

TCP is Competitive Against a Limited Adversary This research was supported in part by grants from NSERC and CITO.

Jeff Edmonds

Computer Science Department  
York University, Toronto, Canada  
jeff@cs.yorku.ca Suprakash Datta

Computer Science Department  
York University, Toronto, Canada  
datta@cs.yorku.ca Patrick Dymond

Computer Science Department  
York University, Toronto, Canada  
patrick@cs.yorku.ca

### Abstract

While the well-known Transport Control Protocol (TCP) is a *de facto* standard for reliable communication on the Internet, and performs well in practice, the question “how good is the TCP/IP congestion control algorithm?” is not completely resolved. In this paper, we provide some answers to this question using the competitive analysis framework. First, we prove that for networks with a single bottleneck (or point of congestion), TCP is competitive to the optimal global algorithm in minimizing the *user-perceived latency* or *flow time* of the sessions. Specifically, we show that with  $\mathcal{O}(1)$  times as much bandwidth and  $\mathcal{O}(1)$  extra time per job, TCP is  $\mathcal{O}(1)$ -competitive against an optimal global algorithm. We motivate the need for allowing TCP to have extra resources by observing that existing lower bounds for non-clairvoyant scheduling algorithms imply that no online, distributed, non-clairvoyant algorithm can be competitive with an optimal offline algorithm if both algorithms were given the same resources. Second, we show that TCP is fair by proving that it converges quickly to allocations where every session gets its fair share of network bandwidth.

## 1 Introduction

Most of the traffic on the Internet today is generated by applications which use the transport control protocol (TCP). According to one study [?], 95% of the bytes and 90% of the packets sent over the Internet use TCP. Since TCP is a large and complex protocol that solves several different problems, including reliable data transfer, flow control, congestion control, and fair allocation of network resources. Most of the research efforts towards evaluating the quality of TCP has concentrated on evaluating (either empirically or analytically) the value of some performance metric. In this paper we take a different approach towards the problem, and evaluate the performance of TCP using the traditional competitive analysis framework [?]. Thus, we compare the performance of TCP (which is an online algorithm) with that of an optimal offline algorithm. In this paper, we study the performance of TCP on single-bottleneck networks, i.e., in a network where there is a single point of congestion. We prove that (a simplified version of) TCP satisfies the transmission requests of all users in an *efficient* and

*fair* way. We show that existing lower bounds from non-clairvoyant scheduling theory imply that no online, distributed, non-clairvoyant algorithm can be competitive with an all-powerful adversary. This suggests that the power of the adversary should be limited to level the playing field. The most popular approach for doing this is to explicitly limit the adversary in some way, e.g., by limiting its freedom in choosing its inputs. In this paper, we use an alternative approach that has been utilized very successfully in scheduling theory [?, ?, ?], viz., giving the online algorithm strictly more resources compared to the adversary. Using this approach, we first prove that TCP performs competitively against any all-powerful adversary if it is given a constant times more bandwidth and either (a) some extra time, or (b) we assume that no job is smaller than a certain size. Second, we study the fairness properties of TCP and prove that TCP converges quickly to allocations where every session gets its fair share of network bandwidth.

An interesting byproduct of our paper is that it emphasizes and exploits a very natural connection between the TCP congestion control algorithm and the theory of online algorithms and non-clairvoyant scheduling. This provides a new theoretical approach for analyzing TCP variants and exploring parameter settings under different network assumptions.

This paper is organized as follows. Section 2 describes the our simplified model of TCP. Section 3 describes how TCP can be viewed as a scheduling algorithm, and introduces the scheduling model. Section 4 surveys the literature on the performance analysis of TCP and results from scheduling theory relevant to this paper. Section 5 presents the main results of the paper.

## 2 The TCP congestion control algorithm

While TCP solves several problems, this paper focuses on the congestion control (prevention, detection, and reaction to congestion in the network) algorithm of TCP. TCP runs at every sender/receiver in a distributed manner. There is no communication between different sessions. In this paper (as in most papers in the literature), we model only the basic algorithmic strategy used by TCP. This strategy (commonly referred to as additive-increase-multiplicative-decrease or AIMD) is very simple: at every step, each source is allowed to increase its sending rate by an additive constant  $\alpha$ . When it detects congestion, the TCP algorithm requires the sender to cut its sending rate by a multiplicative constant  $\beta$ . In this paper, we refer to this action as an *adjustment*.

Detection of network congestion is a difficult task, since no support from switches and routers is assumed (TCP does not receive any messages from the switches or routers that packets pass through *en route* to their destinations. Congestion can only be inferred (perhaps incorrectly) from end-to-end observations. TCP sends acknowledgments to the sender for every packet that reaches its destination, and infers congestion from the late arrival or non-arrival of acknowledgments.

We emphasize that in reality, TCP uses windows of packet sequence numbers and a “self-clocking” mechanism instead of actual transmission rates. TCP also has many other details that we do not model in this paper. The reader is referred to the books by Kurose [?] and Stevens [?] for more information on the protocol.

Our model of the TCP congestion control algorithm is as follows. The sender of each job  $J_i$  uses TCP to determine the rate  $b_i^T(t)$  at which it transmits its job  $J_i$  (Throughout the paper, we use the superscript  $T$  to denote that this bandwidth was allocated by TCP). This sender has no knowledge about the other jobs; it starts with  $b_i^T(t) = 0$ , and increases its transmission rate linearly at a constant rate of  $\partial b_i^T(t)/\partial t = \alpha$  (typically  $\alpha = 1$ ) until it infers that the bottleneck has reached capacity (from non-acknowledgment of some packet). At this point, the sender cuts its own rate  $b_i^T(t)$  by a multiplicative factor of  $\beta$  (typically  $\beta = \frac{1}{2}$ ). We call this an *adjustment point*. After each such

adjustment, the sender resumes increasing the rate linearly at a rate of  $\alpha$ . When the total transmission rate through the bottleneck exceeds its capacity, the bottleneck “loses” randomly-selected data beyond this capacity. For simplicity, we will assume that there is a fixed delay of  $\delta$  between when a bottleneck loses some transmission and when the senders learn about it.

### 3 TCP viewed as a scheduling algorithm

We model the transmission of a file of data through a network as the flow of a fluid that can be allocated a variable bandwidth/transmission rate. Similarly, we model the execution of a fully parallelizable job that can be allocated a variable (real) number of processors. Within these abstractions, the problem of scheduling bandwidth to a number of transmission sessions is identical to that of scheduling a number of processors to a set of parallelizable jobs. The latter problem has a rich history of results [?, ?, ?, ?]. This paper applies and extends those results to the former problem.

#### 3.1 The scheduling problem

We assume that there is a single bottleneck in our network<sup>1</sup> which causes all data losses. In reality this bottleneck may be a link or a router. We assume that the bottleneck has a maximum capacity  $B$ ; if data arrives at a rate higher than  $B$ , the excess data is lost. The input consists of a set of jobs (or sessions)  $\mathcal{J} = \{J_i\}$ . Each job  $J_i$  is defined by its arrival time  $a_i$  and its length (or size)  $l_i$ .

A scheduling algorithm  $\text{ALG}_s$  must schedule the transmission of the jobs at speed  $s$  (i.e. with  $s$  times the resources given to the all-powerful adversary). At each time  $t$ , the algorithm allocates  $b_i^A(t)$  bandwidth to job  $J_i$  passing through it. Since the bottleneck has capacity  $sB$ ,  $\sum_{i \in \mathcal{J}_t} b_i^A(t) \leq sB$ . A job of length  $l_i$  completes at time  $c_i^A$  if the algorithm allocates enough bandwidth so that  $\int_{a_i}^{c_i^A} b_i^A(t) dt = l_i$ .

We use the flow time  $\mathcal{L}(\text{ALG}_s)$  of a scheduling algorithm  $\text{ALG}_s$  as a measure of its performance. The *flow time* [?, ?, ?, ?] is the average time between the arrival time and completion time of a job, i.e.  $\text{Avg}_{i \in \mathcal{J}} [c_i^A - a_i]$ . This measure is sometimes called the *user-perceived latency* in the Systems literature. As mentioned before, we measure the performance of an algorithm by its *competitive ratio* which is defined as the ratio of the flow time of  $\text{ALG}_s$  to the flow time of the optimal algorithm for the worst set of jobs, i.e.,  $\max_{\mathcal{J}} \frac{\mathcal{L}(\text{ALG}_s(\mathcal{J}))}{\mathcal{L}(\text{OPT}_1(\mathcal{J}))}$ . In this paper, we allow algorithm  $\text{ALG}_s$  to have some extra time  $D(\mathcal{J})$  as well. In this case, the competitive ratio is defined as  $\max_{\mathcal{J}} \frac{\mathcal{L}(\text{ALG}_s(\mathcal{J}))}{\mathcal{L}(\text{OPT}_1(\mathcal{J}) + D(\mathcal{J}))}$ . We emphasize that the subscript for the optimal algorithm  $\text{OPT}$  is used to remind the reader that the optimal algorithm is given less resources than the online algorithm.

Any algorithm that solves the preceding scheduling problem must be an *online* algorithm [?] (since it must allocate its bandwidth as jobs arrive, without knowledge of future arrivals), *non-clairvoyant* [?, ?, ?] (it only knows when a job arrives and when it completes, but does not know the amount of work remaining or the parallelizability of the jobs) and *distributed* (the sender of a job has no knowledge of the other jobs or even the maximum bandwidth of the bottlenecks). It only receives limited feedback about his own transmission loss due to bottleneck overflow. While it is known that the Shortest-Remaining-Work-First is the optimal scheduling algorithm for this problem when all the jobs are fully parallelizable, neither non-clairvoyant nor distributed scheduling algorithms have enough information to execute the Shortest-Remaining-Work-First algorithm. The *optimal* scheduler, in contrast, knows everything about all jobs. Equivalently, we can assume that the optimal scheduler is the adversary that chooses the worst-case input which is good for itself and bad for the online algorithm.

<sup>1</sup>For preliminary results on the general case (multiple bottlenecks), please see the manuscript [?]

## 4 Related work and our results

As we have mentioned before, most of the existing analyses of TCP attempt to evaluate explicitly some performance metric after making some probabilistic assumptions on the inputs. In contrast, competitive analysis makes no assumptions about inputs and provides worst-case results. We survey relevant previous work in both areas in the following two subsections. Then, we turn to relevant work on the fairness of the TCP congestion control algorithm. Relevant work from the theory of non-clairvoyant scheduling algorithms are surveyed next. The final subsection presents our results.

### 4.1 Probabilistic analysis of TCP congestion control

Many papers study the efficiency of TCP by evaluating its throughput. Most of these make the simplifying assumption that every packet gets dropped with probability  $p$  independent of all other packet drops. Under this assumption, several papers [?, ?] show that for low values of  $p$ , TCP throughput is proportional to  $1/\sqrt{p}$ . The same result was proved in [?] for more elaborate models of TCP. [?] showed that throughput decreases faster (roughly proportional to  $1/p$ ) at higher values of  $p$ . All these papers assume constant round-trip times. Misra and Ott [?] incorporated state-dependent packet loss probabilities into their model and studied the stationary distribution of the congestion window.

Kelly et al [?, ?] consider multibottleneck models but constant round-trip times. Using control-theoretic methods, they prove that a simplified version of the TCP congestion control algorithm is *stable* – i.e., it converges to a stable equilibrium point which results in fair allocations of the bottleneck capacities. Johari and Tan [?] and Massoulié [?] study the effect of variable round-trip times on stability and show that stability is achieved even under this assumption. It is worth noting that all these papers deal with rates instead of windows of sequence numbers.

### 4.2 Competitive analysis of TCP congestion control

The problem of analyzing TCP congestion control using competitive analysis was suggested by Karp *et al.* [?], who attempted to find theoretical reasons for the empirically observed superiority of TCP to most of the proposed alternatives. In their model, a TCP session attempts to guess a threshold  $u$  (intuitively the *ideal* transmission rate for the session), and the network (modeled as an adversary) imposes a cost  $c(x, u)$  on the session where  $x$  is the current guess of the session. They assume  $u$  to be a positive integer and that the algorithm knows an upper bound  $n$  on  $u$ . They also consider a dynamic version of the problem where the threshold  $u$  can change over time and study ways in which the power of the adversary can be limited in changing the threshold  $u$ . Recently, Arora et al [?] studied the same model and proposed an optimal randomized algorithm for the same problem.

### 4.3 Fairness of TCP congestion control

Chiu and Jain [?] studied fairness and efficiency of the additive-increase-multiplicative-decrease rate adjustment algorithm from a control-theoretic standpoint. The main result of their paper is that as long as no jobs arrive or complete, TCP converges towards fair allocations (they show that the global measure  $(\sum_{i \in \mathcal{J}} b_i^T(t))^2 / (n \sum_{i \in \mathcal{J}} (b_i^T(t))^2)$  converges to 1). Subsequently, several papers (e.g., [?, ?]) studied fairness issues in TCP.

## 4.4 Previous Scheduling Results

Kalyanasundaram and Pruhs [?] present a simple non-clairvoyant algorithm *Balance* and prove that for every set of jobs it performs within a factor of  $\frac{s}{s-1} = 1 + \frac{1}{\epsilon}$  of the optimal schedule as long as it is given  $s = 1 + \epsilon$  times the speed. Such an algorithm is said to be a  $\mathcal{O}(1)$ -speed  $\mathcal{O}(1)$ -competitive algorithm. *Equi-partition* (EQUI) is a simple, natural and fair scheduling algorithm that is used extensively in practice. It allocates an equal (potentially non-integer) number of processors to each unfinished job. Edmonds [?, ?] proves that EQUI is competitive as long as it is given  $s = 2 + \epsilon$  times the speed. Rather surprisingly, EQUI is competitive even for jobs with phases of different “natural” speed-up curves, e.g., *fully parallelizable*  $\Gamma(x) = x$  and *sequential*  $\Gamma(x) = 1$  work. Motwani *et al.* [?] prove that for every deterministic non-clairvoyant scheduler, there is a set of  $n$  jobs on which the scheduler does a factor of  $\Omega(n^{1/3})$  worse than the optimal schedule. For EQUI, this ratio is  $\Omega(\frac{n}{\log n})$ . It is likely this lower bound holds for all distributed schedulers.

## 4.5 Our Results

In this paper, we view TCP as a very simple and natural online, non-clairvoyant and distributed scheduling algorithm. We show that (our simplified version of) TCP is competitive against an optimal offline scheduler, provided we limit the power of the adversary.

Chiu *et al* [?] proved that if no jobs come in or leave, TCP converges quickly to EQUI with  $(\frac{1+\beta}{2})B$  total bandwidth. In this paper, we extend their results and allow jobs to arrive and leave over time. Under these assumptions, TCP takes longer to converge to EQUI. For periods of time after the arrival or the completion of jobs, some jobs may not be given their fair share of the bandwidth. Therefore, the algorithm may no longer be competitive, especially if these starved jobs are short.

Our first result, (in Section 5.2), proves that for TCP with  $s = \mathcal{O}(1)$  extra bandwidth and  $D(\mathcal{J})$  extra time,  $\frac{\mathcal{L}(\text{TCP}_s(\mathcal{J}))}{\mathcal{L}(\text{OPT}_1(\mathcal{J})+D(\mathcal{J}))} = \mathcal{O}(1)$ .<sup>2</sup> Here  $D(\mathcal{J})$  is some extra time which can be crudely upper bounded by  $\mathcal{O}\left(\frac{|\mathcal{J}|(1-\beta)B}{\alpha}\right)$ , where  $|\mathcal{J}|$  is the number of jobs in  $\mathcal{J}$ . Intuitively, this captures the fact that TCP needs  $\mathcal{O}(1)$  extra time per job to be competitive. We will defer the actual definition of  $D(\mathcal{J})$  to Section 5.2 for simplicity of exposition, where we will show that  $D(\mathcal{J})$  is typically much smaller than this upper bound. Our second result, (in Section 5.3), shows that TCP converges quickly to the allocations produced by EQUI. In particular, it bounds the total time that a job is not getting its fair share to being at most a few adjustment periods at the beginning and the end of each job. The length of an adjustment period is at most  $\mathcal{O}\left(\frac{1-\beta}{\alpha} \frac{B}{n(t)}\right)$  when there are  $n(t)$  jobs in the system, because it only takes this much time for  $n(t)$  jobs increasing their individual bandwidth at a rate of  $\alpha$  to increase the total bandwidth from the decreased total of  $(1-\beta)B$  back to the bottleneck’s capacity  $B$ . We expect that this will typically be a small fraction of the job’s total life.

Our results can be interpreted as follows. First, TCP is  $\mathcal{O}(1)$ -competitive if that the adversary is limited in the manner described before *and* all sessions are of a certain minimum length. In the presence of short sessions, the competitive ratio may not be a constant, in keeping with the intuition that EQUI (and of course OPT) may finish small jobs much faster than TCP. To our knowledge, this is the first result on the competitiveness of the TCP congestion control algorithm (we note that the results of Dooly *et al* [?] were for a different problem, even though it concerned TCP and involved competitive analysis). Another interesting aspect of our work is that we prove our results by comparing TCP to EQUI instead of the optimal algorithm. In our model, EQUI captures precisely the notion

---

<sup>2</sup>We point out that an equivalent form of our result is  $\frac{\mathcal{L}(\text{TCP}_s(\mathcal{J}))-D(\mathcal{J})}{\mathcal{L}(\text{OPT}_1(\mathcal{J}))} = \mathcal{O}(1)$ . However, this version is less meaningful if  $\mathcal{L}(\text{TCP}_s(\mathcal{J})) \leq D(\mathcal{J})$ .

of fairness used by TCP,<sup>3</sup> and this allows us to prove fairness properties as a byproduct. It is worth pointing out that our fairness results are subsumed by similar results proved for multi-bottleneck networks in [?, ?, ?, ?].

Finally, our results hold for any constant  $\alpha, \beta$  satisfying  $\alpha > 0$  and  $0 \leq \beta < 1$ . We can quantify some of the tradeoffs involved in choosing the parameters  $\alpha$  and  $\beta$ . For lack of space, we defer this discussion to Section B in the Appendix.

## 5 Fairness and efficiency of TCP

Due to space limitations, we will prove our results for the case  $\delta = 0$ , corresponding to the assumption that senders receive instantaneous feedback when bottleneck capacity is exceeded. In the full version, we prove our results for  $\delta > 0$ . The Appendix also contains some results for  $\delta > 0$ .

### 5.1 Lower bounds for non-clairvoyant schedulers

TCP, being on-line, non-clairvoyant, and distributed, is not always able to compete against an optimal scheduler. We motivate the need to give TCP a constant times extra bandwidth and some extra time to adjust by showing that lower bounds for non-clairvoyant schedulers imply lower bounds for TCP.

Motwani *et al.* [?] prove that for every deterministic non-clairvoyant scheduler (of which TCP is an example), the competitive ratio is  $\Omega(n^{1/3})$ . Thus TCP needs at least a constant factor more resources than the optimal algorithm in order to be competitive.

For EQUI, the competitive ratio is known to be  $\Omega(\frac{n}{\log n})$ . Kalyanasundaram and Pruhs [?] prove that even with speed  $s = 2 - \epsilon$ , EQUI has competitive ratio of  $\Omega(n^\epsilon)$ . It is only with speed  $s = 2 + \epsilon$ , that Edmonds [?] proves that EQUI is  $\mathcal{O}(1)$ -competitive. Since EQUI has more information than TCP, it is reasonable to expect that TCP also needs a speed  $s$  satisfying  $s \geq 2 + \epsilon$ .

### 5.2 Extra bandwidth and adjustment time

Ideally one would like to show that TCP, even though it is on-line, non-clairvoyant, and distributed, always has a competitive user perceived latency (or flow time). However, this is not true. We will prove that TCP is competitive if it is given more resources than the optimal algorithm. The extra resources are a constant times more bandwidth and either some extra time (equal to a constant number of adjustment periods per job). The latter is unnecessary if all the jobs live for at least a constant number of adjustment periods.

We prove our results by comparing TCP to EQUI, which we already know is competitive if it has a constant factor more bandwidth than the optimal. We now describe the intuitive reasons for the extra powers needed by TCP in order to be perform as well with EQUI (and thus be competitive). Since no sender in our model knows about the other senders, it takes a while for the system to adjust to jobs arriving and completing. In contrast, EQUI adjusts its allocations instantly. We prove that despite this, TCP converges towards EQUI exponentially fast for each job. We show that at all adjustment points, at least  $q$  periods after a job arrives, the bandwidth allocated by TCP to the job is at least a factor of  $1 - \beta^q$  of that allocated by EQUI (see Theorem 5.3). We will choose some constant  $q$  and compensate TCP for this remaining gap by giving it an extra factor of  $\frac{1}{1 - \beta^q}$  bandwidth.

Further, because TCP is a distributed algorithm, it is difficult for the algorithm to continually utilize all of the available bandwidth. For the AIMD algorithm, the total bandwidth used varies

---

<sup>3</sup>We remind the reader that TCP congestion control uses windows of sequence numbers and not rates. Therefore, two sessions may have the same window sizes but use different bandwidths if they have different roundtrip times.

linearly between  $\beta B$  and  $B$ , where  $B$  is the capacity bandwidth of the bottleneck. Therefore, TCP utilizes on average only  $\frac{\beta+1}{2}$  of the available bandwidth. It follows that TCP needs a factor of  $\frac{2}{\beta+1}$  extra bandwidth to compete with any non-distributed algorithm.

Finally, an extra  $(1 + \frac{1}{q})$  factor is required to compensate for the effect of other jobs arriving and completing. Combining all of these factors, TCP needs to be given a factor of  $s = (2+\epsilon)(\frac{1}{1-\beta^q})(\frac{2}{\beta+1})(1 + \frac{1}{q})$  more bandwidth than the optimal scheduler is given.

In order for TCP to be competitive we must also either give each job the extra time of a constant number of adjustment periods to adjust or require all the jobs to live at least a constant number of adjustment periods. An *adjustment period* is the period between consecutive adjustment points. Lemma B.1 shows that the length of an adjustment period is  $\frac{(1-\beta)B}{\alpha n^T(t)}$ , where  $n^T(t)$  denotes the (average) number jobs alive under TCP during the period<sup>4</sup>. Adjustment periods may vary in length, and so we make precise the notion of “the time of a constant number of adjustment periods” below.

When a job first arrives, EQUI allocates it a fair share of the bandwidth. The optimal scheduler may allocate all the bandwidth to the job with the shortest remaining work in order to complete it quickly. In contrast, TCP forces the new job to start with a transmission rate of zero. Nevertheless, a job’s allocation converges exponentially quickly to that given by EQUI. In particular a job needs to wait  $q$  complete adjustment periods for its rate to be at least a factor of  $1 - \beta^q$  of that of EQUI.

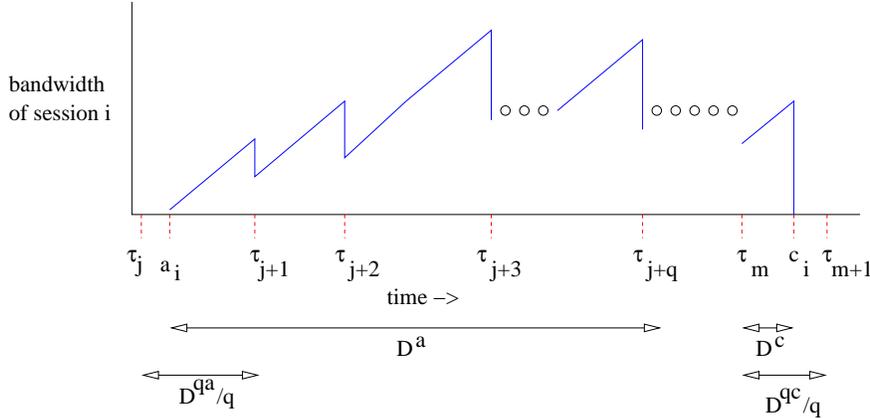


Figure 1: The time of a constant number of adjustment periods per jobs.

More formally, consider a set of jobs (TCP requests)  $\mathcal{J} = \{J_i\}$ . Let  $\tau_j$ ,  $j = 0, \dots$  be the times of the *adjustment points*. Let  $j_i^a$  denote the index of the first adjustment time  $\tau_{(j_i^a)}$  after job  $J_i$  arrives, i.e.  $j_i^a \stackrel{\text{def}}{=} \min_j \{j \mid \tau_j \geq a_i\}$ . (See Figure 1.) Let  $D_i^a$  denote the length of the first  $q$  complete adjustment periods of job  $i$ , i.e.,  $D_i^a \stackrel{\text{def}}{=} \tau_{(j_i^a+q)} - a_i$ .

A job may not get its fair share of the bandwidth in the last adjustment period when the multiplicative constant  $\beta$  is close to zero. With this setting, all jobs drastically decrease their transmission rate when the bottleneck reaches capacity. If a job completes shortly after this adjustment, it does not have a reasonable allotment of bandwidth during this last fractional adjustment period. Let  $j_i^c$  denote the index of the last adjustment time  $\tau_{(j_i^c)}$  before job  $J_i$  completes, i.e.  $j_i^c \stackrel{\text{def}}{=} \max_j \{j \mid \tau_j \leq c_i^T\}$ . Let  $D_i^c$  denote the fractional time of this last adjustment period, i.e.  $D_i^c \stackrel{\text{def}}{=} c_i^T - \tau_{(j_i^c)}$ .

<sup>4</sup>The length of an adjustment period is upper bounded by a (possibly large) constant, viz.,  $\frac{(1-\beta)B}{\alpha}$ . We express this length in terms of  $n^T(t)$  to demonstrate that the length is much smaller if there is a number of jobs or sessions in progress.

Even when a job is neither arriving nor completing, we will see that having other jobs arrive or complete may temporarily cause a job to receive less than its fair allotment of bandwidth. We will have these other jobs “pay” for this. Let  $D_i^{qa}$  and  $D_i^{qc}$  denote  $q$  times the complete length of the first and last adjustment periods that job  $J_i$  is in, namely, let  $D_i^{qa} \stackrel{\text{def}}{=} q \cdot (\tau_{(j_i^a)} - \tau_{(j_i^a-1)})$  and  $D_i^{qc} \stackrel{\text{def}}{=} q \cdot (\tau_{(j_i^c+1)} - \tau_{(j_i^c)})$ .

Summing up, let  $D(\mathcal{J}) \stackrel{\text{def}}{=} \sum_{i \in \mathcal{J}} (D_i^a + D_i^c + D_i^{qa} + D_i^{qc})$  denote the sum of these times over all jobs. Note that this is  $\mathcal{O}(q)$  adjustment periods per job. Our main result states that if TCP is given the constant factor  $s$  more bandwidth and  $D(\mathcal{J})$  extra time to adjust, then  $\frac{\mathcal{L}(\text{TCP}_s(\mathcal{J}))}{\mathcal{L}(\text{OPT}_1(\mathcal{J})+D(\mathcal{J}))} = \mathcal{O}(1)$ . Equivalently we could write  $\frac{\mathcal{L}(\text{TCP}_s(\mathcal{J}))-D(\mathcal{J})}{\mathcal{L}(\text{OPT}_1(\mathcal{J}))} = \mathcal{O}(1)$ .

Note that in keeping with [?], our theorems are proved for general jobs – we allow each job  $J_i$  to have an arbitrary number of phases and each phase to have an arbitrary nondecreasing sublinear speedup function  $\Gamma_{i,k}(b)$ , representing the rate at which work is executed for phase  $k$  of job  $i$  when allocated  $b$  processors. Thus, our results hold even if each transmission request came with a specified upper and lower bound on the rate at which the data could be transmitted or received.

**Theorem 5.1** *Let  $\alpha > 0$ ,  $\beta \in [0, 1)$ ,  $\delta \geq 0$ ,  $q \geq 1$  be an integer,  $s = (2 + \epsilon)(\frac{1}{1-\beta^q})(\frac{2}{\beta+1})(1 + \frac{1}{q})$ , and  $\mathcal{J}$  be any set of jobs in which each phase of each job can have an arbitrary sublinear-nondecreasing speedup function. Let  $D(\mathcal{J})$  be the length of  $\mathcal{O}(q)$  adjustment periods per job, with each adjustment period being of length  $\frac{(1-\beta)B}{\alpha n^T(t)} + (1-\beta)\delta$ . For any non-fully parallelizable phase job, give  $\text{TCP}_s(\mathcal{J})$  the speedup function  $\frac{4\beta+4}{5\beta+3}\Gamma_{i,k}(b)$  whenever  $\text{OPT}_1(\mathcal{J})$  is given  $\Gamma_{i,k}(b)$ . Alternatively, give the same speedup functions, but change the factor of  $(\frac{2}{\beta+1})$  within  $s$  to  $\frac{1}{\beta}$  (which is reasonable unless  $\beta$  is close to zero.) Then  $\frac{\mathcal{L}(\text{TCP}_s(\mathcal{J}))}{\mathcal{L}(\text{OPT}_1(\mathcal{J})+D(\mathcal{J}))} = \mathcal{O}\left(1 + \frac{1}{\epsilon}\right)$ .*

Alternatively, we could require all jobs to live at least a constant number of adjustment periods. In this case, the extra time to adjust is not needed because if each job lives for  $\mathcal{O}(q)$  adjustment periods, then  $D(\mathcal{J}) \stackrel{\text{def}}{=} \sum_{i \in \mathcal{J}} \mathcal{O}(q)$  adjustment periods  $\leq \mathcal{O}(\sum_{i \in \mathcal{J}} c_i^T - a_i) \stackrel{\text{def}}{=} \mathcal{O}(\mathcal{L}(\text{TCP}_s(\mathcal{J})))$ .

**Corollary 5.2** *Let  $q \geq 1$  be an integer,  $s = (2 + \epsilon)(\frac{1}{1-\beta^q})(\frac{2}{\beta+1})(1 + \frac{1}{q})$ , and  $\mathcal{J}$  be any set of jobs in which each job lives for  $\mathcal{O}(q)$  adjustment periods. Then  $\frac{\mathcal{L}(\text{TCP}_s(\mathcal{J}))}{\mathcal{L}(\text{OPT}_1(\mathcal{J}))} = \mathcal{O}\left(1 + \frac{1}{\epsilon}\right)$ .*

### 5.3 TCP converges to EQUI

We prove that TCP converges to EQUI by comparing what TCP and EQUI would allocate on a job-by-job, moment-by-moment basis as jobs arrive and complete.

First, we prove that at all adjustment points, at least  $q$  periods after a job arrives, the bandwidth allocated by TCP to the job is at least a factor of  $(1 - \beta^q)$  of what EQUI would allocate given the same speed. Interestingly, at this point a job could still have a constant fraction of the total bandwidth,  $\beta^q B$ , which might be considerably more than its share  $\frac{sB}{n^T(t)}$ . We must wait  $\mathcal{O}(\log n + q)$  phases until we are sure that TCP allocates no more than a factor  $(1 + \beta^q)$  of what EQUI would allocate.

**Theorem 5.3** *Let  $q \geq 1$  be an integer,  $s$  be any value, and  $\mathcal{J}$  be any set of jobs. For each job  $J_i$  and for all times  $t = \tau_{j_i^a+q+j}$ ,  $j \geq 0$ ,  $b_i^T(t) \geq (1 - \beta^q)\frac{sB}{n^T(t)}$ , where  $b_i^T(t)$  denotes the bandwidth allocated by  $\text{TCP}_s(\mathcal{J})$  to job  $J_i$  at time  $t$  and  $n^T(t)$  denotes the number jobs alive at this time. On the other hand, at all times  $t \geq \tau_{j_i^a+\log(n)/\log(1/\beta)+q}$ ,  $b_i^T(t) \leq (1 + \beta^q)\frac{sB}{n^T(t)}$ .*

The proof of this theorem is given in the Appendix.

While it may seem from Theorem 5.3 if we give TCP  $\frac{2}{\beta+1}$  times as much bandwidth, all jobs get their fair share of the bandwidth, this does not hold, due to the arrivals and departures of other

jobs. For example, when jobs complete, it takes TCP some time to allocate the freed bandwidth. In contrast, EQUI would instantly reallocate the freed bandwidth to the existing jobs. We now bound the amount of time during which such a discrepancy occurs. We will define  $Less(\mathcal{J})$  to be the total time over which  $TCP_s$  allocates a job less bandwidth than  $EQUI_{2+\epsilon}$  would allocate, and prove that this is at most  $\mathcal{O}(q)$  adjustment periods per job, i.e.  $Less(\mathcal{J}) \leq D(\mathcal{J})$ .

More formally, for each job  $J_i$  and each adjustment period  $[\tau_j, \tau_{j+1}]$ , we say that the job during this period receives *less* allocation than  $EQUI_{2+\epsilon}$  would give if the job's average transmission rate during this phase is less under  $TCP_s(\mathcal{J})$  than it would be under  $EQUI_{2+\epsilon}$ , i.e.  $\text{Avg}_{t \in [\tau_j, \tau_{j+1}]} b_i^T(t) < \text{Avg}_{t \in [\tau_j, \tau_{j+1}]} \frac{(2+\epsilon)B}{n^T(t)}$ . We consider the average over the phase, in order to compensate for the fact that TCP decreases and increases the transmission rates each phase.

Let  $Less_i$  denote the sum of the lengths of all the adjustment periods during the life of job  $J_i$  for which the job receives less. Let  $Less(\mathcal{J}) \stackrel{\text{def}}{=} \sum_{i \in \mathcal{J}} Less_i$  denote the total time over which a job is allocated less than its share.

**Theorem 5.4** *Let  $q \geq 1$  be an integer,  $s = (2 + \epsilon)(\frac{1}{1-\beta^q})(\frac{2}{\beta+1})(1 + \frac{1}{q})$ , and  $\mathcal{J}$  be any set of jobs. Then  $Less(\mathcal{J}) \leq D(\mathcal{J})$ .*

We refer the reader to the Appendix for a proof of this theorem when the jobs are fully parallelizable. In the full version, we show how we can extend this result to hold for arbitrary sublinear-nondecreasing speedup functions.

## 5.4 Proof of the competitiveness of TCP

In this section, we prove Theorem 5.1, i.e.,  $TCP_s$  is competitive when given both extra bandwidth and extra time to adjust. This is done by proving that it is competitive against  $EQUI_{2+\epsilon}$  which we already know is competitive against the optimal scheduler  $OPT_1$ .

**Proof of Theorem 5.1:** Given any job set  $\mathcal{J}$ , we construct another set of jobs  $\mathcal{J}'$  and follow the following proof “outline”.

$$\frac{\mathcal{L}(TCP_s(\mathcal{J}))}{\mathcal{L}(OPT_1(\mathcal{J})) + D(\mathcal{J})} \leq \frac{\mathcal{L}(EQUI_{(2+\epsilon)}(\mathcal{J}'))}{\mathcal{L}(OPT_1(\mathcal{J}'_{\text{par}})) + \mathcal{L}(OPT_1(\mathcal{J}'_{\text{seq}}))} = \mathcal{O}\left(1 + \frac{1}{\epsilon}\right)$$

The first inequality to prove is  $\mathcal{L}(TCP_s(\mathcal{J})) = \mathcal{L}(TCP_s(\mathcal{J}')) \leq \mathcal{L}(EQUI_{2+\epsilon}(\mathcal{J}'))$ . The set of jobs  $\mathcal{J}'$  is designed specifically so that  $TCP_s(\mathcal{J})$  and  $TCP_s(\mathcal{J}')$  are identical computations yet  $TCP_s(\mathcal{J}')$  always has completed at least as much as  $EQUI_{2+\epsilon}(\mathcal{J})$  on every job. By definition,  $TCP_s(\mathcal{J})$  allocates more bandwidth to jobs than  $EQUI_{2+\epsilon}$  would allocate in all adjustment periods except for those during which the job receives *less*.  $\mathcal{J}'$  is designed to be exactly the same as  $\mathcal{J}$  except that the work completed during these *less* adjustment periods is deleted and replaced with a *sequential* phase lasting the length of the period. Sequential phases have speedup functions  $\lfloor \_$ , namely  $\Gamma(b) = 1$ . No scheduler can get ahead on a sequential phase of a job, because no matter how much resource (here the resource is the bandwidth and not processors), the phase gets completed at a fixed rate. By design, the computation times do not change from  $TCP_s(\mathcal{J})$  to  $TCP_s(\mathcal{J}')$  and hence  $\mathcal{L}(TCP_s(\mathcal{J})) = \mathcal{L}(TCP_s(\mathcal{J}'))$ . Lemma 5.1 below formally proves that  $TCP_s(\mathcal{J}')$  is never behind  $EQUI_{2+\epsilon}(\mathcal{J})$  on any job and hence its user-perceived latency (flow time) is competitive, i.e.  $\mathcal{L}(TCP_s(\mathcal{J}')) \leq \mathcal{L}(EQUI_{2+\epsilon}(\mathcal{J}'))$ . This gives the first inequality required in the above proof outline.

By definition,  $\mathcal{J}'_{\text{seq}}$  and  $\mathcal{J}'_{\text{par}}$  contain respectively only the sequential and the non-sequential phases of jobs in  $\mathcal{J}'$ . The inequality  $\mathcal{L}(OPT_1(\mathcal{J})) \geq \mathcal{L}(OPT_1(\mathcal{J}'_{\text{par}}))$  holds since  $\mathcal{J}$  has more work than  $\mathcal{J}'_{\text{par}}$ .

The last inequality to prove is  $D(\mathcal{J}) \geq Less(\mathcal{J}) = \mathcal{L}(OPT_1(\mathcal{J}'_{\text{seq}}))$ .  $\mathcal{L}(OPT_1(\mathcal{J}'_{\text{seq}}))$ , by definition, is the flow time of this set of purely sequential jobs under the optimal scheduler. Independent of the

bandwidth allocated, this is simply the sum of the sequential work in each job. By design this is the total time that  $\text{TCP}_s(\mathcal{J})$  allocates a job less bandwidth than  $\text{EQUI}_{2+\epsilon}$  would allocate during a phase, which by definition is  $\text{Less}(\mathcal{J})$ . Hence,  $\mathcal{L}(\text{OPT}_1(\mathcal{J}'_{\text{seq}})) = \text{Less}(\mathcal{J})$ . Finally, Theorem 5.4 proves that  $\text{Less}(\mathcal{J}) \leq D(\mathcal{J})$ . This completes all the inequalities required in the proof outline above. ■

**Lemma 5.1**  $\mathcal{L}(\text{TCP}_s(\mathcal{J}')) \leq \mathcal{L}(\text{EQUI}_{2+\epsilon}(\mathcal{J}'))$ .

**Proof of Lemma 5.1:**  $\text{TCP}_s(\mathcal{J}')$  allocates more bandwidth to the non-sequential phases than  $\text{EQUI}_{2+\epsilon}$  would allocate. We must now prove that this is also more than  $\text{EQUI}_{2+\epsilon}(\mathcal{J}')$  actually does allocate. We prove by induction on  $t$  that at each point in time  $\text{TCP}_s(\mathcal{J}')$  has completed at least as much work on each job as  $\text{EQUI}_{2+\epsilon}(\mathcal{J}')$ , i.e.,  $\mathcal{L}(\text{TCP}_s(\mathcal{J}')) \leq \mathcal{L}(\text{EQUI}_{2+\epsilon}(\mathcal{J}'))$ . Note that this also bounds the number of jobs active at time  $t$ , i.e.  $n^T(t) \leq n_t^E$ . Consider the next time instance. If the next phase of a job in  $\mathcal{J}'$  is non-sequential then it must be completed under  $\text{TCP}_s(\mathcal{J}')$  during an adjustment period during which the job does not receive *less* allocation than  $\frac{(2+\epsilon)B}{n^T(t)}$  that  $\text{EQUI}_{2+\epsilon}$  would give in the same circumstances. By the induction hypothesis,  $n^T(t) \leq n_t^E$  and hence the job does not receive less allocation than  $\frac{(2+\epsilon)B}{n_t^E}$ , which is what  $\text{EQUI}_{2+\epsilon}(\mathcal{J}')$  does allocate. On the other hand, if the next phase of a job is sequential, the job completes at the same fixed rate, irrespective of how much bandwidth is allocated to the job. Hence, we conclude that  $\text{TCP}_s(\mathcal{J}')$  completes at least as much work on each job as  $\text{EQUI}_{2+\epsilon}(\mathcal{J}')$  for the next  $\partial t$  time. This completes the inductive proof. ■

The last inequality in Theorem 5.1 follows directly from the competitiveness of  $\text{EQUI}$ , which was proved in the following theorem in [?] and improved slightly [?] in order to be used here. Also, [?] allows the optimal scheduler to complete the fully parallelizable work independently from the sequential work.

**Theorem 5.5** ([?]) *Let  $\mathcal{J}$  be any set of jobs in which each phase of each job can have an arbitrary sublinear-nondecreasing speedup function.  $\frac{\mathcal{L}(\text{EQUI}_{(2+\epsilon)}(\mathcal{J}))}{2\mathcal{L}(\text{OPT}_1(\mathcal{J}))} \leq \frac{\mathcal{L}(\text{EQUI}_{(2+\epsilon)}(\mathcal{J}))}{\mathcal{L}(\text{OPT}_1(\mathcal{J}_{\text{par}})) + \mathcal{L}(\text{OPT}_1(\mathcal{J}_{\text{seq}}))} \leq \mathcal{O}(1 + \frac{1}{\epsilon})$ , where  $\mathcal{J}_{\text{par}}$  and  $\mathcal{J}_{\text{seq}}$  contain respectively only the non-sequential and the sequential phases of the jobs  $\mathcal{J}$ .*

## A Proofs of theorems

We re-state and prove Theorem 5.3.

**Theorem 1** *Let  $q \geq 1$  be an integer,  $s$  be any value, and  $\mathcal{J}$  be any set of jobs. For each job  $J_i$  and for all times  $t = \tau_{j_i^a + q + j}$ ,  $j \geq 0$ ,  $b_i^T(t) \geq (1 - \beta^q) \frac{sB}{n^T(t)}$ , where  $b_i^T(t)$  denotes the bandwidth allocated by  $\text{TCP}_s(\mathcal{J})$  to job  $J_i$  at time  $t$  and  $n^T(t)$  denotes the number jobs alive at this time. On the other hand, at all times  $t \geq \tau_{j_i^a + \log(n)/\log(1/\beta) + q}$ ,  $b_i^T(t) \leq (1 + \beta^q) \frac{sB}{n^T(t)}$ .*

Note that  $\frac{sB}{n^T(t)}$  is the amount that  $\text{EQUI}_s$  would allocate to the job were it in this situation. However, it may not be the amount  $\frac{sB}{n_t^E}$  that  $\text{EQUI}_s(\mathcal{J})$  does allocate at this time within its computation on the set of jobs  $\mathcal{J}$ , because with different bandwidth allocations jobs may complete at different times under  $\text{TCP}_s(\mathcal{J})$  and  $\text{EQUI}_s(\mathcal{J})$  and hence the number of jobs  $n^T(t)$  and  $n_t^E$  alive under them at time  $t$  may be different. We use  $\text{EQUI}_s$  vs  $\text{EQUI}_s(\mathcal{J})$  to differentiate between “would” and “does”.

**Proof of Theorem 5.3:** Fix some job  $J_i$ . We will classify each unit of bandwidth allocation as either being *adjusted* or *unadjusted* depending on whether the bandwidth is allocated fairly from this job’s

$J_i$ 's perspective. We prove that the amount of adjusted bandwidth converges exponentially to being all the bandwidth and then prove that our job  $J_i$  is allocated a fair share of the adjusted bandwidth.

When the job  $J_i$  first arrives it is initially allocated no bandwidth. Hence, it considers all bandwidth allocation to be unadjusted. When at a rate of  $\alpha$  each job is allocated more bandwidth, this new bandwidth allocation is considered to be adjusted.

At adjustment points, we assume that both the job's adjusted and unadjusted bandwidth allocations are decreased by this factor  $\beta$ . At job  $J_i$ 's first adjustment point,  $\tau_{j_i^a}$ , the total unadjusted bandwidth in the system is at most  $sB$ , this being the capacity of the bottleneck. At each adjustment point, every job decreases its unadjusted bandwidth by a factor  $\beta$  and never increases its unadjusted bandwidth again. Hence, at the time of job  $J_i$ 's  $(q+1)^{st}$  adjustment point,  $\tau_{j_i^{a+q}}$ , the total unadjusted bandwidth in the system is at most  $\beta^q sB$ . At points of adjustment, the total bandwidth of any kind in the system is exactly the capacity  $sB$  of the bottleneck. It follows that at time  $\tau_{j_i^{a+q}}$  the total adjusted bandwidth in the system is at least  $(1 - \beta^q)sB$ .

When job  $J_i$  first arrives at time  $a_i$ , no job has any adjusted bandwidth. At each point in time, each job alive increases its adjusted bandwidth at the same rate  $\alpha$  and hence they continue to have the same amount. Jobs that arrive after our job  $J_i$  may have less adjusted bandwidth than  $J_i$  and jobs that complete release all of their bandwidth, but these events only make it better for  $J_i$ . The point is that  $J_i$  has at least as much adjusted bandwidth as any other job. It follows that at time  $\tau_{j_i^{a+q}}$ , the amount of adjusted bandwidth that  $J_i$  has is  $b_i^T(t) \geq (1 - \beta^q) \frac{sB}{n^T(t)}$ .

We consider two strategies for dealing with a time delay of  $\delta$  before the senders adjust. The above proof assumes the first strategy, namely that in which each sender decreases its transmission rate to the fraction  $\beta$  of its current rate of *sending* data independent of how much of the data is being lost. Now consider the second strategy, in which each sender decreases its transmission rate to a fraction  $\beta$  of the current rate that data passes through the bottleneck without being dropped. Here the rate at which a sender loses data affects its next adjusted transmission rate. Define the transmission that is being lost as being neither unadjusted nor adjusted so that during the  $\delta$  delay the total amounts of unadjusted and adjusted bandwidth stay fixed. However, during this time, the adjusted bandwidth gets shifted from the jobs/senders with more than their share to those with less. Though the bottleneck is at capacity, each sender continues to increase its transmission rate. This is considered adjusted bandwidth. Simultaneously the bottleneck increases the rate that the sender's transmission is lost, which decreases the adjusted bandwidth. The bottleneck is assumed to drop each packet with a fixed probability. Hence, a sender's rate of transmission loss is proportional to its current transmission rate. Hence, senders with less than their share of bandwidth lose less. This effect only helps to ensure that our job/sender  $J_i$  has at least its share of the adjusted bandwidth and helps to speed up the overall rate of convergence to EQUIL.

For completeness, we will now give an upper bound on the bandwidth that an individual job might be allocated. Suppose that initially a particular job has all  $sB$  of the bandwidth. After  $q$  adjustments, the job still has  $\beta^q sB$  of the bandwidth. However, after  $\frac{\log(n)}{\log(1/\beta)} + q$  adjustment phases, the unadjusted bandwidth has decreased to at most  $\beta^q \frac{sB}{n^T(t)}$ . Hence,  $b_i^T(t) \leq (1 + \beta^q) \frac{sB}{n^T(t)}$ . ■

We re-state and prove Theorem 5.4.

**Theorem 2** *Let  $q \geq 1$  be an integer,  $s = (2 + \epsilon)(\frac{1}{1-\beta^q})(\frac{2}{\beta+1})(1 + \frac{1}{q})$ , and  $\mathcal{J}$  be any set of jobs. Then  $\text{Less}(\mathcal{J}) \leq D(\mathcal{J})$ .*

**Proof of Theorem 5.4:** Under  $\text{TCP}_s(\mathcal{J})$ , a job may be allocated less than its fair share of the bandwidth at any point during its life. We classify these times into three types based on whether they occur when the job first arrives, in the middle of its life, or as it is completing. Let  $\text{Less}_i^a$  denote

the amount of time  $t \in [a_i, \tau_{(j_i^a+q)}]$  within job  $J_i$ 's first  $q$  adjustment phases that it is allocated less than  $\frac{(2+\epsilon)B}{n^T(t)}$  (on average over the adjustment period). Let  $Less_i^m$  denote the same within the middle of its life,  $t \in [\tau_{(j_i^a+q)}, \tau_{(j_i^c)}]$ . Finally, let  $Less_i^c$  denote the same within the job's last adjustment phase,  $t \in [\tau_{(j_i^c)}, c_i]$ . Clearly,  $Less_i = Less_i^a + Less_i^m + Less_i^c$ .

Recall that  $D(\mathcal{J}) = \sum_{i \in \mathcal{J}} (D_i^a + D_i^c + D_i^{qa} + D_i^{qc})$ . Simply by definition,  $Less_i^a \leq D_i^a$  and  $Less_i^c \leq D_i^c$ , because these are the times of the first  $q$  and the last one adjustment phases during which job  $J_i$  is alive. What remains in proving  $Less(\mathcal{J}) \leq D(\mathcal{J})$  is to prove that  $\sum_{i \in \mathcal{J}} Less_i^m \leq \sum_{i \in \mathcal{J}} (D_i^{qa} + D_i^{qc})$ . This will not be proved on a job-by-job basis but on an adjustment phase-by-adjustment phase basis with the jobs that are arriving and completing "paying" for the jobs that are not.

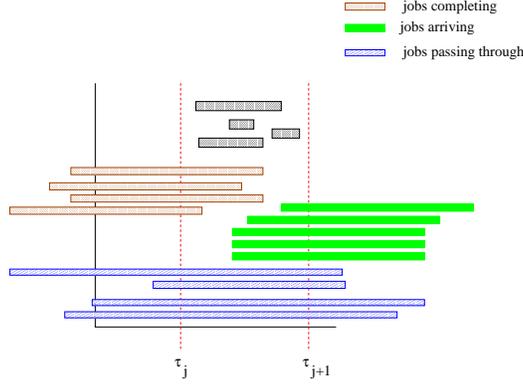


Figure 2:  $n_j^c$ ,  $n_j^a$ , and  $n_j^p$  denotes the number of jobs that respectively complete, arrive, and pass through the adjustment period  $[\tau_j, \tau_{j+1}]$ .

Consider some adjustment period at time  $[\tau_j, \tau_{j+1}]$ . Let  $n_j^c$  denote the number of jobs that are active at the beginning of this period and which complete during it,  $n_j^a$  the number that arrive during it and are active at the end of it, and  $n_j^p$  the number that pass through the entire period. We ignore the jobs that arrive and complete within the period. (See Figure 2.) We will consider two cases.

**Case 1:**  $n_j^c + n_j^a \geq \frac{1}{q}n_j^p$ .

The only jobs that can contribute to  $\sum_{i \in \mathcal{J}} Less_i^m$  are the  $n_j^p$  jobs that are passing through the phase, because contributing jobs must be in the middle of their lives. These jobs may not contribute anything either because they are allocated sufficient bandwidth or because they are not actually in the middle of their lives. However, the most that they can contribute is the full length of this adjustment period.

The  $n_j^c + n_j^a$  jobs that either complete or arrive during this adjustment period each contribute  $q$  times the full length of this adjustment period to  $\sum_{i \in \mathcal{J}} (D_i^{qa} + D_i^{qc})$ .

It follows that the contribution of this adjustment period to  $\sum_{i \in \mathcal{J}} Less_i^m$  is at most  $n_j^p \cdot (\tau_{j+1} - \tau_j)$ , which because of the case assumption is at most  $q(n_j^c + n_j^a) \cdot (\tau_{j+1} - \tau_j)$ , which is at most the contribution of this adjustment period to  $\sum_{i \in \mathcal{J}} (D_i^{qa} + D_i^{qc})$ .

**Case 2:**  $n_j^c + n_j^a < \frac{1}{q}n_j^p$  or more specifically  $n_j^c < \frac{1}{q}n_j^p$  and  $n_j^a < \frac{1}{q}n_j^p$ .

For this case, we will prove that this adjustment period contributes nothing to  $\sum_{i \in \mathcal{J}} Less_i^m$ , because all the jobs that are past their first  $q$  adjustment phases are allocated on average at least  $\frac{(2+\epsilon)B}{n^T(t)}$  during this adjustment period  $[\tau_j, \tau_{j+1}]$ . Consider any such job  $J_i$ .

By definition, we know that the number of jobs alive at the beginning of the phase is  $n_{\tau_j}^T = n_j^c + n_j^p$ . Hence, by Theorem 5.3 we know that immediately before this beginning adjustment point  $t = \tau_j$ , job  $J_i$

is allocated at least  $(1 - \beta^q) \frac{sB}{n_j^c + n_j^p}$  bandwidth. However, being an adjustment point, the job decreases its transmission rate by a factor of  $\beta$ . Hence the rate at the beginning of the phase is  $\beta(1 - \beta^q) \frac{sB}{n_j^c + n_j^p}$ . By the assumption of the claim, this is at least  $\beta(1 - \beta^q) \frac{sB}{\frac{1}{q}n_j^p + n_j^p} = \beta(1 - \beta^q) \left(\frac{1}{1+1/q}\right) \frac{sB}{n_j^p}$ . Similarly, the number of jobs alive at the end of the phase is  $n_{\tau_{j+1}}^T = n_j^p + n_j^a$  and hence at this time the bandwidth  $J_i$  is allocated is at least  $(1 - \beta^q) \frac{sB}{n_j^p + n_j^a} \geq (1 - \beta^q) \left(\frac{1}{1+1/q}\right) \frac{sB}{n_j^p}$ . During the phase, the allocation to job  $J_i$  is increased linearly. Hence, the average (effective) transmission rate for the job during this phase is the average of these beginning and the ending rates. This average is  $\left(\frac{\beta+1}{2}\right)(1 - \beta^q) \left(\frac{1}{1+1/q}\right) \frac{sB}{n_j^p}$ . Because  $s$  is set to  $\left(\frac{2}{\beta+1}\right) \left(\frac{1}{1-\beta^q}\right) \left(1 + \frac{1}{q}\right) (2 + \epsilon)$ , this average transmission rate is at least  $\frac{(2+\epsilon)B}{n_j^p}$ . (Note that having a  $\delta$  time delay while the bottleneck stays at capacity only helps this average.)

We must now bound what  $\text{EQUI}_{2+\epsilon}$  would allocate jobs. We know that for each point in time  $t$  during this phase, the number of jobs  $n^T(t)$  alive is at least  $n_j^p$ , because by definition this is the number that passes through the phase. It follows that that this average rate  $\frac{(2+\epsilon)B}{n_j^p}$  that  $\text{TCP}_s(\mathcal{J})$  allocates job  $J_i$  is at least the amount  $\frac{(2+\epsilon)B}{n^T(t)}$  that  $\text{EQUI}_{2+\epsilon}$  would allocate at any point in time  $t \in [\tau_j, \tau_{j+1}]$  during the adjustment period. Hence job  $J_i$  does not receive *less* during this phase and hence this adjustment period contributes nothing to  $\sum_{i \in \mathcal{J}} \text{Less}_i^m$ .

From these two cases, we can conclude that  $\sum_{i \in \mathcal{J}} \text{Less}_i^m \leq \sum_{i \in \mathcal{J}} (D_i^{qa} + D_i^{qc})$  and hence that  $\text{Less}(\mathcal{J}) \leq D(\mathcal{J})$ . ■

## B Tradeoffs with settings of the TCP parameters $\alpha$ and $\beta$

We will now describe the tradeoffs involved in choosing the parameters  $\alpha$  and  $\beta$ . The key effects are as follows. Setting the multiplicative constant  $\beta$  is a tradeoff between TCP's utilization of the bandwidth and the rate of the convergence of TCP to EQUI. These effects are reflected in Theorem 5.1 by the extra speed  $s = (2 + \epsilon) \left(\frac{1}{1-\beta^q}\right) \left(\frac{2}{\beta+1}\right) \left(1 + \frac{1}{q}\right)$  and the extra time  $D(\mathcal{J}) = \mathcal{O}\left(\frac{(1-\beta)B}{\alpha n^T(t)} + \delta\right)$  required by TCP to be competitive. Setting the additive constant  $\alpha$  is a trade off between the amount of packet loss and the rate of the convergence.

The following lemma derives the dependence of the length of an adjustment period on  $\alpha$  and  $\beta$ . Large adjustment periods decrease the frequency of adjustments, but short adjustment periods decrease the time  $D(\mathcal{J})$  that jobs must wait until they gets their fair allocation of bandwidth.

**Lemma B.1** *The length of an adjustment period is  $|\tau_{j+1} - \tau_j| = \frac{(1-\beta)B}{\alpha n^T(t)} + (1 - \beta)\delta$ , where  $n^T(t)$  denotes the (average) number jobs alive under TCP during the period.*

**Proof of Lemma B.1:** At the point in time when the bottleneck reaches capacity, the total bandwidth allocated to jobs is clearly the bottleneck's capacity  $B$ . If there is a delay of  $\delta$  in time before the senders detect packet loss, then during this time each sender continues to increase its transmission rate at the additive rate of  $\alpha$ . The total transmission rate after this delay will be  $B + \alpha n^T(t)\delta$ .

We consider two strategies that TCP might take at this point. The first strategy is for the sender to decrease its transmission rate to the fraction  $\beta$  of its current rate of *sending* data. Doing this would decrease the total transmission rate to  $\beta(B + \alpha n^T(t)\delta)$ . (It is problematic if this delay  $\delta$  is so big that this adjusted rate is still bigger than the capacity  $B$  of the bottleneck.) The second strategy is to decrease its transmission rate to a fraction  $\beta$  of the current rate that data passes through the bottleneck without getting dropped. Doing this would decrease the total transmission rate to only  $\beta B$ . (Here there is no limit on how large the delay  $\delta$  can be.)

With either strategy, the total bandwidth allocated continues to increase at a rate of  $\alpha n^T(t)$ . The time required for the total to increase again to  $B$  is  $\frac{B-\beta(B+\alpha n^T(t)\delta)}{\alpha n^T(t)}$  in the first strategy and only  $\frac{B-\beta B}{\alpha n^T(t)}$  in the second. The total length of the adjustment period is this plus the  $\delta$  delay time, which is either  $|\tau_{j+1} - \tau_j| = \frac{(1-\beta)B}{\alpha n^T(t)} + (1-\beta)\delta$  or  $|\tau_{j+1} - \tau_j| = \frac{(1-\beta)B}{\alpha n^T(t)} + \delta$ . ■

**Lemma B.2** *The scheduling algorithm TCP becomes precisely the scheduling algorithm EQUI in the limit as  $\beta \rightarrow 1$ ,  $\alpha \rightarrow \infty$ , and  $\delta = 0$ .*

**Proof of Lemma B.2:** With  $\alpha$  increased towards infinity, TCP converges instantly to EQUI. This instantly-converging TCP still decreases its bandwidth allocation by a factor of  $\beta$  each adjustment point for an average total bandwidth utilization of  $\frac{\beta+1}{2}B$ . Increasing the multiplicative constant  $\beta$  towards 1 increases this utilized bandwidth towards the full  $B$ . ■

**Lemma B.3** *For the extreme parameter settings  $\beta \rightarrow 1$ ,  $\alpha \rightarrow \infty$ , and  $\delta = 0$ , when TCP is precisely EQUI, our new Theorem 5.1, which bounds the competitiveness of TCP, is tight with the old Theorem 5.5, which bounds the competitiveness of EQUI.*

**Proof of Lemma B.3:** By setting  $q = \frac{1}{(1-\beta)^2}$  and increasing  $\beta$  to one, the extra bandwidth  $s = (2+\epsilon)(\frac{1}{1-\beta q})(\frac{2}{\beta+1})(1+\frac{1}{q})$  required in Theorem 5.1 goes to  $s = (2+\epsilon)$  in the limit. This is precisely the extra speed that EQUI needs. The extra time  $D(\mathcal{J})$  is  $\mathcal{O}(q)$  adjustment periods per job, which is  $w = \mathcal{O}(q \cdot \frac{(1-\beta)B}{\alpha n^T(t)})$ . By setting  $\alpha = q$  and increasing  $\beta$  to one, this goes to zero in the limit. Finally, the required change  $\frac{4\beta+4}{5\beta+3}\Gamma_{i,k}(b)$  in the speedup function and/or the new factor of  $\frac{1}{\beta}$  within  $s$  both disappear with  $\beta = 1$ . ■

**Lemma B.4** *If one sender has a longer delay  $\delta$  before adjusting than other senders, then the rate at which it sends data will be the same, but it will experience less packet loss than the other senders.*

**Proof of Lemma B.4:** The sender with the longer delay  $\delta$  uses the same parameters  $\alpha$  and  $\beta$  as the other senders and has the same time  $|\tau_{j+1} - \tau_j|$  between adjustments. Hence, the rate that it transmits will increase and decrease in the same way, except shifted forward in time. This will mean that this sender is having its peak transmission rate at a latter point in time after the other have already decreased their rates. It follows that this sender will have less packet loss. ■