A LOWER BOUND FOR ON-LINE FILE TRANSFER ROUTING AND SCHEDULING*

Jessen T. Havill, Weizhen Mao, and Rahul Simha
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187-8795

Abstract

In this paper, we study the On-Line File Transfer Routing and Scheduling problem. Given a sequence of file transfer requests and a graph that represents a network, the problem is to determine both a route and schedule for each file transfer in the sequence so as to minimize a suitable objective function. We require that an algorithm be on-line in the sense that it must respond to each request in the order given and before future requests are known. We show that there is no on-line algorithm which produces solutions with network congestion smaller than $\lfloor \log k \rfloor + 1$ times that of the optimal solution or makespan smaller than $\frac{\lfloor \log k \rfloor}{2} + 1$ times that of the optimal solution, where k is the number of file transfer requests. We explain that this bound also holds for randomized on-line algorithms versus an adaptive on-line adversary. We discuss briefly the performance of several greedy on-line algorithms both theoretically and in practice.

1 Introduction and Motivation

We consider the problem of computing routes and schedules for the transfer of files across a network. The problem originated from the work of Coffman, Garey, Johnson and LaPaugh [6], and others [2, 3, 4, 5, 15, 9], who studied how to schedule file transfers over direct connections while obeying port constraints at the nodes. Coffman, et al. [6] showed that, in general, this problem is NP-hard while some special cases have polynomial time solutions. Subsequently, the problem with routing was studied and shown to be NP-hard even for very simple cases with unit file sizes [10, 11, 13, 14]. Mao and Simha [10] proposed three greedy list scheduling heuristics and showed that even the best of them has an approximation guarantee of at least $\Omega(\sqrt{k})$, where k is the number of file transfer requests.

These past approaches to the file transfer routing and scheduling problem have been *off-line*: it is assumed that all file transfer requests are known *a priori* to the scheduling algorithm. However, in practice, file transfer requests are typically scattered in time and are rarely made together. Moreover, at any given time, nothing is known about future requests. To handle these more realistic situations, we consider the *on-line* version of the problem. The efficiency of an on-line algorithm is measured by its *competitive ratio* (see [8]), defined to be the ratio of its performance to that of an optimal off-line algorithm. To make such a comparison we can use any one of several metrics. The most commonly used metric is the maximum completion time of the file transfers, also called the *makespan*; another metric of interest is a measure of link congestion in the network. We show that the competitive ratio of *any* on-line deterministic algorithm is at least $\frac{\lfloor \log k \rfloor}{2} + 1$ with respect to makespan and $\lfloor \log k \rfloor + 1$ with respect to congestion. These lower bounds hold even when link speeds are identical, file sizes are identical and the network graph is a simple 3-layer digraph. In addition, the results hold for randomized on-line algorithms against an adaptive on-line adversary, which may issue requests based on the algorithm's coin tosses but must make its own decisions on-line after each decision of the algorithm. A similar lower bound, applicable to our problem with respect to network congestion, was given by Aspnes, et al. [1] in the context of on-line virtual circuit routing. They prove that the competitive ratio of any algorithm is at least $\frac{\log(k+1)}{2}$. Our lower bound proof uses a different technique and achieves a result that is tighter by a factor of 2. The problem of finding a fast (and simple) on-line algorithm that achieves the lower bound remains an interesting open problem. We provide some evidence of how hard the problem is: standard greedy on-line algorithms that are used in many scheduling problems can be shown to perform very poorly in the file transfer routing and scheduling problem.

While the individual problems of routing and scheduling have long received considerable attention in the literature, the problem of jointly devising routes and schedules has received less. Efficient management of file transfers is one application which requires consideration of both routing and scheduling simultaneously. It is motivated by data transfer applications which require the transfers of large files across networks, such as remote system backups, data distribution in databases, stored video transfers, and data acquisition and distribution in scientific applications. The traditional approach to managing data transfers in these applications is to break up the files into packets and use standard packet-based network services to send the packets across. To send packets across, it is naturally inefficient to collect global information and schedule the transmissions of individual packets. However, large files are a different matter. The time needed to collect limited global information and compute a schedule is a small fraction of the time spent in transferring the files. The benefits can be significant, since the traditional packet routing method cannot (for lack of global information) prevent overlapping file transfers on links, whereas a centralized routing and scheduling method can carefully spread out file transfers to reduce overall completion times. Note that it is possible to implement centralized scheduling in a partially-distributed manner by using a number of "scheduling servers" as we discuss below.

The practicability of file transfer scheduling is not as modest as it may first appear. The following observations

*This research is supported in part by NSF grant NCR-9505963.

strengthen our belief that a distributed scheduling mechanism could significantly pay off in terms of efficient bandwidth usage. First, as noted above, there is the volume of applications. While it is often pointed out that video traffic will dominate future networks, there is no evidence to assume all such traffic will be real-time; transfers of multimedia (including video data) documents are just as likely to take place. Furthermore, the scheduling of file transfers will impact available bandwidth for real-time "streams". Second, it appears fairly straightforward to the authors to implement some form of distributed scheduling using today's file transfer software. For example, a simple switch on the internet `ftp` protocol would indicate that the user is going to transfer large files from a certain directory. The local `ftp` program would then know the approximate size of the data and would be able to first transmit this information to designated "scheduling servers", which would then use available global information in determining a route and schedule. This extra work could be done in time on the order of a few hundred milliseconds, while presumably the transfer itself would take several minutes or longer. The "servers" could be patterned along the lines of similar servers used for multicasting [12].

We organize the rest of this paper as follows. In Section 2, we give a formal definition of our problem. In Section 3, we prove the lower bounds on the competitive ratio with respect to both the makespan and congestion metrics. In Section 4, we discuss on-line algorithms for the problem, all based on the popular greedy method. Finally in Section 5, we summarize our results.

2 Problem Description

Our problem of interest, the *On-Line File Transfer Routing and Scheduling* problem, is defined by two components: a sequence of file transfer requests and a network represented as a weighted graph. To formally describe the problem, we first introduce some notation.

A file transfer request is denoted by

$$f_i = (s_i, t_i, l_i, a_i)$$

where s_i is the source node in the network from which the file transfer is originated, t_i is the destination node in the network to which the file must be sent, l_i is the length (size) of the file, and a_i is the time at which the request originates. The simplest case for the file transfer requests occurs when $l_i = 1$ and $a_i = 0$ for $i = 1, 2, \ldots, k$, which implies that unit-length files, available at time 0, are to be transferred in the network. (We require that requests with the same arrival time still be handled on-line in the order they appear in the request sequence.) We denote such requests simply by (s_i, t_i). The lower bound is shown to hold even with these simplifying assumptions.

Let $G = (V, E, s, m)$ be the (directed) graph that represents the network, where V is the set of nodes (processors) and E is the set of edges (links). The function $s : E \to \{1, 2, \ldots\}$ describes the speed of the links: file transfer request f_i requires $l_i/s(e)$ time units to traverse edge e. The function $m : V \to \{0, 1, \ldots\}$ denotes the buffer sizes at the nodes. When $m(v) = 0$, a file has to be sent away immediately after it arrives at v since there is no buffer space available to hold the file temporarily. At any time, the total length (size) of the files held at v should not exceed $m(v)$. However, there is no restriction on the sizes of files in transit. The lower bound is shown to hold for the simple version in which $s(e) = 1$ for all $e \in E$ and $m(v) = \infty$ for all $v \in V$, i.e. links have unit speed and buffer sizes are so large that the memory constraint at each node can be ignored.

We assume in this paper that at any time only one file can be transferred via a link. In practice, however, a link resource is packet-switched and hence, any number of files may be using a link simultaneously, each rotating the use of the link on a packet-by-packet basis. In this case, although it seems that multiple files can share a single link, a link is still actually used sequentially, one packet at a time. Therefore, our assumption of the exclusive usage of a link by a file at any moment is a reasonable one.

The On-Line File Transfer Routing and Scheduling problem is to assign a route (path), denoted P_i, to each file transfer request f_i and also construct a schedule (timetable) for each file transfer request to minimize a suitable objective function (see below). The problem is on-line in the sense that file transfer requests must be processed in the order the requests appear in the sequence given. Suppose the request sequence is $\sigma = (f_1, f_2, \ldots, f_k)$. In processing f_i, an on-line algorithm has no knowledge about $f_{i+1}, f_{i+2}, \ldots, f_k$.

The first objective function of interest is the makespan, which is defined to be the maximum completion time over all file transfers, i.e.

$$\max_i \{C_i\}$$

where C_i is the time file transfer f_i reaches its destination. The makespan gives the total elapsed time to transfer all k files through the network. The second objective function we discuss is the congestion, defined to be the maximum delay over all links in the network, i.e.

$$\max_{e \in E} \sum_{i : e \in P_i} \frac{l_i}{s(e)}.$$

Notice that when $l_i = 1$, for all $i = 1, 2, \ldots, k$, and $s(e) = 1$, for all $e \in E$, congestion is equivalent to the maximum link usage in the network.

3 A Lower Bound

3.1 Analysis of on-line algorithms

Competitive analysis is commonly used to judge the quality of solutions produced by an on-line algorithm. Let A be the on-line algorithm under consideration and OPT be the optimal algorithm for the same problem. Furthermore, let $C_A(I)$ and $C_{OPT}(I)$ be the values of the objective function (makespan or congestion) when algorithms A and OPT, respectively, are applied to instance I. If there are c (a constant or a function of the size of I) and a (a constant), such that

$$C_A(I) \leq c \cdot C_{OPT}(I) + a$$

for any I, then we say that on-line algorithm A is c-competitive. The c-competitiveness of an algorithm indicates that in the worst case the value of the objective function of a solution produced by the on-line algorithm is at most c times that of the optimal solution. The competitive ratio of an algorithm is defined to be

$$\sup_I \frac{C_A(I)}{C_{OPT}(I)}.$$

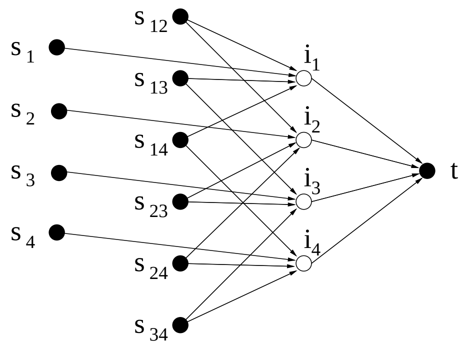

Figure 1: Network graph G_2 ($k = 4$).

A lower bound c for a (minimization) problem is established when one proves that there is no on-line algorithm with competitive ratio better than c. To do so, an adversary strategy is often used. For our problem, consider an arbitrary on-line algorithm A which has complete knowledge of the network. Algorithm A, however, does not know the file transfer request sequence σ in advance; instead, it relies on an adversary to provide the requests one at a time. That is, algorithm A will be initially informed of the first file transfer, say, f_1 with source s_1 and destination t_1. After the algorithm assigns a route and designs a schedule for f_1, the adversary then supplies it with the next file transfer request. What request the adversary supplies depends on the routes and schedules A chooses for the previous requests. The strategy of the adversary is to force A to make bad choices, thus increasing the value of the objective function of the solution A produces. This is the basic idea in the proof we shall present. Throughout this section, we assume that all file transfers have unit length, all links in the network have unit speed and all buffers have infinite capacity, i.e. $l_i = 1$ for $i = 1, 2, \ldots, k$, $s(e) = 1$ for all $e \in E$, and $m(v) = \infty$ for all $v \in V$. Clearly, the lower bound established for such a simple case also applies to more general cases where file lengths and link speeds are arbitrary.

3.2 The network graph

The network graph we use in the lower bound proof can be defined in two equivalent ways; we will describe both. Assume the adversary chooses k to be 2^j for any integer $j > 0$.

The network graph is a 3-layer digraph.[1] Let $S \cup I \cup T$ be the vertex set where

$$S = \{s_{ab} : 1 \le a < b \le k\} \cup \{s_a : 1 \le a \le k\}$$

is the set of source nodes,

$$I = \{i_1, i_2, \ldots, i_k\}$$

is the set of intermediate nodes, and

$$T = \{t\}$$

is the set containing a single destination node. Clearly,

$$n = |S \cup I \cup T| = \frac{1}{2}k(k-1) + 2k + 1 = \frac{1}{2}k^2 + \frac{3}{2}k + 1.$$

[1] A 3-layer digraph is a generalization of a bipartite digraph, which is also a 2-layer digraph, in which the set of nodes V can be partitioned into 3 sets S, I, and T such that, for all directed edges $e = (u, v) \in E$, either $u \in S$ and $v \in I$ or $u \in I$ and $v \in T$.

Let $E_1 \cup E_2$ be the directed edge set where

$$E_1 = \{(s_{ab}, i_a), (s_{ab}, i_b) : 1 \le a < b \le k\} \cup \{(s_a, i_a) : 1 \le a \le k\}$$

is the set of edges between S and I, and

$$E_2 = \{(i_a, t) : 1 \le a \le k\}$$

is the set of edges between I and T. Clearly,

$$m = |E_1 \cup E_2| = k(k-1) + 2k = k^2 + k.$$

For any $j > 0$, we will call the associated graph G_j. For example, Figure 1 shows the network graph G_2.

We can also define the network graph recursively; the recursive definition is more convenient to use in the proof but more difficult to visualize. Let G_0 be a graph with three nodes — a source node s, an intermediate node i, and a destination node t — and two edges (s, i) and (i, t). For any $j > 0$, G_j is constructed recursively as follows: First, take two G_{j-1}'s; second, for any two intermediate nodes x and y, one in each G_{j-1}, add a source node z and edges (z, x) and (z, y); third, contract the two destination nodes in the G_{j-1}'s into one destination node t; fourth, label the intermediate nodes and source nodes appropriately. We label a source node s_{ab}, with $a < b$, if it is connected to intermediate nodes i_a and i_b, and s_a if it is only connected to intermediate node i_a. Figure 2 shows G_0, G_1 and G_2.

We observe that there are two routes a file transfer can take between source s_{ab} and destination t: via intermediate nodes i_a or i_b. Which of the two is chosen depends completely on the particular on-line algorithm. Since the adversary is going to issue its requests based on earlier choices made by the on-line algorithm and we want the proof to hold for any algorithm, we will need notation to allow for arbitrary choices. We define the operator '$|$' as follows: $a|b$ returns a if the on-line algorithm selected a when it had the choice between a or b; $a|b$ returns b otherwise. We define a complementary operator '$\bar{|}$' as follows: $a\bar{|}b$ returns a if $a|b$ returns b, and it returns b if $a|b$ returns a. Hence, for file transfer (s_{ab}, t), an on-line algorithm will assign route $\langle s_{ab}, i_{a|b}, t \rangle$. Note that $a|b$ is different from "a or b" in that once the algorithm makes its decision, the value returned from $a|b$ is fixed.

3.3 The request sequence

The request sequence, provided by the adversary, consists of a recursively defined sequence followed by one last request. Before giving the formal definition of the request sequence, we consider a few examples of the recursively defined sequence. If the network graph is G_1, the sequence, denoted by σ_1, trivially contains just one request, (s_{12}, t). Responding to this request, the algorithm will assign route $\langle s_{12}, i_{1|2}, t \rangle$. If the network graph is G_2, the sequence, denoted by σ_2, will contain two σ_1 sequences, with nodes labeled accordingly, plus one more file transfer request. To be specific, the first request in σ_2 is (s_{12}, t) and the route assigned is $\langle s_{12}, i_{1|2}, t \rangle$. The second request is (s_{34}, t) and the route assigned by the algorithm is $\langle s_{34}, i_{3|4}, t \rangle$. So far, each link in G_2 is used at most once. The third request given by the adversary will be $(s_{(1|2)(3|4)}, t)$. This way the algorithm will have no choice but to select a link that has been used before, forcing the maximum link usage to reach 2, and thus increasing the makespan and congestion. Similarly, the sequence σ_3 for G_3 contains the first 7 requests in Table 1.

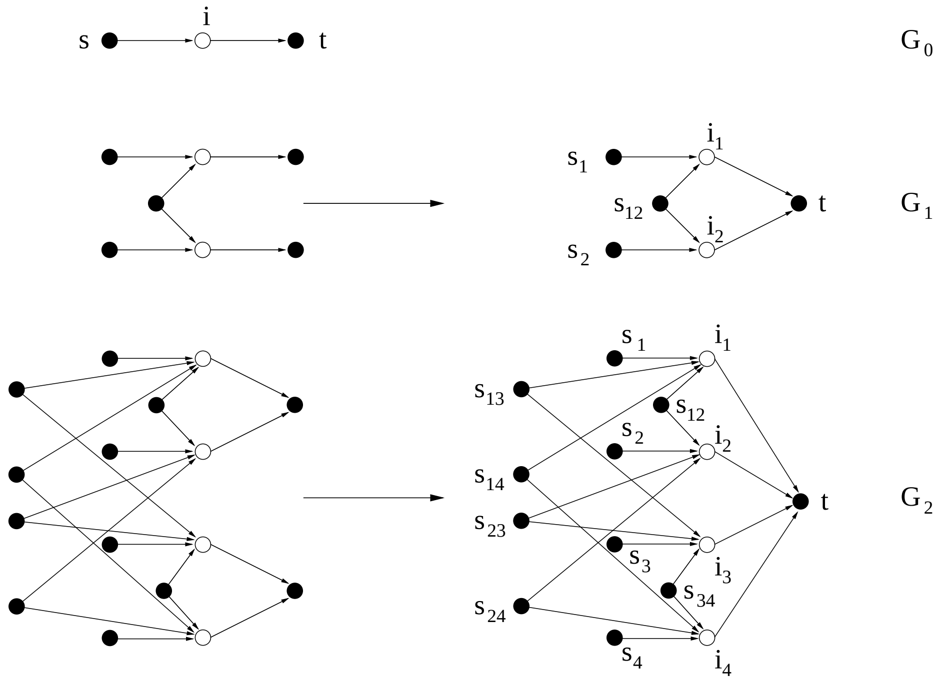

Figure 2: Network graphs G_0, G_1 and G_2.

σ'_3	File transfers	Route assigned
f_1	(s_{12}, t)	$\langle s_{12}, i_{1\|2}, t\rangle$
f_2	(s_{34}, t)	$\langle s_{34}, i_{3\|4}, t\rangle$
f_3	$(s_{(1\|2)(3\|4)}, t)$	$\langle s_{(1\|2)(3\|4)}, i_{(1\|2)\|(3\|4)}, t\rangle$
f_4	(s_{56}, t)	$\langle s_{56}, i_{5\|6}, t\rangle$
f_5	(s_{78}, t)	$\langle s_{78}, i_{7\|8}, t\rangle$
f_6	$(s_{(5\|6)(7\|8)}, t)$	$\langle s_{(5\|6)(7\|8)}, i_{(5\|6)\|(7\|8)}, t\rangle$
f_7	$(s_{((1\|2)\|(3\|4))((5\|6)\|(7\|8))}, t)$	$\langle s_{((1\|2)\|(3\|4))((5\|6)\|(7\|8))}, i_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, t\rangle$
f_8	$(s_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, t)$	$\langle s_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, i_{((1\|2)\|(3\|4))\|((5\|6)\|(7\|8))}, t\rangle$

Table 1: Definition of σ'_3 and the routes assigned by an arbitrary on-line algorithm.

In general, to define σ_j for graph G_j, we take two σ_{j-1}'s with labels of the sources modified accordingly. Let s_{ab} and s_{cd} be the source nodes of the last file transfer requests in each of the two σ_{j-1} sequences. We then add a last file transfer to σ_j from source $s_{(a|b)(c|d)}$ to destination t.

The final request sequence we will give an algorithm will be denoted σ'_j. Let s_{yz} be the source node of the last file transfer in σ_j. Then the request sequence σ'_j is defined to be σ_j followed by the single request $(s_{y|z}, t)$. Notice that request sequence σ'_j contains 2^j requests. The requests in σ'_3 and corresponding routes are given in Table 1.

3.4 Proof of the lower bounds

In this subsection we will examine the makespan and congestion of a solution produced by an arbitrary on-line algorithm and those of the optimal solution, using σ'_j as the request sequence and G_j as the network graph. As a result, we establish a lower bound for our problem.

Let us first consider the example of routing and scheduling σ'_3 in network G_3. Since $j = 3$, there are $k = 2^3 = 8$ files to transfer. As shown in Table 1, f_1 and f_2 use parallel routes $\langle s_{12}, i_{1|2}, t\rangle$ and $\langle s_{34}, i_{3|4}, t\rangle$, respectively, causing the makespan to be 2 (2 hops for the files to reach v) and the congestion to be 1. Request f_3 is next. We observe that *any* on-line algorithm must choose a route for f_3 that passes through a previously used link. This forces the makespan to reach 3 (since a link can only be used by one file transfer at any moment), and the congestion to reach 2. Similarly, the transfer of f_4, f_5 and f_6, which are file transfers in the lower half of the network, results in the same makespan and congestion, thus keeping the overall makespan and congestion unchanged (3 and 2, respectively). When f_7 arrives, because of the way σ'_3 and G_3 are designed, the algorithm has no choice but to

σ_3'	File transfers	Optimal route assigned
f_1	(s_{12}, t)	$\left\langle s_{12}, i_{1\bar{\mid}2}, t \right\rangle$
f_2	(s_{34}, t)	$\left\langle s_{34}, i_{3\bar{\mid}4}, t \right\rangle$
f_3	$\left(s_{(1\mid 2)(3\mid 4)}, t\right)$	$\left\langle s_{(1\mid 2)(3\mid 4)}, i_{(1\mid 2)\bar{\mid}(3\mid 4)}, t \right\rangle$
f_4	(s_{56}, t)	$\left\langle s_{56}, i_{5\bar{\mid}6}, t \right\rangle$
f_5	(s_{78}, t)	$\left\langle s_{78}, i_{7\bar{\mid}8}, t \right\rangle$
f_6	$\left(s_{(5\mid 6)(7\mid 8)}, t\right)$	$\left\langle s_{(5\mid 6)(7\mid 8)}, i_{(5\mid 6)\bar{\mid}(7\mid 8)}, t \right\rangle$
f_7	$\left(s_{((1\mid 2)\mid(3\mid 4))((5\mid 6)\mid(7\mid 8))}, t\right)$	$\left\langle s_{((1\mid 2)\mid(3\mid 4))((5\mid 6)\mid(7\mid 8))}, i_{((1\mid 2)\mid(3\mid 4))\bar{\mid}((5\mid 6)\mid(7\mid 8))}, t \right\rangle$
f_8	$\left(s_{((1\mid 2)\mid(3\mid 4))\mid((5\mid 6)\mid(7\mid 8))}, t\right)$	$\left\langle s_{((1\mid 2)\mid(3\mid 4))\mid((5\mid 6)\mid(7\mid 8))}, i_{((1\mid 2)\mid(3\mid 4))\mid((5\mid 6)\mid(7\mid 8))}, t \right\rangle$

Table 2: Optimal routes assigned to requests in σ_3'.

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 the algorithm must assign it to its only route, which passes over the most heavily used link in the graph, causing the makespan and congestion to increase to 5 and 4, respectively. In general, we have the following lemma:

LEMMA 1 *Let j be any positive integer. When given network graph G_j and request sequence σ_j' of $k = 2^j$ file transfers,* any *on-line algorithm will construct a schedule with makespan at least $j+2 = \log k + 2$ and congestion at least $j+1 = \log k + 1$.*

PROOF By induction on j. ∎

Next we show that, using the same instance, the optimal makespan and congestion are 2 and 1, respectively. Different from an on-line algorithm, the optimal algorithm is able to see the entire request sequence σ_j' in advance and thus avoids the mistakes made by an on-line algorithm.

Let us again consider the example of transferring the 8 files in σ_3' on network G_3. The optimal routes chosen by the optimal algorithm are given in Table 2. In this route assignment, no single link will be used more than once, thus resulting in the optimal makespan 2 (2 hops for any file to reach its destination) and the optimal congestion 1.

Now consider the situation in general. We recall that $a|b$ refers to a choice of a and b made by the on-line algorithm, and that $a\bar{|}b$ refers to the choice among a and b *not* made by the on-line algorithm. Because of the way the network graph is constructed, for each file transfer there are exactly two routes. If the on-line algorithm chooses one route, denoted by $\langle s_{ab}, i_{a|b}, t\rangle$, the optimal algorithm will always choose the other route, denoted by $\left\langle s_{ab}, i_{a\bar{|}b}, t\right\rangle$. When the last file transfer arrives, the edges on its only path are unused. Therefore, the optimal schedule uses no edge more than once. In general, we have the following lemma:

LEMMA 2 *Let j be any positive integer. When given network graph G_j and request sequence σ_j', an optimal off-line algorithm will construct a schedule with makespan 2 and congestion 1.*

PROOF By induction on j. ∎

The above two lemmas imply the following lower bound theorem for the On-Line File Transfer Routing and Scheduling problem.

THEOREM 1 *For the On-line File Transfer Routing and Scheduling problem, even when all files have unit length and share one destination, and the network is a simple 3-layer graph with unit-speed links, there is no on-line algorithm with competitive ratio better than $\frac{\lfloor \log k \rfloor}{2} + 1$ with respect to makespan and $\lfloor \log k \rfloor + 1$ with respect to congestion, where k is the number of file transfers.*

It is worth mentioning that these lower bounds also hold for any randomized on-line algorithm against an adaptive on-line adversary. An adaptive on-line 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. In this case the randomized on-line algorithm under consideration chooses randomly one route for the current file transfer request while the adversary defines the next request adaptively based on the route selected by the on-line algorithm. The proof works for the randomized algorithms with little change needed. Thus we also have the following theorem:

THEOREM 2 *For the On-line File Transfer Routing and Scheduling problem, even when all files have unit length and share one destination, and the network is a simple 3-layer graph with unit-speed links, there is no randomized on-line algorithm with competitive ratio better than $\frac{\lfloor \log k \rfloor}{2} + 1$ with respect to makespan and $\lfloor \log k \rfloor + 1$ with respect to congestion against an adaptive on-line adversary, where k is the number of file transfers.*

Lastly, we point out that our lower bounds can also be written in terms of n, the number of nodes in the network. This formulation may make more sense in many situations. Specifically, we can say that any on-line algorithm will have competitive ratio at least $\left\lceil \frac{\log n + 4}{4} \right\rceil$ with respect to makespan and $\left\lceil \frac{\log n + 2}{2} \right\rceil$ with respect to congestion.

4 Greedy On-line Algorithms

The results from the previous section raise the natural question: is there an on-line algorithm that achieves the lower bound? In other words, is there an on-line algorithm that produces a solution with makespan or congestion that is $O(\log k)$ times that of an optimal solution? Consider the following greedy on-line algorithms:

- Algorithm 1 ($A1$): For file transfer request f_i, assign any route P between s_i and t_i in the network graph, and then schedule f_i along the route in the most efficient way.
- Algorithm 2 ($A2$): For file transfer request f_i, assign a route P between s_i and t_i with the fewest links that have been used before, and then schedule f_i along the route in the most efficient way.
- Algorithm 3 ($A3$): For file transfer request f_i, assign a route P between s_i and t_i with the smallest increase in congestion over all edges, and then schedule f_i along the route in the most efficient way.
- Algorithm 4 ($A4$): For file transfer request f_i, assign a route P between s_i and t_i which results in the minimum value of $\max_{e\in P} \sum_{j\leq i: e\in P_j} \frac{l_j}{s(e)}$, and then schedule f_i along the route in the most efficient way.

By "the most efficient way" we mean that a file stays at an intermediate node only when the link it chooses to use in the next time step is busy transferring a file with a smaller index.

In [10], the authors prove that the competitive ratio, with respect to both makespan or congestion, of $A1$ and $A2$ is at least $\Omega(k)$, and the competitive ratio of $A3$ is at least $\Omega(\sqrt{k})$. This implies that algorithms $A1$, $A2$, and $A3$ all fail to achieve the lower bound proved in Section 3. Our simulation experiments show that in practice both $A2$ and $A3$ outperform $A1$ by a large margin while $A3$ constructs file transfer schedules just a little better than $A2$ does. As for $A4$, a preliminary study [7] shows that the algorithm achieves a $O(\log k)$ competitive ratio for at least a small class of request sequences and network graphs. Determining the competitive ratio for $A4$ is one of many open problems for our future research.

5 Conclusions

In this paper, we study the On-Line File Transfer Routing and Scheduling problem. In particular, we establish a lower bound on the competitive ratio of $\frac{\lfloor \log k \rfloor}{2} + 1$ with respect to makespan and $\lfloor \log k \rfloor + 1$ with respect to network congestion. Alternatively, in terms of n, we can say that the competitive ratio of any algorithm is at least $\lceil \frac{\log n + 4}{4} \rceil$ with respect to makespan and $\lceil \frac{\log n + 2}{2} \rceil$ with respect to congestion. By considering greedy on-line algorithms and their performance, we have argued that the problem of finding a good on-line algorithm is a hard one. File Transfer Routing and Scheduling is an important problem in computer communications and networks. It is also applicable in other areas, such as manufacturing. By treating nodes in the network graph as machines and file transfers as jobs, the file transfer problem can be formulated as a a parallel job shop scheduling problem. For the future, there remains the open problem of designing a simple on-line algorithm that achieves the lower bound, as well as considering distributed scheduling algorithms.

References

[1] 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 *Proc. ACM Symposium on Theory of Computing*, pages 623–631, 1993.

[2] H.-A. Choi. *Scheduling Data Transfers in Networks*. PhD thesis, Northwestern University, 1985.

[3] H.-A. Choi and S. L. Hakimi. Scheduling file transfers for trees and odd cycles. *SIAM Journal on Computing*, 16:162–168, 1987.

[4] H.-A. Choi and S. L. Hakimi. Data transfers in networks. *Algorithmica*, 3:223–245, 1988.

[5] H.-A. Choi and S. L. Hakimi. Data transfers in networks with tranceivers. *Networks*, 18:223–251, 1988.

[6] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. LaPaugh. Scheduling file transfers. *SIAM Journal on Computing*, 14(3):744–780, 1985.

[7] J. T. Havill and W. Mao. Greedy on-line file transfer routing and scheduling on layered networks. Submitted for publication.

[8] R. M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? Technical Report TR-92-044, ICSI, Berkeley, CA, 1992.

[9] W. Mao. Directed file transfer scheduling. In *Proceedings of the 31st ACM Southeast Conference*, pages 199–203, 1993.

[10] W. Mao and R. Simha. Routing and scheduling file transfers in packet-switched networks. *Journal of Computing and Information*, 1:559–574, 1994.

[11] W. Mao, R. Simha, and D. Nicol. A general file transfer scheduling problem. In *TIMS/ORSA Joint Conference*, Chicago, IL, May 1993.

[12] J. Moy. Multicast routing extensions for OSPF. *Communications of the ACM*, 37(8):61–66, Aug. 1994.

[13] P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe. Scheduling data redistribution in distributed databases. In *Proc. 6th International Conference on Data Engineering*, pages 166–173. IEEE Computer Society, Feb. 1990.

[14] P. I. Rivera-Vega, R. Varadarajan, and S. B. Navathe. Scheduling file transfers in fully connected networks. *Networks*, 22:563–588, 1992.

[15] J. Whitehead. The complexity of file transfer scheduling with forwarding. *SIAM Journal on Computing*, 19(2):222–245, Apr. 1990.