ARTICLE IN PRESS

Theoretical Computer Science ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On the competitiveness of AIMD-TCP within a general network

Jeff Edmonds *

Department of Computer Science, York University, North York, ONT M3J 1P3, Canada

ARTICLE INFO

Article history:
Received 14 July 2005
Received in revised form 30 August 2011
Accepted 30 July 2012
Communicated by G. Italiano

Keywords:
AIMD
TCP
Online competitive ratio
Flow time
Fairness
Multi-bottleneck

ABSTRACT

This paper studies the performance of AIMD (Additive Increase Multiplicative Decrease) TCP as an online distributed scheduling algorithm for allocating transmission rate to sessions/jobs running on a general network. The network consists of a set of routers which in this context act only as bottlenecks, i.e. when a router's capacity has been reached, it informs the jobs passing through it to multiplicatively back off transmission rates. The analysis is easier when this AIMD algorithm is modeled by a continuous algorithm. We improve on that presented by Kelly to better capture the interconnectedness of the network. Extending the paper by Edmonds, Datta, and Dymond that solves the single-bottleneck case, we prove that with extra resources, this algorithm AIMDEQUI is competitive against the optimal global algorithm in minimizing the average transmission time of the jobs. We also bound the fairness of this resource allocation according to three different definitions of fairness.

1. Introduction

AIMD (Additive Increase Multiplicative Decrease) is the core algorithmic component of TCP (Transport Control Protocol) for allocating bandwidth or transmission rate to the different jobs. This paper considers a network modeled as a general graph where each node is a router that acts as a bottleneck with a given capacity B_k. The set of sessions/jobs that arrive and complete throughout time on the network must be allocated network bandwidth. The amount of bandwidth that each of them will be allocated depends on the bottlenecks it passes through. In this algorithm, each job J_i increases his bandwidth $b_{\langle i,t\rangle}$ linearly at a rate of $\delta b_{\langle i,t\rangle}/\delta t = \alpha$ (typically $\alpha = 1$) until he detects that one of the bottlenecks that his transmission passes through has reached capacity, at which point, he cuts his bandwidth by a multiplicative factor of β (typically $\beta = \frac{1}{2}$).

This scheduling problem is understood quite well when the network is restricted to a single bottleneck. In this case, there is a limited amount B of resource, be it bandwidth or processors, which must be allocated at each point in time to a set of online jobs. We mention processors here only because this is the context within which the scheduling community tends to write [3]. The standard measure of the successful utilization of the resource in both the systems and the scheduling communities is the average flow/response/waiting time of the jobs, computed as $\text{Avg}_{i\in\mathcal{J}}[c_i - a_i]$. The optimal scheduling algorithm is Shortest Remaining Job First which gives all B of the bandwidth/processors to the job which is closest to completing. This algorithm is online, in that it does not need to know what jobs will arrive in the future, but it is not non-clairvoyant in that it needs to know the amount of work remaining in the jobs currently in the system. A common non-clairvoyant scheduling algorithm is EQUI which simply allocates, at each point in time t, to each job currently alive, its share $\frac{B}{n_t}$ of bandwidth/processors. Though this is fair to all users, [32] proves the *competitive ratio* of this online, non-clairvoyant scheduler can be as bad as $\Theta(\frac{n}{\log n})$ when measured against the optimal all-powerful, all-knowing, off-line scheduler, which in this case is Shortest Remaining Work First. When there is such a negative result, a typical way to prove that the scheduler does perform well is to give it some extra resources before comparing it to the optimal scheduler, [23]. (See Section 5 for

* Tel.: +1 416 736 2100; fax: +1 416 736 5872.
E-mail addresses: jeff@cse.yorku.ca, jeff@cs.yorku.ca.

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.07.033

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

additional motivation.) [9] does this proving that EQUI is $(2+\epsilon)$-speed $\mathcal{O}(1+\frac{1}{\epsilon})$-competitive, meaning that when EQUI is given $2+\epsilon$ times as much bandwidth/processors, it performs within a constant as well as the optimal.

Another complication is that the setting in which the AIMD algorithm is intended is distributed in that each job/sender has no global knowledge of the state of the other jobs in the network. The algorithm EQUI, which instantly reallocates the amount of bandwidth/processors to each job as jobs arrive and complete, is not implementable in this setting. In contrast, AIMD can be implemented distributively as long as each job/sender receives feedback when the bottleneck is at its capacity. Despite this restriction, [7] proves the AIMD algorithm quickly reconverges to EQUI. AIMD, however, is different from EQUI in that its allocations continually increase and decrease and it takes some (small) time for it to reconverge after jobs arrive or depart. [11] proves that if AIMD is given a constant number of adjustment periods per job to converge then it is also $\mathcal{O}(1)$-speed $\mathcal{O}(1)$-competitive.

The main purpose of this paper is to extend these results to the multi-bottleneck case. Unlike [11], which takes into account the fact that AIMD constantly increases additively and decreases multiplicatively, for the multi-bottleneck case, we want to simplify AIMD in two ways. The first simplification is to smooth out the quick changes using approximating differential equations. Because the resulting algorithm is still distributed, the allocations still change gradually when a job arrives or leaves, unlike EQUI which instantly knows to reallocate. But just as the single-bottleneck AIMD quickly converges to the global allocation state given by EQUI when the job set does not change, it is our strong belief that the multi-bottleneck AIMD also quickly converges to some global allocation state. Surprisingly, however, there has not previously been a description of this converged to allocation. Kelly et al. in [27,25] does a good job, but the algorithm they consider is different. In their AIMD, the frequency at which a bottleneck drops packets, instructing its jobs to decrease their bandwidth changes as a fixed function that depends only on the current total traffic through the bottleneck in question. In contrast, in the standard AIMD algorithm for TCP, a bottleneck instructs its jobs to back off only when it reaches its capacity. The frequency at which this occurs is a much more complex function of what the other bottlenecks are doing. In Section 3, we define a new continuous model of how AIMD evolves on a general network within this setting and also define the scheduler, AIMDEQUI, to be that to which it converges.

This paper then proves that this global on-line non-clairvoyant multi-bottleneck rate allocation algorithm AIMDEQUI is competitive when given extra resources. Our proof technique is to reduce this problem to proving that the algorithm EQUI in the single-bottleneck case is competitive. The result in [9], which was actually written for the processor allocation problem, then applies directly. Comparing AIMDEQUI to EQUI requires some notion of the *fairness* of bandwidth allocation. Because different jobs pass through different bottlenecks with different capacities, such a notion is not clearly defined. The standard definition of fairness is referred to as *max–min fairness* [12,13,17,19,20]. We call this the *socialistic* view of fairness because it attempts to give each job the same bandwidth. In Section 4, we consider two other notions of fairness as well. *Local* fair is similar but considers only local information. *Free Market* fair penalizes jobs that use many bottlenecks that are in high demand.

According to a *socialistic* view of fairness, [14] proves that AIMD can be unfair by a factor of m, where m is a bound on the number of bottlenecks that a job goes through. Whether this is tight is open. We do, however, show that according to a *local* view of fairness, it is never more than a factor of m unfair and that according to a *free market* view, it is perfectly fair.

Finally, Section 5 proves that AIMDEQUI is $\mathcal{O}(m^3)$-speed $\mathcal{O}(m)$-competitive, meaning that with $\mathcal{O}(m^3)$ times the bandwidth, the flow time under AIMDEQUI is within a factor of $\mathcal{O}(m)$ of that of the optimal all-knowing scheduler. We believe that it is reasonable to assume that m is a constant because within the actual internet no transmission hops more than half a dozen times. We are also able to prove that AIMDEQUI is $\mathcal{O}(1)$-speed $\mathcal{O}(1)$-competitive independent of m. However, this result requires the assumption that the adjustment frequencies of the bottlenecks do not change much within the life of an individual job. This we believe is a reasonable assumption because the adjustment frequencies are a global property that should not be greatly affected by the arrival and the completion of individual jobs. We believe that the result is true without this assumption or minimally when given speed $s=\mathcal{O}(m)$, however, as of yet this has been unattainable.

There has been little work done for AIMD within multi-bottleneck networks. Some, however, has been done for more general bandwidth scheduling algorithms [2,4–6,12,13,17,18,21,22,31]. Hahne [17,18] proves that if each bottleneck/router relays the packets of the jobs in a round-robin way, then the bandwidths converge to max–min fairness between the jobs. However, they do not consider the worst case packet arrival or jobs arriving and leaving, but assume either Bernoulli packet arrivals or the case in which there are always packets waiting to enter the system. Fatourou et al. [12,13] prove that another class of algorithms converge to max–min fairness. These, however, have a global scheduler dictate an order in which the jobs "update", where such an update requires the job to tell any other job that shares a bottleneck with it to decrease its bandwidth.

2. Scheduling models and competitive ratio

In this section, we review the formal definitions used by the scheduling community. Generally, they consider the problem of partitioning the time of a single processor amongst the online jobs currently alive. Each job completes in time proportional to the fraction of the processors' time allocated to it. This problem maps directly onto the single-bottleneck version of the problem considered in this paper, namely the problem of partitioning the bandwidth of a single bottleneck among the jobs. Edmonds [9] extends this notion to that of scheduling multiple processors to jobs which unbeknownst to the scheduler may or may not be able to effectively utilize all of the processors given to it. Similarly in the problem of transmitting files, each

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

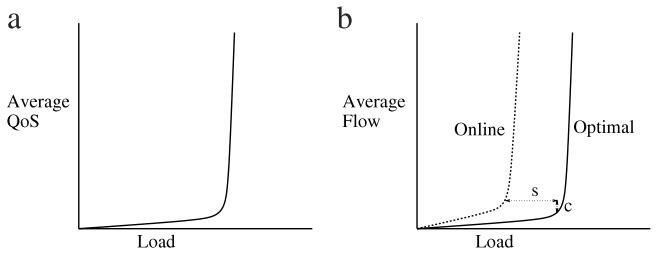


Fig. 1. (a) Standard performance curve, and (b) The worst possible performance curve of an s-speed c-competitive online algorithm.

sender may have a different upper bound on the rate at which it can transmit data. This can be modeled by representing the transmission with a job whose speedup function is fully parallelizable up to the senders capacity and then becomes sequential for any additional bandwidth allocated to it, namely $\llcorner$. In such ways, this paper does use the stronger result about arbitrary speed-up functions in its reduction to the bandwidth problem. For this reason, we will start with a review of definitions introduced in [9].

An instance consists of a collection $\mathcal{J} = \{J_1, \ldots, J_n\}$ where *job* J_i has a *release/arrival time* a_i and a sequence of phases $\langle J_i^1, J_i^2, \ldots, J_i^{q_i}\rangle$. Each phase is an ordered pair $\langle w_i^q, \Gamma_i^q\rangle$, where w_i^q is a positive real number that denotes the amount of *work* in the phase and Γ_i^q is a function, called the *speed-up function*, that maps a nonnegative real number to a nonnegative real number. $\Gamma_i^q(p)$ represents the rate at which work is processed for phase q of job J_i when run on p processors running at speed 1. If these processors are running at speed s, then work is processed at a rate of $s\Gamma_i^q(p)$.

A phase of a job is *parallelizable* if its speed-up function is $\Gamma(p) = p$. Increasing the number of processors allocated to a parallelizable phase by a factor of s increases the rate of processing by a factor of s. A phase is *sequential* if its speed-up function is $\Gamma(p) = 1$, for all $p \geq 0$. The rate that work is processed in a sequential phase is independent of the number of processors, even if it is zero. A speed-up function Γ is *nondecreasing* if and only if $\Gamma(p_1) \leq \Gamma(p_2)$ whenever $p_1 \leq p_2$. A speed-up function Γ is *sublinear* if and only if $\Gamma(p_1)/p_1 \geq \Gamma(p_2)/p_2$ whenever $p_1 \leq p_2$. We assume all speed-up functions Γ in the input instance are nondecreasing and sublinear.

A schedule specifies for each time, and for each job, a nonnegative real number specifying the number of processors assigned to that job. The total number of processors assigned at any time can be at most the number of processors. A *nonclairvoyant* algorithm only knows the past arrival and completion of jobs. In particular, a nonclairvoyant algorithm does not know the current phase q, its work w_i^q, or its speed-up function Γ_i^q.

The *completion time* of a job J_i, denoted c_i, is the first point of time when all the work of the job J_i has been processed. Note that in the language of scheduling, we are assuming that preemption is allowed, that is, a job maybe be suspended and later restarted from the point of suspension. The *response/flow time* of job J_i is $c_i - a_i$, which is the length of the time interval during which the job is active. Let n_t be the number of active jobs at time t. Another formulation of total flow time is $\int_0^\infty n_t \delta t$.

Let A be an algorithm and $\mathcal{J}$ an instance. We denote the schedule output by A with speed s processors on $\mathcal{J}$ as $A_s(\mathcal{J})$. Let $Opt(\mathcal{J})$ be the optimal schedule with unit speed processors on input $\mathcal{J}$. We let $F(S)$ denote the total response time incurred in schedule S,

To understand the worst-case analysis results in the literature, we need to introduce and motivate resource augmentation analysis [23]. A scheduling algorithm A is said to be s-speed c-competitive if

$$\max_{\mathcal{J}} \frac{A_s(\mathcal{J})}{\mathrm{OPT}_1(\mathcal{J})} \leq c$$

where $A_s(\mathcal{J})$ denotes the average flow time for the schedule given by A with a speed s on input $\mathcal{J}$, and similarly $\mathrm{OPT}_1(\mathcal{J})$ denotes the flow time of the adversarial schedule for $\mathcal{J}$ with a unit speed.

Our analysis philosophy is to put first priority on minimizing the speed, while keeping the competitive ratio reasonable. The reason for this is that average QoS curves such as those in Fig. 1(a) are ubiquitous in server systems [28]. That is, the average QoS at loads below capacity is negligible, and the average QoS above capacity is intolerable. The concept of load is not so easy to formally define, but generally reflects the number of users of the system. So in some sense, one can specify the performance of such a system by simply giving the value of the capacity of the system. In this setting, $A_s(\mathcal{J})$ is at most c times optimal average flow time with s times higher load, since slowing down the speed by a factor of s is the same as increasing the load by a factor of s. But since the optimal flow time is almost always negligible or intolerable, a modest c times either negligible or intolerable, still gives you negligible or intolerable. So an s-speed c-competitive algorithm should perform reasonably well up to load $1/s$ of the capacity of the system as long as c is of modest size. Thus usually the goal is to find a server scheduling algorithm that is $(1+\epsilon)$-speed $O(1)$-competitive; We call such an algorithm *almost fully scalable* since it should perform well up to almost peak load.

Though most scheduling papers consider the allocation of a fixed number of processors between the active jobs, the results hold for our setting of allocating the fixed bandwidth of a single-bottleneck network. It is shown in [9] that the algorithm, EQUI, which devotes an equal amount of processing power to each job, is a $(2+\epsilon)$-speed $O(1+1/\epsilon)$-competitive

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

algorithm for scheduling of jobs with "natural" speed-up curves. The result in the original paper [9] stated $\frac{\text{EQUI}_s(\mathcal{J})}{\text{OPT}_1(\mathcal{J})} \leq \frac{2s}{(s-2)}$. This was improved in [10] for the purpose of proving Theorem 2 to $1+\mathcal{O}(\frac{\sqrt{s}}{s-2})$, which does not change the result $\mathcal{O}(\frac{1}{\epsilon})$ when the speed s is $2+\epsilon$, but when the speed s is large, the improvement is from $2+\mathcal{O}(\frac{1}{s})$ to $1+\mathcal{O}(\frac{1}{\sqrt{s}})$. It is likely that the competitive ratio should be $1+\mathcal{O}(\frac{1}{s})$, but as of now that is unattainable. Another improvement needed to prove Theorem 2 that [10] provides over [9] is that it allows the optimal scheduler to complete the fully parallelizable work and the sequential work independently. The formal statement needed is as follows.

Theorem 1 (*[10]*). *Let $\mathcal{J}$ be any set of jobs in a single-bottleneck network in which each phase of each job can have an arbitrary sublinear-nondecreasing speed-up function.* $\frac{\text{EQUI}_s(\mathcal{J})}{\text{OPT}_1(\mathcal{J}_{par})+\text{OPT}_1(\mathcal{J}_{seq})} \leq 1+\mathcal{O}(\frac{\sqrt{s}}{s-2})$, *where* $\text{OPT}_1(\mathcal{J}_{par})$ *is the flow of the optimal schedule to complete only the parallelizable phases* $\mathcal{J}_{par}$ *of the jobs in* $\mathcal{J}$ *and* $\text{OPT}_1(\mathcal{J}_{seq})$ *only the sequential phases.*

3. The continuous AIMD model for general networks

In this section, we propose two new models of AIMD through a general network. The first model is a set of differential equations similar to those given by Kelly in [27,25]. We argue, however, that ours is a better model of how changes in one part of the network can affect other parts. Unlike Kelly, however, we are unable to prove that the system converges, though we have strong arguments that it does. To avoid this problem, we will simply define another model, denoted AIMDEQUI, which is the previous model at its steady state. It is this second model that we prove is competitive against the optimal bandwidth scheduling algorithm. We use the following notation:

- $\mathcal{B}$ is a set of routers that act as bottlenecks, the kth of which has maximum bandwidth B_k. When the scheduler has "speed" s, this maximum bandwidth of each bottleneck is increased to $s \cdot B_k$.
- $\mathcal{J} = \{J_i\}$ is the set of jobs (or sessions). Each job J_i is defined by its arrival time a_i, its file length l_i, and as done in [27,25], the subset of the bottlenecks $\mathcal{B}_i$ that it passes through. Conversely let $\mathcal{J}_k$ denote the set of jobs J_i that pass through the kth bottleneck and $\mathcal{J}_{\langle k,t\rangle}$ to be those active at time t. Note that as a simplifying assumption, we are ignoring the path that a job takes through these bottlenecks and any delays caused by transmission times. In particular, we are ignoring the fact that different jobs may have different transmission times.
- We denote by $b_{\langle i,t\rangle}$ the bandwidth or transmission rate used by job J_i at time t. The restriction for the kth bottleneck is that $\sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle} \leq sB_k$.

  As done in [1], AIMD is not represented at the packet level but via simple fluid equations. Though AIMD decreases the bandwidth allocation of a session non-continuously at discrete points in time, we will amortize this change to make all the equations continuous, as done in [27,25].
- We denote by c_i the time that the transmission of job J_i is completed. To accomplish this, the algorithm must allocate enough bandwidth so that $\int_{t\in[a_i,c_i]} b_{\langle i,t\rangle} = l_i$.
- We measure the quality of a scheduling algorithm using the average flow/response/waiting time of the jobs, i.e. $\text{Avg}_{i\in\mathcal{J}}[c_i - a_i]$.
- α is the *additive increase* and β the *multiplicative decrease* parameter set by the AIMD algorithm. Namely, each user increases his transmission rate linearly at a constant rate of $\delta b_{\langle i,t\rangle}/\delta t = \alpha$ (typically $\alpha = 1$) until he detects that one of the bottlenecks that his transmission passes through has reached capacity. At this point, the sender cuts his own rate $b_{\langle i,t\rangle}$ by a multiplicative factor of β (typically $\beta = \frac{1}{2}$).
- $f_{\langle k,t\rangle}$, the *adjustment frequency*, will denote the instantaneous frequency at time t at which the event occurs in which the kth bottleneck reaches capacity and instructs its users to back off.

The equations relating these values are as follows:

$$\forall \text{ bottlenecks } k \quad \left(f_{\langle k,t\rangle} \geq 0 \text{ and } \sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle} = sB_k\right) \text{ or } \left(f_{\langle k,t\rangle} = 0 \text{ and } \sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle} < sB_k\right) \tag{1}$$

$$\forall \text{ active jobs } i \quad \frac{\delta b_{\langle i,t\rangle}}{\delta t} = \alpha - (1-\beta) b_{\langle i,t\rangle} \sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle} \tag{2}$$

Eq. (1) states that the total bandwidth $\sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle}$ through the kth bottleneck is bounded by its capacity sB_k. Moreover, this bottleneck instructs its users to back off if and only if it is at capacity. Eq. (2) states that each job J_i continually increases his bandwidth linearly at a rate of $\delta b_{\langle i,t\rangle}/\delta t = \alpha$ and approximates the effect of the multiplicative decreases. When any one of the bottlenecks that J_i passes through reaches capacity, its bandwidth $b_{\langle i,t\rangle}$ decreases by a multiplicative factor of β, i.e. from $b_{\langle i,t\rangle}$ to $\beta b_{\langle i,t\rangle}$, which is a decrease of $(1-\beta)b_{\langle i,t\rangle}$. The number of times that this occurs during a time period of length δt is $\left[\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle}\right]\delta t$ for a total decrease of $\left[(1-\beta)b_{\langle i,t\rangle}\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle}\right]\delta t$. Clearly, Eq. (2) is only a differential approximation of the decreases that occur at discrete points in time. This same approximation was made in [27,25]. As was said, the advantage of this approximation is that it smooths out the constant oscillations between a bottleneck reaching full capacity and then

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

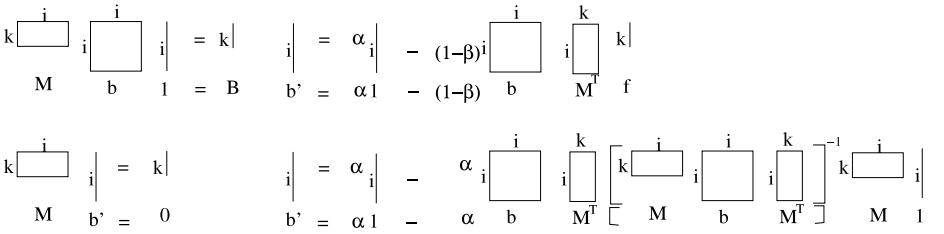


Fig. 2. Matrix representation for computing next $b_{\langle i,t\rangle}$.

suddenly having all of its allocations decrease by a factor of β. In [11], we prove that, at least for the single-bottleneck case, the effect of these oscillations can be compensated by multiplicative changing the capacity by a constant.

The main difference between this model and Kelly's in [27,25] is that Kelly has a single equation $(f_{\langle k,t\rangle}) = \mu_k = p_k(\sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle})$ defining a bottleneck's adjustment frequency $f_{\langle k,t\rangle}$ as a function p_k of the total flow $\sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle}$ through the bottleneck. Though Kelly defines μ_k instead to be "the proportion of marked packets", it is used in the same way in Eq. (2) as we do and we assume that this quantity reflects the proportion of the jobs passing through the bottleneck that will adjust and hence is related to our frequency $f_{\langle k,t\rangle}$. Moreover, Kelly does not speak of the bottlenecks having a capacity, but presumably this fixed non-negative, continuous, strictly increasing function p_k can be such that as this total flow increases towards the bottleneck's capacity, a sufficiently strong message is given to the jobs to back off that this capacity is never exceeded.

In contrast, our model does not have a single equation defining a bottleneck's adjustment frequency $f_{\langle k,t\rangle}$. We feel that this is a better model for AIMD when it is driven by bottleneck capacities, because when an individual bottleneck adjusts in practice does depend in an intricate way on when the other bottlenecks adjust. For example, having a job pass through a long line of m bottlenecks with the same capacities, should be equivalent to passing through only one. In Kelly's model, each of these bottlenecks will send the same message as if it were the only bottleneck and hence the job will back off m times more often. On the other hand, in our model, it is irrelevant and undefined which one of the bottlenecks will adjust. We can only make claims about $\sum_k f_{\langle k,t\rangle}$.

Given the current bandwidth allocations $b_{\langle i,t\rangle}$, the next values are determined by first solving a system of equations for the adjusting frequencies $f_{\langle k,t\rangle}$ and then using these to compute $\delta b_{\langle i,t\rangle}/\delta t$. The following matrix notation is useful. Let M denote the 0/1 matrix such that $M_{\langle k,i\rangle} = 1$ iff the ith job is in the kth bottleneck. Similarly, define the vectors $B = \langle B_k\rangle$, $f = \langle f_{\langle k,t\rangle}\rangle$, $b' = \langle \delta b_{\langle i,t\rangle}/\delta t\rangle$, $0_K = \langle 0,\ldots,0\rangle$, and $1_n = \langle 1,\ldots,1\rangle$. In contrast, represent the bandwidths $b_{\langle i,t\rangle}$ as an $n\times n$ matrix b with diagonals $b_{\langle i,t\rangle}$ and the rest zero. (For ease of notation, we drop the t subscript on f, b', and b.) Eqs. (1) and (2) translate into $Mb1_n = sB$ and $b' = \alpha 1_n - (1-\beta)bM^Tf$. Note there is one equation and one unknown b'_i and $f_{\langle k,t\rangle}$ for each job and for each bottleneck. We can solve these as follows. Differentiating the first gives $Mb' = 0_K$. Substituting the second into this gives $M(\alpha 1_n - (1-\beta)bM^Tf) = 0_K$ or $\alpha M1_n = (1-\beta)MbM^Tf$. Solving this gives the required values $f = \frac{\alpha}{(1-\beta)}(MbM^T)^{-1}M1_n$. These values are used in $b' = \alpha 1_n - (1-\beta)bM^Tf = \alpha 1_n - \alpha bM^T(MbM^T)^{-1}M1_n$ to compute b'. These in turn gives us the next values for $b_{\langle i,t\rangle}$, namely $b_{\langle i,t+\delta t\rangle} = b_{\langle i,t\rangle} + b'_{\langle i,t\rangle}\delta t$. (This can't easily be represented as a matrix because b is square and b' is a vector.) (See Fig. 2.)

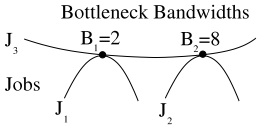
The steady state of this system occurs when $\delta b_{\langle i,t\rangle}/\delta t = 0$. Eq. (2) then gives $b_{\langle i,t\rangle} = \frac{\alpha}{(1-\beta)}/\left(\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle}\right)$. It is our strong belief that this system quickly converges to this state. If the dynamic system allocates job J_i an amount that is different from this then Eq. (2) automatically moves it closer. Assume, for example that job J_i just arrived and hence, $b_{\langle i,t_0\rangle} = 0$. If we assume that the total frequency $f_{\langle i,t\rangle} = \sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle}$ remains relatively constant for a few adjustment periods, then the single differential equation $\delta b_{\langle i,t\rangle}/\delta t = \alpha - (1-\beta)b_{\langle i,t\rangle}f_{\langle i,t\rangle}$ can be solved in isolation from the others, giving $b_{\langle i,(t_0+d)\rangle} = \frac{\alpha}{1-\beta}\frac{1}{f_{\langle i,t\rangle}}(1-e^{-(1-\beta)f_{\langle i,t\rangle}d})$. The time until the AIMD allocation to the job is within a factor $1-e^{-(1-\beta)q} \approx 1-\beta^{-q}$ of the steady state allocation is $d_i = \frac{q}{f_{\langle i,t\rangle}}$. In the single-bottleneck case, this equals q adjustment periods, which corresponds exactly to the results given in [11]. This gives that for the single-bottleneck case, this system quickly converges. Whether this occurs is not as clear (at least for me) for the multi-bottleneck case. To avoid this problem, we will simply define another model, denoted AIMDEQUI, which is the previous model at its steady state. Replacing Eq. (2) with Eq. (4) gives the equations defining AIMDEQUI to be:

$$\forall \text{ bottlenecks } k \quad \left(f_{\langle k,t\rangle} \geq 0 \text{ and } \sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle} = sB_k\right) \text{ or } \left(f_{\langle k,t\rangle} = 0 \text{ and } \sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle} < sB_k\right) \tag{3}$$

$$\forall \text{ active jobs } i \quad b_{\langle i,t\rangle} = \frac{\alpha}{(1-\beta)}\Big/\left(\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle}\right) \tag{4}$$

Note that once one determines which bottlenecks are at capacity, there is one equation and one unknown $b_{\langle i,t\rangle}$ and $f_{\langle k,t\rangle}$ for each job and bottleneck. The simplest example, of course, is when there is a single bottleneck. Eq. (4) states that this single

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033



Allocated Bandwidths	b_1	b_2	b_3
Socialistic Fairness	1	7	1
Local Fairness	1	4	1
Free Market Fairness	1.07	7.07	0.93

Fig. 3. The following example demonstrates Socialistic, Local, and Free Market Fairness. There are two bottlenecks with capacities $B_1 = 2$ and $B_2 = 8$. There are three jobs with J_1 passing through B_1, J_2 through B_2 and J_3 through both. Socialistic Fairness allocates $b_1 = b_3 = 1$ bandwidth to jobs J_1 and J_3, fairly splitting the bandwidth of $B_1 = 2$. Job J_2 is given the remaining B_2 bandwidth $b_2 = B_2 - b_3 = 8 - 1 = 7$ because doing so does not hurt the other jobs. Local Fairness allocation is the same in this example, except that J_2 is only given his fair share of B_2's bandwidth which is $b_2 = \frac{B_2}{n_2} = \frac{8}{2} = 4$. Note that the second bottleneck has $B_2 - b_2 - b_3 = 8 - 4 - 1 = 3$ bandwidth left unused. AIMDEQUI or Free Market Fairness allocates price f_1 to the bandwidth of the first bottleneck and f_2 to that of the second. The key requirements are that the bottlenecks are at capacity, giving $b_1 + b_3 = B_1 = 2$ and $b_2 + b_3 = B_2 = 8$ and that all players pay the same for their bandwidth giving that $f_1 \cdot b_1 = f_2 \cdot b_2 = (f_1 + f_2) \cdot b_3 = \frac{\alpha}{(1-\beta)}$. Changing this constant simply scales the prices and hence wlog $\frac{\alpha}{(1-\beta)} = 1$. This gives us a total of $2 + 3 = 5$ equations and $2 + 3 = 5$ unknowns. Solving them gives us that the price for the bandwidths are $f_1 = 0.96$ and $f_2 = 0.14$. Note that as expected the bandwidth through B_1 is much more expensive because there is less of it. J_3 gets $b_3 = 0.93$ bandwidth, which is less than the others because he is paying more. This leaves $b_1 = 1.07$ and $b_2 = 7.07$ for the other jobs.

bottleneck would charge some fixed amount $f_{\langle 1,t\rangle}$ for its bandwidth and all jobs would be allocated an amount so that they all pay the same, giving that all allocations are the same $b_{\langle i,t\rangle} = \frac{\alpha}{(1-\beta)f_{\langle 1,t\rangle}} = \frac{sB_1}{n_t}$. A slightly more complex example of such a calculation appears in Fig. 3. In the matrix notation, these translate into $Mb1_n = B$ and $bM^T f = \frac{\alpha}{(1-\beta)}1_n$.

4. Socialistic, local, and free market views of fairness

It is clear what a fair distribution is of a single resource like the bandwidth of a bottleneck. However, when different jobs are restricted by different bottlenecks with different capacities, it is not clear what is "fair". This section defines three views of fairness: *Socialistic*, *Local*, and a *Free Market*, with corresponding "Equal Partition" schedulers: S-EQUI, L-EQUI, and F-EQUI. AIMDEQUI will be evaluated with respect to each. See Fig. 3.

We refer to the standard definition of fairness, *max–min fairness* [12,13,17,19,20] as the *socialistic* view of fairness because it attempts to give each job the same bandwidth. It allocates the bandwidths in the unique way so that no job could be allocated more bandwidth without decreasing that of some other job which has the same or less. S-EQUI achieves such a distribution of bandwidth as follows. Starting with zero bandwidth to each job, increase the bandwidth of each job equally, except fixing that to any job passing through a bottleneck that is at capacity. According to this view, [14] proves that AIMD can be unfair by a factor of m to jobs that pass through m bottlenecks. An open problem is to prove that this is the worst case.

In the *local* view, a bottleneck never gives a job more bandwidth than is fair from its local information. In the scheduler L-EQUI, the kth bottleneck tries to allocate a fair share $\frac{sB_k}{n_{\langle k,t\rangle}}$ of its bandwidth to each of the $n_{\langle k,t\rangle} = |\mathcal{J}_{\langle k,t\rangle}|$ jobs that pass through it. A job, however, may not be able to receive this high a bandwidth because of the constraints of its other bottlenecks. Therefore, L-EQUI allocates to job J_i the minimum allocated by each of the bottlenecks through which it passes, i.e. $b_{\langle i,t\rangle} = \min_{k\in\mathcal{B}_i} \frac{sB_k}{n_{\langle k,t\rangle}}$. This locality of the fairness is used to reduce a schedule on the general network $\mathcal{B}$ to one separate single-bottleneck network for each of $\mathcal{B}$'s bottlenecks. Using this, Theorem 3 proves that though L-EQUI sometimes allocates less bandwidth than it could, it is $\mathcal{O}(m^2)$-speed $\mathcal{O}(m)$-competitive. The same result automatically applies for S-EQUI because it never allocates less bandwidth to any job. Lemma 3 proves that AIMDEQUI allocates at least $\frac{1}{m}$ as much. Theorem 2, stating that AIMDEQUI is $\mathcal{O}(m^3)$-speed $\mathcal{O}(m)$-competitive, follows.

We will now argue that the natural definition of *free market fair* is accomplished exactly by our algorithm AIMDEQUI. This view of fair argues that it is not fair to allocate the same bandwidth to every job when the jobs pass through different numbers of bottlenecks with different demands on their bandwidth. Instead, in this view each job is charged by each bottleneck it passes through for the bandwidth that it uses. In a supply and demand way the market fluctuates so that each job is allocated the same cost of bandwidth. This system has $K + n_t$ unknowns and $K + n$ equations, where K is the number of bottlenecks and n_t is the number of jobs active at time t. As such, there should be a unique solution to the equations. More specifically, for each bottleneck, there is the price $f_{\langle i,t\rangle}$ at which its bandwidth is charged and for each job J_i, there is bandwidth $b_{\langle i,t\rangle}$ that it is allocated. Any bottleneck that is fully utilized has its bandwidth fully allocated, i.e. has $\sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle} = sB_k$. On the other hand, any bottleneck that is not fully utilized should not, by supply and demand, charge much for its bandwidth. In fact, such a bottleneck is not really acting like a bottleneck at all and hence should be discounted, giving that it charges $f_k = 0$ for its bandwidth. Note that this requirement is exactly expressed by Eq. (4) for AIMDEQUI. The other requirement is that each job is charged the same amount. Being charged for its bandwidth by each bottleneck it passes through, Job J_i is charged a total of $(\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})b_{\langle i,t\rangle}$. Eq. (4) then enforces that the allocations of bandwidth are such that this charge is the same for all jobs.

The classic property of economics is that each price f_k decreases proportional to the *supply*, namely its capacity sB_k, and increases proportional to the *demand*, namely the number of jobs $n_{\langle k,t\rangle} = |\mathcal{J}_{\langle k,t\rangle}|$ passing through it or perhaps on the number $n^{\max}_{\langle k,t\rangle}$ that are constrained the most by it. The adjustment frequency $f_{\langle k,t\rangle}$ of a bottleneck has this property because

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

Lemma 2 proves that it is bounded within $\frac{\alpha}{(1-\beta)}[\frac{1}{m}\frac{n^{\max}_{\langle k,t\rangle}}{sB_k}, \frac{n_{\langle k,t\rangle}}{sB_k}]$ and Lemma 1 proves equality of this relationship on average, i.e. $n_t = \frac{s(1-\beta)}{\alpha}\sum_k f_{\langle k,t\rangle}B_k$.

The global aspect of this view of fairness is used to reduce AIMDEQUI on the entire network $\mathcal{B}$ to a single network with a single bottleneck. This is used to prove Theorem 4, which states that AIMDEQUI is $\mathcal{O}(1)$-speed $\mathcal{O}(1)$-competitive when these adjustment frequencies $f_{\langle k,t\rangle}$ do not change much within the life of an individual job.

5. The competitiveness of AIMDEQUI

The main result of this paper is that AIMDEQUI despite being online, non-clairvoyant, and distributed is $\mathcal{O}(m^3)$-speed $\mathcal{O}(m)$-competitive.

Theorem 2. *Let $\mathcal{B}$ be any general network. Let $\mathcal{J}$ be any set of jobs in which each phase of each job can have an arbitrary sublinear-nondecreasing speed-up function. Let m denote the maximum number of bottlenecks that a job passes through. It follows that* $\frac{\text{AIMDEQUI}_{\mathcal{O}(m^3)}(\mathcal{J})}{\text{OPT}_1(\mathcal{J})} \le \mathcal{O}(m)$.

Proof of Theorem 2. The result follows from Theorem 3 stating that LEQUI is $\mathcal{O}(m^2)$-speed $\mathcal{O}(m)$-competitive and from Lemma 3 stating that AIMDEQUI allocates at least $\frac{1}{m}$ as much bandwidth as LEQUI to each job. □

Recall that the scheduling algorithm LEQUI, so that no bottleneck ever gives a job more bandwidth than is fair from its local information, allocates to job J_i the bandwidth $b_{\langle i,t\rangle} = \min_{k\in\mathcal{B}_i}\frac{sB_k}{n_{\langle k,t\rangle}}$. This notation is used now to reduce a schedule on the general network $\mathcal{B}$ to one separate single-bottleneck network for each of $\mathcal{B}$'s bottlenecks.

Theorem 3. $\frac{\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{J})}{\text{OPT}_1(\mathcal{J})} \le \mathcal{O}(m)$.

Proof of Theorem 3. This proof uses the fact that LEQUI is locally fair at each bottleneck. It is a reduction to many instances of Theorem 1. For each bottleneck within the general network $\mathcal{B}$, the proof reduces what occurs in that bottleneck to a separate single bottleneck network with capacity B_k on a job set defined below which we will denote $\mathcal{J}^k$. Because these networks have a single bottleneck, it is equivalent to think of them as a set of jobs $\mathcal{J}^k$ completed by B_k processors using the scheduling algorithm Equal Partition. Also note the difference between the notations $\mathcal{J}_k$ and $\mathcal{J}^k$. $\mathcal{J}_k$ consists of those jobs in the general network $\mathcal{B}$ that pass through the kth bottleneck. In contrast, $\mathcal{J}^k$ consists of those jobs in the kth single bottleneck network with capacity B_k. Though there is a one-to-mapping between these jobs, they should be differentiated between. In many contexts, we will used the notation $A_s(M,\mathcal{J})$ (for example $\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ or $\text{EQUI}_{\mathcal{O}(m^2)}(B_k,\mathcal{J}^k)$) to be the flow time, where M is the model, $\mathcal{J}$ is the set of jobs being scheduled, A is the scheduling algorithm, and s is the speed of this algorithm. The steps used in this proof can be followed in the summery at the bottom.

Given $\mathcal{B}$ and $\mathcal{J}$ modify them so that each job appears in exactly m bottlenecks by putting jobs through new "fake" bottlenecks. Define $\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J}) = \sum_i[c_i^L - a_i]$ to be the total flow/response/waiting time of the jobs, instead of the average. This change does not change the competitive ratio. For the kth bottleneck in $\mathcal{B}$, define $\text{LEQUI}^{\mathcal{J}_k}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J}) = \sum_{i\in\mathcal{J}_k}[c_i^L - a_i]$ to be the same but only for those jobs $\mathcal{J}_k$ that pass through the kth bottleneck. Because each job appears in exactly m bottlenecks, it follows that $\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J}) = \frac{1}{m}\sum_k \text{LEQUI}^{\mathcal{J}_k}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$.

For each bottleneck k, we define a set of jobs $\mathcal{J}^k$ that is a mirror of the set of jobs $\mathcal{J}_k$ that pass through the bottleneck. In this way, each job appears in m of these sets of jobs. However, at each point in time only one of these copies is a true copy with fully parallelizable work. The remaining $m-1$ copies are replaced by sequential work that act more as place holders. The motivation is as follows. Recall that the kth bottleneck in LEQUI attempts to allocate a fair share $(sB_k)/n^L_{\langle k,t\rangle}$ of its bandwidth to each of the $n^L_{\langle k,t\rangle} = |\mathcal{J}_{\langle k,t\rangle}|$ jobs that pass through it at time t. However, those jobs that are constrained by other bottlenecks are unable to utilize all of this bandwidth and are allocated only $b^L_{\langle i,t\rangle} = \min_{k'\in\mathcal{B}_i}(sB_{k'})/n^L_{k',t}$ bandwidth. Hence, it is reasonable for the unconstrained bottlenecks to view such phases of such jobs as being sequential. Recall that sequential jobs complete at a fixed rate even when allocated more resources. More formally, for each job J_i and time t, let $k(i,t)$ denote the index of the bottleneck that constrains the job, i.e. $k(i,t)$ is the k' minimizing $\min_{k'\in\mathcal{B}_i}(sB_{k'})/n^L_{k',t}$. Break ties arbitrarily. For each bottleneck k, let $\mathcal{J}^{\min}_{\langle k,t\rangle} \subseteq \mathcal{J}_{\langle k,t\rangle}$ denote those jobs J_i passing through the kth bottleneck for which $k(i,t) = k$. For jobs J_i in $\mathcal{J}^{\min}_{\langle k,t\rangle}$, the work completed at time t by $\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ will be copied without change to the job $J_i^k \in \mathcal{J}^k$. For the other jobs J_i in $\mathcal{J}_{\langle k,t\rangle} \setminus \mathcal{J}^{\min}_{\langle k,t\rangle}$, a sequential phase is inserted into the job J_i^k so that it completes in the same time that the work completed at time t by $\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ completes.

The next step is to prove that $\text{LEQUI}^{\mathcal{J}_k}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J}) \le \text{EQUI}_{\mathcal{O}(m^2B_k)}(B_k,\mathcal{J}^k)$. Recall that the LHS is the partial sum of the flows associated with the kth bottleneck for the general network when running the algorithm LEQUI with extra speed $\mathcal{O}(m^2)$ while the RHS is the sum of flows associated with the kth single-bottleneck network when running the algorithm EQUI with extra speed $\mathcal{O}(m^2B_k)$. By way of induction on t, suppose that at time t, $\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ has completed at least as much work on each job in $\mathcal{J}_{\langle k,t\rangle}$ as $\text{EQUI}_{\mathcal{O}(m^2B_k)}(B_k,\mathcal{J}^k)$. The bandwidth allocated by $\text{LEQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ to job J_i

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

is $b^L_{\langle i,t\rangle} = \min_{k'\in\mathcal{B}_i}(\mathcal{O}(m^2)B_{k'})/n^L_{k',t}$. If $J_i \in \mathcal{J}^{\min}_{\langle k,t\rangle}$, then this is equal to $(\mathcal{O}(m^2)B_k)/n^L_{\langle k,t\rangle}$. By the induction hypothesis, $\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ is not behind and hence $n^L_{\langle k,t\rangle} \le n^{E,k}_t$, giving that $b^L_{\langle i,t\rangle} \ge (\mathcal{O}(m^2)B_k)/n^{E,k}_t$ which is the bandwidth $b^{E,k}_t$ allocated by $\text{EQUI}_{\mathcal{O}(m^2B_k)}(B_k,\mathcal{J}^k)$. We can conclude that $\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ completes at least as much work on the jobs currently in $\mathcal{J}^{\min}_{\langle k,t\rangle}$. For the other jobs in $\mathcal{J}_{\langle k,t\rangle}$, $\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J})$ may allocate less bandwidth than this because of another bottleneck. $\text{EQUI}_{\mathcal{O}(m^2B_k)}(B_k,\mathcal{J}^k)$, however, will take just as much time on this phase of this job because it is working on a sequential job which takes this same fixed amount of time independent of the bandwidth allocated to it. This completes the proof by induction.

The next inequality $\text{EQUI}_{\mathcal{O}(m^2B_k)}(B_k,\mathcal{J}^k) \le (1+\frac{1}{2m})(\text{OPT}_{B_k}(B_k,\mathcal{J}^k_{par}) + \text{OPT}_{B_k}(B_k,\mathcal{J}^k_{seq}))$ is given by Theorem 1 by setting $s = \mathcal{O}(m^2)$. Recall that $\text{OPT}_{B_k}(B_k,\mathcal{J}^k_{par})$ is the flow of the optimal schedule with extra speed B_k to complete only the parallelizable phases $\mathcal{J}^k_{par}$ of the jobs in $\mathcal{J}^k$ for the kth single-bottleneck network. Similarly, $\text{OPT}_{B_k}(B_k,\mathcal{J}^k_{seq})$ that for only the sequential phases. In contrast, $\text{OPT}_1(\mathcal{B},\mathcal{J}_{par})$ is the flow of the optimal schedule with extra speed 1 to complete only the parallelizable phases $\mathcal{J}_{par}$ of the jobs in $\mathcal{J}$ for the general network $\mathcal{B}$. Similarly, $\text{OPT}_1(\mathcal{B},\mathcal{J}_{seq})$.

The steps $\sum_k \text{OPT}_{B_k}(B_k,\mathcal{J}^k_{par}) \le \sum_k \text{OPT}'_{B_k}(B_k,\mathcal{J}^k_{par}) \le m\text{OPT}_1(\mathcal{B},\mathcal{J}_{par})$ are proved as follows. For each bottleneck k, define $\text{OPT}'_{B_k}(B_k,\mathcal{J}^k_{par})$ to be the single-bottleneck scheduler that allocates each job in $\mathcal{J}^k$ the exact bandwidth that $\text{OPT}_1(\mathcal{B},\mathcal{J}_{par})$ allocates its counterpart in $\mathcal{J}$. Because these jobs all pass through the kth bottleneck, the total amount allocated to these jobs by $\text{OPT}_1(\mathcal{B},\mathcal{J}_{par})$ is at most B_k. Hence, $\text{OPT}'_{B_k}(B_k,\mathcal{J}^k_{par})$ too does not exceed the capacity of its single bottleneck and hence is a valid scheduler. It will follow that the optimal scheduler for $\mathcal{J}^k$ can only be better, i.e. $\text{OPT}_{B_k}(B_k,\mathcal{J}^k_{par}) \le \text{OPT}'_{B_k}(B_k,\mathcal{J}^k_{par})$.

We will now show $\sum_k \text{OPT}'_{B_k}(B_k,\mathcal{J}^k_{par}) \le m\text{OPT}_1(\mathcal{B},\mathcal{J}_{par})$. Because job J^k_i in $\mathcal{J}^k_{par}$ has no more fully parallelizable work than its counterpart in $\mathcal{J}_{par}$, and because the schedulers allocate the same bandwidth to these jobs, it follows that the first completes no later, i.e. $c_i^{\langle O',k\rangle} \le c_i^{\langle O,\mathcal{B}\rangle}$. From this the bound follows. $\sum_k \text{OPT}'_{B_k}(B_k,\mathcal{J}^k_{par}) = \sum_k \sum_{i\in\mathcal{J}_k}(c_i^{\langle O',k\rangle} - a_i) = \sum_i \sum_{k\in\mathcal{B}_i}(c_i^{\langle O',k\rangle} - a_i) \le \sum_i m(c_i^{\langle O,\mathcal{B}\rangle} - a_i) = m\text{OPT}_1(\mathcal{B},\mathcal{J}_{par})$. Greater understanding, however, can be gained by seeing when this extra factor of m is needed and when not. Recall that for each point in time t under $\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J}))$ only one of the m copies of job J_i actually has fully parallelizable work. If the bottleneck $k = k(i,t)$ that constrains job J_i does not change throughout its life, then only one copy of J_i contributes to the sum and $\sum_{k\in\mathcal{B}_i}(c_i^{\langle O',k\rangle} - a_i) = (c_i^{\langle O,\mathcal{B}\rangle} - a_i)$. However, when $k = k(i,t)$ changes over time, the fully parallelizable work from J_i is partitioned between the m copies. If these optimal schedulers allocated some fixed bandwidth during the life of the job, then the same equality, $\sum_{k\in\mathcal{B}_i}(c_i^{\langle O',k\rangle} - a_i) = (c_i^{\langle O,\mathcal{B}\rangle} - a_i)$, would hold. However, suppose that $\text{OPT}_1(\mathcal{B},\mathcal{J}_{par}))$ delays job J_i for a long time after it arrives. Then if the job has a little fully parallelizable work in $\mathcal{J}^k_{par}$ for each of its m bottlenecks, then this delay will contribute m times to the sum and this factor of m is needed.

The next inequality is $\sum_k \text{OPT}_{B_k}(B_k,\mathcal{J}^k_{seq})) = \text{OPT}_1(\mathcal{B},\mathcal{J}_{seq}) + (m-1)\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J}))$. First note that the statement of the theorem allows for arbitrary sublinear-nondecreasing speed-up functions, but it is easy to prove that the worst case is when each phase of each job is either sequential or parallelizable. Then there are two types of sequential work in a set of jobs $\mathcal{J}^k$. The first type was originally in $J_i \in \mathcal{J}$ and was copied as is to $J^{k(i,t)}_i \in \mathcal{J}^{k(i,t)}$. This sequential work appears once in $\sum_k \text{OPT}_{B_k}(B_k,\mathcal{J}^k_{seq}))$ and once in $\text{OPT}_1(\mathcal{B},\mathcal{J}_{seq})$. The other type of sequential work arises because, for each piece of work in $\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(\mathcal{B},\mathcal{J}))$, whether fully parallelizable or sequential, a piece of sequential work lasting the same time is added to J^k_i for the $m-1$ values of k other than $k(i,t)$. The equality follows.

The final inequalities $\text{OPT}_1(\mathcal{B},\mathcal{J}_{par}) \le \text{OPT}_1(\mathcal{B},\mathcal{J})$ and $\text{OPT}_1(\mathcal{B},\mathcal{J}_{seq}) \le \text{OPT}_1(\mathcal{B},\mathcal{J})$ are true because OPT has strictly less work to do in each case.

The above steps result in an expression that can be rearranged to give the last line. All these steps are summarized as follows:

$$\begin{aligned}
\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(G,\mathcal{J}) &= \frac{1}{m}\sum_k \text{L}\mathcal{EQUI}^{\mathcal{J}_k}_{\mathcal{O}(m^2)}(G,\mathcal{J}) \\
&\le \frac{1}{m}\sum_k \text{EQUI}_{\mathcal{O}(m^2B_k)}(B_k,\mathcal{J}^k) \\
&\le \frac{1}{m}\left(1+\frac{1}{2m}\right)\sum_k(\text{OPT}_{B_k}(B_k,\mathcal{J}^k_{par}) + \text{OPT}_{B_k}(B_k,\mathcal{J}_{\langle k,seq\rangle})) \\
&\le \frac{1}{m}\left(1+\frac{1}{2m}\right)(m\text{OPT}_1(G,\mathcal{J}_{par}) + \text{OPT}_1(G,\mathcal{J}_{seq}) + (m-1)\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(G,\mathcal{J})) \\
&\le \left(1+\frac{2}{m}\right)\text{OPT}_1(G,\mathcal{J}) + \left(1-\frac{1}{2m}\right)\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(G,\mathcal{J})) \\
\frac{1}{2m}\text{L}\mathcal{EQUI}_{\mathcal{O}(m^2)}(G,\mathcal{J}) &\le \left(1+\frac{2}{m}\right)\text{OPT}_1(G,\mathcal{J}) \quad \square
\end{aligned}$$

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

Theorem 2 can be completely tightened giving that AIMDEQUI is $(2+\epsilon)$-speed $\mathcal{O}(1)$-competitive if we assume that the adjustment frequencies of the bottlenecks do not to change much within the life of an individual job. This we believe is a reasonable assumption. Recall that the adjustment frequency $f_{\langle k,t\rangle}$ is supposed to represent how the bottleneck k at time t is "charging" all the jobs that pass through it for their bandwidth. This amount is a global property that should not be greatly affected by the arrival and the completion of individual jobs.

Recall that our algorithm AIMDEQUI is perfectly *free market fair*, meaning that each job is charged f_k by each bottleneck it passes through for the bandwidth that it uses. In a supply and demand way the market fluctuates so that each job is allocated the same cost of bandwidth. We now use the global aspect of this view of fairness to reduce AIMDEQUI on the entire network $\mathcal{B}$ to a single network with a single bottleneck. This is used to prove the following.

Theorem 4. *Let $\mathcal{B}$ be any general network. Let $\mathcal{J}$ be any set of jobs in which each phase of each job can have an arbitrary sublinear-nondecreasing speed-up function. Suppose for each job J_i, the ratio $(\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})/(\sum_k f_{\langle k,t\rangle} B_k)$ between adjustment frequencies does not change by more than a factor of r throughout the life of a job, where $r \geq 1$ is some constant. It follows that* $\frac{\text{AIMDEQUI}_{r(2+\epsilon)}(\mathcal{J})}{\text{OPT}_1(\mathcal{J})} \leq \mathcal{O}(1+\frac{1}{\epsilon})$.

Proof of Theorem 4. This proof uses the fact that AIMDEQUI is Free Market Fair. It is a simple reduction to Theorem 1 by reducing everything that is occurring within the general network $\mathcal{B}$ to a single network with a single bottleneck with capacity $B = 1$, namely $\frac{\text{AIMDEQUI}_{r(2+\epsilon)}(\mathcal{B},\mathcal{J})}{\text{OPT}_1(\mathcal{B},\mathcal{J})} \leq \frac{\text{EQUI}_{(2+\epsilon)}(1,\mathcal{J}^f)}{\text{OPT}_1(1,\mathcal{J}^f)} = \mathcal{O}\left(1+\frac{1}{\epsilon}\right)$. Using the same notation $A_s(M,\mathcal{J})$, we have that $\text{AIMDEQUI}_{r(2+\epsilon)}(\mathcal{B},\mathcal{J})$ is the total flow where the model is the general network $\mathcal{B}$, the job set is $\mathcal{J}$, and the algorithm is AIMDEQUI with speed $r(2+\epsilon)$. Similarly, $\text{EQUI}_{(2+\epsilon)}(1,\mathcal{J}^f)$ is the total flow where the model is the single-bottleneck network "1", job set is $\mathcal{J}^f$, which is defined shortly, and the algorithm is EQUI with speed $(2+\epsilon)$. The last step is a direct application of Theorem 1.

Define $F_{\langle i,t\rangle} = (\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})/(\sum_k f_{\langle k,t\rangle} B_k)$ to be a needed comparison between the adjusting frequency of job J_i at time t and that of the overall network. By the statement of the theorem, this does not change by more than r throughout the life of the job and hence $F_i \leq F_{\langle i,t\rangle} \leq rF_i$ for some F_i.

Though the theorem states that each phase of each jobs in $\mathcal{J}$ can have an arbitrary sublinear-nondecreasing speed-up function, [9] proves that in the worst case each phase is either sequential or parallelizable. Hence, we can construct another set of jobs $\mathcal{J}^f = \{J_i^{F_i} \mid J_i \in \mathcal{J}\}$ by taking each job $J_i \in \mathcal{J}$ and creating the job $J_i^{F_i}$ by scaling the work of each parallelizable phase by this constant F_i and keeping each sequential phase the same.

The first step is to prove that $\text{AIMDEQUI}_{r(2+\epsilon)}(\mathcal{B},\mathcal{J}) \leq \text{EQUI}_{(2+\epsilon)}(1,\mathcal{J}^f))$. By induction on t, assume that at time t $\text{AIMDEQUI}_{r(2+\epsilon)}(\mathcal{B},\mathcal{J})$ has completed at least as much work on each job as $\text{EQUI}_{(2+\epsilon)}(1,\mathcal{J}^f)$. We prove as follows that the first algorithm allocates at least $\frac{1}{F_i}$ times more bandwidth to job J_i at this time than the second does, i.e. $b^A_{\langle i,t\rangle} \geq \frac{1}{F_i}\cdot b^E_{\langle i,t\rangle}$. By the bound on $F_{\langle i,t\rangle}$ given by the theorem and that on $b^A_{\langle i,t\rangle}$ given in Eq. (4), $F_i\cdot b^A_{\langle i,t\rangle} \geq \frac{1}{r}F_{\langle i,t\rangle}\cdot b^A_{\langle i,t\rangle} = \frac{1}{r}\left[(\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})/(\sum_k f_{\langle k,t\rangle}B_k)\right]\cdot\left[\frac{\alpha}{(1-\beta)}/(\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})\right] = (2+\epsilon)/\left(r(2+\epsilon)\frac{(1-\beta)}{\alpha}(\sum_k f_{\langle k,t\rangle}B_k)\right)$. By Lemma 1 below, this is $(2+\epsilon)/n^A_t$. By the induction hypothesis $n^A_t \leq n^E_t$ and hence this is at least $(2+\epsilon)/n^E_t$, which is the bandwidth $b^E_{\langle i,t\rangle}$, allocated by $\text{EQUI}_{(2+\epsilon)}(1,\mathcal{J}^f)$. Because $b^A_{\langle i,t\rangle} \geq \frac{1}{F_i}\cdot b^E_{\langle i,t\rangle}$, $\text{AIMDEQUI}_{r(2+\epsilon)}(\mathcal{B},\mathcal{J})$ continues to keep up. The second algorithm has F_i times as much fully parallelizable work and by definition sequential work completes at a fixed rate independent of the number of processors allocated. This completes the proof by induction.

The final step in the proof is to compare the optimal algorithms $\text{OPT}_1(\mathcal{B},\mathcal{J}) \geq \text{OPT}_1(1,\mathcal{J}^f)$. This is done by constructing another algorithm OPT^f_1 for which $\text{OPT}_1(\mathcal{B},\mathcal{J}) = \text{OPT}^f_1(1,\mathcal{J}^f)$. Because $\text{OPT}_1(1,\mathcal{J}^f)$ is the optimal algorithm, $\text{OPT}^f_1(1,\mathcal{J}^f) \geq \text{OPT}_1(1,\mathcal{J}^f)$. $\text{OPT}^f_1(1,\mathcal{J}^f))$ is defined to be the same as $\text{OPT}_1(\mathcal{B},\mathcal{J})$ except that the bandwidth allocated to job J_i is scaled by F_i, i.e. $b^f_{\langle i,t\rangle} = F_i\cdot b^{\mathcal{B}}_{\langle i,t\rangle}$. $\text{OPT}_1(\mathcal{B},\mathcal{J}) = \text{OPT}^f_1(1,\mathcal{J}^f)$, because both the amount of parallel work and the number of processors have been scaled by F_i. What remains is to show that $\text{OPT}^f_1(1,\mathcal{J}^f)$ does not allocate more than a total of one bandwidth at any given time, namely $\sum_i b^f_{\langle i,t\rangle} = \sum_i F_i\cdot b^{\mathcal{B}}_{\langle i,t\rangle} \leq \sum_i F_{\langle i,t\rangle}\cdot b^{\mathcal{B}}_{\langle i,t\rangle} = \sum_i(\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})/(\sum_k f_{\langle k,t\rangle}B_k)\cdot b^{\mathcal{B}}_{\langle i,t\rangle} = (\sum_k f_{\langle k,t\rangle}(\sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b^{\mathcal{B}}_{\langle i,t\rangle}))/(\sum_k f_{\langle k,t\rangle}B_k)$. Because $\text{OPT}_1(\mathcal{B},\mathcal{J})$ cannot exceed the capacity of the kth bottleneck, this is at most $(\sum_k f_{\langle k,t\rangle}(B_k))/(\sum_k f_{\langle k,t\rangle}B_k) = 1$. This completes all the required steps of the proof. □

Lemma 1. *The number n_t of jobs active at time t under* AIMDEQUI_s *is* $n_t = \frac{s(1-\beta)}{\alpha}\sum_k f_{\langle k,t\rangle}B_k$.

Proof of Lemma 1. $n_t = \sum_{\text{active } i} 1$, which by both the left and right sides of Eq. (4) equals $\sum_i\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle}b_{\langle i,t\rangle}\frac{(1-\beta)}{\alpha} = \frac{(1-\beta)}{\alpha}\sum_k f_{\langle k,t\rangle}(\sum_{i\in\mathcal{J}_{\langle k,t\rangle}} b_{\langle i,t\rangle})$, which by Eq. (3) equals $\frac{(1-\beta)}{\alpha}\sum_k f_{\langle k,t\rangle}sB_k$. □

Lemma 2. *For each k, the adjustment frequency for the kth bottleneck is bounded by* $f_{\langle k,t\rangle} \in \frac{\alpha}{(1-\beta)}\left[\frac{1}{m}\frac{n^{\max}_{\langle k,t\rangle}}{sB_k}, \frac{n_{\langle k,t\rangle}}{sB_k}\right]$.

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

Proof of Lemma 2. Multiplying both sides of Eq. (3) by $f_{\langle k,t\rangle}$ gives $sf_{\langle k,t\rangle}B_k = \sum_{i\in\mathcal{J}_{\langle k,t\rangle}} f_{\langle k,t\rangle}b_{\langle i,t\rangle}$ (both for the equality and the inequality versions). $f_{\langle k,t\rangle}b_{\langle i,t\rangle} \le (\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})b_{\langle i,t\rangle}$, which by Eq. (4) is equal to $\frac{\alpha}{(1-\beta)}$. Thus $sf_{\langle k,t\rangle}B_k \le \sum_{i\in\mathcal{J}_{\langle k,t\rangle}} \frac{\alpha}{(1-\beta)} = n_{\langle k,t\rangle}\frac{\alpha}{(1-\beta)}$. Rearranging gives the upper bound $f_{\langle k,t\rangle} \le \frac{\alpha}{1-\beta}\frac{n_{\langle k,t\rangle}}{sB_k}$.

For each job J_i, let $k'(i,t)$ denote the index of the bottleneck with the highest adjusting frequency that the job passes through, i.e. $k'(i,t)$ is the k maximizing $\max_{k\in\mathcal{B}_i} f_{\langle k,t\rangle}$. Let $\mathcal{J}^{\max}_{\langle k,t\rangle} \subseteq \mathcal{J}_{\langle k,t\rangle}$ denote those jobs $J_i \in \mathcal{J}_{\langle k,t\rangle}$ passing through the kth bottleneck for which $k = k'(i,t)$. Let $n^{\max}_{\langle k,t\rangle} = |\mathcal{J}^{\max}_{\langle k,t\rangle}|$ denote the number of such jobs. Similar to that done before, $sf_{\langle k,t\rangle}B_k = \sum_{i\in\mathcal{J}_{\langle k,t\rangle}} f_{\langle k,t\rangle}b_{\langle i,t\rangle} \ge \sum_{i\in\mathcal{J}^{\max}_{\langle k,t\rangle}} f_{\langle k,t\rangle}b_{\langle i,t\rangle}$. Applying the simplification $(mf_{k'(i,t)})b_{\langle i,t\rangle} \ge (\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})b_{\langle i,t\rangle} = \frac{\alpha}{(1-\beta)}$ of Eq. (4) gives that this is at most $\sum_{i\in\mathcal{J}^{\max}_{\langle k,t\rangle}} \frac{1}{m}\frac{\alpha}{(1-\beta)} = n^{\max}_{\langle k,t\rangle}\frac{1}{m}\frac{\alpha}{(1-\beta)}$. Rearranging gives the lower bound $f_{\langle k,t\rangle} \ge \frac{\alpha}{1-\beta}\frac{1}{m}\frac{n^{\max}_{\langle k,t\rangle}}{sB_k}$. □

Lemma 3. *The bandwidth allocated by* AIMDEQUI$_s(\mathcal{J})$ *to job* J_i *at time* t *is at least* $\frac{1}{m}$ *that allocated by* L-EQUI$_s(\mathcal{J})$, *i.e.* $b_{\langle i,t\rangle} \ge \frac{1}{m}\min_{k\in\mathcal{B}_i}\frac{sB_k}{n_{\langle k,t\rangle}}$. *It follows that* $\frac{\text{AIMDEQUI}_{ms}(\mathcal{J})}{\text{L-EQUI}_s(\mathcal{J})} \le 1$.

Proof of Lemma 3. Consider a job J_i. Eq. (4) gives $(m\max_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})b_{\langle i,t\rangle} \ge (\sum_{k\in\mathcal{B}_i} f_{\langle k,t\rangle})b_{\langle i,t\rangle} = \frac{\alpha}{(1-\beta)}$ and hence $b_{\langle i,t\rangle} \ge \frac{\alpha}{1-\beta}\frac{1}{m}\min_{k\in\mathcal{B}_i}\frac{1}{f_{\langle k,t\rangle}}$. Applying the bound in Lemma 2 gives $b_{\langle i,t\rangle} \ge \frac{1}{m}\min_{k\in\mathcal{B}_i}\frac{sB_k}{n_{\langle k,t\rangle}}$. □

6. Open problems

What make me slightly uncomfortable about this paper is the following. I do not know if or how quickly the differential equations in Eqs. (1) and (2) converge. For Eqs. (3) and (4), it is not clear to me how one determines which bottlenecks are at capacity. Once this is known, there is one equation and one unknown $b_{\langle i,t\rangle}$ and $f_{\langle k,t\rangle}$ for each job and bottleneck. However, I don't even know whether these equations have a solution, not to mention a solution in which both $b_{\langle i,t\rangle}$ and $f_{\langle k,t\rangle}$ are positive. The obvious open question is to prove Theorem 4 without its restrictions that the adjustment frequencies do not change much throughout the life of a job. Also, according to the *max–min socialistic fairness*, [14] proves that AIMD can be unfair by a factor of m to jobs that pass through m bottlenecks. An open problem is to prove that this is the worst case.

For further reading

[1,8,15,16,24,26,29,30,33–36].

Acknowledgments

This research is partially supported by NSERC grants. I would like to thank Steven Watson for his extensive help on this paper.

References

[1] F. Baccelli, D. Hong, AIMD, fairness and fractal scaling of TCP traffic RR-4155 INRIA rocquencourt and infocom, June 2002.
[2] F. Bonomi, K. Fendick, The rate-based flow control for available bit rate ATM service, IEEE Network 9 (2) (1995) 24–39.
[3] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.
[4] A. Charny, An algorithm for rate allocation in a packet-switching network with feedback, Technical Report MIT/LCS/TR-601, Laboratory for Computer Science, Massachusetts Institute of Technology, April 1994.
[5] A. Charny, D.D. Clark, R. Jain, Congestion control with explicit rate indication, in: Proceedings of the IEEE 30th International Conference on Communications, June 1995, pp. 1954–1963.
[6] A. Charny, K.K. Ramakrishnan, A. Lauck, Time scale analysis and scalability issues for explicit rate allocation in ATM networks, IEEE/ACM Transactions on Networking, 4 (4) (1996) 569–581.
[7] D.M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computer Networks and ISDN Systems 17 (1) (1989) 1–14.
[8] Daniel R. Dooly, Sally A. Goldman, Stephen D. Scott, TCP dynamic acknowledgement delay: theory and practice, in: ACM Symposium on Theory of Computing, 1998.
[9] Jeff Edmonds, Scheduling in the dark, Journal of Theoretic Computer Science (1999); ACM Symposium on Theory of Computing (1999) 179–188.
[10] Jeff Edmonds, Scheduling in the dark—improved results: manuscript, http://www.cs.yorku.ca/~jeff, 2001.
[11] J. Edmonds, S. Datta, P. Dymond, TCP is competitive against a limited adversary, in: Proc. 15th Ann. ACM Symp. of Parallelism in Algorithms and Architectures, 2003, pp. 174–183.
[12] P. Fatourou, M. Mavronicolas, P. Spirakis, The global efficiency of distributed, Rate Based flow control algorithms, in: Proceedings of the 5th Colloqium on Structural Information and Communication Complexity, Carleton University Press, June 1998, pp. 244–258.
[13] P. Fatourou, M. Mavronicolas, P. Spirakis, Max–min fair flow control sensitive to priorities, in: Proceedings of the 2nd International Conference on Principles of Distributed Systems, December 1998, pp. 45–59.
[14] S. Floyd, Connections with multiple congested gateways in packet-switched networks, Part I: one-way traffic, Computer communications review 21 (5) (1991) 30–47.
[15] Sally Floyd, Van Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (4) (1993) 397–413.
[16] V. Jacobson, Congestion avoidance and control, in: Proceedings of the Sigcomm'88 Symposium in Stanford, CA, August, 1988, ACM Computer Communication Review 18 (4) (1988) 314–329.
[17] E. Hahne, Round-robin scheduling for MaxMin fairness in data networks, IEEE Journal on Selected Areas in Communications 9 (7) (1991) 1024–1039.

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033

[18] E. Hahne, R.G. Gallager, Round-robin scheduling for fair flow control in data communication networks, in: Proceedings of the IEEE 21st International Conference on Communications, June 1986, pp. 103–107.
[19] J.M. Jaffe, Bottleneck flow control, IEEE Transactions on Communications COM-29 (7) (1981) 954–962.
[20] J.M. Jaffe, Flow control power is nondecentralizable, IEEE Transactions on Communications COM-29 (9) (1981) 1301–1306.
[21] R. Jain, S. Kalyanaraman, R. Viswanathan, The OSU scheme for congestion avoidance using explicit rate indication, Technical Report ATM-FORUM/95-0883, The Ohio State University, September 1994.
[22] L. Kalampoukas, A. Varma, K.K. Ramakrishnan, An efficient rate allocation algorithm for ATM networks providing max–min fairness, in: Proceedings of the 6th IFIP International Conference on High Performance Networking, September 1995, pp. 143–154.
[23] Bala Kalyanasundaram, Kirk Pruhs, Speed is as powerful as Clairvoyance, Journal of the ACM 47 (4) (2000) 617–643.
[24] R. Karp, E. Koutsoupias, C. Papadimitriou, S. Shenker, Optimization problems in congestion control, in: IEEE Symposium on Foundations of Computer Science, 2000, pp. 66–74.
[25] F. Kelly, Mathematical modelling of the internet, in: Bjorn Engquist, Wilfried Schmid (Eds.), Mathematics Unlimited—2001 and Beyond, Springer, 2001.
[26] F. Kelly, Fairness and stability of end-to-end congestion control, in: European Control Conference, Cambridge, 2003.
[27] F. Kelly, A. Maulloo, D. Tan, Rate control in communication networks: shadow prices, proportional fairness and stability, Journal of the Operational Research Society 49 (1998).
[28] J. Kurose, K. Ross, Computer Networking: A Top–Down Approach Featuring the Internet, Addison-Wesley, 2002.
[29] T.V. Lakshman, U. Madhow, The performance of networks with high bandwidth-delay products and random loss, IEEE/ACM Transactions on Networking 5 (3) (1997).
[30] A. Misra, T. Ott, The window distribution of idealized tcp congestion avoidance with variable packet loss, in: Proceedings of INFOCOM, March 1999.
[31] J. Mosely, Asynchronous distributed flow control algorithms, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, October 1984.
[32] R. Motwani, S. Phillips, E. Torng, Non-clairvoyant scheduling, Theoretical computer science 130 (1994) 17–47 (Special issue on dynamic and on-line algorithms).
[33] T. Ott, J. Kemperman, M. Mathis, The Stationary Behavior of Ideal TCP Congestion Avoidance, August 1996.
[34] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling tcp throughput: a simple model and its empirical validation, in: Proceedings of SIGCOMM, 1988, pp. 303–314.
[35] Kevin Thompson, Gregory J. Miller, Rick Wilder, Wide-area internet traffic patterns and characteristics, IEEE Network 11 (6) (1997) 10–23.
[36] C. Phillips, C. Stein, E. Torng, J. Wein, Optimal time-critical scheduling via resource augmentation, Algorimica 32 (2) (2002) 163–200.

Please cite this article in press as: J. Edmonds, On the competitiveness of AIMD-TCP within a general network, Theoretical Computer Science (2012), doi:10.1016/j.tcs.2012.07.033