first request will be $\left(u^{1}_{h-1}, v^{1}_{h}\right)$. Since GREEDY_ROUTE1 may assign either of the two possible routes to the request, suppose it assigns the route $\left\langle u^{1}_{h-1}, x^{1}_{1,h-1}, x^{1}_{2,h-1}, \ldots, x^{1}_{h-1,h-1}, v^{1}_{h}\right\rangle$. The next request will be $\left(u^{1}_{h-1}, v^{1}_{h-1}\right)$. Each of this request's two possible routes contains the link $\left(u^{1}_{h-1}, x^{1}_{1,h-1}\right)$ and so assigning the request to either route will increase the maximum congestion to 2. Therefore GREEDY_ROUTE1 may assign the route $\left\langle u^{1}_{h-1}, x^{1}_{1,h-1}, x^{1}_{2,h-1}, \ldots, x^{1}_{h-1,h-1}, v^{1}_{h-1}\right\rangle$. Now consider the next request, $\left(u^{1}_{h-2}, v^{1}_{h-2}\right)$. As was the case previously, each of the routes for this request contains a common link, in this case $\left(x^1_{1,h-1}, x^1_{2,h-1}\right)$, which has the maximum congestion. Thus, GREEDY_ROUTE1 may assign the route $\left\langle u^1_{h-2}, x^1_{1,h-1}, x^1_{2,h-1}, \dots, x^1_{h-1,h-1}, v^1_{h-2}\right\rangle$. The maximum congestion is now 3. In general, to request $(u^1_\kappa, v^1_\kappa) \in B^1_h$, $\kappa = 1, 2, \dots, h-2$, GREEDY_ROUTE1 may assign the route

$$\left\langle u^1_\kappa, x^1_{1,\kappa+1}, \dots, x^1_{h-\kappa-2,\kappa+1}, x^1_{h-\kappa-1,h-1}, \dots, x^1_{h-1,h-1}, v^1_\kappa \right\rangle .$$

When the last request $(u^1_1, v^1_1) \in \sigma^1_h$ arrives, it is assigned the route

$$\left\langle u^1_1, x^1_{1,2}, \dots, x^1_{h-3,2}, x^1_{h-2,h-1}, x^1_{h-1,h-1}, v^1_1 \right\rangle ,$$

which causes the maximum congestion to be h.

Induction step ($\iota > 1$). Let us now consider how GREEDY_ROUTE1 might assign the requests in $\sigma^1_h \circ \sigma^2_h \circ \dots \circ \sigma^\iota_h$. By the induction hypothesis, after assigning the requests in $\sigma^1_h \circ \sigma^2_h \circ \dots \circ \sigma^{\iota-1}_h$, GREEDY_ROUTE1 could have incurred congestion $(h-1)(\iota-1)+1$. Let us say that this has happened. Now we consider what can happen when the requests in σ^ι_h arrive. The paths assigned to these requests will all be contained in the subgraph $(U_{h,\iota} \cup V_{h,\iota} \cup X_{h,\iota}, E_{h,\iota})$. By construction, none of the links in $E_{h,\iota}$ has been traversed by a previous request. Therefore, none of the first $(h-1)(\iota-1)+1$ requests in σ^ι_h (the requests in A^ι_h) will increase the maximum congestion. Thus, GREEDY_ROUTE1 may assign each such request $(u^\iota_{h-1}, v^\iota_\kappa) \in A^\iota_h$, $\iota = h, h+1, \dots, (h-1)\iota+1$, to an arbitrary route, say $\left\langle u^\iota_{h-1}, x^\iota_{1,h-1}, x^\iota_{2,h-1}, \dots, x^\iota_{h-1,h-1}, v^\iota_\kappa \right\rangle$. As a result of this assignment, the links $\left(u^\iota_{h-1}, x^\iota_{1,h-1}\right), \left(x^\iota_{1,h-1}, x^\iota_{2,h-1}\right), \dots, \left(x^\iota_{h-2,h-1}, x^\iota_{h-1,h-1}\right)$ will each have congestion $(h-1)(\iota-1)+1$.

The next request will be $(u^{\iota}_{h-1}, v^{\iota}_{h-1})$. Each of this request's two possible routes contains the link $\left(u^{\iota}_{h-1}, x^{\iota}_{1,h-1}\right)$, which currently has the maximum congestion. Therefore GREEDY_ROUTE1 may assign the route $\left\langle u^{\iota}_{h-1}, x^{\iota}_{1,h-1}, x^{\iota}_{2,h-1}, \cdots, x^{\iota}_{h-1,h-1}, v^{\iota}_{h-1}\right\rangle$. As a result, the maximum congestion is now $(h-1)(\iota-1)+2$. Now consider the next request, $(u^{\iota}_{h-2}, v^{\iota}_{h-2})$. As was the case previously, each of the routes for this request contains a common link, in this case $\left(x^{\iota}_{1,h-1}, x^{\iota}_{2,h-1}\right)$, which has the maximum congestion. Thus, GREEDY_ROUTE1 may assign the route $\left\langle u^{\iota}_{h-2}, x^{\iota}_{1,h-1}, x^{\iota}_{2,h-1}, \cdots, x^{\iota}_{h-1,h-1}, v^{\iota}_{h-2}\right\rangle$. The maximum congestion is now $(h-1)(\iota-1)+3$. In general, for request $(u^{\iota}_{\kappa}, v^{\iota}_{\kappa}) \in B^{\iota}_{h}$, $\kappa = 1, 2, \ldots, h-2$, GREEDY_ROUTE1 may assign the route

$$\left\langle u^{\iota}_{\kappa}, x^{\iota}_{1,\kappa+1}, \cdots, x^{\iota}_{h-\kappa-2,\kappa+1}, x^{\iota}_{h-\kappa-1,h-1}, \cdots, x^{\iota}_{h-1,h-1}, v^{\iota}_{\kappa}\right\rangle .$$

When the last request in $(u^{\iota}_{1}, v^{\iota}_{1}) \in \sigma^{\iota}_{h}$ arrives, it is assigned the route

$$\left\langle u^{\iota}_{1}, x^{\iota}_{1,2}, \cdots, x^{\iota}_{h-3,2}, x^{\iota}_{h-2,h-1}, x^{\iota}_{h-1,h-1}, v^{\iota}_{1}\right\rangle ,$$

which causes the maximum congestion to be $(h-1)(\iota-1)+h = (h-1)\iota+1$. ■

Finally, we can use Lemmas 3.6 and 3.7 to prove Theorem 3.3:

Proof of Theorem 3.3

By Lemma 3.7, after GREEDY_ROUTE1 has finished routing the entire request sequence $\sigma_{h,i} = \sigma^{1}_{h} \circ \sigma^{2}_{h} \circ \cdots \circ \sigma^{i}_{h}$, it could have incurred congestion

$$\mu = (h-1)i + 1. \tag{3.14}$$

To rewrite μ in terms of m, we first notice from (3.13) that

$$i \geq \sqrt{\frac{m}{h^2+h-1}}. \tag{3.15}$$

By substituting (3.15) into (3.14), we see that

$$\begin{aligned}\mu &\geq (h-1)\sqrt{\frac{m}{h^2+h-1}}+1\\ &= \Omega\left(\sqrt{m}\right).\end{aligned}$$

Since $\mu^* = 1$ by Lemma 3.6, it follows that the competitive ratio of GREEDY_ROUTE1 is $\Omega\left(\sqrt{m}\right)$ on layered networks. ■

Chapter 4

Online Algorithm GREEDY_ROUTE2

4.1 Introduction

In this chapter, we consider a more refined greedy online algorithm for routing permanent virtual circuits.[1] GREEDY_ROUTE2 differs from GREEDY_ROUTE1 in that it bases its routing decision only on the maximum congestion of the routes that can be assigned to the request, rather than on the maximum congestion in the network. We will show that there are classes of networks on which GREEDY_ROUTE2 is guaranteed to have a competitive ratio that is polylogarithmic in the number of network nodes. Furthermore, GREEDY_ROUTE2 is as simple and fast as GREEDY_ROUTE1 in the worst case. (However, GREEDY_ROUTE1 is still faster than GREEDY_ROUTE2 in the best case.)

4.1.1 The Algorithm

The online algorithm we study in this chapter is formally defined as follows:

[1] A preliminary version of the results in this chapter appeared in [43].

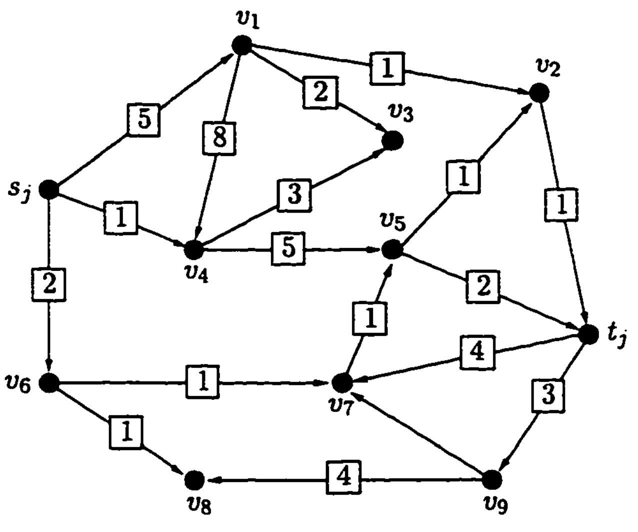


Figure 4.1: An example of a network graph.

Algorithm GREEDY_ROUTE2

For request f_j, assign any route $P \in \mathcal{P}_j$ which minimizes

$$\max_{e \in P} \left\{ \mu_j(e) + \frac{l_j}{w} \right\}.$$

Ties are broken arbitrarily.

In other words, GREEDY_ROUTE2 considers all routes for a request and assigns the one that would have the minimum congestion if the request were assigned to it. Consider again the network from the previous chapter, redisplayed in Figure 4.1. Recall that if an online algorithm receives the request $(s_j, t_j, 1, \tau)$, it can assign any of the following six routes:

1. $\langle s_j, v_1, v_4, v_5, t_j \rangle$,
2. $\langle s_j, v_1, v_4, v_5, v_2, t_j \rangle$,
3. $\langle s_j, v_1, v_2, t_j \rangle$,

Route	$a = 1.01$	$a = 1.1$	$a = 1.5$	$a = 1.9$	$a = 1.99$	Rank
1.	0.00400198	0.0401842	0.203469	0.368897	0.406263	5
2.	0.00499751	0.0497599	0.244778	0.434895	0.477138	6
3.	0.00299353	0.0293806	0.136864	0.232743	0.253113	1
4.	0.00399005	0.0390489	0.179956	0.303428	0.329424	4
5.	0.00299452	0.0294732	0.138647	0.237429	0.258549	2
6.	0.00398806	0.0388584	0.175934	0.292109	0.316116	3

Table 4.1: Costs computed by EXP_ROUTE for the example.

4. $\langle s_j, v_4, v_5, v_2, t_j \rangle$,

5. $\langle s_j, v_4, v_5, t_j \rangle$, or

6. $\langle s_j, v_6, v_7, v_5, t_j \rangle$.

If an online algorithm used either of the first two routes, the resulting congestion on either route would be 9. If an online algorithm used any of the next three routes, the resulting congestion on each route would be 6. However, if the algorithm used the last route, the resulting congestion on that route would be only 3. GREEDY_ROUTE2 would therefore select the last route. On the other hand, GREEDY_ROUTE1 could have chosen any of the last four routes because no matter which it chose, the resulting network congestion would be 8 (on link (v_1, v_4)).

For comparison, if $w = 10$, the algorithm EXP_ROUTE [4] would choose the third route, for a wide variety of values for a. The exponential costs that EXP_ROUTE computes for each route are shown in Table 4.1. Notice that EXP_ROUTE favors short routes, until the congestion on those routes reaches a high enough threshold. Also notice how the algorithm distinguishes between routes 3 and 5, which have almost identical congestion on their links.

Since GREEDY_ROUTE2 limits its consideration to just the links that could possibly carry the request to its destination, an adversary like the one used to prove Theorem 3.2 cannot fool GREEDY_ROUTE2 into choosing bad routes in the same way that it can fool GREEDY_ROUTE1. Consider what happens when GREEDY_ROUTE2 is presented with the constructions used to prove Theorems 3.2 and 3.3. When presented with network $G_{h,1}$ and request sequence $\sigma_{h,1}$ (in the proof of either Theorem 3.2 or 3.3), GREEDY_ROUTE2 can incur congestion h in the same way that GREEDY_ROUTE1 can. But when presented with network $G_{h,i}$ and request sequence $\sigma_{h,i}$, for $i > 1$, GREEDY_ROUTE2 will incur congestion *at most* h, whereas GREEDY_ROUTE1 could incur congestion $(h-1)i+1$.

The behavior of GREEDY_ROUTE2 on network $G_{h,1}$ and request sequence $\sigma_{h,1}$ proves only that the competitive ratio of GREEDY_ROUTE2 is $\Omega(m)$, where m is the number of network links, *if* the length of the longest path is $\Omega(m)$, in effect proving a lower bound on the order of the longest path length $d = \max_j \max_{P \in \mathcal{P}_j} |P|$. In Section 4.3, we will strengthen this result by showing that the competitive ratio of GREEDY_ROUTE2 is $\Omega(d + \log(n-d))$. Therefore, the greedy algorithm, when applied to routing, no longer has a logarithmic competitive ratio for general instances.

We can also use a result from the previous chapter to infer an upper bound on the competitive ratio of GREEDY_ROUTE2, since Theorem 3.1 holds for GREEDY_ROUTE2 just as it does for GREEDY_ROUTE1. The proof of this fact is identical to the proof of Theorem 3.1, primarily because Lemma 3.2 holds for GREEDY_ROUTE2 also. We restate this result for GREEDY_ROUTE2 here for completeness.

Theorem 4.1

The competitive ratio of GREEDY_ROUTE2 is $O\left(\sqrt{\mathcal{D}\mathcal{L}m}\right)$.

In Section 4.2, we will show that a stronger upper bound on the competitive ratio of

GREEDY_ROUTE2 is possible on some networks. Specifically, we will prove that the competitive ratio of GREEDY_ROUTE2 is $O(d \log n)$ if there exist pairwise edge disjoint routes for the requests. (The actual optimal path requirement is slightly less strict and arises from the proof technique. The same result holds if the ratio of the maximum to minimum bandwidth requirement is constant and the maximum number of optimal paths that intersect is bounded by a polynomial in n.) We believe that a similar (or better) upper bound will still hold if the optimal paths are not disjoint. We provide some intuitive support for this conjecture at the end of Section 4.2.

Consider for a moment instances that obey this optimal route requirement. In this context, our result indicates that if the network has paths whose lengths are $O(\log n)$ (or $O(\log m)$) (e.g., splitter networks) then the competitive ratio of GREEDY_ROUTE2 is guaranteed to be a polylogarithmic function of the number of network nodes, which comes very close to matching the problem lower bound of $\Omega(\log n)$. If the length of the paths is constant then GREEDY_ROUTE2 is asymptotically optimal.

4.1.2 Tradeoffs Between Time Complexity and Routing Efficiency

Consider the pseudocode for GREEDY_ROUTE2 in Figure 4.2. Similar to GREEDY_ROUTE1, GREEDY_ROUTE2 requires only $\sum_{P \in \mathcal{P}_j} |P|$ comparisons and $\sum_{P \in \mathcal{P}_j} |P|$ additions in the worst case to assign a route to request f_j.[2] However, the best case time for GREEDY_ROUTE2 is the same as the worst case, unlike GREEDY_ROUTE1 which can require as few as $|\mathcal{P}_j|$ comparisons and additions. Summarizing, GREEDY_ROUTE1 can be faster than GREEDY_ROUTE2,

[2]As was the case with GREEDY_ROUTE1, if the algorithm has available to it $O(m)$ bytes of working memory, then the values $\left(\mu_j(e) + \frac{l_i}{w}\right)$ can be computed once for each path, bounding the number of additions by $O(m)$.

```
GREEDY_ROUTE2
   for each $f_j \in \sigma$ do
      $P_j \leftarrow$ Assign_GREEDY_ROUTE2($j$)
   end for

function Assign_GREEDY_ROUTE2($j$)
   $\mu' \leftarrow \infty$
   $P' \leftarrow \emptyset$
   for all $P \in \mathcal{P}_j$ do
      temp $\leftarrow \max_{e \in P} \left\{ \mu_j(e) + \frac{l_j}{c(e)} \right\}$
      if temp < $\mu'$ then
         $\mu' \leftarrow$ temp
         $P' \leftarrow P$
      end if
   end for
   return $P'$
```

Figure 4.2: A pseudocode representation of GREEDY_ROUTE2.

but GREEDY_ROUTE2 can assign better routes than GREEDY_ROUTE1. As was discussed in the previous chapter, both algorithms make their decisions more quickly than EXP_ROUTE[4]. The real time difference would be accentuated on networks in which a large number of exponential computations is required.

As was the case with GREEDY_ROUTE1, GREEDY_ROUTE2 can be expected to perform worse than EXP_ROUTE when paths can be relatively long. This factor of d makes intuitive sense in the competitive ratio of GREEDY_ROUTE2 when one compares GREEDY_ROUTE2 to EXP_ROUTE. When all routes for a request have unit length, both algorithms will choose a route with the minimum congestion. As the maximum lengths of the routes increase, EXP_ROUTE will be more discriminating than GREEDY_ROUTE2 and the gap between their competitive ratios will increase.

This can be illustrated with a few examples. First, suppose that both algorithms are confronted with a choice of two routes — one containing one link with congestion c and another containing 100 links with maximum congestion c. Whereas GREEDY_ROUTE2 will not discriminate between the two routes, EXP_ROUTE will choose the shortest one. EXP_ROUTE will be more discriminating even if the two paths were to have the same length. For example, consider two paths with 5 links. The first path has congestion 5 on one link and no congestion on any other link. The second path has congestion 5, 4, 3, 2, and 1 on its five links. EXP_ROUTE will clearly choose the first path, which is intuitively a much better choice, while GREEDY_ROUTE2 will not discriminate. This phenomenon was also illustrated in the example associated with Figure 4.1. We quantify in this chapter the degree to which the competitive ratios of GREEDY_ROUTE2 and EXP_ROUTE differ.

As was discussed previously, the competitive ratio of GREEDY_ROUTE2 is polylogarithmic when $d = O(\log n)$ (at least for feasible instances). Therefore, we should expect that GREEDY_ROUTE2 would perform well in these cases. It seems reasonable to expect large networks with arbitrary topologies to have relatively short paths between most nodes. For example, by using the `traceroute` program, one can see that between most pairs of randomly selected IP addresses in the United States, there is a site to site route containing at most 15 nodes. While one could probably concoct a route with many more nodes, only these relatively short routes are used in practice. By restricting GREEDY_ROUTE2 or GREEDY_ROUTE1 to a set of relatively short routes *a priori*, one would expect good performance based on our results for these algorithms (assuming the optimal algorithm is also restricted to these routes). This method of considering restricted sets of paths might be used with success on any network with small diameter. Another good scenario for GREEDY_ROUTE2 might be one in which the traffic pattern favors source/destination pairs which have short routes.

4.2 An Upper Bound

In this section, we prove our new upper bound on the competitive ratio of GREEDY_ROUTE2. The method we use to prove the result is adapted from a technique used by Azar, Naor and Rom [17, 18] to show that an online algorithm similar to GREEDY_ROUTE2 is $O(\log n)$ competitive for an online load balancing problem in which each job can be assigned to exactly one of a subset of n identical machines. Intuitively, their idea was to conceptualize the online algorithm's assignment on each machine as a partition of a number of successive *layers*.[3] The sum of the weights of the jobs in each layer (which we will call the layer's *width*) is equal to the optimal load, except for possibly the last layer which contains the remaining weight less than the optimal load. (All subsequent layers after this last nonzero one have width 0.) They showed that, for any i, the sum of the widths of all the i^{th} layers is at least as large as the sum of the widths of all subsequent layers. From this step, the logarithmic competitive ratio follows in a relatively straightforward way. In our proof, we adapt this method to work with paths, rather than single machines. Our idea is to consider the final online congestion caused by GREEDY_ROUTE2 on each *optimal path* and partition the congestion on each of these optimal paths into layers with width at most $\tilde{\Lambda}$, the maximum bandwidth requirement divided by the capacity of the network links. We then show that, for any i, the sum of the widths of all the i^{th} layers, multiplied by a certain factor, is at least as large as the sum of the widths of all subsequent layers, and our result follows.

[3]The meaning of *layer* in this section is unrelated to *layered* graphs discussed in previous chapters.

4.2.1 Notation

Before formally stating our theorem and its proof, we will present the notation we will use, along with several useful facts. (Some of this notation was defined in the previous chapter; we restate it here for convenience.)

- P_j^* is the path chosen for request f_j by an optimal algorithm.
- μ^* is the network congestion incurred by an optimal algorithm.
- $d_j = \min_{P \in \mathcal{P}_j} |P|$ is the length of the shortest path between nodes s_j and t_j.
- $D_j = \max_{P \in \mathcal{P}_j} |P|$ is the length of the longest path between nodes s_j and t_j.
- $\mathcal{D} = \max_{1 \leq j \leq k} \frac{D_j}{d_j}$.
- $d = \max_{1 \leq j \leq k} D_j$ is the length of the longest path that can be assigned to any request.
- $\lambda = \min_{1 \leq j \leq k} l_j$ and $\Lambda = \max_{1 \leq j \leq k} l_j$ are the minimum and maximum bandwidth requirements, respectively.
- $\widetilde{\Lambda} = \frac{\Lambda}{w}$, where w is the width of the network links. Also, for any j, $1 \leq j \leq k$, let $\widetilde{l_j} = \frac{l_j}{w}$.

  Fact 4.1

  $\mu^* \geq \widetilde{\Lambda}$.

  Fact 4.2

  $\mu^* \geq \frac{1}{m} \sum_{j=1}^{k} \left(d_j \widetilde{l_j} \right)$.
- $\mathcal{L} = \frac{\Lambda}{\lambda}$ is the ratio of the maximum to minimum bandwidth requirement.

- $e_j^* \in P_j^*$ is an edge $e \in P_j^*$ satisfying $\mu(e) = \mu(P_j^*)$. Ties are broken in favor of the link with the smallest index.

- $\mathcal{I} = \max_{e \in E} \left|\left\{j : e_j^* = e, 1 \le j \le k\right\}\right|$ is the maximum number of optimal paths P_j^* whose edge e_j^* is the same.

The following fact is true since $\mathcal{I}$ is a lower bound on the maximum number of optimal paths that intersect at the same edge.

Fact 4.3

$\mu^* \ge \frac{\mathcal{I}\lambda}{w}$.

- For all $i \ge 1$, $1 \le j \le k$, $W_{ij} = \begin{cases} \widetilde{\Lambda}, & \text{if } \mu(P_j^*) \ge i\widetilde{\Lambda} \\ \mu(P_j^*) - (i-1)\widetilde{\Lambda}, & \text{if } (i-1)\widetilde{\Lambda} < \mu(P_j^*) < i\widetilde{\Lambda} \\ 0, & \text{otherwise} \end{cases}$.

As discussed above, we partition the congestion on each edge e_j^* into layers with width at most $\widetilde{\Lambda}$. W_{ij} is the width of the i^{th} such layer.

Fact 4.4

For all j, $\mu(P_j^*) = \sum_i W_{ij}$.

- W_{ij}^r is the portion of W_{ij} on edge e_j^* that was added by request f_r, $1 \le r \le k$, in the online route assignment (divided by w).

The following fact is true because if $W_{ij}^r > 0$ for more than two layers, then $\widetilde{l_r} > \widetilde{\Lambda}$, which is impossible.

Fact 4.5

$W_{ij}^r > 0$ for at most 2 values of i, which must be consecutive.

- $S_{ij} = \{r : W^j_{lr} > 0 \text{ for some } l > i\}$ is the set of indices of optimal paths P^*_r such that f_j traverses e^*_r in a layer greater than i in the online route assignment.

  The following fact is true because f_j traverses $|P_j| \leq D_j$ links, each of which may be e^*_r for at most $\mathcal{I}$ optimal paths P^*_r.

  Fact 4.6

  For all i and j, $|S_{ij}| \leq |S_{0j}| \leq D_j\mathcal{I}$.

- $R_{ij} = \sum_{r \in S_{ij}} \left(\tilde{l_j} - W^j_{ir}\right)$ is the total congestion incurred by f_j in all layers greater than i.

  Let us explain this definition in more detail. The quantity we are describing is at most equal to $\tilde{l_j}$ times the number of edges e^*_r to which f_j adds congestion after layer i (which is $|S_{ij}| \cdot \tilde{l_j}$). We must say "at most" because it is possible, if f_j adds congestion to layer $i+1$ on some edge e^*_r, that some congestion was also added to layer i. (No congestion could have been added to a layer before i by Fact 4.5.) Thus, to get R_{ij}, we must subtract off this quantity, W^j_{ir}, for each edge on which this is the case. On the other hand, if f_j does not add any congestion to any edges e^*_r in layer $i+1$, then $R_{ij} = |S_{ij}| \cdot \tilde{l_j}$. The equation above takes into account both of these possibilities because in the second case, $W^j_{ir} = 0$.

  Fact 4.7

  For all i and j, $R_{ij} \leq |S_{ij}| \cdot \tilde{l_j} \leq D_j\mathcal{I}\tilde{\Lambda}$.

- $W_i = \sum_{j=1}^{k} W_{ij}$.

- $R_i = \sum_{j=1}^{k} R_{ij}$.

The next fact follows simply from the definitions. On the other hand, Fact 4.9 requires an explicit proof.

Fact 4.8

$R_0 \leq \mathcal{I} \sum_{j=1}^{k} \left(D_j \tilde{l}_j \right)$.

Fact 4.9

$R_i = \sum_{l>i} W_l$.

Proof

Notice that

$$\begin{aligned}
R_i &= \sum_{j=1}^{k} R_{ij} && \text{by the definition of } R_i \\
&= \sum_{j=1}^{k} \sum_{r \in S_{ij}} \left(\tilde{l}_j - W_{ir}^{j} \right) && \text{by the definition of } R_{ij} \\
&= \sum_{r=1}^{k} \sum_{j : r \in S_{ij}} \left(\tilde{l}_j - W_{ir}^{j} \right) && \text{by changing the order of summation} \\
&= \sum_{r=1}^{k} \sum_{l>i} W_{lr} && \text{(see below)} \\
&= \sum_{l>i} \sum_{r=1}^{k} W_{lr} && \text{by changing the order of summation} \\
&= \sum_{l>i} W_l && \text{by the definition of } W_i.
\end{aligned}$$

A more lengthy explanation is in order for the fourth equality. The term

$$\sum_{j : r \in S_{ij}} \left(\tilde{l}_j - W_{ir}^{j} \right)$$

refers to the total bandwidth of all requests f_j (divided by w) that cross e_r^* after layer i, minus the portions that fall in layer i. In other words, this is the total congestion on edge e_r^* after layer i. (See the text accompanying the definition of R_{ij} for an explanation of the "$- W_{ir}^j$".) By definition, this quantity is simply the total width of all the layers of e_r^* after layer i, or $\sum_{l>i} W_{lr}$. ■

The last fact follows from Fact 4.9.

Fact 4.10

$R_i = R_{i-1} - W_i$.

4.2.2 Proof of the Upper Bound

We can now formally state our upper bound result for GREEDY_ROUTE2:

Theorem 4.2

The competitive ratio of GREEDY_ROUTE2 is $O\left(d \min\{\mathcal{L},\mathcal{I}\} \log\left(\frac{n\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right)\right)$.

We will use the following three lemmas to prove Theorem 4.2.

Lemma 4.1

$\mu \leq \max_j \mu(P_j^*) + \tilde{\Lambda}$.

Proof

Consider an edge $e \in E$ which satisfies $\mu(e) = \mu$. Let f_y be the last request assigned to a path containing e (so $e \in P_y$). Notice that since $\tilde{\Lambda} \geq \tilde{l}_y$,

$$\mu_y(P_y) = \mu_y(e) = \mu - \tilde{l}_y \geq \mu - \tilde{\Lambda}. \tag{4.1}$$

Also, by the definition of GREEDY_ROUTE2, we know that, for all $P \in \mathcal{P}_y$, including P_y^*,

$$\mu_y(P) \geq \mu_y(P_y). \tag{4.2}$$

Combining (4.1) and (4.2), we see that $\mu_y(P_y^*) \geq \mu - \widetilde{\Lambda}$, which implies that

$$\mu \leq \mu_y(P_y^*) + \widetilde{\Lambda} \leq \max_j \mu(P_j^*) + \widetilde{\Lambda}. \quad \blacksquare$$

Lemma 4.2

For all i and j, $D_j\mathcal{I} \cdot W_{ij} \geq R_{ij}$.

Proof

We divide the proof into three cases. The lemma follows trivially in the first two cases; the main part of the proof is contained in the third case.

Case 1: $R_{ij} = 0$.

In this case, the lemma is clearly true since the lefthand side is always non-negative.

Case 2: $W_{ij} = \widetilde{\Lambda}$.

In this case, the lemma is also clearly true since, by Fact 4.7, $R_{ij} \leq D_j\mathcal{I}\widetilde{\Lambda}$.

Case 3: $R_{ij} > 0$ and $W_{ij} < \widetilde{\Lambda}$.

Since $R_{ij} > 0$, we know, by definition, that $S_{ij} \neq \emptyset$. Let r be an arbitrary member of S_{ij}. Then, by the definition of S_{ij}, for some $l > i$, $W_{lr}^j > 0$. This means that f_j crossed edge e_r^* and therefore

$$e_r^* \in P_j.$$

Now notice that since $W_{ij} < \widetilde{\Lambda}$, the i^{th} layer of e_j^* is the last nonzero layer, and thus $\mu(P_j^*) < i\widetilde{\Lambda}$ by the definition of W_{ij}. So clearly, $\mu_j(P_j^*) < i\widetilde{\Lambda}$ and, since $\mu_j(P_j) \leq \mu_j(P_j^*)$

by the definition of GREEDY_ROUTE2 and $e_r^* \in P_j$, we know that

$$\mu_j(e_r^*) < i\widetilde{\Lambda}. \tag{4.3}$$

Also, notice that, since $W_{lr}^j > 0$ for some $l > i$,

$$\mu_{j+1}(e_r^*) > i\widetilde{\Lambda}. \tag{4.4}$$

In other words, (4.3) and (4.4) tell us that just before f_j was assigned to route P_j, the edge $e_r^* \in P_j$ had congestion less than $i\widetilde{\Lambda}$ (but greater than $(i-1)\widetilde{\Lambda}$ since $\widetilde{l_j} \leq \widetilde{\Lambda}$), and just after f_j was assigned, e_r^* had congestion greater than $i\widetilde{\Lambda}$. Thus, the value $\widetilde{l_j}$ added to the congestion on edge e_r^* is divided between layers i and $i+1$: W_{ir}^j is added to layer i and $W_{(i+1)r}^j$ is added to layer $i+1$. Thus,

$$\begin{aligned}\mu_j(P_j) &\geq \mu_j(e_r^*)\\ &= (i-1)\widetilde{\Lambda} + (\widetilde{\Lambda} - W_{ir}^j).\end{aligned} \tag{4.5}$$

We next need to show that

$$W_{(i-1)j} = \widetilde{\Lambda}. \tag{4.6}$$

Assume, for contradiction, that $W_{(i-1)j} < \widetilde{\Lambda}$. This implies that $\mu(P_j^*) < (i-1)\widetilde{\Lambda}$, and therefore $\mu_j(P_j^*) < (i-1)\widetilde{\Lambda}$. But this means that f_j should not have been assigned to P_j since $\mu_j(P_j) > (i-1)\widetilde{\Lambda}$. Thus, (4.6) is true. Now, by combining (4.6) with the assumption that $W_{ij} < \widetilde{\Lambda}$, we see that

$$\mu(P_j^*) = (i-1)\widetilde{\Lambda} + W_{ij}. \tag{4.7}$$

Using the facts above, we can deduce the following:

$$\begin{aligned}
(i-1)\tilde{\Lambda} + W_{ij} &= \mu(P_j^*) && \text{by (4.7)} \\
&\geq \mu_j(P_j^*) \\
&\geq \mu_j(P_j) && \text{by the definition of GREEDY\_ROUTE2} \\
&\geq (i-1)\tilde{\Lambda} + (\tilde{\Lambda} - W_{ir}^j) && \text{by (4.5).}
\end{aligned}$$

Thus,

$$W_{ij} \geq \tilde{\Lambda} - W_{ir}^j. \tag{4.8}$$

Finally, we can conclude that

$$\begin{aligned}
R_{ij} &= \sum_{r \in S_{ij}} \left(\tilde{l}_j - W_{ir}^j\right) && \text{by the definition of } R_{ij} \\
&\leq \sum_{r \in S_{ij}} \left(\tilde{\Lambda} - \min_{r' \in S_{ij}} W_{ir'}^j\right) \\
&= |S_{ij}| \left(\tilde{\Lambda} - \min_{r' \in S_{ij}} W_{ir'}^j\right) \\
&\leq D_j \mathcal{I} \cdot W_{ij} && \text{by Fact 4.6 and (4.8).}
\end{aligned}$$

■

Lemma 4.3

For any i, $R_i \leq \left(\frac{d\mathcal{I}}{d\mathcal{I}+1}\right)^i R_0$.

Proof

By Lemma 4.2 and the definitions of W_i and R_i, we know that

$$\begin{aligned} W_i &= \sum_{j=1}^{k} W_{ij} \\ &\geq \frac{1}{\mathcal{I}} \sum_{j=1}^{k} \frac{R_{ij}}{D_j} \\ &\geq \frac{1}{d\mathcal{I}} \sum_{j=1}^{k} R_{ij} \\ &= \frac{1}{d\mathcal{I}} R_i. \end{aligned} \tag{4.9}$$

From (4.9) and Fact 4.10, it follows that

$$R_i \leq R_{i-1} - \frac{1}{d\mathcal{I}} R_i.$$

Therefore,

$$R_i \leq \frac{d\mathcal{I}}{d\mathcal{I}+1} R_{i-1}. \tag{4.10}$$

Finally, by recursively applying inequality (4.10), we conclude that

$$R_i \leq \left(\frac{d\mathcal{I}}{d\mathcal{I}+1}\right)^i R_0.$$ ■

Finally, we can prove the main result of this section:

Proof of Theorem 4.2

First, let

$$b = \left\lceil \log_{\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)} \left(\frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right) \right\rceil \geq \log_{\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)} \left(\frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right).$$

Thus,

$$\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)^b \geq \frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}. \tag{4.11}$$

Then we can deduce that

$$\begin{aligned}
R_b &\leq \left(\frac{d\mathcal{I}}{d\mathcal{I}+1}\right)^b R_0 && \text{by Lemma 4.3}\\
&\leq \frac{1}{\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)^b}\,\mathcal{I}\sum_{j=1}^{k}\left(D_j\tilde{l}_j\right) && \text{by Fact 4.8}\\
&\leq \left(\frac{\min\{\mathcal{L},\mathcal{I}\}}{m\mathcal{I}}\right)\mathcal{I}\sum_{j=1}^{k}\left(D_j\tilde{l}_j\right) && \text{by (4.11)}\\
&\leq \mathcal{D}\min\{\mathcal{L},\mathcal{I}\}\frac{\sum_{j=1}^{k}\left(d_j\tilde{l}_j\right)}{m}\\
&\leq \mathcal{D}\min\{\mathcal{L},\mathcal{I}\}\,\mu^* && \text{by Fact 4.2.}
\end{aligned} \tag{4.12}$$

Now notice that, for all j,

$$\begin{aligned}
\mu(P_j^*) &= \sum_i W_{ij} && \text{by Fact 4.4}\\
&= \sum_{i=1}^{b} W_{ij} + \sum_{i>b} W_{ij}\\
&\leq \sum_{i=1}^{b} W_{ij} + \sum_{i>b} W_i\\
&= \sum_{i=1}^{b} W_{ij} + R_b && \text{by Fact 4.9}\\
&\leq b\tilde{\Lambda} + \mathcal{D}\min\{\mathcal{L},\mathcal{I}\}\,\mu^* && \text{by (4.12) and the definition of } W_{ij}.
\end{aligned}$$

Thus,

$$\begin{aligned}\max_j \mu(P_j^*) &\le b\tilde{\Lambda} + \mathcal{D}\min\{\mathcal{L},\mathcal{I}\}\mu^* \\ &= \left\lceil \log_{\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)} \left(\frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right)\right\rceil \tilde{\Lambda} + \mathcal{D}\min\{\mathcal{L},\mathcal{I}\}\mu^*.\end{aligned}$$

By Lemma 4.1, this implies that

$$\mu \le \left\lceil \log_{\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)} \left(\frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right)\right\rceil \tilde{\Lambda} + \tilde{\Lambda} + \mathcal{D}\min\{\mathcal{L},\mathcal{I}\}\mu^*.$$

So the competitive ratio of GREEDY_ROUTE2 is

$$\begin{aligned}\frac{\mu}{\mu^*} &\le \frac{\left\lceil \log_{\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)} \left(\frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right)\right\rceil \tilde{\Lambda} + \tilde{\Lambda}}{\mu^*} + \mathcal{D}\min\{\mathcal{L},\mathcal{I}\} \\ &\le \frac{\left\lceil \log_{\left(\frac{d\mathcal{I}+1}{d\mathcal{I}}\right)} \left(\frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right)\right\rceil \tilde{\Lambda} + \tilde{\Lambda}}{\max\left\{\frac{\mathcal{I}\lambda}{w},\tilde{\Lambda}\right\}} + \mathcal{D}\min\{\mathcal{L},\mathcal{I}\} && \text{by Facts 4.1 and 4.3} \\ &\le \frac{d\mathcal{I}\tilde{\Lambda}\log\left(\frac{m\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right) + 2\tilde{\Lambda}}{\max\left\{\frac{\mathcal{I}\lambda}{w},\tilde{\Lambda}\right\}} + \mathcal{D}\min\{\mathcal{L},\mathcal{I}\} && \text{by Lemma A.1 (in Appendix A)} \\ &= O\left(d\min\{\mathcal{L},\mathcal{I}\}\log\left(\frac{n\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right)\right).\end{aligned}$$

■

Combining Theorem 4.1 and Theorem 4.2, we have the following result.

Theorem 4.3

The competitive ratio of GREEDY_ROUTE2 is $O\left(\min\left\{d\min\{\mathcal{L},\mathcal{I}\}\log\left(\frac{n\mathcal{I}}{\min\{\mathcal{L},\mathcal{I}\}}\right), \sqrt{\mathcal{D}\mathcal{L}m}\right\}\right)$.

We can state the following corollaries giving tighter upper bounds for circumstances in which the optimal congestion is small. The first result follows directly from Theorem 4.2.

Corollary 4.1

On instances in which the optimal routes are pairwise edge disjoint, the competitive ratio of GREEDY_ROUTE2 is $O(d \log n)$.

The second corollary is more general. Recall that a *feasible* request sequence is one for which there exists a route assignment with congestion $\mu^* \leq 1$.

Corollary 4.2

On feasible request sequences, the competitive ratio of GREEDY_ROUTE2 is

$$\begin{cases} O\left(d\mathcal{L}\log\left(\frac{nw}{\Lambda}\right)\right), & \mathcal{L} \leq \mathcal{I} \\ O\left(d\frac{w}{\lambda}\log n\right), & \mathcal{I} \leq \mathcal{L} \end{cases} = O\left(d\frac{w}{\lambda}\log\left(\frac{nw}{\Lambda}\right)\right).$$

Proof

If $\mu^* \leq 1$, then by Facts 4.1 and 4.3, $\Lambda \leq w$ and $\mathcal{I} \leq \frac{w}{\lambda}$. The corollary follows by substituting into the result in Theorem 4.2. ∎

On feasible instances in which the minimum required bandwidth of a request is at least a constant fraction of the link capacity w, the competitive ratio of GREEDY_ROUTE2 is $O(d \log n)$. Specifically, suppose that, for all j, $l_j \geq \frac{w}{\beta}$, for some $\beta = O(1)$. Then, $\frac{w}{\Lambda} \geq \frac{w}{\lambda} \geq \beta$, and the competitive ratio of GREEDY_ROUTE2 is $O(d\beta \log(\beta n)) = O(d \log n)$. We can also claim that the competitive ratio of GREEDY_ROUTE2 is $O(d \log n)$ on feasible instances if $w = O(1)$.

4.2.3 Final Notes

The existence of the factor $\mathcal{I}$ in the previous theorem is not intuitive and we do not think that it belongs in the competitive ratio of GREEDY_ROUTE2. It is easy to show that when the problem instance is such that all of the optimal paths intersect (and $\mathcal{I} = k$), the competitive ratio of GREEDY_ROUTE2 is at most $\mathcal{L}$. To see this, notice that when all of the optimal paths intersect, the optimal congestion is at least $\frac{k\lambda}{w}$. The congestion of any route assignment is at most $\frac{k\Lambda}{w}$. Thus, the competitive ratio of any algorithm is at most $\mathcal{L}$.

Theorem 4.4

If the optimal paths for an instance all intersect at one link, then any algorithm is $\mathcal{L}$ competitive for that instance.

This result gives credence to the idea that as $\mathcal{I}$ increases, the competitive ratio decreases, contrary to the theorem.

Another interesting, but unrelated, phenomenon is evident by looking at instances which force GREEDY_ROUTE2 to assign routes which all overlap. The following theorem states that if GREEDY_ROUTE2 is given an instance that forces it to increase its congestion with every request, then the competitive ratio of GREEDY_ROUTE2 for that instance cannot be worse than d. The instance (whatever it is) that generates the worst case for the competitive ratio is thus not the worst case for GREEDY_ROUTE2 and therefore the competitive ratio of GREEDY_ROUTE2 is less than k.

Theorem 4.5

If the paths chosen by GREEDY_ROUTE2 for an instance all intersect at one link, then GREEDY_ROUTE2 is d competitive for that instance.

Proof

For all j, $1 \leq j \leq k$, let $\tilde{l}_j = \frac{l_j}{w}$. Also, let $z_j = \sum_{i=1}^{j} \tilde{l}_i$. Notice that each request f_j adds $\tilde{l}_j$ to the congestion of each link on its path. Also notice that, in order for the congestion of the instance to be z_k, the assignment of every request f_j must increase the congestion by exactly $\tilde{l}_j$ (assuming the routes are acyclic). In other words, after every request f_j,

$1 \leq j \leq k$, the congestion must equal z_j on some link $e \in P_j$. Therefore, P_j must intersect every previously assigned path P_i, $1 \leq i < j$, at at least link e. Furthermore, in order for GREEDY_ROUTE2 to have chosen P_j, *every* path $P \in \mathcal{P}_j$ must have had congestion z_{j-1} after request f_{j-1}, and therefore every path $P \in \mathcal{P}_j$ must intersect every previously assigned path P_i, $1 \leq i < j$.

Let $I \leq d$ be the number of links that P_2 has in common with P_1. First, suppose $I = d$. Then GREEDY_ROUTE2 must have assigned f_2 to $P_2 = P_1$ because all routes $P \in \mathcal{P}_1 = \mathcal{P}_2$ intersect P_1. Therefore, in any assignment, the routes for f_1 and f_2 must intersect P_1. In addition, as stated above, all routes for every subsequent request must also intersect P_1. Thus, since P_1 contains d links, the optimal congestion must be at least $\lceil \frac{z_k}{d} \rceil$.

Now suppose $I \leq d-1$. Then, from above, for $2 \leq j \leq k$, every path $P \in \mathcal{P}_j$ must intersect P_1 at at least one of these I links. In this case, the optimal congestion is at least $\lceil \frac{z_{k-1}}{I} \rceil \geq \left\lceil \frac{z_{k-1}}{d-1} \right\rceil$.

Therefore, the competitive ratio, with respect to congestion, is at most

$$\frac{z_k}{\min\left\{\lceil \frac{z_k}{d} \rceil, \left\lceil \frac{z_{k-1}}{d-1} \right\rceil\right\}}.$$

If $z_k \leq d$, then the competitive ratio is at most

$$\frac{z_k}{\left\lceil \frac{z_{k-1}}{d-1} \right\rceil} = z_k \leq d.$$

(This follows because $z_k \leq d \Rightarrow z_{k-1} \leq d \Rightarrow \left\lceil \frac{z_{k-1}}{d-1} \right\rceil \geq \lceil \frac{z_{k-1}}{d} \rceil = 1$.) On the other hand, if $z_k > d$, then the competitive ratio is at most

$$\frac{z_k}{\lceil \frac{z_k}{d} \rceil} \leq d.$$

Therefore, in all cases, the competitive ratio is at most d. ■

We point out that if GREEDY_ROUTE2 breaks ties in favor of the route with the fewest maximum congestion links, then the d in Theorem 4.5 becomes $d-1$.

Corollary 4.3

If $k > d$, the competitive ratio of GREEDY_ROUTE2 is strictly less than k.

Proof

Suppose that the competitive ratio is at least k. Then the congestion incurred by GREEDY_ROUTE2 is $\mu \geq k\mu^* \geq k\tilde{\Lambda}$. Since this is the maximum possible congestion, it must be the case that $\mu = k\tilde{\Lambda}$. But then, by Theorem 4.5, the competitive ratio is at most $d < k$, which is a contradiction.

■

4.3 A Lower Bound

In this section, we present a lower bound on the competitive ratio of GREEDY_ROUTE2. Our lower bound improves upon the lower bound of d that can be inferred from the construction in the proof of Theorem 3.2. (See the introduction to this chapter.) For small values of d, the lower bound is close to the upper bound in Theorem 4.2. Our experience indicates that the true competitive ratio of GREEDY_ROUTE2 may be closer to this lower bound than to the upper bound in Theorem 4.2.

Theorem 4.6

The competitive ratio of GREEDY_ROUTE2 is $\Omega(d + \log(n-d))$ for arbitrarily large values of d and n.

To prove the theorem, we will first construct an arbitrarily large network $G_{h,i}$ (for any $h \geq 1$ and $i \geq 1$) and a corresponding request sequence that can cause GREEDY_ROUTE2 to incur congestion equal to $h + i$. We will then show that this quantity is equivalent to the quantity in Theorem 4.6.

4.3.1 The Network

For any integers $h \geq 1$ and $i \geq 1$, $G_{h,i}$ is a directed network with unit capacity links. The vertex set of $G_{h,i}$ is defined to be

$$\begin{aligned} V(G_{h,i}) = {} & \{u_\iota : 1 \leq \iota \leq 2^i - 1\} \cup \{x_{0,\iota} : 1 \leq \iota \leq 2^i\} \\ & \cup \{x_{\iota,2^i} : 1 \leq \iota \leq h-1\} \cup \{v_\iota : 1 \leq \iota \leq h\}. \end{aligned}$$

The directed edge set of $G_{h,i}$ is defined to be

$$\begin{aligned} & E(G_{h,i}) = \\ & \bigcup_{\iota=1}^{i} \left\{ \left(u_{2^{\iota-1}}, x_{0,2^{\iota-1}}\right), \left(u_{3\cdot 2^{\iota-1}}, x_{0,3\cdot 2^{\iota-1}}\right), \left(u_{5\cdot 2^{\iota-1}}, x_{0,5\cdot 2^{\iota-1}}\right), \ldots, \left(u_{2^i-2^{\iota-1}}, x_{0,2^i-2^{\iota-1}}\right) \right\} \\ & \cup \bigcup_{\iota=1}^{i} \left\{ \left(u_{2^{\iota-1}}, x_{0,2\cdot 2^{\iota-1}}\right), \left(u_{3\cdot 2^{\iota-1}}, x_{0,4\cdot 2^{\iota-1}}\right), \left(u_{5\cdot 2^{\iota-1}}, x_{0,6\cdot 2^{\iota-1}}\right), \ldots, \left(u_{2^i-2^{\iota-1}}, x_{0,2^i}\right) \right\} \\ & \cup \left\{ \left(x_{\iota,2^i}, x_{\iota+1,2^i}\right) : 0 \leq \iota \leq h-2 \right\} \cup \left\{ \left(x_{0,\iota}, v_1\right) : 1 \leq \iota \leq 2^i - 1 \right\} \\ & \cup \left\{ \left(x_{h-1,2^i}, v_1\right) \right\} \cup \left\{ \left(x_{\iota,2^i}, v_{\iota+1}\right) : 1 \leq \iota \leq h-1 \right\} \cup \left\{ (v_1, v_\iota) : 2 \leq \iota \leq h \right\}. \end{aligned}$$

As an example, $G_{4,3}$ is shown in Figure 4.3. Notice that $G_{h,i}$ has

$$n = 2^{i+1} + 2h - 2 \tag{4.13}$$

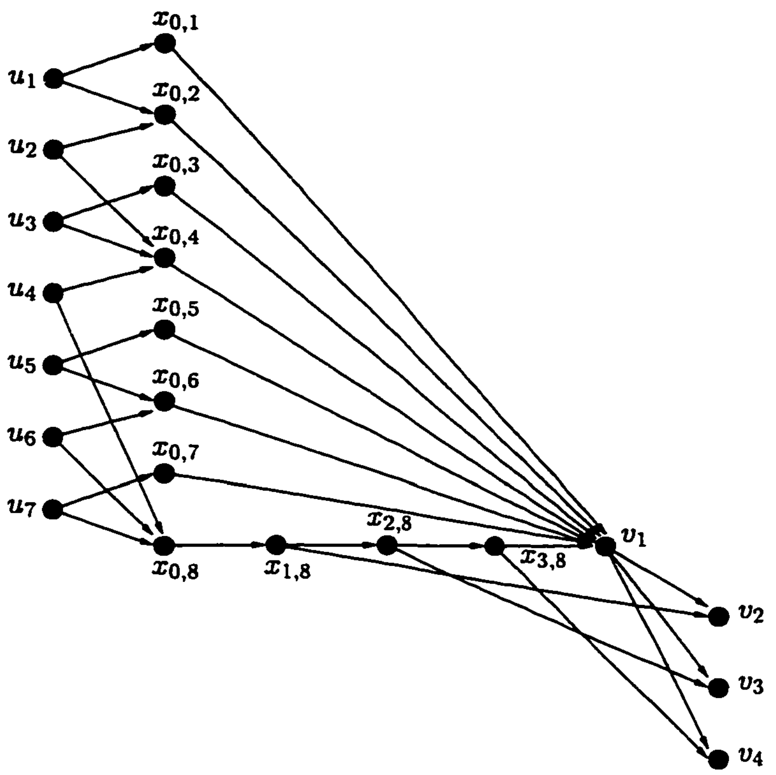


Figure 4.3: The network $G_{4,3}$.

vertices and

$$m = 2^{i+1} + 2^i + 3h - 5$$

edges.

4.3.2 The Request Sequence

We will define the request sequence $\sigma_{h,i}$ for network $G_{h,i}$ to be the concatenation of two subsequences σ_i and σ_h^i. Each request f_j requires unit bandwidth and has arrival time 0, and so we denote each request simply by (s_j, t_j). The subsequence σ_i consists of $2^i - 1$

requests organized into i phases. In phase ι, $1 \le \iota \le i$, we issue the $2^{i-\iota}$ requests

$$(u_{2^{\iota-1}}, v_1), (u_{3 \cdot 2^{\iota-1}}, v_1), (u_{5 \cdot 2^{\iota-1}}, v_1), \ldots, (u_{2^i - 2^{\iota-1}}, v_1).$$

For example, when $i = 3$ (as in Figure 4.3), we have the following 7 requests:

- Phase 1: $(u_1, v_1), (u_3, v_1), (u_5, v_1), (u_7, v_1)$
- Phase 2: $(u_2, v_1), (u_6, v_1)$
- Phase 3: (u_4, v_1)

The request subsequence σ_h^i contains the following h requests:

$$(x_{0,2^i}, v_2), (x_{1,2^i}, v_3), \ldots, (x_{h-2,2^i}, v_h), (x_{h-1,2^i}, v_1).$$

Notice that the longest path that can be assigned to a request in $G_{h,i}$ has length

$$d = h + 1. \tag{4.14}$$

Thus, we are issuing

$$k = |\sigma_{h,i}| = 2^i + h - 1 = 2^i + d - 2$$

requests on the network $G_{h,i}$.

4.3.3 Proof of the Lower Bound

We can now use the preceding construction to prove Theorem 4.6. To this end, we first present three lemmas that describe the behavior of GREEDY_ROUTE2 and the optimal algorithm when they are given this instance. Lemma 4.4 and Lemma 4.5 together imply that GREEDY_ROUTE2 can incur congestion $h + i$. Lemma 4.6 shows that the optimal route assignment has congestion 1.

Lemma 4.4

Suppose we issue the request sequence σ_i on network $G_{h,i}$, for any integers $h \geq 1$ and $i \geq 1$. At the end of phase ι, $1 \leq \iota \leq i$, GREEDY_ROUTE2 can incur congestion ι on each of the links in

$$\left\{(x_{0,2^\iota}, v_1), (x_{0,2\cdot2^\iota}, v_1), (x_{0,3\cdot2^\iota}, v_1), \ldots, (x_{0,2^i-2^\iota}, v_1)\right\} \cup$$
$$\left\{(x_{0,2^\iota}, x_{1,2^\iota}), (x_{1,2^\iota}, x_{2,2^\iota}), \ldots, (x_{h-2,2^\iota}, x_{h-1,2^\iota}), (x_{h-1,2^\iota}, v_1)\right\}$$

and no link in the network can have congestion greater than ι.

Proof

The proof is by induction on ι.

Base case ($\iota = 1$). The requests in phase 1 are

$$(u_1, v_1), (u_3, v_1), (u_5, v_1), \ldots, (u_{2^i-1}, v_1).$$

For each request (u_κ, v_1), $\kappa = 1, 3, \ldots, 2^i - 3$, GREEDY_ROUTE2 will arbitrarily choose either route — $\langle u_\kappa, x_{0,\kappa}, v_1\rangle$ or $\langle u_\kappa, x_{0,\kappa+1}, v_1\rangle$ — since no links on either route have been used before. For request (u_{2^i-1}, v_1), GREEDY_ROUTE2 may also arbitrarily choose either route — $\langle u_{2^i-1}, x_{0,2^i-1}, v_1\rangle$ or $\langle u_{2^i-1}, x_{0,2^i}, x_{1,2^i}, \cdots, x_{h-1,2^i}, v_1\rangle$ — for the same reason. By choosing the latter route for each request, the base case is satisfied.

Induction step ($\iota > 1$). The requests in phase ι are

$$(u_{2^{\iota-1}}, v_1), (u_{3\cdot2^{\iota-1}}, v_1), (u_{5\cdot2^{\iota-1}}, v_1), \ldots, (u_{2^i-2^{\iota-1}}, v_1).$$

Each request $(u_{\kappa\cdot 2^{\iota}-1}, v_1)$, $\kappa = 1, 3, \ldots, 2^{i-\iota+1} - 3$, must be assigned to one of the following routes: $\langle u_{\kappa\cdot 2^{\iota}-1}, x_{0,\kappa\cdot 2^{\iota}-1}, v_1\rangle$ or $\langle u_{\kappa\cdot 2^{\iota}-1}, x_{0,(\kappa+1)\cdot 2^{\iota}-1}, v_1\rangle$. Request $(u_{2^i-2^{\iota}-1}, v_1)$ must be assigned to either $\langle u_{2^i-2^{\iota}-1}, x_{0,2^i-2^{\iota}-1}, v_1\rangle$ or $\langle u_{2^i-2^{\iota}-1}, x_{0,2^i}, x_{1,2^i}, \cdots, x_{h-1,2^i}, v_1\rangle$. By the induction hypothesis, each of the links in the set

$$\bigcup_{\kappa=1,3,\ldots,2^{i-\iota+1}-3} \left\{\left(x_{0,\kappa\cdot 2^{\iota}-1}, v_1\right), \left(x_{0,(\kappa+1)\cdot 2^{\iota}-1}, v_1\right)\right\} \cup \left\{\left(x_{0,2^i-2^{\iota}-1}, v_1\right)\right\} \cup$$
$$\left\{\left(x_{0,2^i}, x_{1,2^i}\right), \left(x_{1,2^i}, x_{2,2^i}\right), \ldots, \left(x_{h-2,2^i}, x_{h-1,2^i}\right), \left(x_{h-1,2^i}, v_1\right)\right\}$$

can have congestion $\iota - 1$ prior to the arrival of any requests in phase ι. Let us suppose that this has happened. Then, when each request in phase ι arrives, both of the routes that can be assigned to this request have congestion $\iota - 1$. Thus, GREEDY_ROUTE2 may arbitrarily assign each request to either route. If GREEDY_ROUTE2 assigns each request to its latter route (as presented above), then every link in the set

$$\left\{(x_{0,2^{\iota}}, v_1), (x_{0,2\cdot 2^{\iota}}, v_1), (x_{0,3\cdot 2^{\iota}}, v_1), \ldots, \left(x_{0,2^i-2^{\iota}}, v_1\right)\right\} \cup$$
$$\left\{\left(x_{0,2^i}, x_{1,2^i}\right), \left(x_{1,2^i}, x_{2,2^i}\right), \ldots, \left(x_{h-2,2^i}, x_{h-1,2^i}\right), \left(x_{h-1,2^i}, v_1\right)\right\}$$

will have congestion ι. Furthermore, since, by the induction hypothesis, no link in the network had congestion greater than $\iota - 1$ prior to phase ι, no link can have congestion greater than ι after phase ι. ■

Lemma 4.5

For any integer $h \geq 1$ and $i \geq \iota$, suppose every link of network $G_{h,i}$ in the set

$$\left\{\left(x_{0,2^{\iota}}, x_{1,2^{\iota}}\right), \left(x_{1,2^{\iota}}, x_{2,2^{\iota}}\right), \ldots, \left(x_{h-2,2^{\iota}}, x_{h-1,2^{\iota}}\right), \left(x_{h-1,2^{\iota}}, v_1\right)\right\}$$

has congestion c and every other link in the network has congestion at most c. Then, if we issue the requests in request sequence σ_h^i, GREEDY_ROUTE2 can assign routes such that, after the j^{th} request in σ_h^i, every link in the set

$$\{(x_{j-1,2^i}, x_{j,2^i}), (x_{j,2^i}, x_{j+1,2^i}), \cdots, (x_{h-2,2^i}, x_{h-1,2^i}), (x_{h-1,2^i}, v_1)\}$$

will have congestion $c + j$ and no link in the network will have congestion greater than $c + j$.

Proof

The proof is by induction on j.

Base case ($j = 1$). The first request in σ_h^i is $(x_{0,2^i}, v_2)$. GREEDY_ROUTE2 must assign to this request either the route $\langle x_{0,2^i}, x_{1,2^i}, v_2\rangle$ or the route $\langle x_{0,2^i}, x_{1,2^i}, \cdots, x_{h-1,2^i}, v_2\rangle$. Since both routes contain the link $(x_{0,2^i}, x_{1,2^i})$, both routes have congestion c. Thus, GREEDY_ROUTE2 may arbitrarily assign either route to the request. If GREEDY_ROUTE2 assigns the latter route, then every link in the set

$$\{(x_{0,2^i}, x_{1,2^i}), (x_{1,2^i}, x_{2,2^i}), \cdots, (x_{h-2,2^i}, x_{h-1,2^i}), (x_{h-1,2^i}, v_1)\}$$

will have congestion $c + 1$. Since every other link in the network had congestion at most c before the request arrived, every link must have congestion at most $c + 1$ after the request has been assigned the route.

Induction step ($j > 1$). We partition the induction step into two cases.

Case 1 ($j < h$): If $j < h$, then the j^{th} request in σ_h^i is $(x_{j-1,2^i}, v_{j+1})$. GREEDY_ROUTE2 must assign one of two routes — $\langle x_{j-1,2^i}, x_{j,2^i}, v_{j+1}\rangle$ or $\langle x_{j-1,2^i}, x_{j,2^i}, \cdots, x_{h-1,2^i}, v_{j+1}\rangle$ —

to this request. By the induction hypothesis, every link in the set

$$\{(x_{j-2,2^i}, x_{j-1,2^i}), (x_{j-1,2^i}, x_{j,2^i}), \dots, (x_{h-2,2^i}, x_{h-1,2^i}), (x_{h-1,2^i}, v_1)\}$$

can have congestion $c+j-1$ when this request arrives. Assume that this is the case. Then, since both routes contain the link $(x_{j-1,2^i}, x_{j,2^i})$ and no link can have congestion greater than $c+j-1$, both routes have congestion $c+j-1$. Thus, GREEDY_ROUTE2 may arbitrarily assign either route to the request. If GREEDY_ROUTE2 assigns the latter route, then every link in the set

$$\{(x_{j-1,2^i}, x_{j,2^i}), (x_{j,2^i}, x_{j+1,2^i}), \dots, (x_{h-2,2^i}, x_{h-1,2^i}), (x_{h-1,2^i}, v_1)\}$$

will have congestion $c+j$.

Case 2 ($j = h$): If $j = h$, then the j^{th} request in σ_h^i is $(x_{h-1,2^i}, v_1)$. GREEDY_ROUTE2 must assign the route $\langle x_{h-1,2^i}, v_1\rangle$ to this request. Since, by the induction hypothesis, the link $(x_{h-1,2^i}, v_1)$ has congestion $c+h-1$ when the request arrives, it will have congestion $c+h$ after the assignment. ■

Lemma 4.6

For all integers $h > 1$ and $i \geq 1$, the set of optimal routes for request sequence $\sigma_{h,i}$ on network $G_{h,i}$ is pairwise edge disjoint.

Proof

To prove the theorem, we simply present the pairwise edge disjoint route assignment. First, to each request $(u_\iota, v_1) \in \sigma_i$, $\iota = 1, 2, \dots, 2^i - 1$, we assign the route $\langle u_\iota, x_{0,\iota}, v_1\rangle$. These routes are clearly disjoint since the values of ι are all different. Second, for each request

$(x_{\iota,2^i}, v_{\iota+2}) \in \sigma_h^i$, $\iota = 0, 1, \ldots, h-2$, we assign the route $\langle x_{\iota,2^i}, x_{\iota+1,2^i}, v_{\iota+2}\rangle$. Again, these routes are clearly disjoint because the values of ι are all different. They are also clearly disjoint from the set of routes assigned to the requests in σ_i. Lastly, for request $(x_{h-1,2^i}, v_1) \in \sigma_h^i$, we assign the route $\langle x_{h-1,2^i}, v_1\rangle$. ■

Finally, we can use the preceding three lemmas to prove Theorem 4.6.

Proof of Theorem 4.6

Recall that the request sequence $\sigma_{h,i}$ consists of the concatenation of the subsequences σ_i and σ_h^i. By Lemma 4.4, GREEDY_ROUTE2 can incur congestion i after the i^{th} (and last) phase of subsequence σ_i. Specifically, GREEDY_ROUTE2 can incur congestion i on every link in the set

$$\left\{ \left(x_{0,2^i}, x_{1,2^i}\right), \left(x_{1,2^i}, x_{2,2^i}\right), \ldots, \left(x_{h-2,2^i}, x_{h-1,2^i}\right), \left(x_{h-1,2^i}, v_1\right) \right\}.$$

By Lemma 4.5 then, after GREEDY_ROUTE2 has assigned routes to all the requests in $\sigma_{h,i}$, the link $(x_{h-1,2^i}, v_1)$ has congestion $i+h$.

Now notice that, by (4.13) and (4.14),

$$i = \log(n - 2d + 4) - 1.$$

Thus,

$$i + h = \log(n - 2d + 4) + d - 2.$$

On the other hand, by Lemma 4.6, an optimal algorithm can always find a set of edge disjoint routes with congestion 1. Therefore the competitive ratio of GREEDY_ROUTE2 is $\Omega(d + \log(n-d))$. ■

4.4 A Lower Bound for Layered Networks

In this section, we consider the case where all the paths for all the requests have length d, and prove a lower bound on the competitive ratio of GREEDY_ROUTE2 that is almost as high as that in Theorem 4.6. Similar to the result in Section 3.4, this lower bound also implies a lower bound on the variation of GREEDY_ROUTE2 which breaks ties in favor of the shortest path.

Theorem 4.7

The competitive ratio of GREEDY_ROUTE2 is $\Omega\left(d + \log\left(\frac{n}{d} - d\right)\right)$ on layered networks for arbitrarily large values of d and n.

To prove the theorem, we will first construct an arbitrarily large network $G_{h,i}$ (for any $h \geq 1$ and $i \geq 1$) and a corresponding request sequence that can cause GREEDY_ROUTE2 to incur congestion equal to $h + i$. We will then show that this quantity is equivalent to the quantity in Theorem 4.7. The proof of this result is very similar to the proof of Theorem 4.6.

4.4.1 The Network

For any integers $h \geq 1$ and $i \geq 1$, $G_{h,i}$ is a directed network with unit capacity links. The vertex set of $G_{h,i}$ is defined to be

$$V(G_{h,i}) = \left\{u_\iota : 1 \leq \iota \leq 2^i + h - 1\right\} \cup \{v_\iota : 1 \leq \iota \leq h\}$$
$$\cup \left\{x_{\kappa,\iota} : 0 \leq \kappa \leq h, 1 \leq \iota \leq 2^i + h - 1\right\} \setminus \left\{x_{\kappa,\iota} : 1 \leq \kappa \leq h-1, \iota = \kappa + 2^i\right\}.$$

The directed edge set of $G_{h,i}$ is defined to be

$$\begin{aligned}
E(G_{h,i}) = & \bigcup_{\iota=1}^{i} \left\{ \left(u_{2^{\iota-1}}, x_{0,2^{\iota-1}}\right), \left(u_{3\cdot 2^{\iota-1}}, x_{0,3\cdot 2^{\iota-1}}\right), \left(u_{5\cdot 2^{\iota-1}}, x_{0,5\cdot 2^{\iota-1}}\right), \dots, \left(u_{2^i-2^{\iota-1}}, x_{0,2^i-2^{\iota-1}}\right) \right\} \\
\cup & \bigcup_{\iota=1}^{i} \left\{ \left(u_{2^{\iota-1}}, x_{0,2\cdot 2^{\iota-1}}\right), \left(u_{3\cdot 2^{\iota-1}}, x_{0,4\cdot 2^{\iota-1}}\right), \left(u_{5\cdot 2^{\iota-1}}, x_{0,6\cdot 2^{\iota-1}}\right), \dots, \left(u_{2^i-2^{\iota-1}}, x_{0,2^i}\right) \right\} \\
\cup & \left\{ (x_{\kappa,\iota}, x_{\kappa+1,\iota}) \colon 0 \le \kappa \le h-1, 1 \le \iota \le 2^i \right\} \cup \left\{ (x_{h,\iota}, v_1) \colon 1 \le \iota \le 2^i \right\} \\
\cup & \left\{ \left(u_{2^i}, x_{0,2^i}\right) \right\} \cup \left\{ \left(x_{h,2^i}, v_\iota\right) \colon 2 \le \iota \le h \right\} \\
\cup & \left\{ (u_\iota, x_{0,\iota}), (x_{0,\iota}, x_{1,\iota}), \dots, (x_{\iota-N-2,\iota}, x_{\iota-N-1,\iota}), \left(x_{\iota-N-1,\iota}, x_{\iota-N,2^i}\right), \right. \\
& \left(x_{\iota-N,2^i}, x_{\iota-N+1,\iota}\right), (x_{\iota-N+1,\iota}, x_{\iota-N+2,\iota}), \dots, (x_{h-1,\iota}, x_{h,\iota}), \\
& \left. (x_{h,\iota}, v_{\iota-N+1}) \colon 2^i + 1 \le \iota \le 2^i + h - 1, N = 2^i \right\}.
\end{aligned}$$

As an example, $G_{4,3}$ is shown in Figure 4.4. Notice that $G_{h,i}$ has

$$n = (h+2)(2^i + h - 1) + 1 \tag{4.15}$$

vertices and

$$m = (h+3)2^i + h^2 + 2h - 4$$

edges.

4.4.2 The Request Sequence

As in the previous section, we will define the request sequence $\sigma_{h,i}$ for network $G_{h,i}$ to be the concatenation of two subsequences σ_i and σ_h^i. Each request f_j requires unit bandwidth and has arrival time 0, and so we denote each request simply by (s_j, t_j). The subsequence

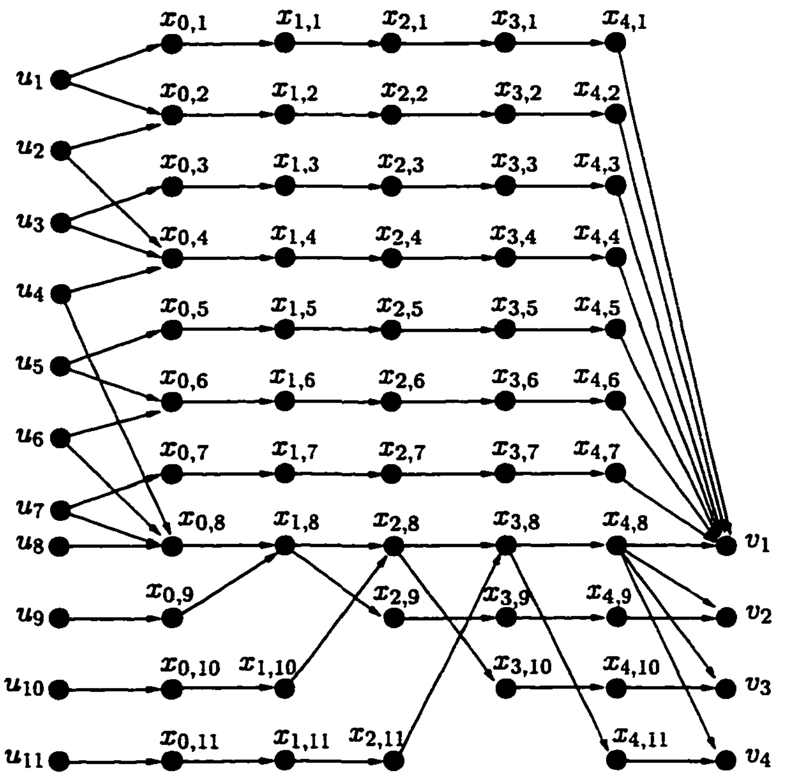


Figure 4.4: The layered network $G_{4,3}$.

σ_i is identical to that in the previous section and consists of $2^i - 1$ requests organized into i phases. In phase ι, $1 \le \iota \le i$, we issue the $2^{i-\iota}$ requests

$$(u_{2^{\iota-1}}, v_1), (u_{3\cdot 2^{\iota-1}}, v_1), (u_{5\cdot 2^{\iota-1}}, v_1), \ldots, (u_{2^i - 2^{\iota-1}}, v_1).$$

The request subsequence σ_h^i contains the following h requests:

$$(u_{2^i}, v_2), (u_{2^i+1}, v_3), \ldots, (u_{2^i+h-2}, v_h), (x_{2^i+h-1}, v_1).$$

The subsequence σ_h^i differs from that in the previous section in that each source node $x_{\iota,2^i}$ is replaced by $u_{2^i+\iota}$.

Notice that the longest path that can be assigned to a request in $G_{h,i}$ has length

$$d = h + 2. \tag{4.16}$$

Thus, we are issuing

$$k = |\sigma_{h,i}| = 2^i + h - 1 = 2^i + d - 3$$

requests on the network $G_{h,i}$.

4.4.3 Proof of the Lower Bound

Lemmas 4.4, 4.5, and 4.6 from the previous section apply to the situation here with few modifications. The modifications arise from the introduction of layers into the network and the corresponding changes in the request sequence.

Proof of Theorem 4.7

As in the proof of Theorem 4.6, we can infer that GREEDY_ROUTE2 incurs congestion $i + h$. Notice that, by (4.15) and (4.16),

$$i = \log\left(\frac{n-1}{d} - d + 3\right).$$

Thus,

$$i + h = \log\left(\frac{n-1}{d} - d + 3\right) + d - 2.$$

On the other hand, an optimal algorithm can always find a set of edge disjoint routes with congestion 1. Therefore the competitive ratio of GREEDY_ROUTE2 is $\Omega\left(d + \log\left(\frac{n}{d} - d\right)\right)$. ■

Chapter 5

Online Packet Routing and Scheduling

5.1 Introduction

Up to this point we have considered online greedy algorithms for routing virtual circuit requests. In this chapter, we consider online greedy store-and-forward scheduling algorithms for moving packets in a packet-switched network. As discussed in Chapter 1, we are interested in a model in which each packet is assigned a route and a schedule at the source. This model has the advantage of simplifying switches on the network, and also has theoretical interest stemming from its proposed use in past literature (e.g., [72, 82]).

Simplifying the notation from Section 1.1, we will denote a sequence of packets as $\sigma = p_1, p_2, \ldots, p_k$, where each $p_j = (s_j, t_j, a_j)$. The values s_j, t_j, and a_j are as defined earlier: s_j is the packet's source node, t_j is the packet's destination node, and a_j is the packet's arrival time. If P_j is the route assigned to packet p_j, then let $P_j(i)$ denote the

i^{th} link in the route. A schedule for packet p_j is a function $S_j : \{1, 2, \ldots, |P_j|\} \rightarrow \{a_j + 1, a_j + 2, \ldots, \}$ where $S_j(i)$ is the time step during which packet p_j will cross link $P_j(i)$. If $S_j(i) \neq S_j(i-1) + 1$ then p_j waits in the queue, or is *delayed*, at the head of link $P_j(i)$ during time steps $S_j(i-1)+1, S_j(i-1)+2, \ldots, S_j(i)-1$. The schedule S_j must be *feasible*, that is, it must obey the capacity constraints of both the links and the queues during each time step.

Like most related models in the literature, we will assume that clocks are synchronous and time is measured in discrete *time steps*. Each time step $t \geq 1$ refers to the continuous time interval $(t-1, t]$. During one time step, up to w packets may cross each network link. If a packet arrives during time step a_j, the earliest it will cross the first link on its path is during time step $a_j + 1$. We consider the dynamic problem in which arrival times are arbitrary and any number of packets may originate or be delivered to any node. This is in contrast to a static permutation routing problem in which $a_j = 0$, for all j, and an equal number of packets originate at every node. Whereas a permutation routing problem is a good approximation to a system in which nodes produce equal numbers of packets at the same rate, a dynamic problem resembles systems with less predictable and more general behavior. Let C_j denote the completion time of packet p_j, which is the time when p_j reaches its destination. The goal of an algorithm is to minimize the makespan of its schedule, defined to be $\max_j C_j$.

In this chapter, we will assume that packets are not delayed due to full queues. Equivalently, we assume that $q(v) \geq \mu w \cdot \text{indegree}(v)$ for all $v \in V$, where μ is the congestion of the online route assignment. We note that this assumption is quite common in the literature (e.g., [68, 31, 72, 82, 83, 88, 64]), in which bounds are proven on the expected size of queues *a posteriori.*

5.2 Bounds on Optimal Makespan

We can state both lower and upper bounds on the makespan of any optimal schedule. In general, let $\delta = \max_j \left\{a_j + \left|P_j^*\right|\right\}$, where P_j^* is the route assigned to packet p_j by an optimal algorithm, and let $\Delta = \max\left\{\lceil\mu^*\rceil, \delta\right\}$, where μ^* is the congestion of the set of routes assigned by an optimal algorithm. The following states a trivial lower bound on the makespan for any schedule.

Fact 5.1

The makespan of any optimal schedule is at least Δ.

It turns out that, for any static packet scheduling instance, the optimal makespan is always within a constant factor of this lower bound. The following theorem is due to Leighton, Maggs, and Rao [59, 57].

Theorem 5.1 ([59, 57])

Let $\mathcal{P}$ be a set of routes with maximum length d and congestion μ for a static packet routing instance on an arbitrary network. Then there always exists a schedule for this instance with makespan $O(\mu + d) = O(\Delta)$ that uses only constant length queues.

A schedule whose makespan matches this bound can be found by a centralized offline algorithm [58].

5.3 Greedy Fixed Priority Scheduling

Define GFP to be the online scheduling algorithm that schedules each request immediately in the order it appears in the sequence and does not delay a packet at any link e during time step t unless there are w previously scheduled packets already assigned to e during t. Recall that this is the simple scheduling heuristic proposed by Mao and Simha [72] and Rivera-Vega, et al. [82].

GFP creates a *greedy, fixed priority* schedule. The schedule is *greedy* in the sense that packet p_j is delayed at a link e during time step t only if w other packets are already traversing e during t. It is a *fixed priority* schedule because it is as if packet p_j has priority j and is only delayed by packets with higher priority $i < j$.[1]

This scheduling technique has advantages over other techniques, primarily from an engineering point of view. First, each packet need only carry its priority to encode its schedule. On the other hand, other global scheduling algorithms (e.g., [57, 91, 8]) need to have packets carry more explicit information about their entire schedule. Second, very little work is required locally at the network switches to forward packets. During each time step, a switch need only examine the packets waiting for each adjacent link e and send the w packets with the highest priorities. (The algorithm need not decide which packets to send on which links since the route for each packet has been assigned in advance.) Other local scheduling algorithms, such as the one which forwards packets with the farthest distance yet to travel, require slightly more work and bookkeeping.

It has been shown in the literature [82, 30, 31, 67, 68] that greedy schedules in general

[1]We use *greedy* in this way to reflect the terminology used in the literature. Notice that a global synchronous time base is necessary for the priority scheme to work properly.

exhibit quite good worst case guarantees on certain sets of routes. Mansour and Patt-Shamir [67, 68] proved the following bound on the makespan of *any* greedy schedule on shortest paths. The result holds even for dynamic instances.

Theorem 5.2 ([68])

In any greedy schedule that follows shortest paths, each packet p_j arrives at its destination within $d_j + \lfloor \frac{k-1}{w} \rfloor$ time steps.

Previously, Rivera-Vega, et al. [82] had shown that GFP specifically achieves the above bound on shortest paths when $w = 1$.

Note that this result is not useful for analyzing the competitive ratio of an arbitrary greedy scheduling algorithm. For example, consider a graph consisting of 2 nodes and k unit capacity parallel links. An algorithm assigning all packets to the same link would be using shortest paths, and would therefore deliver all packets within k time steps. However, an optimal algorithm would assign each packet to its own link, giving a competitive ratio of k, the worst possible.

Without using this result, we can however prove a trivial upper bound on the competitive ratio of any online greedy scheduling algorithm that always uses the same route as the optimal algorithm. (This would be the case on a tree, for example.) Let $d = \max_{1 \le j \le k} |P_j|$ denote the length of the longest path assigned to a packet.

Theorem 5.3

The competitive ratio of any greedy online scheduling algorithm which always chooses the same route as the optimal algorithm is at most $\min\{\mu, d + 1\}$.

Proof

For a particular problem instance, let p_j denote the packet that finishes last in the online schedule. Notice that

$$C_j \leq a_j + d_j + (\mu - 1)d_j \leq \delta + (\mu - 1)d.$$

According to Fact 5.1, the optimal makespan is at least Δ. Therefore, the competitive ratio of any algorithm is at most

$$\frac{\delta + (\mu - 1)d}{\Delta} \leq \frac{\Delta + (\mu - 1)\Delta}{\Delta} = \mu. \tag{5.1}$$

Alternatively, we see that

$$\frac{\delta + (\mu - 1)d}{\Delta} \leq \frac{\Delta + (\Delta - 1)d}{\Delta} \leq d + 1. \tag{5.2}$$

Combining (5.1) and (5.2), we can conclude that the competitive ratio of any greedy online scheduling algorithm is at most $\min\{\mu, d + 1\}$. ■

5.4 Linear Array Networks

In this section, we analyze the competitive ratio of GFP on full-duplex linear array networks. A network is called full-duplex if packets can move in both directions concurrently between each pair of adjacent nodes. On the other hand, a network is called simplex if packets can move in only one direction between each pair of adjacent nodes. Formally, a full-duplex linear array network consists of n nodes $\{v_0, v_1, \ldots, v_{n-1}\}$ and $2(n - 1)$ directed links $\{(v_i, v_{i+1}), (v_{i+1}, v_i) : i = 0, 1, \ldots, n - 2\}$, each with capacity w. We will naturally assume that all packets follow acyclic routes.

Recall that we are interested in situations in which an arbitrary number of packets may be requested between any source/destination pair, and packets can arrive at arbitrary times. Some simpler cases are trivial. For example, if there is only one source and all arrival times are different, there is never any congestion. This is also true in a permutation routing problem.

Our results actually apply more generally to any network in which each node has indegree one. This includes simplex linear arrays, simplex ring networks in which all links are directed either clockwise or counter-clockwise, and arborescences directed from the root toward the leaves. The latter network might model communication from a server process at the root to client processes in a client/server system. In the context of more general networks, the results in this section apply to situations in which the network can be partitioned into lines of nodes, and each packet's route resides entirely on one line. Moving packets on linear arrays is also a fundamental problem in the design of many algorithms for multidimensional arrays. For example, in a two dimensional array, or mesh, packets are often routed along their row and then their column, or vice versa. Each half of the route is equivalent to a linear array.

We will prove the following theorem, showing a tight competitive ratio for GFP on linear arrays (and the other types of networks discussed above).

Theorem 5.4

The competitive ratio of GFP is $2 - \frac{1}{\Delta}$ on linear array networks.

5.4.1 The Lower Bound

We first bound the competitive ratio of GFP from below. We construct an instance on a simplex linear array since this network is contained in any network in indegree one.

Lemma 5.1

The competitive ratio of GFP is at least $2 - \frac{1}{\Delta}$ on a simplex linear array network.

Proof

Assume $w = 1$. Let $k = n - 1$ and each $p_j = (v_0, v_j, 0)$, $1 \leq j \leq k$. Note that $\delta = n - 1$ and $\mu = \mu^* = n - 1$ since all packets must cross link (v_0, v_1). GFP schedules the packets in this instance in order, so that the last packet arrives at vertex v_{n-1} at time $(n-2) + (n-1) = 2n - 3$. On the other hand, an optimal algorithm will schedule the packets in reverse order, achieving a makespan of $n - 1$.[2] Thus, the competitive ratio of GFP is at least

$$\frac{2n-3}{n-1} = 2 - \frac{1}{n-1} = 2 - \frac{1}{\Delta}. \qquad \blacksquare$$

5.4.2 The Upper Bound

In this section, we show that the lower bound in Lemma 5.1 is tight. Recall that we can infer from Theorem 5.2 above that the makespan of a schedule constructed by GFP is at most $a_j + d_j + \frac{k-1}{w}$, if p_j is the last packet to reach its destination. Much earlier, Valiant and Brebner [94] noted that the specific greedy algorithm that gives priority to packets that

[2]We note that this result holds in more general situations as well: in any instance, if there is at most one packet destined for each node and the packet that has the farthest distance yet to travel is given priority at each step, then every packet will reach its destination in at most $n - 1$ steps ([54], page 161).

have moved at least one step toward their destinations delays each packet at most $k-1$ times on a linear array. Kaufmann and Sibeyn [51] showed that if $w = 1$ and all packets arrive at the same time, then *any* priority scheme will result in a packet being delayed at most $\mu - 1$ times on a linear array. We note however that in a dynamic problem, an arbitrary priority scheme could result in a packet being delayed up to $(n-1)(\mu-1)$ times. This will happen if, as a packet enters each node on its path, $\mu - 1$ new packets with higher priorities arrive at that node. By assigning priorities that correspond to arrival times, we can avoid this situation; we will show that packets are delayed far fewer times with GFP. Kaufmann and Sibeyn also proved that the scheduling algorithm that always forwards the packet that has the farthest yet to travel is optimal for the static problem in that it always achieves a makespan that is at most the worst case optimal.

The following lemma bounds from above the maximum completion time of a packet in a schedule constructed by GFP. From this lemma we will infer an upper bound on the competitive ratio of GFP, which will, together with Lemma 5.1, imply Theorem 5.4.

To simplify notation, we define

$$c_j = \lceil \mu_{j+1}(P_j) \rceil = \left\lceil \frac{\max_{e \in P_j} |\{i : 1 \le i \le j, e \in P_i\}|}{w} \right\rceil .$$

Lemma 5.2

For any packet p_j, $C_j \le a_j + d_j + c_j - 1$.

Proof

Denote the completion time of p_j as $C_j = a_j + d_j + x$, $x \ge 0$. For contradiction, suppose that $x > c_j - 1$. Consider link $P_j(l)$, where p_j is delayed last, and notice that

$$S_j(l) = a_j + x + l. \tag{5.3}$$

Also, notice that $l > 1$. Otherwise, by the definition of GFP, w earlier packets must cross $P_j(1)$ during each time step $a_j + 1, \ldots, a_j + x$, which implies that

$$c_j \geq \left\lceil \frac{((a_j + x) - (a_j + 1) + 1)w + 1}{w} \right\rceil = x + 1,$$

a contradiction.

Let $X(\tau) = \{p_i : i < j \text{ and } p_i \text{ crosses link } P_j(l) \text{ during time step } \tau\}$. Also, to simplify notation, let $t = S_j(l-1) + 1$. Then $X(t), X(t+1), \ldots, X(S_j(l) - 1)$ are the sets of packets (each with cardinality w) that delay p_j on link $P_j(l)$. Additionally, there may be sets of packets $X(r), X(r+1), \ldots, X(t-1)$ with cardinality w for consecutive time steps $r, r+1, \ldots, t-1$. If there are no such additional sets, then let $r = t$. Formally, we define $r = \min\{\tau : 2 \leq \tau \leq t, |X(\tau)| = w \text{ for all } \tau, \tau + 1, \ldots, t\}$.

We first prove the following lemma under the assumption that $x > c_j - 1$.

Lemma 5.3

For all packets $p \in X(\tau)$, $r \leq \tau \leq S_j(l) - 1$, p must have crossed link $P_j(l-1)$ previously.

Proof

Suppose that there is a packet $p_i \in X(\tau)$, for some $r \leq \tau \leq S_j(l) - 1$, that did not cross link $P_j(l-1)$. Then, $P_i(1) = P_j(l)$. Therefore, by the definition of GFP, w earlier packets must cross $P_j(l)$ during each time step $a_i + 1, \ldots, \tau - 1$. Since w packets also cross $P_j(l)$

during each time step $\tau, \ldots, S_j(l) - 1$ and p_j crosses $P_j(l)$ during time step $S_j(l)$,

$$\begin{aligned} c_j &\geq \left\lceil \frac{((S_j(l) - 1) - (a_i + 1) + 1)w + 1}{w} \right\rceil \\ &= (S_j(l) - 1) - (a_i + 1) + 2 \\ &= (a_j + x + l) - a_i && \text{by (5.3)} \\ &> x + 1 && \text{since } a_j \geq a_i \text{ and } l > 1. \end{aligned}$$

But this is a contradiction. ■

We now continue with the proof of Lemma 5.2. Let $X = \bigcup_{r \leq \tau \leq t} X(\tau) \cup \{p_j\}$. By Lemma 5.3, the $(t - r + 1)w + 1$ packets in X crossed link $P_j(l - 1)$ before time step t. Thus, at least one packet $p \in X$ must have crossed link $P_j(l - 1)$ before time step $r - 1$. Since p crossed link $P_j(l)$ during a time step greater than $r - 1$, p must have been delayed by w packets crossing link $P_j(l)$ during time step $r - 1$. This set of packets must be the set $X(r - 1)$. But, by definition, $X(r - 1)$ contains strictly less than w packets, a contradiction. Therefore, $x \leq c_j - 1$, which implies that $C_j \leq a_j + d_j + c_j - 1$. This concludes the proof of Lemma 5.2. ■

Lemma 5.4

The competitive ratio of GFP is at most $2 - \frac{1}{\Delta}$.

Proof

Let p_j denote the last packet to complete in the schedule constructed by GFP. By Lemma 5.2, we know that $C_j \leq a_j + d_j + c_j - 1$. On the other hand, by Fact 5.1, the optimal makespan is at least $\Delta = \max\{\lceil \mu \rceil, \delta\}$, where $\delta = \max_j\{a_j + d_j\}$. Therefore, the competitive ratio of

GFP is at most

$$\frac{a_j + d_j + c_j - 1}{\max\{\lceil\mu\rceil, \delta\}} \leq \frac{\delta + \lceil\mu\rceil - 1}{\max\{\lceil\mu\rceil, \delta\}} \leq 2 - \frac{1}{\Delta}. \qquad \blacksquare$$

5.5 Tree Networks

In this section, we analyze the performance of GFP on trees. A tree is an undirected graph with a unique simple path between any pair of nodes. In a full-duplex tree network, each undirected edge is replaced by two directed links, one in each direction. In a simplex tree network, each undirected edge is replaced by one directed link. Any result for full-duplex tree networks can be applied to general networks by having packets follow paths that lie on a spanning tree.

Recall from the discussion at the beginning of Section 5.4 that Theorem 5.4 applies to trees directed away from a designated root. In the following subsection, we will consider trees directed toward the root. Then we will discuss how to combine these results to schedule packets on arbitrary full-duplex trees.

5.5.1 Trees Directed Toward the Root

We can infer from Theorem 5.3 that the competitive ratio of GFP on a balanced tree is $O(\log n)$ since the length of the longest path is $O(\log n)$. In this section, we show that this result holds tight for GFP on any tree directed toward the root. We prove the following theorem.

Theorem 5.5

The competitive ratio of GFP is $\Theta(\log n)$ on trees directed toward the root.

The Upper Bound

Consider a tree with links directed toward the root. For any link e in such a network, let

$$\Pi(e) = \{e\} \cup \{e' \colon \text{there is a path between the tail of } e' \text{ and the head of } e\}.$$

We will use the following lemma to prove an upper bound on the competitive ratio of GFP for this case.

Lemma 5.5

Suppose $\mu > 1$. If $S_j(h) \geq a_j + 2l\Delta$, for any integer $l \geq 0$, then $|\Pi(P_j(h))| \geq 2^l - 1$.

Proof

The proof is by induction on l.

Base case ($l = 0$). When $l = 0$ we need only show that $|\Pi(P_j(h))| \geq 0$, which is trivially true.

Induction step ($l > 0$). Let $g = \max\{i \colon p_j \text{ was delayed at link } P_j(i),\ 1 \leq i \leq h\}$. Note that there exists such a g since p_j was delayed for at least $2l\Delta - h \geq \Delta > 1$ time steps before crossing link $P_j(h)$. (We note that this also implies that $g \geq 2$, since p_j could have been delayed at most $\mu - 1 \leq \Delta - 1$ times on one link.) Since, by definition, p_j is not delayed between the time it crosses link $P_j(g)$ and the time it crosses link $P_j(h)$, we know that

$$\begin{aligned} S_j(g) &= S_j(h) - (h-g) \\ &\geq S_j(h) - (d_j - 2) \\ &> a_j + 2l\Delta - \Delta \\ &= a_j + (2l-1)\Delta. \end{aligned} \tag{5.4}$$

Now consider the time step during which p_j crosses link $P_j(g-1)$. Using (5.4) and the fact that p_j could have been delayed at link $P_j(g)$ for at most $\mu - 1$ time steps, we see that

$$\begin{aligned} S_j(g-1) &\geq S_j(g) - \mu \\ &> a_j + (2l-1)\Delta - \mu \\ &\geq a_j + 2(l-1)\Delta. \end{aligned}$$

Applying the induction hypothesis, it follows that

$$|\Pi(P_j(g-1))| \geq 2^{l-1} - 1. \tag{5.5}$$

Now let $X(\tau) = \{p_i : i < j \text{ and } p_i \text{ crosses link } P_j(g) \text{ during time step } \tau\}$. Also, to simplify notation, let $t = S_j(g-1) + 1$. Then $X(t), X(t+1), \ldots, X(S_j(g)-1)$ are the sets of packets, each with cardinality w, that delay p_j at the head of link $P_j(g)$. Additionally, there may be sets of packets $X(r), X(r+1), \ldots, X(t-1)$ with cardinality w for consecutive time steps $r, \ldots, t-1$. If there are no such additional sets, then let $r = t$. Formally, we define $r = \min\{\tau : 2 \leq \tau \leq t,\ |X(\tau)| = w \text{ for all } \tau, \tau+1, \ldots, t\}$. Notice that, by definition,

$$r \geq S_j(g) - (\mu - 1). \tag{5.6}$$

To continue, we prove the following two lemmas.

Lemma 5.6

For all packets $p \in X(\tau)$, $r \leq \tau \leq S_j(g) - 1$, p must have crossed some link before it crossed link $P_j(g)$.

Proof

Suppose there exists a packet $p_i \in X(\tau)$, $r \leq \tau \leq S_j(g) - 1$, that did not cross link $P_j(g)$. Then $P_i(1) = P_j(g)$ and, by the definition of GFP, w earlier packets must cross link $P_j(g)$ during each time step $a_i + 1, a_i + 2, \ldots, \tau - 1$. Since w packets also cross link $P_j(g)$ during each time step $\tau, \tau + 1, \ldots, S_j(g) - 1$ and p_j crosses $P_j(g)$ during time step $S_j(g)$,

$$\begin{aligned}
\lceil \mu \rceil &\geq \left\lceil \frac{((S_j(g) - 1) - (a_i + 1) + 1)w + 1}{w} \right\rceil \\
&= (S_j(g) - 1) - (a_i + 1) + 2 \\
&> a_j + (2l - 1)\Delta - a_i && \text{by (5.4)} \\
&\geq \Delta && \text{since } a_j \geq a_i \text{ and } l > 0.
\end{aligned}$$

But this is a contradiction. ■

Lemma 5.7

There is at least one packet $p_i \in X(q)$, for some q, $r \leq q \leq t$, that must have crossed a link other than $P_j(g - 1)$ immediately prior to crossing link $P_j(g)$.

Proof

We know from the previous lemma that all packets $p \in X(\tau)$, $r \leq \tau \leq t$, crossed some link before crossing link $P_j(g)$. For contradiction, suppose that all these packets crossed link $P_j(g-1)$. Let $X = \bigcup_{r \leq \tau \leq S_j(g)-1} X(\tau) \cup \{p_j\}$. Notice that, since the $(t-r+1)w+1$ packets in X crossed link $P_j(g-1)$ before time step t, at least one packet $p \in X$ must have crossed link $P_j(g-1)$ before time step $r-1$. Since p crossed link $P_j(g)$ during a time step greater than $r-1$, p must have been delayed by w packets crossing link $P_j(g)$ during time step $r-1$. But this set of packets must be the set $X(r-1)$ which, by definition, has cardinality less than w, a contradiction. ■

We now continue with the proof of Lemma 5.5. By Lemma 5.7, there exists a packet $p_i \in X(q)$, $r \le q \le t$, that crossed a link $P_i(f) \ne P_j(g-1)$ immediately prior to crossing link $P_j(g)$. Packet p_i crosses $P_i(f)$ no earlier than during time step $r-1$. Otherwise, GFP would have assigned p_i to link $P_j(g)$ during time step $r-1$ at the latest, since fewer than w packets that appeared before p_i cross $P_j(g)$ during time step $r-1$. Using (5.4) and (5.6), we see that

$$\begin{aligned} t_i(f) &\ge r-1 \\ &\ge S_j(g) - (\mu-1) - 1 \\ &> a_j + (2l-1)\Delta - (\mu-1) - 1 \\ &\ge a_i + 2(l-1)\Delta. \end{aligned}$$

Hence, by applying the induction hypothesis, it follows that

$$|\Pi(P_i(f))| \ge 2^{l-1} - 1. \tag{5.7}$$

Lastly, we need to show that $\Pi(P_i(f)) \cap \Pi(P_j(g-1)) = \emptyset$. For contradiction, suppose that there exists a link e' such that $e' \in \Pi(P_i(f))$ and $e' \in \Pi(P_j(g-1))$. Then there is a path from the tail of e' to the heads of both $P_i(f)$ and $P_j(g-1)$. Since both $P_i(f)$ and $P_j(g-1)$ are immediate predecessors of $P_j(g)$, and $P_i(f) \ne P_j(g-1)$, these are two distinct paths between link e' and the tail of $P_j(g)$. But this contradicts our assumption that the network is a tree. Thus, by combining (5.5) and (5.7), we can conclude that

$$\begin{aligned}|\Pi(P_j(h))| &\geq |\Pi(P_j(g))| \\ &\geq |\Pi(P_i(f))| + |\Pi(P_j(g-1))| + 1 \\ &\geq 2^l - 1.\end{aligned}$$

The 1 in the second inequality counts $P_j(g) \notin \Pi(P_i(f)) \cup \Pi(P_j(g-1))$. ■

Lemma 5.8

$\max_j C_j \leq O(\log n)\Delta$.

Proof

If $\mu = 1$, then no packets will be delayed and competitive ratio is 1. Thus, assume $\mu > 1$. Consider an arbitrary schedule constructed by GFP and let p_j be the last packet to finish in this schedule. Let $l \geq 0$ be the integer satisfying

$$2l\Delta \leq C_j - a_j < 2(l+1)\Delta.$$

Thus, $S_j(d_j) \geq a_j + 2l\Delta$. By Lemma 5.5, we know that $|\Pi(P_j(d_j))| \geq 2^l - 1$. Thus,

$$\begin{aligned}C_j &< a_j + 2(l+1)\Delta \\ &\leq a_j + 2\left(\log\left(|\Pi(P_j(d_j))| + 1\right) + 1\right)\Delta \\ &\leq 2(\log(m+1))\Delta + 3\Delta \\ &= O(\log n)\Delta.\end{aligned}$$ ■

The Lower Bound

Leighton, Maggs, and Rao [57] constructed an instance that shows that the competitive ratio of any deterministic algorithm that chooses the order in which packets pass through a node independent of the paths that the packets take after they pass through the node (a nonpredictive strategy) is $\Omega\left(\frac{\log n}{\log\log n}\right)$ on a binary tree. Notice that this result applies to a number of intuitive scheduling algorithms, including fixed priority algorithms, FIFO, and any algorithm that bases its decision on how long a packet has been waiting in a queue. It does not however apply to the algorithm which gives priority to packets with the farthest yet to travel. By using the same network with a different request sequence, we will show that the competitive ratio of GFP is $\Omega(\log n)$ on a binary tree.

Theorem 5.6

The competitive ratio of GFP is $\Omega(\log n)$ on a binary tree.

Proof

For arbitrary positive integers c and d, we consider a full, directed binary tree of height $d-1$ with one extra node x connected to the root by a single edge. The edge between any two nodes is directed toward the node on the higher level. The edge connecting x to the root is directed toward x. We will say that x is on level 0, the root of the full binary tree is on level 1, etc. Label the $l = 2^{d-1}$ leaves $v_1, v_2, \ldots, v_l$ from left to right.

The request sequence consists of the $k = c \cdot l$ packets in the set

$$\{p_{i,j} : 1 \leq i \leq l,\ 1 \leq j \leq c\}.$$

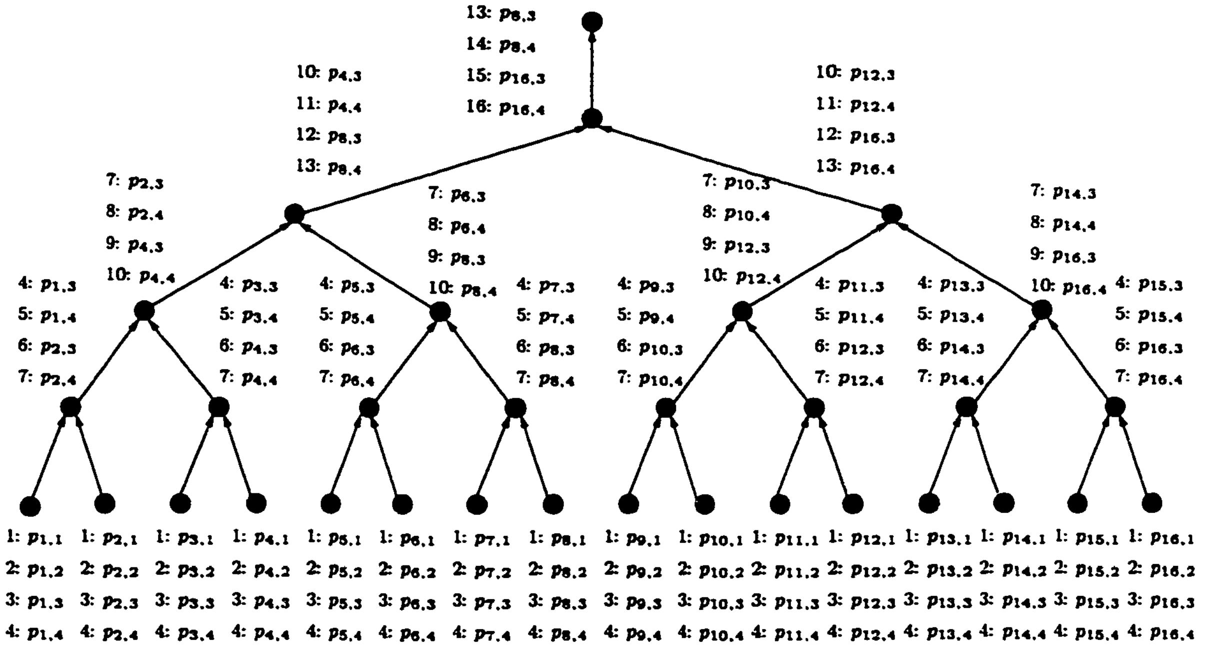

13: $p_{8,3}$ 14: $p_{8,4}$ 15: $p_{16,3}$ 16: $p_{16,4}$
10: $p_{4,3}$ 11: $p_{4,4}$ 12: $p_{8,3}$ 13: $p_{8,4}$
10: $p_{12,3}$ 11: $p_{12,4}$ 12: $p_{16,3}$ 13: $p_{16,4}$
7: $p_{2,3}$ 8: $p_{2,4}$ 9: $p_{4,3}$ 10: $p_{4,4}$
7: $p_{6,3}$ 8: $p_{6,4}$ 9: $p_{8,3}$ 10: $p_{8,4}$
7: $p_{10,3}$ 8: $p_{10,4}$ 9: $p_{12,3}$ 10: $p_{12,4}$
7: $p_{14,3}$ 8: $p_{14,4}$ 9: $p_{16,3}$ 10: $p_{16,4}$
4: $p_{1,3}$ 5: $p_{1,4}$ 6: $p_{2,3}$ 7: $p_{2,4}$
4: $p_{3,3}$ 5: $p_{3,4}$ 6: $p_{4,3}$ 7: $p_{4,4}$
4: $p_{5,3}$ 5: $p_{5,4}$ 6: $p_{6,3}$ 7: $p_{6,4}$
4: $p_{7,3}$ 5: $p_{7,4}$ 6: $p_{8,3}$ 7: $p_{8,4}$
4: $p_{9,3}$ 5: $p_{9,4}$ 6: $p_{10,3}$ 7: $p_{10,4}$
4: $p_{11,3}$ 5: $p_{11,4}$ 6: $p_{12,3}$ 7: $p_{12,4}$
4: $p_{13,3}$ 5: $p_{13,4}$ 6: $p_{14,3}$ 7: $p_{14,4}$
4: $p_{15,3}$ 5: $p_{15,4}$ 6: $p_{16,3}$ 7: $p_{16,4}$
1: $p_{1,1}$ 2: $p_{1,2}$ 3: $p_{1,3}$ 4: $p_{1,4}$
1: $p_{2,1}$ 2: $p_{2,2}$ 3: $p_{2,3}$ 4: $p_{2,4}$
1: $p_{3,1}$ 2: $p_{3,2}$ 3: $p_{3,3}$ 4: $p_{3,4}$
1: $p_{4,1}$ 2: $p_{4,2}$ 3: $p_{4,3}$ 4: $p_{4,4}$
1: $p_{5,1}$ 2: $p_{5,2}$ 3: $p_{5,3}$ 4: $p_{5,4}$
1: $p_{6,1}$ 2: $p_{6,2}$ 3: $p_{6,3}$ 4: $p_{6,4}$
1: $p_{7,1}$ 2: $p_{7,2}$ 3: $p_{7,3}$ 4: $p_{7,4}$
1: $p_{8,1}$ 2: $p_{8,2}$ 3: $p_{8,3}$ 4: $p_{8,4}$
1: $p_{9,1}$ 2: $p_{9,2}$ 3: $p_{9,3}$ 4: $p_{9,4}$
1: $p_{10,1}$ 2: $p_{10,2}$ 3: $p_{10,3}$ 4: $p_{10,4}$
1: $p_{11,1}$ 2: $p_{11,2}$ 3: $p_{11,3}$ 4: $p_{11,4}$
1: $p_{12,1}$ 2: $p_{12,2}$ 3: $p_{12,3}$ 4: $p_{12,4}$
1: $p_{13,1}$ 2: $p_{13,2}$ 3: $p_{13,3}$ 4: $p_{13,4}$
1: $p_{14,1}$ 2: $p_{14,2}$ 3: $p_{14,3}$ 4: $p_{14,4}$
1: $p_{15,1}$ 2: $p_{15,2}$ 3: $p_{15,3}$ 4: $p_{15,4}$
1: $p_{16,1}$ 2: $p_{16,2}$ 3: $p_{16,3}$ 4: $p_{16,4}$

Figure 5.1: An example with $c = 4$ and $d = 5$.

Let $s_{i,j}$ and $t_{i,j}$ denote the source and destination of packet $p_{i,j}$, respectively. The sources and destinations of the packets are defined as follows: $s_{i,j} = v_i$ and

$$t_{i,j} = \begin{cases} \text{ancestor of } v_i \text{ on level } d-1, & 1 \leq j \leq \lceil \frac{c}{2} \rceil \\ \text{ancestor of } v_i \text{ on level } d-h-1, & \lceil \frac{c}{2} \rceil + 1 \leq j \leq c,\ \frac{i}{2^{h-1}} \text{ odd},\ h = 1, 2, \ldots, d-2 \\ x, & \lceil \frac{c}{2} \rceil + 1 \leq j \leq c,\ i = 2^{d-2}, 2^{d-1} \end{cases}.$$

The packets all arrive at time 0 and are ordered lexicographically by index in the sequence.

The last packet, $p_{l,c}$, arrives at its destination at time $c + (d-1)(c-1) = (c-1)d + 1$. For example, see Figure 5.1. An optimal schedule has length $d + \frac{c}{2} - 1$. Therefore, if we set $c = d$, the competitive ratio is $\Omega(d) = \Omega(\log n)$. ■

5.5.2 Arbitrary Trees

Let us now consider arbitrary full-duplex tree networks. We naturally assume that packets follow acyclic routes. For any packet p_j, let top(j) denote the node in P_j that is closest to the root. Notice that each path in an arbitrarily directed tree can be partitioned into two subpaths. First, a packet follows a (possibly empty) subpath that is directed toward the root from node s_j to node top(j) and then it follows a (possibly empty) subpath that is directed away from the root from node top(j) to node t_j. If an instance consists of packets whose routes are directed only toward or only away from the root, then either Theorem 5.8 or Theorem 5.4 applies to that instance, respectively. For more general cases however, neither of these results applies directly.

Consider what happens if we use GFP to schedule packets on paths which are directed toward and then away from the root of a tree. In order to apply the previous results, we need to view the instance in terms of two request sequences: the original request sequence and one describing how each packet arrives at the node at the top of its path. The first sequence is considered to be on a tree directed toward the root and the second is on a set of linear arrays directed away from the root. By Theorem 5.8, each packet will arrive at its top node within $O(\log n)C^*$ time steps. However, Theorem 5.4 does not apply to the schedule assigned to the second (virtual) sequence because the priorities that the packets were assigned at their sources may not correspond to their arrival times at the top nodes on their paths.

We can fix this problem by reassigning a priority to each packet at its top node based on the time the packet arrives at its top node. Since GFP knows these times, it can assign these second priorities at the source. In other words, GFP assigns a priority to a packet based the packet's arrival time and then, based on the time the packet will reach the top node on its path, GFP assigns the packet a second priority for use on its subpath away from the root. Let $\hat{C}_j$ denote the time p_j reaches top(j) in the schedule constructed by GFP. Then, by Theorem 5.4, p_j will reach its destination by time $C_j \leq \hat{C}_j + d + \mu - 1 \leq O(\log n)C^*$. The only caveat about this approach is that the schedule for a packet p_j on its path away from the root is not necessarily known when the packet leaves its source, since packets may arrive later than p_j at their source nodes but earlier than p_j at their top nodes and thus be assigned higher priorities and eventually delay p_j.

5.6 Ring Networks

In this section, we consider full duplex ring networks. In a full duplex ring, there are exactly two routes that any packet can follow to reach its destination. Therefore, we will need to combine routing and scheduling for the first time.

Because GREEDY_ROUTE1 and GREEDY_ROUTE2 can choose arbitrarily long routes on arbitrary networks, a routing and scheduling algorithm which combines either GREEDY_ROUTE1 or GREEDY_ROUTE2 with GFP scheduling can perform badly on a ring. For example, consider an n node ring with nodes indexed $v_0, v_1, \ldots, v_{n-1}$. Given one request from node v_0 to v_{n-1}, both GREEDY_ROUTE1 and GREEDY_ROUTE2 may assign the long route, taking $n-1$ time steps, while an optimal algorithm will clearly assign the short route, taking just one time step. Therefore, we have the following theorem.

Theorem 5.7

The competitive ratio of any algorithm that chooses routes with either GREEDY_ROUTE1 or GREEDY_ROUTE2 is $\Omega(n)$ on ring networks.

As an alternative, we need to consider algorithms that give some preference to short routes. In the next subsection, we will consider the online algorithm that always chooses the short route and schedules packets with GFP. For convenience, we will call this algorithm RING_ROUTE and prove that it is at most $3 - \frac{1}{\Delta}$ competitive. In the special case that all packets originate at the same node, we will show that the algorithm is at most 2 competitive. Later we will show that the competitive ratio of any algorithm is at least $2 - \epsilon$ if the algorithm always assigns a packet to its shorter path if the path has length at most βn, for some $0 < \beta \leq \frac{1}{2}$. This result applies to the dynamic algorithm of Sibeyn [88], the only dynamic algorithm for packet routing on a ring of which we are aware. To beat this lower bound (if possible), we will need an algorithm that is adaptive in some sense.

5.6.1 Minimum Hop Routing and GFP Scheduling

Every simple path on a full duplex ring is contained in one of two disjoint simplex rings. We will call the simplex ring with links pointing clockwise the *right ring* and the simplex ring with links pointing counter-clockwise the *left ring*. A routing algorithm must decide whether to route a packet to its destination via the left ring or the right ring. The algorithm RING_ROUTE will always choose the simplex ring which contains the shorter route for each request. The packet's schedule is then assigned by GFP.

Before we prove our result, we define some useful notation. Let

$$A = \{j : P_j = P_j^*\} \text{ and } B = \{j : P_j \neq P_j^*\}.$$

Furthermore, let $A = A_l \cup A_r$ where A_l is the subset of indices in A of packets which the online algorithm routes in the left ring and A_r is the subset of indices in A of packets which the online algorithm routes in the right ring. Similarly, $B = B_l \cup B_r$. Therefore, RING_ROUTE routes the packets with indices in $A_l \cup B_l$ in the left ring and the optimal algorithm routes the packets with indices in $A_l \cup B_r$ in the left ring. The symmetric property holds for the right ring.

We will use the following two lemmas in the proof of our upper bound result. For a set of requests S, let

$$\mu_S = \max_{e \in E} \frac{|\{j \in S : e \in P_j\}|}{w}.$$

μ_S^* is defined analogously for the optimal route assignment.

Lemma 5.9

$\mu_{B_l} \leq \mu_{B_l}^*$ and $\mu_{B_r} \leq \mu_{B_r}^*$.

Proof

We will show that $\mu_{B_l} \leq \mu_{B_l}^*$. The argument for B_r is symmetric. Recall that the set of requests in B_l are assigned shortest paths in the left ring in the algorithm's route assignment, but are assigned long paths in the right ring in the optimal route assignment. Let $(v_i, v_{(i+1)\bmod n})$ denote the link that satisfies the definition of μ_{B_l}. Let $S \subseteq B_l$ be the subset of requests that are assigned to $(v_i, v_{(i+1)\bmod n})$ in the algorithm's route assignment. (So

$\mu_{B_l} = |S|$.) Now partition the nodes of the ring into two sets: let

$$X = \left\{v_\iota : \iota = \left(i + \left\lceil \frac{n}{2} \right\rceil + 1\right) \bmod n, \dots, i\right\}$$

and let

$$Y = \left\{v_\iota : \iota = (i+1) \bmod n, \dots, \left(i + \left\lceil \frac{n}{2} \right\rceil\right) \bmod n\right\}.$$

Notice that the source of every request in S is in X and the destination of every request in S is in Y, since the requests are following shortest paths. Therefore, since all the requests in S are assigned to the opposite direction in the optimal route assignment, every request in S must be assigned to the link $\left(v_{(i+\lceil \frac{n}{2} \rceil + 1) \bmod n}, v_{(i+\lceil \frac{n}{2} \rceil) \bmod n}\right)$ in the optimal route assignment. Thus, $\mu^*_{B_l} \geq |S| = \mu_{B_l}$. ■

Lemma 5.10

$\mu \leq 2\mu^*$.

Proof

In the worst case, the total congestion incurred by the online algorithm is

$$\mu \leq \max\left\{\mu_{A_l} + \mu_{B_l},\ \mu_{A_r} + \mu_{B_r}\right\}.$$

The inequality is an equality if, in the simplex ring with the maximum congestion (say the left ring), the link e that satisfies $\mu_{A_l}(e) = \mu_{A_l}$ also satisfies $\mu_{B_l}(e) = \mu_{B_l}$.

Without loss of generality, suppose $\mu_{A_l} + \mu_{B_l} \geq \mu_{A_r} + \mu_{B_r}$. Then,

$$\begin{aligned} \mu &\leq \mu_{A_l} + \mu_{B_l} \\ &\leq \mu_{A_l} + \mu^*_{B_l} \\ &\leq 2\mu^*. \end{aligned}$$

The second inequality follows from Lemma 5.9. ■

We can use the preceding lemma to prove the following theorem.

Theorem 5.8

RING_ROUTE is $3 - \frac{1}{\Delta}$ competitive on a full duplex ring.

Proof

Let p_j denote the packet that completes last in the online schedule. Since the schedule on the full duplex ring is equivalent to two disjoint schedules, each on a simplex ring, we know from Lemma 5.2 that

$$\begin{aligned} C_j &\le a_j + |P_j| + c_j - 1 \\ &\le \max_i \{a_i + |P_i^*|\} + \lceil \mu \rceil - 1 \\ &\le \delta + 2\lceil \mu^* \rceil - 1. \end{aligned}$$

The second inequality follows because packets are assigned to their shortest routes by the online algorithm. The last inequality follows from Lemma 5.10. Therefore, the competitive ratio of RING_ROUTE is at most

$$\frac{\delta + 2\lceil \mu^* \rceil - 1}{\max\{\lceil \mu^* \rceil, \delta\}} \le 3 - \frac{1}{\Delta}.$$

■

In the special case that all packets arrive at the same time and originate at the same node, we can show that RING_ROUTE is 2 competitive.

Theorem 5.9

If all packets arrive at time 0 and have the same source, then RING_ROUTE is 2 competitive on a full duplex ring.

Proof

Recall that RING_ROUTE assigns the requests whose indices are in A_l and B_l to the left ring and the requests whose indices are in A_r and B_r to the right ring. Therefore, by Lemma 5.2, the makespan of the algorithm's schedule is at most

$$\max\{d_l + (|A_l| + |B_l|) - 1,\ d_r + (|A_r| + |B_r|) - 1\},$$

where d_l and d_r are the lengths of the longest paths in the online schedule in the left and right rings, respectively. The optimal algorithm assigns the requests whose indices are in A_l and B_r to the left ring and the requests whose indices are in A_r and B_l to the right ring. Notice that all of the requests in B_l and B_r follow paths of length at least $\frac{n}{2}$ in the optimal schedule. Therefore, since all such requests have the same source, the optimal makespan can be bounded from below by both $\frac{n}{2} + |B_l| - 1$ and $\frac{n}{2} + |B_r| - 1$, if $B_l \neq \emptyset$ and $B_r \neq \emptyset$, respectively. The optimal makespan can also be bounded from below by both $|A_l|$ and $|A_r|$. Without loss of generality, suppose that $d_l + (|A_l| + |B_l|) - 1 \geq d_r + (|A_r| + |B_r|) - 1$. If $B_l = \emptyset$ then the competitive ratio of RING_ROUTE is at most

$$\frac{d_l + |A_l| - 1}{\max\{d_l, |A_l|\}} < 2.$$

Otherwise, the competitive ratio of RING_ROUTE is at most

$$\frac{\frac{n}{2} + (|A_l| + |B_l|) - 1}{\max\{\frac{n}{2} + |B_l| - 1, |A_l|\}} \leq 2.$$

■

5.6.2 A Lower Bound

We can show that the competitive ratio of any online algorithm which consistently assigns certain packets to their shortest paths must be at least $2 - \epsilon$, regardless of the scheduling algorithm.

Theorem 5.10

The competitive ratio of any routing algorithm which always assigns a request to its shortest path if the shortest path has length at most βn, for any $0 < \beta \leq \frac{1}{2}$, is at least $2 - \epsilon$ for arbitrarily small positive ϵ.

Proof

Consider a ring with n nodes and a sequence of $k \geq (1 - 2\beta)n$ packets, all with source v_0, destination $v_{\lfloor \beta n \rfloor}$, and arrival time 0. The algorithm will assign all these requests to their shortest path, resulting in makespan $\lfloor \beta n \rfloor + k - 1$. On the other hand, an optimal algorithm will assign $\lceil \alpha k \rceil$ packets to the shortest path and $\lfloor (1-\alpha)k \rfloor$ packets to the long path, where $\alpha = \frac{1}{2} + \frac{(1-2\beta)n}{2k}$. The makespan of this schedule will be less than $\frac{n+k}{2}$. Therefore, the competitive ratio for this instance is at least

$$2\left(\frac{\lfloor \beta n \rfloor + k - 1}{n + k}\right).$$

For an arbitrarily large value of k, this quantity approaches 2. ■

Therefore, among the class of algorithms considered in Theorem 5.10, RING_ROUTE is asymptotically optimal if all packets arrive at the same time and originate at the same node. This is somewhat remarkable since one would think that an algorithm that gives priority to the packet that has the farthest yet to travel might beat an algorithm using GFP scheduling. However, in the worst cases (when many paths have the same length), GFP is just as good, and simpler.

Sibeyn [88] designed an algorithm for the dynamic case that assigns a packet to its shortest path if the shortest path has length at most $\frac{n}{3}$ and alternates the directions assigned to other packets. Packets are forwarded locally by giving preference to the packet that has the farthest yet to travel. Sibeyn conjectured that this algorithm might be worst case optimal for k-k distributions based on examples. While this may be true when packets are evenly distributed over the network, it follows from Theorem 5.10 that the algorithm is clearly not optimal for arbitrary distributions of packets.

Theorem 5.10 implies that any deterministic algorithm with a competitive ratio better than 2 must be adaptive in some sense. An adaptive algorithm would examine the network state and then assign a route accordingly. This is an interesting area for future research.

5.7 Arbitrary Networks

In Chapter 2, we showed that the competitive ratio of any online routing and scheduling algorithm on a layered network is $\Omega(\log n)$ with respect to makespan. This result is based entirely on the network congestion the instance forces the routing algorithm to incur.

Cidon, Kutten, Mansour, and Peleg [30, 31] constructed a route assignment on a certain non-layered network which forces a greedy, fixed priority scheduling algorithm to assign a schedule with makespan $\Omega\left(k + d\sqrt{k}\right)$, where d is the dilation of the set of routes. In the construction, both d and k are $O(n)$. If either GREEDY_ROUTE1 or GREEDY_ROUTE2 were given as input the same underlying network and request sequence, they could choose the same bad set of paths because every path for every packet shares a common first link. The optimal makespan for the instance is $O(d+k)$ since all the packets can be routed along a line contained in the network. Thus, the competitive ratio of GREEDY_ROUTE1 or GREEDY_ROUTE2 combined with GFP scheduling is

$$\Omega\left(\frac{k+d\sqrt{k}}{d+k}\right) = \Omega\left(\sqrt{n}\right),$$

even when $d, k = O(n)$.

It is interesting to note, however, that if GREEDY_ROUTE1 or GREEDY_ROUTE2 chooses the shortest minimum congestion route for every packet in this instance, it will choose the optimal set of routes, which all lie along the aforementioned line of nodes.

5.8 Other Algorithms

In this chapter, we explored the feasibility of scheduling arbitrary dynamic packet sequences according to a priority scheme based on arrival times. This method was shown to perform well on a few simple, but common networks. On a linear array, for example, an arbitrary priority scheme can result in worst case makespan for a dynamic instance while GFP constructs a schedule with makespan less than twice optimal. In this section, we briefly discuss other possible algorithm ideas that fit into our model of interest.

5.8.1 Greedy Algorithms in General

Is GFP the only online greedy scheduling algorithm that can construct schedules in advance? Recall that a greedy schedule must not delay a packet at link e during time step t unless there are w packets assigned to e during time step t. GFP is the natural way to construct such a schedule. However, there are other possibilities. For example, a greedy, fixed priority algorithm need not assign priorities based on arrival times. An algorithm could assign the priority of a packet at the source so that local nodes are forced to let it go in front of packets that arrived earlier. Another idea would be to leave "gaps" in the schedule to allow future requests to jump ahead of past requests, but then the nodes would need to locally "compact" the schedule if the gaps are not filled when a packet that has been scheduled to wait arrives at the node. For example, suppose packet p arrives at node v at time t and has not been scheduled to cross its next link (v, w) until time step $t+2$. But at time t only $w-1$ packets have arrived at v that are scheduled to cross (v, w) during time step $t+1$. Then the node would choose to send p (or another packet in the same situation) across (v, w) during time step $t+1$, earlier than it was originally scheduled. In this scenario, the schedule allocates time *guarantees* rather than absolute times. Packets could cross links ahead of schedule, and arrive at their next links earlier than originally scheduled. Notice that a schedule is allowed to change from its original form by moving packets earlier only; packets may not be moved later than originally scheduled.

There are many interesting questions regarding GFP on general networks. For example, is there a set of paths in any network for which GFP gives a short schedule relative to the optimal schedule? Is there an online algorithm that can find a good set of paths? We could use a minimum spanning tree in an arbitrary network, but then we would have to show a relationship between this set of routes and an optimal set of routes to prove anything about the competitive ratio.

5.8.2 An Exponential Scheduling Algorithm

The question of whether a variant of EXP_ROUTE can be used to route and schedule packets on general networks is compelling. A good algorithm for general networks needs to both choose short paths and minimize congestion, both of which are done by EXP_ROUTE. This approach was applied recently by Awerbuch, Azar, and Fiat [8] with some success to a packet routing model in which both link and queue capacities may be exceeded. They showed that a variant of EXP_ROUTE can produce a schedule that is simultaneously $O\left(\log\left(T\sum_{e\in E} c(e)\right)\right)$ competitive with respect to congestion, where T is the maximum packet delay, and $O(1)$ competitive with respect to average packet delay.

Can this approach be used to construct schedules that do not exceed capacities? Unfortunately, a straightforward modification presents several problems. First, if we minimize the exponential function only over feasible schedules, then the current competitive analysis is no longer valid. Second, the algorithm as it stands has each packet p_j either enter a queue at its source or cross the first link on its path during step $a_j + 1$. But this may not be possible if we respect capacities. Therefore, a packet must be allowed to remain in the initial queue at its source indefinitely. But if we assign initial queues infinite capacity and incorporate this into the exponential function, then the algorithm, as it stands, will always choose a feasible schedule with zero congestion by waiting sufficiently long at the source. This will clearly have a bad impact on makespan, delay, and response time. We have explored several adaptations so far without success.

The exponential algorithm packet routing algorithm does not necessarily send a packet out on the first available route. It may hold the packet until the first link on another, better route is available. It would be interesting to know whether we can prove a lower bound on the competitive ratio of an algorithm that always uses the first available route.

5.8.3 Using an Offline Algorithm

Another approach to our problem is to simply wait for all the packets to arrive and then schedule them optimally offline. If k is finite, then this approach would result in makespan $a_k + C^* \leq 2C^* - 1$, since $C^* \geq a_k + 1$. Therefore, the competitive ratio of this algorithm is $2 - \epsilon$. However, there are a few problems with this technique. First, it is NP-hard. Second, the shortest packet completion time is greater than C^*. On the other hand, our results for GFP include bounds on the completion time of any individual request. Third, this won't work on infinitely long sequences, unless we violate capacity constraints. This approach was taken in [8] to get an algorithm which is 3-competitive with respect to makespan and 2-competitive with respect to congestion.

Chapter 6

Conclusions and Future Work

We have defined a general problem in which data transfer requests on a network become known to an algorithm one at a time and the algorithm must efficiently assign a route and/or a schedule to each data transfer request without knowing future requests. The arrival process may be arbitrary. We measure the efficiency of an online algorithm by its competitive ratio, the maximum ratio, over all request sequences, of the cost of the online algorithm's solution to that of an optimal offline algorithm that knows the entire request sequence in advance.

We identify two distinct variations of this general problem. In the first, data transfer requests are permanent virtual circuit requests and the goal is to minimize the network congestion caused by the route assignment. In the second variation, data transfer requests are packets and the goal is to minimize the completion time of the last packet. In the next two sections, we will summarize our results for these two problems and outline related future research directions. In the last section, we mention a novel new model for routing and scheduling problems that we hope to investigate further in the near future.

6.1 Online Permanent Virtual Circuit Routing

We have given several results concerning the efficiency of algorithms that assign routes to arbitrary sequences of permanent virtual circuit requests in an arbitrary communication network with equal capacity links. We first presented a new proof showing that the competitive ratio of any online algorithm for permanent virtual circuit routing is at least $\frac{\log n+3}{2}$ with respect to network congestion. This result also implies a lower bound of $\frac{\log n+5}{4}$ with respect to makespan for an online routing and scheduling algorithm. The congestion result improves by an additive factor a previous lower bound result of Aspnes, et al. [4, 5]. Compared to that construction, ours requires a smaller network and approximately half as many requests, and shows how to raise the lower bound for a randomized online algorithm against an adaptive online adversary. The lower bound is tight with respect to the number of requests, up to an additive one.

We analyzed the competitive ratio of two greedy online algorithms for permanent virtual circuit routing. The simpler greedy algorithm, GREEDY_ROUTE1, was proposed by Mao and Simha [72]. We showed that the competitive ratio of GREEDY_ROUTE1 is $\Theta\left(\sqrt{\mathcal{D}m}\right)$ on arbitrary networks when the bandwidth requirements of requests are approximately equal, where $\mathcal{D}$ is the ratio of the longest to shortest path for any particular request and m is the number of network links. When bandwidth requirements are arbitrary, our upper bound is increased by a factor of $O\left(\sqrt{\mathcal{L}}\right)$, where $\mathcal{L}$ is the ratio of the maximum to minimum bandwidth requirement. We showed that a second greedy algorithm, GREEDY_ROUTE2, is superior to GREEDY_ROUTE1 when the length of the longest path in the network is small relative to the size of the network. Specifically, at least when the set of optimal routes for a request sequence has a small amount of overlap, GREEDY_ROUTE2 is $\max\left\{O(d\log n), O\left(\sqrt{\mathcal{D}m}\right)\right\}$ competitive, where d is the length of the longest path assigned to a request and n is the number of network nodes. If $d = O(\log n)$, as is the case in many common networks, then the competitive ratio of GREEDY_ROUTE2 is polylogarithmic; if d is constant, it is asymptotically optimal. We also showed that the competitive ratio of GREEDY_ROUTE2 is $\Omega\left(d + \log\left(n - d\right)\right)$ on arbitrary networks and $\Omega\left(d + \log\left(\frac{n}{d} - d\right)\right)$ on layered networks.

These results answer open questions posed by Mao and Simha [72] and also present alternatives to the asymptotically optimal, but computationally more expensive algorithm of Aspnes, et al. [4, 5]. We discussed situations in which the greedy algorithms can be expected to perform well in Chapters 3 and 4. Since there is a tradeoff between speed and efficiency in the choice of algorithms, the decision of which to use really depends on the requirements of the situation. We would like to further investigate these questions in the future through simulations. It will be important to find appropriate networks for this purpose. One can always construct a network that makes the algorithms look good; we would ideally like to find real networks on which these algorithms would be appropriate.

There are a few technical points related to GREEDY_ROUTE1 and GREEDY_ROUTE2 that are topics of continuing research. First, we suspect that the $O\left(\sqrt{\mathcal{L}}\right)$ term does not belong in the true competitive ratio of GREEDY_ROUTE1. In the proof, it appears that inequality (3.5) is weak; we would like to improve it to perhaps get a tighter result. More work also needs to be done on the competitive ratio of GREEDY_ROUTE2. While it was important to show that the competitive ratio of GREEDY_ROUTE2 is polylogarithmic for some cases, it is clear to us that this upper bound is not tight. The two terms in the upper bound of Theorem 4.3 do not really "mesh"; the first term can be much larger than the second term for large values of d. Based on examples, we would conjecture that the competitive ratio of GREEDY_ROUTE2 is closer to the lower bound of $O(d + \log n)$.

There are also more general questions that interest us concerning the greedy algorithms. First, can we improve GREEDY_ROUTE2 by breaking ties in different ways? For example, if GREEDY_ROUTE2 breaks ties in favor of the route with the fewest maximum congestion links, then we lose the additive d in Theorem 4.6. It might also be interesting to look into random tie breaking schemes. Second, what can we say about the algorithms in cases with arbitrary capacity links? The analyses of the greedy load balancing algorithm do not preclude a polylogarithmic competitive ratio in this case. It would also be interesting to learn something about how the two algorithms handle switched virtual circuits. In this case, the definition of congestion changes to incorporate time, and the arrival and departure of network traffic. Lastly, Rivera-Vega, et al. [83] proposed using a greedy algorithm related to GREEDY_ROUTE2 for assigning routes to file transfers in fully connected networks, but were not able to give any analysis for their algorithm. Can our analysis can be applied to this situation?

6.2 Online Packet Routing and Scheduling

We also investigated the quality of schedules constructed by an online greedy packet scheduling algorithm, GFP. We analyzed GFP on arbitrary online (dynamic) request sequences in which any number of packets may originate at or be destined for any node. Each packet is simply assigned a priority based upon its arrival time and then no packet is delayed by a packet with a lower priority during its schedule. We pointed out that this type of scheduling, although not necessarily optimal, is advantageous in that network switches must do minimal work to forward packets. This type of scheduling was also proposed by Mao and Simha [72] and Rivera-Vega, et al. [83] but not analyzed by either group.

We showed that the completion time of any packet scheduled by GFP on a linear array, or any directed network with indegree one, is at most $a_j + d_j + c_j - 1$, where $\hat{c}_j$ is the maximum congestion on any link of P_j up to and including packet p_j. The competitive ratio of GFP is exactly $2 - \frac{1}{\Delta}$, where $\Delta = \max\{\lceil \mu \rceil, \max_j \{a_j + d_j\}\}$. We also showed that the competitive ratio of GFP is $\Theta(\log n)$ on a general directed tree in which there is exactly one path between any two nodes.

On a full-duplex ring, there are exactly two paths that any packet may follow to reach its destination. For this situation, we considered the algorithm that always assigns a packet to its shortest path and assigns a schedule with GFP. For convenience, we called this algorithm RING_ROUTE and showed that it is $3 - \frac{1}{\Delta}$ competitive. If all packets arrive at the same time and originate at the same node then RING_ROUTE is 2 competitive. Lastly, we showed that the competitive ratio of any algorithm for the ring which always assigns a packet to its shorter path if the length of the shorter path is at most βn, for any $0 < \beta < \frac{1}{2}$, is at least $2 - \epsilon$ for an arbitrarily small positive ϵ. This result implies that any deterministic online algorithm that is better than $2 - \epsilon$ competitive must be adaptive: the algorithm must use its knowledge of the past to make routing decisions.

The investigation of adaptive online algorithms for the ring is a primary topic for future research. We would also like to prove a tight competitive ratio for RING_ROUTE. We have so far been unable to construct an instance which causes RING_ROUTE to have a competitive ratio greater than 2. Lastly, we would like to analyze the competitive ratio (or performance ratio, as the case may be) of the algorithms proposed by Makedon and Symvonis [64] and Sibeyn [88]. These algorithms were only analyzed with respect to their worst case makespan. One of the algorithms in [88] is for dynamic instances and would fit into our model if it were not for the scheduling, which is performed by giving priority to the packet with farthest distance yet to travel. From our lower bound result above, we already know that the competitive ratio of this algorithm is at least $2 - \epsilon$. It would be quite interesting to show that both RING_ROUTE and this algorithm match this lower bound. This would imply that farthest first scheduling is no better than GFP scheduling with respect to competitive ratio on a ring. The other algorithms in [64, 88] are designed for static $k-k$ permutations but it would still be interesting to prove bounds on the ratio between the algorithm's makespan and optimal makespan for each particular instance, in the spirit of competitive analysis. This was also mentioned as an interesting open problem by Kaufmann and Sibeyn [50].

There are several ways in which we may attempt to generalize or alter the network model in the future. First, we might consider files of arbitrary length or networks links of arbitrary capacity (although unit length packets and unit capacities are standard in the literature). Second, for long or infinite request sequences, it would be worthwhile to examine the competitive ratio of GFP with respect to the maximum or average delay $(C_j - a_j)$. Third, it would be important to look at more general networks. A related issue would be the consideration of half-duplex networks. A network is *half-duplex* if packets can move between adjacent nodes in both directions, but not concurrently. Fourth, we would be interested in considering more realistic time models. We assumed, as most packet routing models do, that time is synchronous. This assumption is valid in parallel computer architectures, but it is not valid in general distributed systems. For these harder cases, one would need to base a priority scheme on logical clocks rather than real clocks. It might also be possible to assume roughly synchronized clocks. With these more general, distributed assumptions, standard competitive analysis is no longer a good measure of efficiency because, in addition to a lack of knowledge about future requests, there is a lack of knowledge about event order and reliability. New distributed models of competitive analysis have been proposed recently [1, 6].

Lastly, we wish to consider cases in which queue lengths are bounded, and therefore a packet may be delayed because the queue at the head of its next link is full. The routing and scheduling problem with bounded queues is clearly a much more difficult problem [55]. Most scheduling models in the literature assume unbounded queues during the routing and then show expected queue sizes in their analysis. Far fewer papers, to our knowledge (e.g., [59, 56, 63]), have successfully addressed the issue of arbitrarily bounded queues in any non-stochastic packet routing problems. One interesting special case of this problem is *hot potato routing* or *deflection routing*, in which packets are never allowed to wait in intermediate queues. Rather, at each step, a node's queues must be emptied and all packets sent on some adjacent link. Sometimes this means that a packet must be *derouted*, or sent away from its destination temporarily. A special case of hot potato routing is *direct routing*, in which packets are never derouted: once a packet leaves its source, it moves one link closer to its destination with each step. Symvonis [91] and Alstrup, et al. [2] designed offline direct routing algorithms for trees that first construct a certain order for the packets and then assign start times greedily. Each of these algorithms is guaranteed to deliver every packet to its destination in at most $n - 1$ steps. It would be interesting to study the online version of the problem in which packets cannot be ordered before start times are assigned.

6.3 A Different Link Scheduling Model

The classical models of machine scheduling and load balancing do not accurately represent job scheduling in multiprogramming operating systems. The classical models allow for only one job at a time to execute on any machine. Some models allow an algorithm to preempt jobs. Depending upon the model, a preempted job must either continue on the same machine at a later time without penalty [53] or be restarted at a later time, possibly on a different machine [85, 86]. In either case, the preempted job must leave the processor while another job is executing. Although multiprogramming can theoretically be modeled with the former model of preemption, to accurately do so might require that each job be split into thousands, or even millions, of pieces. Instead, we propose a model in which a job may be started on a processor at any time without removing other jobs. Rather, the rate at which each job executes on a machine is simply slowed in proportion to the number of jobs that are currently assigned to the machine.

For example, consider the following example on a single machine: job J_1 arrives at time 0 and has processing requirement M, and jobs J_2, J_3, and J_4 arrive at time ϵ and have processing requirement 1. If J_1 is scheduled first and the others are forced to wait, then the sum of the completion times of the jobs is $M+(M+1)+(M+2)+(M+3)=4M+6$. On the other hand, if jobs J_2, J_3, and J_4 are allowed to start execution when they arrive, then the total completion time is $3(4+\epsilon)+(M+3)=M+15+3\epsilon$, which is close to the optimal time of $(1+\epsilon)+(2+\epsilon)+(3+\epsilon)+(M+3)=M+9+3\epsilon$. This problem can be viewed of as a kind of packing problem, as depicted in Figure 6.1. A machine is represented by a rectangle with height proportional its processing power and length equal to the time interval

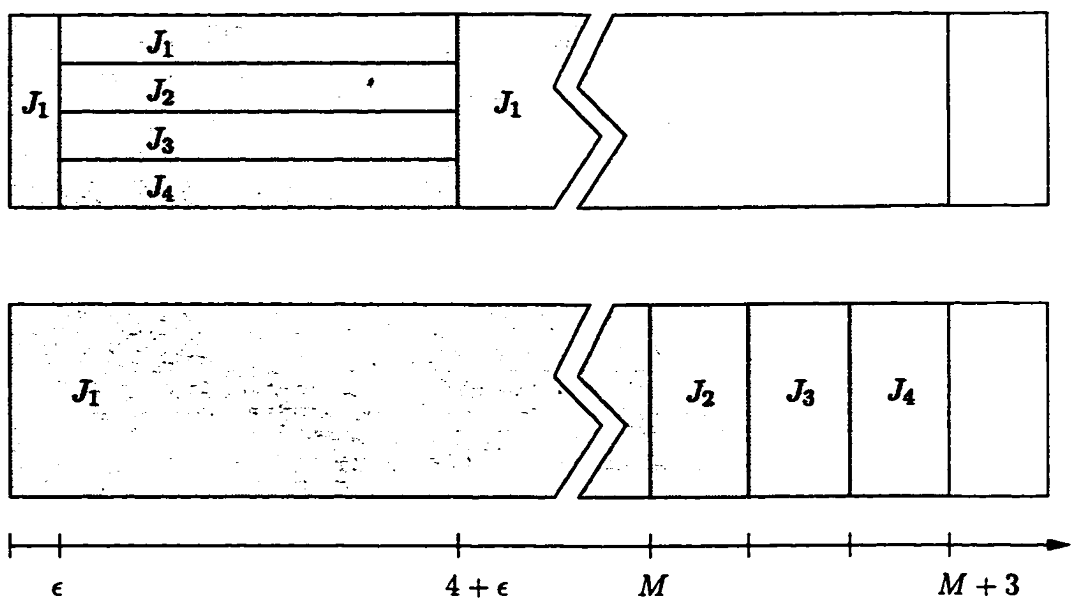


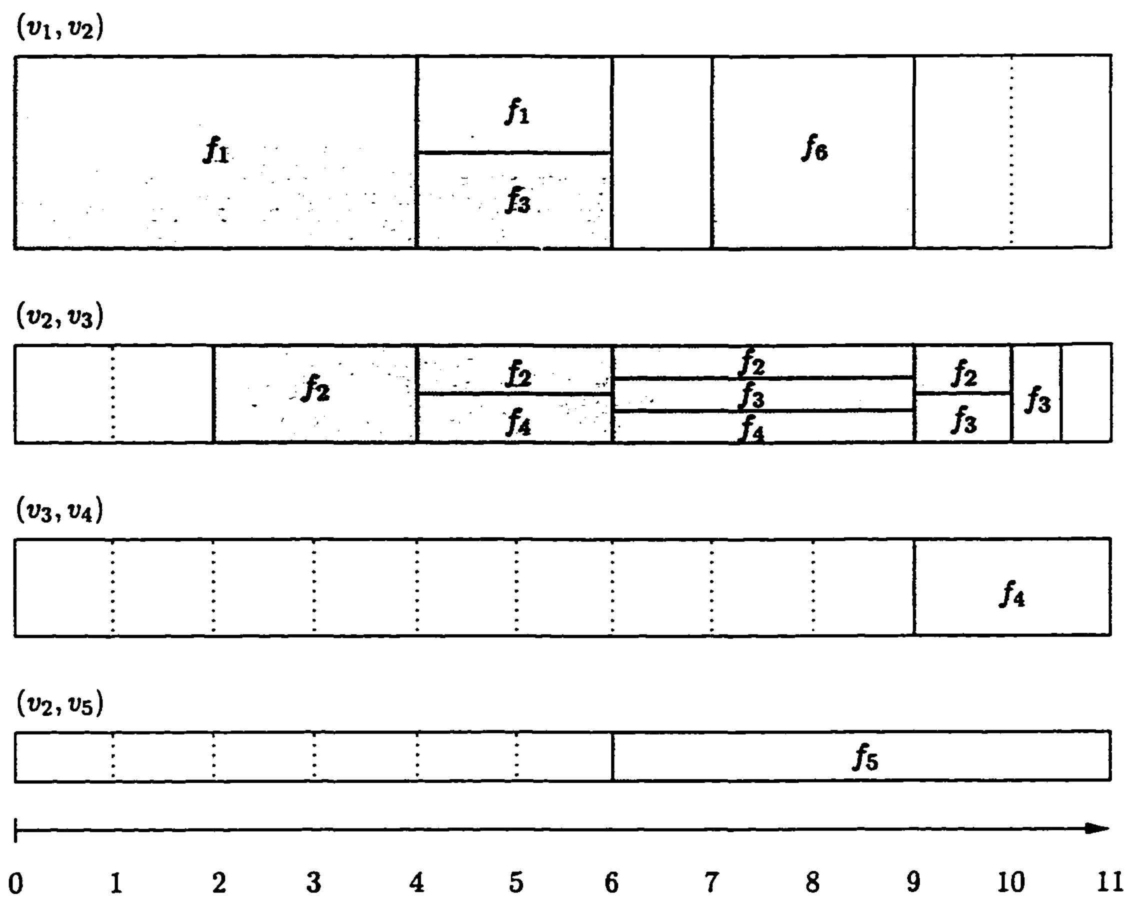
Figure 6.1: An illustration of multiprogramming and serial scheduling models.

considered in the problem. Each job is represented by a rectangle with area equal to its processing requirement. The top illustration in Figure 6.1 depicts the multiprogramming, or shared, model while the bottom illustration depicts the serial model.

We can also apply this idea to the problem of transferring file transfers in a network: we allow several files to share a link at the same time. In the underlying network protocol, the files are transferred simultaneously over the same link by rotating use of the link on a packet-by-packet basis. However, modeling such a transfer at the packet level would be too cumbersome. On the other hand, pretending that links are switched on a file by file basis is unrealistic.

In this model, as long as there is at least one file being transferred over any link, the link's bandwidth is fully utilized and divided evenly among all the files using it. As a result, the speed of each file transfer at a particular moment is slowed in proportion to the number of files using the link. In particular, if at time t, j files are using link e, then the speed of the link is $\frac{c(e)}{j}$ as far as each file is concerned. We note that the bandwidth used by a file may change with time as files finish or more files arrive at the link. Although pipelining files is certainly possible, it is simplest to assume that a complete file arrives at the head of a link before it begins its traversal of the next link on its path. Once a complete file reaches a node, it may be delayed there at the discretion of an algorithm. We note that these assumptions imply that nodes "understand" files in some sense.

Again, it is convenient to depict the construction of a schedule as a kind of packing problem. The bandwidth available on each link is depicted by a rectangle with height equal to its capacity and length equal to the extent of time considered in the problem. Each file transfer f_j is represented by a rectangle with area equal to the file transfer's length l_j. The object is to appropriately pack the area of each of the file transfers into the rectangles representing the links. If a file transfer has length l_j then it must occupy an area equal to l_j on each link in its path before it progresses to the next link in its path. For example, the example in Figure 6.2 illustrates a schedule assigned to an instance consisting of six file transfers — $f_1 = (v_1, v_2, 20, 0)$, $f_2 = (v_2, v_3, 9, 2)$, $f_3 = (v_1, v_3, 4, 4)$, $f_4 = (v_2, v_4, 4, 4)$, $f_5 = (v_2, v_5, 5, 6)$, and $f_6 = (v_1, v_2, 8, 7)$ — on four links — (v_1, v_2) with capacity 4, (v_2, v_3) with capacity 2, (v_3, v_4) with capacity 2, and (v_2, v_5) with capacity 1. Further investigation of this model is left as a topic for future research.



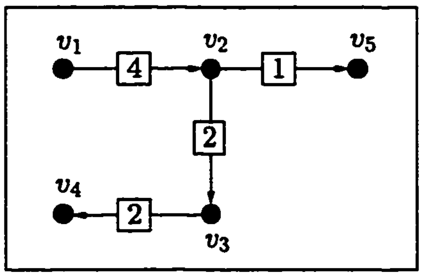


Figure 6.2: An illustration of routing and scheduling in the shared link model.

Appendix A

Proof of Lemma A.1

The following simple lemma is used in the proof of Theorem 4.2 in Chapter 4.

Lemma A.1

For any $a, y \geq 1$, $a \ln y \leq \left\lceil \log_{\left(\frac{a+1}{a}\right)} y \right\rceil \leq a \log_2 y + 1$.

Proof

$$\log_{\left(\frac{a+1}{a}\right)} y \leq \left\lceil \log_{\left(\frac{a+1}{a}\right)} y \right\rceil \leq \log_{\left(\frac{a+1}{a}\right)} y + 1$$

$$\Updownarrow$$

$$\frac{\ln y}{\ln\left(\frac{a+1}{a}\right)} \leq \left\lceil \log_{\left(\frac{a+1}{a}\right)} y \right\rceil \leq \frac{\log_2 y}{\log_2\left(\frac{a+1}{a}\right)} + 1$$

$$\Updownarrow$$

$$\frac{a \ln y}{a \ln\left(\frac{a+1}{a}\right)} \leq \left\lceil \log_{\left(\frac{a+1}{a}\right)} y \right\rceil \leq \frac{a \log_2 y}{a \log_2\left(\frac{a+1}{a}\right)} + 1$$

$$\Updownarrow$$

$$\frac{a \ln y}{\ln\left(\frac{a+1}{a}\right)^a} \leq \left\lceil \log_{\left(\frac{a+1}{a}\right)} y \right\rceil \leq \frac{a \log_2 y}{\log_2\left(\frac{a+1}{a}\right)^a} + 1$$

$$\Updownarrow$$

$$\frac{a \ln y}{\ln e} \leq \left\lceil \log_{\left(\frac{a+1}{a}\right)} y \right\rceil \leq \frac{a \log_2 y}{\log_2 2} + 1$$

$$\Updownarrow$$

$$a \ln y \leq \left\lceil \log_{\left(\frac{a+1}{a}\right)} y \right\rceil \leq a \log_2 y + 1.$$

■

Appendix B

Common Notation

The following is a list of the notation most commonly used in this dissertation.

$\log n = \log_2 n$

a_j is the arrival time of the j^{th} request

$c(e)$ is the capacity of network link $e \in E$

C_j is the completion time of the j^{th} request

$d = \max_{1 \le j \le k} D_j$

$d_j = \min_{P \in \mathcal{P}_j} |P|$

$D_j = \max_{P \in \mathcal{P}_j} |P|$

$\mathcal{D} = \max_{1 \le j \le k} \frac{D_j}{d_j}$

$e \in E$ is a network link (also e_i)

E is a set of network links

f_j is the j^{th} data transfer request

$G = (V, E)$ is a directed graph representing a communication network

$k = |\sigma|$

l_j is the length or bandwidth requirement of the j^{th} request

$\mathcal{L} = \frac{\Lambda}{\lambda}$

$m = |E|$

$n = |V|$

p_j is the j^{th} packet request

$P_j \in \mathcal{P}_j$ is the path assigned to the j^{th} request by an online algorithm

$P_j^* \in \mathcal{P}_j$ is the path assigned to the j^{th} request by an optimal offline algorithm

$P_j(i)$ is the i^{th} link in path P_j

$|P|$ is the number of links in path P

$\mathcal{P}_j$ is the set of paths between s_j and t_j

s_j is the source node of the j^{th} request

S_j is the online schedule assigned to packet p_j

$S_j(i)$ is the time step during which packet p_j crosses link $P_j(i)$

t_j is the destination node of the j^{th} request

V is a set of network nodes

w is a uniform link capacity

$\delta = \max_{1 \le j \le k} \left\{ a_j + \left| P_j^* \right| \right\}$

$\Delta = \max \{ \lceil \mu^* \rceil , \delta \}$

$\lambda = \min_{1 \leq j \leq k} l_j$

$\Lambda = \max_{1 \leq j \leq k} l_j$

$\mu_j(e) = \sum_{i<j: e \in P_i} \frac{l_i}{c(e)}$

$\mu(e) = \mu_{k+1}(e)$

$\mu_j(P) = \max_{e \in P} \mu_j(e)$

$\mu(P) = \mu_{k+1}(P)$

$\mu_j = \max_{e \in E} \mu_j(e)$

$\mu = \mu_{k+1}$ is the network congestion given by an online algorithm for a problem instance

μ^* is the optimal network congestion for a problem instance

σ is a request sequence

Bibliography

[1] M. Ajtai, J. Aspnes, C. Dwork, and O. Waarts. A theory of competitive analysis for distributed algorithms. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 401–411, 1994.

[2] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Direct routing on trees. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 342–349, 1998.

[3] A. Amir, M. Blum, M. Loui, J. Savage, and C. Smith. Contributions of theoretical computer science. *SIGACT News*, 26(4):2–4, Dec. 1995.

[4] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 623–631, 1993.

[5] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the Association for Computing Machinery*, 44(3):486–504, 1997.

[6] J. Aspnes and O. Waarts. A modular measure of competitive performance for distributed algorithms. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 237–246, 1996.

[7] B. Awerbuch and Y. Azar. Local optimization of global objectives: Competitive distributed deadlock resolution ad resource allocation. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 240–249, 1994.

[8] B. Awerbuch, Y. Azar, and A. Fiat. Packet routing via min-cost circuit routing. In *Proceedings of the Israeli Symposium on the Theory of Computing and Systems*, pages 37–42, 1996.

[9] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 32–40, 1993.

[10] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive routing of virtual circuits with unknown duration. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 321–327, 1994.

[11] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 164–173, 1993.

[12] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosen. Competitive non-preemptive call control. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.

[13] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 412–423, 1994.

[14] Y. Azar, A. Z. Broder, and A. R. Karlin. On-line load balancing. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 218–225, 1992.

[15] Y. Azar and L. Epstein. On-line load balancing of temporary tasks on identical machines. In *Proceedings of the Israeli Symposium on the Theory of Computing and Systems*, pages 119–125, 1997.

[16] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. R. Pruhs, and O. Waarts. On-line load balancing of temporary tasks. In *Workshop on Algorithms and Data Structures*, pages 119–130, 1993.

[17] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 203–210, 1992.

[18] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18:221–237, 1995.

[19] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber. Bandwidth allocation with preemption. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 616–625, 1995.

[20] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 51–58, 1992.

[21] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 39–50, 1992.

[22] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 379–386, 1990.

[23] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11:2–14, 1994.

[24] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 373–382, 1987.

[25] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task systems. *Journal of the Association for Computing Machinery*, 39:745–763, 1992.

[26] H.-A. Choi and S. L. Hakimi. Scheduling data transfers in networks with preemptions. In *Proceedings of the Allerton Conference on Communication, Control, and Computing*, pages 720–729, 1985.

[27] H.-A. Choi and S. L. Hakimi. Scheduling file transfers for trees and odd cycles. *SIAM Journal on Computing*, 16:162–168, 1987.

[28] H.-A. Choi and S. L. Hakimi. Data transfers in networks. *Algorithmica*, 3:223–245, 1988.

[29] H.-A. Choi and S. L. Hakimi. Data transfers in networks with tranceivers. *Networks*, 18:223–251, 1988.

[30] I. Cidon, S. Kutten, Y. Mansour, and D. Peleg. Greedy packet scheduling. In *Proceedings of the 4th International Workshop on Distributed Algorithms*, pages 169–183, 1990.

[31] I. Cidon, S. Kutten, Y. Mansour, and D. Peleg. Greedy packet scheduling. *SIAM Journal on Computing*, 24(1):148–157, 1995.

[32] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers. *SIAM Journal on Computing*, 14:744–780, 1985.

[33] A. Feldmann, B. Maggs, J. Sgall, D. D. Sleator, and A. Tomkins. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, Carnegie Mellon University, 1995.

[34] J. A. Garay and I. S. Gopal. Call preemption in communication networks. In *Proceedings of IEEE INFOCOM*, pages 1043–1050, 1992.

[35] J. A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *Proceedings of the Israeli Symposium on the Theory of Computing and Systems*, pages 285–293, 1993.

[36] R. Gawlick, C. Kalmanek, and K. G. Ramakrishnan. On-line routing for permanent virtual circuits. In *Proceedings of IEEE INFOCOM*, pages 278–288, 1995.

[37] R. Gawlick, A. Kamath, S. Plotkin, and K. G. Ramakrishnan. Routing and admission control in general topology networks. Technical Report STAN-CS-TR-95-1548, Stanford University, 1995.

[38] I. S. Gopal, G. Bongiovanni, M. A. Bonucelli, D. T. Tang, and C. K. Wong. An optimal switching algorithm for multibeam satellite systems with variable bandwidth beams. *IEEE Transactions on Communications*, 30(11):2475–2481, 1982.

[39] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[40] B. Hajek. Link schedules, flows, and the multichromatic index of graphs. In *Proceedings of the Conference on Information Sciences and Systems*, pages 498–502, 1984.

[41] B. Hajek and G. Sasaki. Link scheduling in polynomial time. *IEEE Transactions on Information Theory*, 34(5):910–917, 1988.

[42] J. T. Havill and W. Mao. Tight bounds for a simple online routing algorithm. In preparation.

[43] J. T. Havill and W. Mao. Greedy on-line file transfer routing. In *Proceedings of the IASTED International Conference on Parallel and Distributed Systems*, pages 225–230, 1997.

[44] J. T. Havill, W. Mao, and R. Simha. A lower bound for on-line file transfer routing and scheduling. In *Proceedings of the 31st Annual Conference on Information Sciences and Systems*, pages 936–941, 1997.

[45] A. Kamath, O. Palmon, and S. Plotkin. Routing and admission control in general topology networks with poisson arrivals. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 269–78, 1996.

[46] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 132-140, 1994.

[47] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[48] R. M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? Technical Report TR-92-044, International Computer Science Institute, 1992.

[49] M. Kaufmann, S. Rajasekaran, and J. F. Sibeyn. Matching the bisection bound for routing and sorting on the mesh. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 31–40, 1992.

[50] M. Kaufmann and J. F. Sibeyn. Deterministic routing on circular arrays. In *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, pages 376–383, 1992.

[51] M. Kaufmann and J. F. Sibeyn. Optimal multi-packet routing on the torus. In *Proceedings of the 3rd Scandinavian Workshop on Algorithm Theory*, pages 118–129, 1992.

[52] P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23:466–487, 1994.

[53] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. C. Graves, A. H. G. R. Kan, and P. Zipkin, editors, *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*. North-Holland, 1990.

[54] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, CA, 1992.

[55] F. T. Leighton. Methods for message routing in parallel machines. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 77–96, 1992.

[56] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, 1994.

[57] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling in O(congestion + dilation) steps. *Combinatorica*, 14(2):167–180, 1994.

[58] T. Leighton and B. Maggs. Fast algorithms for finding O(congestion + dilation) packet routing schedules. In *28th Hawaii Conference on System Sciences*, 1995.

[59] T. Leighton, B. Maggs, and S. Rao. Universal packet routing algorithms. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 256–269, 1988.

[60] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 101–111, 1991.

[61] S. Leonardi, A. Marchetti-Spaccamela, A. Presciutti, and A. Rosen. On-line randomized call control revisited. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 323–332, 1998.

[62] R. J. Lipton and A. Tomkins. Online interval scheduling. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 302–311, 1994.

[63] B. M. Maggs and R. K. Sitaraman. Simple algorithms for routing on butterfly networks with bounded queues. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 150–161, 1992.

[64] F. Makedon and A. Symvonis. Optimal algorithms for multipacket routing problems on rings. *Journal of Parallel and Distributed Computing*, 22(1):37–43, 1994.

[65] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for online problems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 322–333, 1988.

[66] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[67] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 165–175, 1991.

[68] Y. Mansour and B. Patt-Shamir. Greedy packet scheduling on shortest paths. *Journal of Algorithms*, 14:449–465, 1993.

[69] W. Mao. Directed file transfer scheduling. In *Proceedings of the 31st ACM Southeast Conference*, pages 199–203, 1993.

[70] W. Mao. A parallel multi-operation scheduling problem with machine order constraints. In *Proceedings of the ACM Symposium on Applied Computing*, pages 473–477, 1997.

[71] W. Mao, R. K. Kincaid, and A. Rifkin. On-line single machine scheduling algorithms. In S. Nash and A. Sofer, editors, *The Impact of Emerging Technologies on Computer Science and Operations Research*, chapter 8, pages 157-173. Kluwer Academic Publishers, 1995.

[72] W. Mao and R. Simha. Routing and scheduling file transfers in packet-switched networks. *Journal of Computing and Information*, 1:559–574, 1994.

[73] W. Mao, R. Simha, and D. Nicol. A general file transfer scheduling problem. TIMS/ORSA Joint Conference, 1993.

[74] S. Nakano, T. Nishizeki, and N. Saito. On the f-coloring of multigraphs. *IEEE Transactions on Circuits and Systems*, 35(3):345–353, 1988.

[75] S.-I. Nakano and T. Nishizeki. Scheduling file transfers under port and channel constraints. *International Journal of Foundations of Computer Science*, 4(2):101–115, 1993.

[76] T. Nishizeki and K. Kashiwagi. On the 1.1 edge-coloring of multigraphs. *SIAM Journal on Discrete Mathematics*, 3(3):391–410, 1990.

[77] R. G. Ogier. A decomposition method for optimal link scheduling. In *Proceedings of the Allerton Conference on Communications, Control, and Computing*, pages 822–823, 1986.

[78] R. Ostrovsky and Y. Rabani. Universal O(congestion + dilation + $\log^{1+\epsilon} n$) local control packet switching algorithms. In *Proceedings of the ACM Symposium on Theory of Computing*, 1997.

[79] S. Phillips and J. Westbrook. Online load balancing and network flow. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 402–411, 1993.

[80] S. Plotkin. Competitive routing of virtual circuits in ATM networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1128–1136, 1995.

[81] Y. Rabani and E. Tardos. Distributed packet switching in arbitrary networks. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 366–375, 1996.

[82] P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe. Scheduling data redistribution in distributed databases. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 166–173, 1990.

[83] P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe. Scheduling file transfers in fully connected networks. *Networks*, 22:563–588, 1992.

[84] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the Association for Computing Machinery*, 37(2):318–334, 1990.

[85] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines on-line. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 131–140, 1991.

[86] D. B. Shmoys, J. Wein, and D. P. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995.

[87] J. F. Sibeyn. Routing and sorting on circular arrays. Technical Report MPI-I-93-138, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1993.

[88] J. F. Sibeyn. Deterministic routing and sorting on rings. In *Proceedings of the IEEE International Parallel Processing Symposium*, pages 406–410, 1994.

[89] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the Association for Computing Machinery*, 28(2):202–208, 1985.

[90] A. Symvonis. Optimal algorithms for packet routing on trees. In *Proceedings of the International Conference on Computing and Information*, pages 144–161, 1994.

[91] A. Symvonis. Routing on trees. *Information Processing Letters*, 57(4):215–223, 1996.

[92] A. Symvonis and J. Tidswell. An empirical study of off-line permutation packet routing on 2-dimensional meshes based on the multistage routing method. *IEEE Transactions on Computers*, 45(5):619–625, 1996.

[93] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[94] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 263–277, 1981.

[95] R. Varadarajan and P. I. Rivera-Vega. An efficient approximation algorithm for the file redistribution scheduling problem in fully connected networks. Technical Report 92-020, University of Florida, 1992.

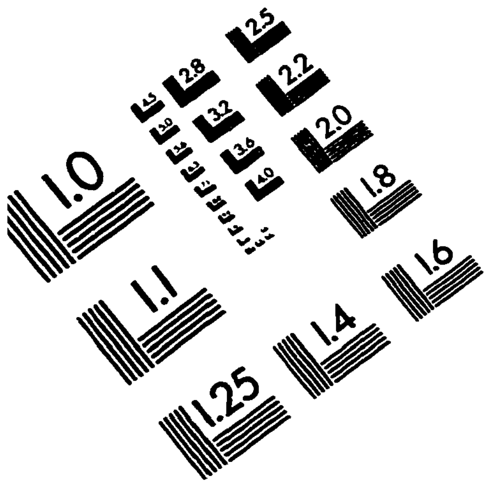
[96] J. Westbrook. Load balancing for response time. In *Proceedings of the European Symposium on Algorithms*, pages 355–368, 1995.

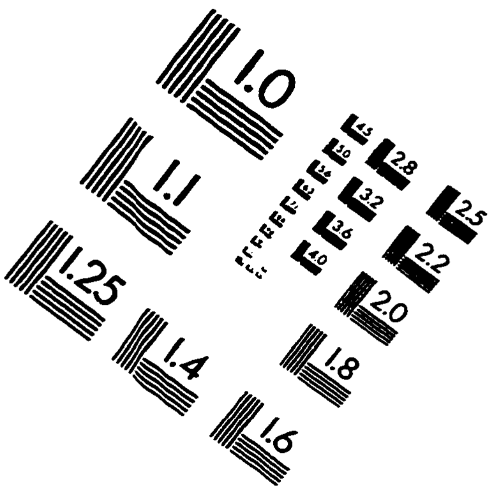
[97] J. Whitehead. The complexity of file transfer scheduling with forwarding. *SIAM Journal on Computing*, 19(2):222–245, 1990.

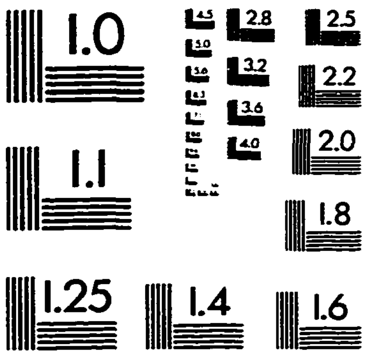
VITA

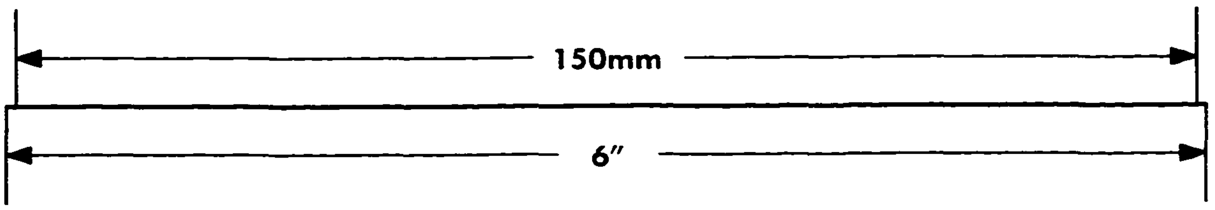
Jessen Tait Havill was born in Madison, Wisconsin on March 6, 1970. He graduated from Shepaug Valley High School in Washington, Connecticut in 1988 and from Bucknell University in Lewisburg, Pennsylvania in 1992 with a Bachelor of Arts degree, majoring in both Computer Science and Religion. Later that year, he matriculated in the Computer Science program at The College of William and Mary in Williamsburg, Virginia as a Teaching Assistant. In 1994, he received the Master of Science degree in Computer Science from The College of William and Mary, and entered the doctoral program at the same institution. In the Fall of 1998, the author will assume the rank of Assistant Professor in the Department of Mathematics and Computer Science at Denison University in Granville, Ohio.

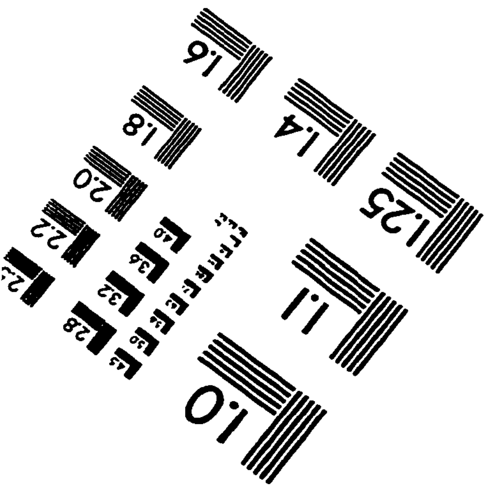
IMAGE EVALUATION TEST TARGET (QA-3)

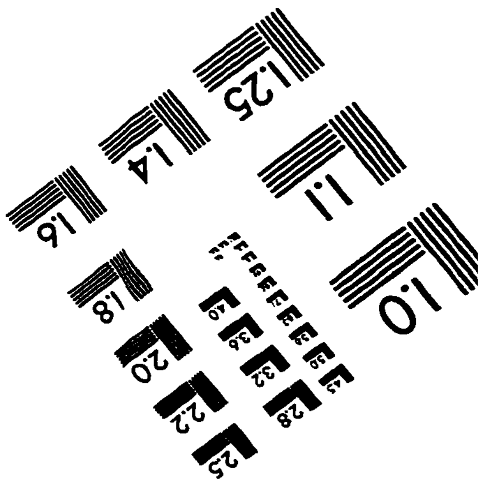












APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989