Scheduling Data Transfers in a Network and the Set Scheduling Problem*

Ashish Goel
University of Southern California[†]

Monika R. Henzinger
Google Inc.[‡]

Serge Plotkin
Stanford University[§]

Eva Tardos
Cornell University[¶]

December 28, 2000

Abstract

In this paper we consider the *online ftp problem.* The goal is to service a sequence of file transfer requests given bandwidth constraints of the underlying communication network. The main result of the paper is a technique that leads to algorithms that optimize several natural metrics, such as max-stretch, total flow time, max flow time, and total completion time. In particular, we show how to achieve optimum total flow time and optimum max-stretch if we increase the capacity of the underlying network by a logarithmic factor.

We show that the resource augmentation is necessary by proving polynomial lower bounds on the max-stretch and total flow time for the case where online and offline algorithms are using same-capacity edges. Moreover, we also give poly-logarithmic lower bounds on the resource augmentation factor necessary in order to keep the total flow time and max-stretch within a constant factor of optimum.

---

*A preliminary version of these results appeared as [14].

†Ashish Goel was at Stanford University when this research was done, and was supported by ARO Grant DAAG55-98-1-0170 and ASSERT award DAAG55-97-1-0221. Part of this research was conducted when the author was at Compaq Systems Research Center. EMail: agoel@cs.usc.edu .

‡Monika Henzinger was at the Compaq Systems Research Center, Palo Alto when this research was conducted. EMail: monika@google.com .

§Department of Computer Science, Stanford University. Supported by ARO Grant DAAG55-98-1-0170 and ONR Grant N00014-98-1-0589. EMail: plotkin@cs.stanford.edu .

¶Department of Computer Science, Cornell University. Supported by NSF Grant CCR-9357949 and ONR Grant N00014-98-1-0589. EMail: eva@cs.cornell.edu .

1 Introduction

Consider the problem of sending large files (eg. bitmap images) through a general topology network. The requests arrive online and the goal is to eventually satisfy all the requests. Since the bandwidth of the links in the network is limited, it makes sense to try to schedule the transmissions in a way that utilizes the available resources optimally.

In this paper we consider the *online ftp problem*, which is a formal abstraction of the above file transfer problem. We assume that each ftp request specifies source/destination nodes and the size of the file. The goal of the online algorithm is to choose a path that will be used for transmitting each file, and to decide on the transmission rate. The main difference between this model and the (well-studied) models for online routing and admission control [11, 1, 12, 2] is that here we do not assume that the sources have prespecified transmission rate requirements, i.e. we can deal with non-streaming types of information.

There are two related measures of performance that can be used to compare different algorithms for the online ftp problem. The first measure is the *total flow time*, i.e. the sum over all jobs of the time that elapses between the instant the ftp request is submitted and the time it is satisfied (including the transmission time). The other measure is the *max-stretch*, which is the maximum over all ratios of the flow time of each request and the smallest time needed to satisfy this request. The second quantity is determined by the link bandwidth and the size of the file. Both measures are useful since they are directly related to the performance of the network perceived by the end-user.

Let n be the number of requests and P the maximum ratio between the sizes of the files. Assume that the smallest request can be processed in one time unit. Let F^*_{MAX} denote the optimum max-flow i.e. the smallest value for the maximum time a request spends in the system. The main results of the paper are algorithms that achieve the *optimum max-stretch* and the *optimum total flow time* using resource augmentation[1]. For the max-stretch algorithm we need to increase capacities by a factor of $O(\log P)$, whereas the total flow time algorithm needs a factor of $O(\log F^*_{MAX})$ increased capacity[2]. The latter algorithm does not only achieve the optimum total flow time, but *simultaneously* optimizes many other objective functions, like the maximum flow time, the total square-of-flow-time, etc.

To justify the need for giving larger capacities to the online algorithm (i.e. resource augmentation), we show polynomial lower bounds on both max-stretch and total flow time for the case where both online and offline algorithms are using the same capacities. Moreover, we show that in order to achieve constant competitive ratio against an adaptive adversary we have to give the online algorithm $\Omega(\log P/\log\log P)$ factor more capacity for the max-stretch metric, and $\Omega(\sqrt{\log\gamma/\log\log\gamma})$ more capacity for the total flow metric, where $\gamma = \min\{n, P\}$.

[1]Throughout this paper, when we refer to an optimum solution, we mean the optimum without any resource augmentation.

[2]Note that $F^*_{MAX} \le nP$ and therefore $\log F^*_{MAX} \le \log n + \log P$.

In the context of machine scheduling, total flow time is known to be a hard metric to approximate [18] and it is only recently that progress has been made towards obtaining algorithms that give total flow time guarantees. In particular, logarithmic-factor resource augmentation was used in [20] to obtain optimum flow time for machine scheduling. Unlike the problem we consider, resource augmentation is not necessary to obtain good approximation ratios for minimizing flow time machine scheduling problem; a logarithmic approximation was obtained in [19]. Max-stretch was recently proposed as a good metric to measure user satisfaction [5]. Our lower bound on the amount of resource augmentation needed for max-stretch holds in the machine scheduling model as well, and therefore our upper bounds for max-stretch are also of interest in the machine scheduling model.

When proving upper bounds, we restrict our algorithms to use a single rate when transmitting a specific file, and do not allow preemption. The competitive ratio is computed against an offline algorithm that does not have these restrictions. Our lower bounds for online flow-time minimization algorithm without resource augmentation (i.e. both the online and the offline algorithms work in the same network) hold even if we remove this restriction, i.e. allow the algorithm to use time-varying rate when transmitting a file. This contrasts with minimizing flow time for machine scheduling, where a $\log P$-competitive preemptive algorithm is known [19]. Also, the lower bound for total flow time is achieved using same-size files. This is in contrast to machine scheduling where the unit jobs case is trivial.

We view the online ftp problem as a special case of the *set scheduling problem*. In this problem we have a set of resources and each job requires a specific subset of these resources (or one of a set of subsets). Set scheduling is a natural generalization of the machine scheduling problem that was extensively studied under several different metrics (See [17] for a survey of offline approximation algorithms, and [19, 20, 5, 15, 16, 8, 21] for a sampling of recent results in online algorithms.). The set scheduling model is similar to the parallel jobs model studied by [10, 23]. We show how to apply several techniques developed in the context of machine scheduling to the set scheduling problem (and hence the online ftp problem) for simpler metrics such as makespan and total completion time. In particular, we use the technique that allows us to convert an offline optimization algorithm that maximizes the number of scheduled jobs into an online algorithm that minimizes total completion time [15, 16, 21]. We also develop new techniques that help us attack more difficult metrics such as total flow time and max-stretch.

The techniques developed in this paper can be better understood when compared to the technique of Hall *et al.* [15, 16]. There the approach is to use offline ρ-approximation algorithms for offline packing problems to construct $O(\rho)$-competitive online algorithms for average completion time. Our techniques allow the transformation of offline packing algorithms that achieve the optimum packing using $O(\rho)$ resource augmentation into online algorithms that achieve the optimum flow time using $O(\rho \cdot \log F^*_{MAX})$ resource augmentation. If the online algorithm is not required to work in polynomial time, then an optimum offline solution ($\rho = 1$) can be used. Unlike the work of Hall *et al.* [15, 16] our techniques apply only when jobs are *malleable* [15, 6, 10, 23] i.e. extra capacity/resources can be used to reduce the processing time of jobs. Two such problems are the parallel jobs problem [6, 10, 23] and the vector scheduling problem [13, 7, 4]. Using our techniques we can obtain non-polynomial-time online algorithms for minimizing the total flow time for these problems.

In Section 2 we explain the models that we use. Section 3 contains the main technical contributions of the paper – the lower and upper bounds on the performance of online algorithms using the total flow time and max-stretch metrics. In Section 4 we describe online algorithms for the ftp problem using the makespan and total completion time metrics. Not all online algorithms in Sections 3 and 4 run in polynomial time; polynomial time online algorithms and offline approximation algorithms are discussed in Section 5. Section 5 also sketches an offline, polynomial time algorithm for minimizing the makespan for the set scheduling problem (and hence the online ftp problem) if the rate at which a request is serviced is allowed to vary arbitrarily.

2 Models and Definitions

In the *set scheduling problem* there are n jobs and m resources. Job j has an arrival time (release date) a_j, a processing time p_j, and a resource requirement S_j where S_j is a subset of S, the set of resources. We define $P = \max_j p_j / \min_j p_j$. The quantity P plays a crucial role in the analysis of our algorithms. As in traditional scheduling, both the preemptive and non-preemptive variants are of interest. The Set Scheduling Problem can be formulated as either an offline or an online problem. As in job shop scheduling and multiprocessor scheduling, the performance of an algorithm for this problem can be studied under several different metrics – most notably makespan, total completion time, total flow time, and max-stretch. In this paper we will concentrate mainly on online algorithms.

The *online ftp problem* is defined as follows. We are given a network $G = (V, E)$ where all edges have identical bandwidths. Assume that the transmission delay along any link is zero, and that there are no buffers in the network. Once a source starts transmitting data to another node, the other node starts receiving it immediately. Of course the rate at which the sender transmits the data is bounded by the minimum available bandwidth along the route over which the transmission is taking place. Let m be the number of links in the network, and n the number of ftp requests. Request j has an arrival time a_j, specifies file size p_j, and a route R_j over which the data needs to be transmitted. We also address the case where instead of the route, the request specifies only the source and the sink nodes. The former model is closer to the IP world, where the routes are determined by an external algorithm, while the second model is closer to the ATM world, where one can use source routing.

Let C_j be the completion time of job j in a schedule. The quantity $F_j = C_j - a_j$ is called the flow time of job j. The makespan of a schedule is $\max_j C_j$; total completion time is $\sum_j C_j$; total flow time is $\sum_j F_j$ and max-stretch is $\max_j F_j/\tau_j$ where τ_j is the time it would take to satisfy job j if it had the whole network to itself. We also permit jobs to have weights w_j. In the presence of weights the total completion time and total flow time metrics are defined as $\sum_j w_j C_j$ and $\sum_j w_j F_j$ respectively. Traditionally, the total flow time and max-stretch metrics are considered to be the hardest. These are also perhaps the most interesting metrics as they most directly measure end user experience.

The following theorem captures the hardness of the set scheduling problem – the reduction is straightforward and is sketched in Appendix A for completeness.

Theorem 1 *The Vertex Color problem reduces (via polynomial time reductions) to Minimizing Makespan for Set Scheduling in an approximation preserving fashion.*

For the vertex color problem lower bounds are known for both the approximation ratio ($\Omega(n^{1-\epsilon})$ unless P=NP [9]) and competitive ratio ($\Omega(n^{1/3})$ [3]). The above reduction also holds for the online ftp problem if the routes as well as the transmission rates are given as input. Thus to make progress with the set scheduling/online ftp problems, we need to relax the model. The first relaxation we propose is to allow rate control for jobs. Thus each job would be assigned a start time s_j ($s_j \geq a_j$) and a rate r_j by the scheduler. The job would execute from time s_j to $s_j + p_j/r_j$ and would consume an r_j fraction of each resource in its resource set S_j during this interval. More than one jobs may use a resource at the same time. However, the total usage of a resource at any time must be at most 1. The term *malleable jobs* is commonly used to describe this property [15, 6, 10, 23]. This relaxation is particularly appropriate to the ftp problem: it is possible to control the rate of a TCP connection; more than one connections can use the same link; further, a connection uses up the same bandwidth on each link along its route[3].

3 Flow time and max-stretch using resource augmentation

3.1 Upper bounds with resource augmentation but no preemption

Assume that all links have the same capacity in the original network; rescale capacities so that this capacity becomes 1. Further rescale time such that the smallest request takes four units of time to finish if it has the entire network to itself. Then the time required to service the largest request (if the request has the entire network to itself) is at most $4P$.

Let n be the number of requests, and m the number of links. Let $K = 3 + \log n + \log P$. We assume that the online algorithm can use a capacity of $5K$ on each link.

[3] Instead of allowing a fixed rate r_j for each job, we could also allow the rate to vary. It turns out that our online algorithms, even though they use just one rate r_j for job j, are competitive against optimal solutions which are allowed to vary the rate. For offline algorithms it may help to vary the rate; we will delve into this a little in Section 5

The online algorithm pretends that there are K copies of the network, $G_0 \ldots G_{K-1}$, each with edge capacities 5. We call this algorithm **MRHP** (Most Recent Highest Priority) since at any given time, connections which have been waiting in the system the shortest are the most likely to get scheduled. The online algorithm does its processing only at integral time instants. Figure 1 describes the behavior of MRHP at time t such that $t = 2^k.t'$, where t' is odd.

```
for i = 0 to min{k, K-1}

    1. Let S_i be the set of requests which arrived in the interval [t-2^i, t)
    2. Find the largest weight subset of S_i that can be completed in the
       network G_i between times t and
       t + 2^i            [Note: This step may not run in polynomial time in general]
    3. Schedule this subset in G_i such that each request
       has starting time t, finishing time t+2^i, and a uniform rate during
       this interval.
```

Figure 1: Algorithm MRHP at time $t = 2^k.t'$, where t' is odd.

The same job may get scheduled by multiple copies of the network. The flow time of such a job is taken to be the smallest flow time from all its copies. All the jobs ultimately get scheduled by the online algorithm, as the G_{K-1} alone has sufficient capacity to schedule all the jobs.

Let w_j be the weight of the jth job, and F_j the total time this job spends in the system. Let Q_k be the total weight of the requests which get scheduled in at least one of the networks $G_0 \ldots G_k$. Let Q_k^* be the total weight of all requests that have a flow time of at most 2^{k+2} in the optimum solution. Let $q_k = Q_k - Q_{k-1}$, and $q_k^* = Q_k^* - Q_{k-1}^*$ (for convenience define Q_{-1} and Q_{-1}^* to be 0.). Each job j which contributes to q_k must have a flow time $F_j \leq 2^{k+1}$ in the MRHP schedule, and each job j which contributes to q_k^* must have a flow time $F_j^* \geq 2^{k+1}$ in the optimum schedule.

Let $\mathcal{F}^*$ and $\mathcal{F}$ denote the total weighted flow times of the optimum and online algorithms, respectively. Clearly, $\mathcal{F}^* \geq \sum_{0 \leq k \leq K} 2^{k+1} q_k^*$. Further, $\mathcal{F} \leq \sum_{0 \leq k \leq K-1} 2^{k+1} q_k$.

Lemma 2 $Q_k \geq Q_k^*$

Proof: Let S_k^* be the set of requests which contribute to Q_k^*. By definition, each of these requests has a flow time of at most 2^{k+2}. Divide time into intervals of the form $[i \cdot 2^k, (i+1) \cdot 2^k)$ for $i \geq 0$. Let $S_k^{(i)^*}$ denote the set of requests from S_k^* which arrive during the i-th interval, and let $Q_k^{(i)^*}$ denote their combined weight. All these jobs are scheduled by the optimum algorithm to finish before time $(i+1) \cdot 2^k + 2^{k+2}$. Hence all these jobs must arrive and finish in the interval $[i \cdot 2^k, (i+1) \cdot 2^k + 2^{k+2})$, which has length $5 \cdot 2^k$. Since G_k has 5 times the original capacity on each edge, and since it has all the jobs in $S_k^{(i)^*}$ available for scheduling during the interval $[(i+1) \cdot 2^k, (i+2) \cdot 2^k)$, it will schedule jobs with a weight of at least $Q_k^{(i)^*}$ during this interval. Summing up over all i, $Q_k \geq Q_k^*$. ■

Let g be any function from $\Re^+$ to $\Re^+$. Let $\mathcal{F}_g^*$ denote the optimum value of $\sum_j w_j g(F_j)$ that can be obtained in an unaugmented network, and $\mathcal{F}_g$ denote the corresponding value obtained by MRHP.

Theorem 3 *$\mathcal{F}_g \leq \mathcal{F}_g^*$, for all non-decreasing functions g.*

Proof: Let $W = \sum_j w_j$. We define $P(k) = Q_k/W$ and $P^*(k) = Q_k^*/W$. P and P^* are probability measures, and Lemma 2 implies that P^* stochastically dominates P. Theorem 3 now follows from the properties of stochastic dominance. ■

Theorem 3 is particularly interesting because it shows that MRHP simultaneously optimizes a very wide class of metrics. In particular, the following results can be obtained as corollaries.

Corollary 3.1 *$\mathcal{F} \leq \mathcal{F}^*$.*

Proof: Let g be the identity function in the statement of Theorem 3. ■

Corollary 3.2 *Let F_{MAX} denote the maximum flow time (max-flow) in the schedule obtained by MRHP and F_{MAX}^* denote the max-flow in the optimum schedule. Then $F_{MAX} \leq F_{MAX}^*$*

Proof: For $p > 0$, define $\mathcal{F}_p$ to be $\sum_j w_j (F_j)^p$. $\mathcal{F}_p^*$ is defined analogously. Theorem 3 implies that $\mathcal{F}_p \leq \mathcal{F}_p^*$ for all $p > 0$. F_{MAX} and F_{MAX}^* are the limiting values of $(\mathcal{F}_p)^{1/p}$ and $(\mathcal{F}_p^*)^{1/p}$ respectively as $p \to \infty$. Therefore $F_{MAX} \leq F_{MAX}^*$. ■

The average stretch of a job can be mimicked using a total weighted flow time objective function with appropriate weights. MRHP does not really need to know K in advance – it can maintain an estimate of K and increment this estimate by one if and when the current value of K does not suffice to schedule all the requests. Let F_{MAX}^* be the optimum max-flow for the given sequence of jobs, given that the shortest job takes one unit time to finish if it has the entire network to itself. Notice that $F_{MAX}^* \leq nP$. The following theorem gives a sharper bound on the amount of resource augmentation needed by MRHP.

Theorem 4 *MRHP needs $O(\log F_{MAX}^*)$ resource augmentation. Further, F_{MAX}^* need not be known in advance.*

The above theorem represents a significant improvement, since n can be arbitrarily large even in a well behaved system with small max-flow. Section 5 shows how to implement the algorithm in expected polynomial time with $O(\log n + \log P + \log m)$ resource augmentation.

We now return to the max-flow metric introduced in Corollary 3.2. The max-flow metric (F_{MAX}) is interesting primarily because it relates to the max-stretch metric. We give a simple online algorithm **MMF** (Minimum Max-Flow) that uses only a constant factor resource augmentation.

More specifically, MMF uses at most five times the capacity of the original network. MMF assumes that the optimum max-flow is at least T and at most $2T$ (Initially, T is assumed to be the time required to complete the very first job in the original network.). At times t which are multiples of $T/2$, MMF looks at all requests which arrived during the last $T/2$ time units. It then assigns to each of these jobs a rate which is just sufficient for this job to finish in the next $T/2$ time units. If the load on any edge exceeds five times the capacity of that edge in the original network, MMF doubles T, aborts the current phase, and waits till the current time becomes a multiple of the new value of $T/2$. The following theorem subsumes Corollary 3.2.

Theorem 5 *The maximum flow time of a job in the schedule produced by MMF is no larger than the optimum max-flow. MMF runs in time polynomial in n, m, and $\log P$.*

Proof: Consider all jobs that arrive in the interval $\left(\frac{T}{2}(i-1), \frac{T}{2}i\right]$. Let the optimum max-flow be F^*_{MAX}. All these jobs must finish in the optimal schedule by the time $(T/2)i + F^*_{MAX}$. If $F^*_{MAX} \leq 2T$ then the total requirement of all the jobs that need edge e can be at most $2T + T/2 = 5T/2$ (recall that the capacity of each edge is one). Hence by *stretching* all these jobs over an interval of length $T/2$, no edge can exceed its capacity by more than a factor of 5. If an edge exceeds capacity by more than a factor of 5, we can conclude that $F^*_{MAX} > 2T$, and hence double our estimate of T. ■

We are now ready to present **MMS** (Minimum Max-stretch) which uses $O(\log P)$ resource augmentation and guarantees a max-stretch that is no worse than the optimum max-stretch. We first observe that MMF can be modified to guarantee a max-flow that is at most half the optimum value if the amount of capacity on each edge is ten times that in the original network. Let p_1 be the amount of data transfer required by the first job. MMS bunches incoming requests into (at most $\log P$) classes, with class i containing all requests which have a data requirement in the range $[p_1 \cdot 2^i, p_1 \cdot 2^{i+1})$ (i may be negative as well). There can be at most $2 + \log P$ classes. For requests within class i MMS invokes a separate copy of modified MMF. Thus the resource augmentation needed by MMS is $O(\log P)$. Note that MMS does not need to know P in advance. The fact that the max-flow obtained within each class is at most half the optimum max-flow for that class is sufficient to guarantee that the max-stretch obtained by MMS is no more than the optimum max-stretch. The following theorem summarizes the claims made in the above discussion.

Theorem 6 *MMS uses $O(\log P)$ resource augmentation and obtains a max-stretch that is no more than the optimum max-stretch. Further, MMS does not need to know P. MMS runs in time polynomial in n, m, and $\log P$.*

Note that neither MRHP, nor MMF, nor MMS need to get the transmission routes R_j as input.

Theorem 7 *MRHP, MMF, and MMS can obtain optimum values for their respective metrics even if the routes R_j are not given as input.*

If routes are not provided as input, MRHP, MMF, and MMS as described above would not run in polynomial time. See theorem 15 for the amount of resource augmentation needed by polynomial time algorithms.

3.2 Lower bounds with preemption but without resource augmentation

We show that without extra capacity, the competitive ratio of any randomized online algorithm which tries to minimize the total flow time (max-stretch, resp.) for the data transfer problem against an oblivious adversary can not be bounded by any function of the network size. The lower bound for the competitive ratio in terms of the number of jobs, n, is $\Omega(\sqrt{n})$ for both metrics. The quantity P is 1 for the flow-time lower bound, and $\sqrt{n}$ for the max-stretch lower bound. The lower bounds hold even if the online algorithm is allowed to do preemption and use fractional capacities on links but the adversary is not.

Consider the length-3 path $A - B - C - D$. Assume that all 3 links have the same bandwidth, u. Each connection will request the same amount of data, r. We rescale time so that $u = r$ i.e. each request can be serviced in exactly one time unit.

Total flow time: The adversary first tosses an unbiased coin. If the outcome is "Heads" it chooses the link $A - B$ as a special link, else it chooses $C - D$. During the first time step, the adversary generates k requests from A to C and k from B to D. The adversary does not do anything for the next $k - 1$ time units. Then for the next k^2 time units the adversary generates one request per time unit over the special link.

Lemma 8 *The expected flow time of any online algorithm on this sequence must be $\Omega(k^3)$, even if preemption is allowed and the online algorithm is allowed to use fractional capacities. Further, the optimum flow time for this sequence is $O(k^2)$ even without using fractional capacities and preemption.*

Proof: During the first k time units, the algorithm can send only k units of data over the edge $B - C$. Hence, k units of data remains unsent at time k. Since the special edge is picked randomly by the adversary and not known to the algorithm, the expected amount of unsent data which needs to traverse the special edge is at least $k/2$. During the next k^2 time units, even if the special edge is kept continuously busy, the expected amount of unsent data waiting to cross the special edge is at least $k/2$. Therefore the expected flow time of the algorithm is at least $k^2 \cdot (k/2) = \Omega(k^3)$. The adversary on the other hand will schedule all the requests that need the special edge during the first k time units to obtain a total flow time of $O(k^2)$. ■

Since the number of jobs is $n = 2k + k^2$, the competitive ratio of any online algorithm must be $\Omega(\sqrt{n})$ which does not depend on the network size.

Max-stretch: Again, the adversary first tosses an unbiased coin. If the outcome is "Heads" it chooses the link $A - B$ as a special link, else it chooses $C - D$. During the first time step, the adversary generates 1 request from A to C and 1 from B to D, each of size k; for the next $k-1$ time units the adversary does nothing. Over the next k^2 time units the adversary generates one request of size 1 every time unit over the special link. The proof of the following lemma is similar to the previous one, and is omitted.

Lemma 9 *The expected max-stretch of any online algorithm on this sequence must be $\Omega(k)$, even if preemption is allowed and the online algorithm is allowed to use fractional capacities. Further, the optimum max-stretch for this sequence is 2 even without using fractional capacities and preemption.*

The ratio $P = p_{max}/p_{min}$ for this sequence is k. Since the number of jobs is $n = 2 + k^2$, the competitive ratio of any online algorithm must be $\Omega(\min\{P, \sqrt{n}\})$ which does not depend on the network size. A lower bound of $\Omega(P^{1/3})$ for the competitive ratio of an online algorithm for the minimum max-stretch problem in the context of machine scheduling was presented in [5].

3.3 Lower bounds on the amount of resource augmentation

In this section we give lower bounds on the amount of resource augmentation needed for any randomized online algorithm to achieve a constant competitive ratio. These lower bounds require an adaptive adversary, and assume that the online algorithm is not allowed to preempt requests or change the rate at which a request is being serviced. Notice that our upper bounds all work against adaptive adversaries, and do not preempt requests.

Theorem 10 *Against an adaptive adversary, any randomized online algorithm that achieves constant competitiveness for max-stretch must use $\Omega(\min\{n, \log P/\log\log P\})$ resource augmentation.*

Proof: The adversary uses a one link network with capacity 1. Let u be the resource augmentation that the online algorithm uses and let k be a parameter chosen suitably below. The sequence of requests created by the adversary consists of subsequences $A_0, A_1, \ldots, A_f$, for some $f \geq 0$. The beginning of a new subsequence A_i is called a *restart*. Initially $i = 0$. Each subsequence A_i consists of requests of size L_i, one every L_i time units where $L_i = (16uk)^{3u-i}$. Define an *i-phase* to be a time interval between the ith and the i+1st restart during which no new jobs of A_i arrive and no old jobs of A_i are completed by the online algorithm. Since the algorithm is not allowed to vary the rates, the adversary can determine at the beginning of an i-phase how long the i-phase would last if no new job arrived. The adversary also knows the bandwidth utilization of the online algorithm during the i-phase. If the adversary encounters an i-phase that would last at least $L_i/(8u)$ time units and where jobs of A_i use more than 1/3 units of bandwidth, the adversary increments i and it restarts. If the adversary does not encounter such an i-phase, it stops when A_i consists of k jobs.

Note that whenever the adversary restarts, the bandwidth available to the online algorithm for jobs created after the restart is reduced by at least 1/3. Thus the adversary restarts at most $3u$ times, i.e. $f \leq 3u$. It can be shown inductively that the optimum algorithm can schedule all jobs in $\cup_{l>i} A_l$ (ie all jobs of size less than L_i) in time at most L_i. Hence delaying the last job of each size by its size gives an algorithm with max-stretch at most 2.

We show next that the max-stretch of the online algorithm is at least k. Let $L_f = (16uk)^{3u-f}$ be the size of the shortest jobs generated by the adversary. When the adversary creates jobs of size L_f no f-phase exists of length at least $L_f/(8u)$ where jobs of A_f use more than 1/3 units of bandwidth. Since k jobs of size L_f are created, there are at most $2k$ f-phases. The total amount of data of jobs in A_f transferred during f-phases where the jobs in A_f use more than 1/3 units of bandwidth is at most $2k \cdot L_f/(8u) \cdot u = L_f k/4$. We consider next f-phases where the jobs in A_f use at most 1/3 units of bandwidth. During the first $2kL_f$ time units of these f-phases at most $2kL_f/3$ data of jobs in A_f is transferred. Therefore the total amount of data of jobs in A_f transferred by the online algorithm during the first $2kL_f$ time units since the last restart is at most $11kL_f/12$. Hence, there are some jobs of A_f left unfinished at time $2kL_f$ and therefore, there must be some job with a stretch of k.

It follows that the competitive ratio is at least $k/2$. Note that the ratio P of the maximum job size to minimum job size is $(16uk)^f$ and that the number n of jobs is at most fk. Since $f \leq 3u$, $n \leq 3ku$ and $P \leq (16uk)^{3u}$. If the competitive ratio is a constant, both $n/3u$ and $P^{1/(3u)}/(16u)$ must be a constant. The first condition translates to $u = \Omega(n)$ and the second translates to $u = \Omega(\log P/\log\log P)$. Therefore

$$u = \Omega(\min\{n, \log P/\log\log P\}).$$

■

Theorem 11 *Let $\gamma = \min\{n, P\}$. Against an adaptive adversary, any randomized online algorithm that achieves constant competitiveness for Total Flow Time must use $\Omega(\sqrt{\log\gamma/\log\log\gamma})$ resource augmentation*[4]*.*

Proof: We define the adversary recursively. Choose a fixed constant $\delta < 1/6$, and let $k > 2$ be a parameter whose value we will specify later. Let u be the amount of resource augmentation; we restrict our attention to those u that are multiples of δ. The network consists of just a single link and the adversary generates requests which need to transfer 1 unit of data over this link; we call these the *long* jobs. Define a basic interval to be of the form $[i/(2u), (i+1)/(2u))$, where i is a non-zero integer. If the online algorithm schedules a job over the duration $[t_1, t_2)$, then we round up t_1 and round down t_2 to be multiples of $1/(2u)$ *without* altering the rate which was assigned to this job by the online algorithm. This can only help the online algorithm. Define a basic interval to be "bad" if the online algorithm uses a capacity of more than δ on the link during this interval.

The adversary generates a long request at time 0. Then, at time $i > 0$, the adversary generates a long request only if the previous time unit $[i-1, i)$ did not contain a bad basic interval. The

[4]In a preliminary version of this paper [14] we claimed a slightly weaker lower bound of $\Omega(\sqrt{\log\gamma}/\log\log\gamma)$.

adversary stops as soon as it sees $k\delta/u$ basic bad intervals or has generated k long requests. Also, during each basic bad interval, the adversary recursively generates the lower bound sequence for $(u-\delta)$ resource augmentation with time scaled down by a factor of $1/(4ku)$.

Let $F^*(u)$ denote the optimal flow time for the above sequence, and $F(u)$ denote the flow time obtained by the online algorithm. Let $N(u)$ and $P(u)$ denote the number of jobs and the ratio of the largest to the smallest job generated by the adversary, respectively. As a base case, we use the lower bound example used in Lemma 8[5], where $u = 1$. For this, choose $N(1) = k$. Then $F(1) = \Theta(k^{1.5})$, $F^*(1) = \Theta(k)$, and $P(1) = 1$.

The optimum way to schedule all the jobs generated by the adversary is to schedule a long job during a time unit which does not contain a basic bad interval. There will never be more than one long job waiting in the system. Further, since there are never more than $k\delta/u$ basic bad intervals, the total duration of the schedule is at most $k + k\delta/u < 2k$. Therefore, the total contribution of the long jobs to the flow time is at most $2k$; this also ensures that all the jobs generated during the recursive procedure fit into the basic interval and do not overflow. During each basic bad interval time is scaled down by a factor of $1/(4ku)$, which implies the recurrence

$$F^*(u) \leq 2k + \frac{1}{4ku}F^*(u-\delta)\frac{k\delta}{u} < 2k + F^*(u-\delta).$$

Using the base case, we obtain

$$F^*(u) < 2ku/\delta + O(k) = O(ku). \tag{1}$$

Also, $P(u) \leq (4ku)P(u-\delta)$, which simplifies to $P(u) = (ku)^{O(u)}$, and $N(u) = k + (k\delta/u)N(u-\delta) = k^{O(u)}$.

We now provide a lower bound on $F(u)$. First, we study the case in which the adversary terminated before finding $k\delta/u$ basic bad intervals. Since the capacity available to the algorithm is u, any job must originally be scheduled for at least $1/u$ time. By rounding up the start and rounding down the end of an interval of length at least $1/u$ to multiples of $1/(2u)$, the length of the interval decreases by a factor of at most 2. Hence, the total data transferred originally by the algorithm is at most twice the data transfer after the rounding. If the number of bad basic intervals is less than $k\delta/u$ then the total data transfer during these bad intervals is at most $k\delta$. Further, the bandwidth used during the good basic intervals is at most $2k\delta$ during the first $2k$ time units. Hence, the total data transferred originally by the algorithm is at most $2(k\delta + 2k\delta) = 6k\delta$ during the first $2k$ time units. Therefore there is $k(1-6\delta)$ amount of data left unsent at the end of the $2k$ time units; the flow time because of this data must be at least $(k(1-6\delta))^2/(2u) = \Omega(k^2/u)$. If on the other hand, the adversary terminates because it found enough bad basic intervals, then the flow time must be at least $(1/4ku)(k\delta/u)F(u-\delta) = F(u-\delta)/(4u^2)$. Therefore,

$$F(u) = \min\left\{(k(1-6\delta))^2/(2u), F(u-\delta)/(4u^2)\right\}.$$

[5] Lemma 8 uses a length three path as opposed to a single edge network used in the current theorem; it is easy to see that the proofs in the current theorem go through even for a length three path. Also, the base case can be modified slightly to work on a length 1 path without changing the lower bounds obtained in the current theorem.

For large k, the second term is always smaller than the first (inductively) and we obtain

$$F(u) = k^{3/2}/u^{O(u)}.$$

Now, the competitive ratio is $C(u) = \sqrt{k}/u^{O(u)}$. For $C(u)$ to be constant, k must equal $u^{O(u)}$. We now lower bound u in terms of $n = N(u)$ and $P = P(u)$, given that $k = u^{O(u)}$. We can increase k as long as neither of the two conditions $N(u) = k^{O(u)}$ and $P(u) = (ku)^{O(u)}$ is satisfied; hence for a constant competitive ratio either $n = k^{O(u)} = u^{O(u^2)}$, or $P = (ku)^{O(u)} = u^{O(u^2)}$. In other words, if $\gamma = \min\{n, P\}$, then $\gamma = u^{O(u^2)}$, or $u = \Omega(\sqrt{\log \gamma / \log\log \gamma})$.

■

4 Online Algorithms for Makespan and Total Completion Time

Standard techniques can be used to obtain constant competitive online algorithms for makespan and average completion time for the online ftp problem without the use of resource augmentation.

Makespan: Define λ as the maximum over all edges, e, of the amount of data that needs to be transferred over e. We rescale time so that one unit of data can be transferred over a link in one unit of time. Let a_{MAX} be the time at which the last request arrives. Let L be the quantity $\max(a_{MAX}, \lambda)$. L is a lower bound on the makespan of any schedule. The online algorithm maintains a guess $\tilde{\lambda}$ for the value of L. We assume that the first request arrives at time 0. The initial value of $\tilde{\lambda}$ is set to p_1, the amount of data transfer needed by the first request. Each time a request arrives, the algorithm recomputes L. If $L > \tilde{\lambda}$, $\tilde{\lambda}$ is reset to $\max(L, 2\tilde{\lambda})$. The online algorithm schedules the newly arrived request to execute from time $\tilde{\lambda}$ to $2\tilde{\lambda}$, with a rate of $1/\tilde{\lambda}$. It is easy to see that the above algorithm does not violate capacity constraints. Let U represent the final value of $\tilde{\lambda}$; by construction U is at most $2L$. The makespan is at most $2U + U + U/2 + \ldots < 4U$. We can now claim the following result.

Theorem 12 *The above algorithm is 8-competitive.*

The above algorithm runs in polynomial time and hence, is also an offline approximation algorithm. However, an offline algorithm "knows" the exact value of L and hence can provide an approximation guarantee of 2. If routes are not given as part of the input, a slight variant of the above online algorithm can still obtain an 8-approximation, but it would not run in polynomial time.

Total Completion Time: The general scaling technique outlined by Hall *et al.* [16, 15] directly results in a 4-competitive online algorithm for the total completion time metric, regardless of whether routes are given as part of the input. Their technique requires an offline algorithm that can pack an optimum number of requests into a given interval. This problem is NP hard, and therefore, our online algorithm does not run in polynomial time. An $O(\log m)$-competitive polynomial time algorithm is outlined in Section 5.

5 Polynomial Time Approximation and Online Algorithms

In this section we give offline algorithms for total completion time, makespan, total flow time, average stretch, maximum flow time, and maximum stretch that run in polynomial time. The algorithms for total completion time and makespan approximate the optimum performance without resource augmentation. The algorithms for the remaining metrics achieve optimum performance using either a constant-factor or a polylogarithmic-factor resource augmentation. We conclude the section by giving polynomial-time algorithms with optimum makespan under two different relaxations of our model: (1) We relax the condition that the rate of a job has to be constant: we give a polynomial-time algorithm that varies the rates and achieves optimum makespan. (2) We assume that the start time s_j is part of the input and show that then the problem can be solved in polynomial time.

Theorem 13 *There exists an algorithm that achieves an $O(\log m)$-approximation of the total completion time for the online ftp problem in time polynomial in n and m, regardless of whether routes are given as part of the input.*

Proof: Consider the problem of maximizing the number of ftp requests (out of a given set of requests, all of which have the same arrival time) that can be scheduled over a given period of time. An $O(\log m)$ approximation to this problem can be obtained using multicommodity flow followed by randomized rounding [22]; plugging this into the general technique of Hall *et al.* [16, 15] results in an $O\log m)$-competitive polynomial time online algorithm for the total completion time of ftp requests. ∎

A polynomial time 2-approximation for the makespan of the ftp problem when routes are given as part of the input follows from the discussion in Section 4; a simple randomized rounding trick results in an $O(\log m)$-approximation if routes are not provided as input.

Theorem 14 *There exists an algorithm that achieves a 2-approximation for makespan in time polynomial in n and m if routes are given as part of the input, and $O(\log m)$ otherwise..*

We now describe how to implement algorithm MRHP in polynomial time. The only step of MRHP which might take super-polynomial time is step 2, finding the largest weight subset A_i of S_i that can be completed between times t and $t + 2^i$. To implement it in expected polynomial time we need to add $\log m + 2eK$ to the capacity of each edge, where $K = \log n + \log P + 3$.

We use first a linear programming relaxation of the problem, then round it probabilistically and finally show that with high probability no edge capacity constraint is violated. The linear program uses for each job j a variable x_j and maximizes $\sum_{\mathrm{J} \in S_i} w_j x_j$ under the constraint that for each edge e, $\sum_{j \text{ uses } e} x_j p_j / 2^i \leq 1$ and that for each j, $x_j \geq 0$. Let x_j^* denote the value of x_j in the solution. We probabilistically round each job j for each network i such $P(j \in A_i) = x_j^*$. Let X_e be the random variable denoting the load of edge e in G. The expected value μ of X_e is

$\sum_{0\leq i<K}\sum_{j \text{ uses } e} x_j^* p_j/2^i \leq K$. Using Chernoff bounds with $\delta = (\log m + 2eK)/\mu - 1$ shows that

$$\begin{aligned} Pr(X_e > \log m + 2eK) &\leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu \\ &\leq \left(\frac{e}{\log m + 2eK}\right)^{\log m + 2eK} \\ &< 1/(m(nP)^{2e}) \end{aligned}$$

Thus, the probability that one of the edge capacities overflows is at most $1/(nP)^{2e}$ in which case we simply redo the rounding step.

Note that both MMF and MMS already run in time polynomial in n, m, and $\log P$ if routes are given as part of the input. The same ideas that we outlined above for total flow time also result in polynomial time algorithms for the max-flow and max-stretch problems when routes are not given as input.

Theorem 15 *There exist (online or offline) algorithms that run in time polynomial in n, m, and $\log P$ and*

- *achieve optimum total flow-time or average stretch with an expected $O(\log n + \log P + \log m)$-factor resource augmentation regardless of whether routes are given as part of the input;*
- *achieve optimum maximum flow time with a constant-factor resource augmentation if the routes are given as part of the input, and expected $O(\log n + \log m)$ resource augmentation otherwise;*
- *achieve optimum maximum stretch with an $O(\log P)$-factor resource augmentation if the routes are given as part of the input, and expected $O(\log n + \log m + \log P)$ resource augmentation otherwise.*

We finally relax some of our conditions. Consider first the case that the rate of jobs can vary. Assume that the optimum makespan is M. We present a linear program that given M checks whether there exists a feasible solution. By performing a binary search over M, with $0 < M \leq nP$ and assuming that time is rescaled so that the shortest job takes one time unit, we get a polynomial time algorithm that finds the optimum makespan.

We assume wlog that the first job arrives at time 0. Break the time from 0 to M into intervals whenever a new job arrives and number the time intervals from 1 to n. Let l_i be the length of interval i and let a_{MAX} be the arrival time of the last job. Note that the length of the last interval is $M - a_{MAX}$. There is a variable $x_{j,i}$ for each interval i and each job j. The linear program checks whether there is a non-negative assignment for the variables $x_{j,i}$ such that

1. For each job j, $\sum_i x_{j,i} l_i \geq p_j$,
2. For each edge e and interval i, $\sum_{j \text{ uses } e} x_{i,j} \leq 1$, and

3. For each job j and interval i such that j arrived after i, $x_{j,i} = 0$.

The linear program can be slightly modified to give a polynomial-time algorithm in the case that the start time s_j is given for each job j.

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. *25th ACM Symposium on Theory of Computing*, pages 623–31, 1993.

[2] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput competitive online routing. *34th IEEE symposium on Foundations of Computer Science*, pages 32–40, 1993.

[3] Y. Bartal, S. Leonardi, and A. Fiat. Lower bounds for on-line graph problems with application to on-line circuit and optical routing. *Proc. of the 28th Symposium on Theory of Computation*, pages 531–540, 1996.

[4] J.E. Beck and D.P. Siewiorek. Modeling multicomputer task allocation as a vector packing problem. *9th International Symposium on Systems Synthesis*, pages 115–120, 1996.

[5] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. *Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, 1998.

[6] S. Chakrabarti, C. Phillips, A. Schulz, D. Shmoys, C. Stein, and J. Wein. Improved scheduling algorithms for minsum criteria. *23rd International Colloquium on Automata, Languages, and Programming*, 1996.

[7] C. Chekuri and S. Khanna. On multi-dimensional packing problems. *Tenth ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[8] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein. Approximation techniques for average completion time scheduling. *Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 609–618, 1997.

[9] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Proceedings of the Eleventh Annual Conference on Complexity Theory, IEEE*, 1996.

[10] A. Feldmann, M. Kao, J. Sgall, and S. Teng. Optimal online scheduling of parallel jobs with dependencies. *J. of Combinatorial Optimization*, 1(4):393–411, 1998.

[11] J. Garay and I. Gopal. Call preemption in communication networks. *IEEE INFOCOM*, pages 1043–1050, 1992.

[12] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. *2nd Annual Israel Conference on Theory of Computing and Systems*, 1993.

[13] M. Garofalakis and Y. Ioannidis. Scheduling issues in multimedia query optimization. *ACM Computing Surveys*, 27(4):590–592, 1995.

[14] A. Goel, M. Henzinger, S. Plotkin, and E. Tardos. Scheduling data transfers in a network and the set scheduling problem. *31st ACM Synposium on Theory of Computing*, 1999.

[15] L.A. Hall, D.B. Shmoys, A.S. Schulz, and J. Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[16] L.A. Hall, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time: off-line and on-line algorithms. *7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 142–151, 1996.

[17] D. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Company, 1997.

[18] H. Kellerer, T. Tautenhahn, and G.J. Woeginger. Approximability and non-approximability results for minimizing total flow time on a single machine. *28th ACM Symposium on Theory of Computing*, pages 418–426, 1996.

[19] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 110–119, 1997.

[20] C. Phillips, C. Stein, E. Torng, and J. Wein. Critical scheduling via resource augmentation. *29th ACM Symposium on Theory of Computing*, 1997.

[21] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. *Lecture Notes in Computer Science 955*, pages 290–301, Springer, Berlin.

[22] P. Raghavan and C. Thompson. Randomized rounding. *Combinatorica*, 7:365–374, 1987.

[23] J. Sgall. Randomized on-line scheduling of parallel jobs. *J. of Algorithms*, 21:149–175, 1996.

A Reduction of Vertex Color to Set Scheduling

In this section we prove Theorem 1:*The Vertex Color problem reduces (via polynomial time reductions) to Minimizing Makespan for Set Scheduling in an approximation preserving fashion.*

Let $G(V, E)$ be an instance of the vertex color problem. We construct an instance of the set scheduling problem with $S = E$. For each vertex v in the original vertex color problem, we introduce a job j_v with arrival time 0, processing time 1, and resource requirement $I(v)$, where $I(v)$ is the set of all edges incident on v. Jobs j_u and j_v need a common resource iff there is an edge between vertices u and v. Thus, jobs j_u and j_v can be scheduled at the same time iff u and v can have the same color. This establishes a one to one correspondence between the makespan of the set scheduling problem and the number of colors needed for the vertex color problem.