An Analysis of AIMD Algorithm with Decreasing Increases

Yunhong Gu, Xinwei Hong, and Robert L. Grossman

Abstract— **In this paper, we introduce a new class of AIMD algorithms in which the size of the additive increases will decrease as the sending rate of the algorithm increases. We call these algorithms decreasing AIMD algorithms or DAIMD algorithms. We show that DAIMD algorithms are efficient, stable and fair. A special case of a DAIMD algorithm is provided by the UDT algorithm, which has been used for a variety of grid-based data intensive applications. UDT satisfies max-min fairness if all concurrent flows have the same bottleneck capacities; otherwise, the unfairness will be lower bounded. It can expand to use from 0 to 90% of available bandwidth in a given fixed time interval. We use both simulations and experiments to examine these characteristics (i.e., fairness, efficiency, and stability).**

Index Terms—**UDT, congestion control, transport protocol, grid networking.**

I. Introduction

One of the research objectives of grid networking is to design a new congestion control algorithm that is highly scalable to networks with high bandwidth-delay product (BDP). The original TCP algorithm [7] and its current most popular version (TCP NewReno) have notably poor efficiency and fairness in high BDP environments [15]. Researchers have presented a series of TCP improvements, including Scalable TCP [5], HighSpeed TCP [4], Bic TCP [3], and FAST TCP [2]. These new congestion control algorithms more or less improve the efficiency of standard TCP (TCP NewReno) and can be used in high-speed networks. Katabi, et al. [6] proposed a different solution for future high BDP networks by providing explicit information from routers, such as the number of concurrent flows and the available bandwidth.

In our previous work [14, 27, 28, 29], we have introduced a new application level data transport protocols SABUL (Simple Available Bandwidth Utilization Library) and its successor UDT (UDP-based Data Transfer) and provided some simulation and experimental results. We are continuing the work and present in this paper a detailed analysis of the rate-based congestion control algorithm used in UDT.

We generalize a type of rate-based AIMD (additive increase multiplicative decrease) algorithm, named DAIMD, whose increments are decreasing functions of the current sending rates. The first "D" in DAIMD indicates that the additive parameter is decreasing. We show that this type of control algorithm is fair and can be efficient if the parameter is properly tuned.

UDT uses a rate control algorithm by specifying the increment function of DAIMD. It also specifies a constant rate control interval and introduces a supportive window-based flow control mechanism. We use experiments and simulation to examine the issues of efficiency, fairness, and stability. In addition, we will also describe the automatic parameter tuning and the impact of parameter estimation error.

The rest of this paper is organized as follows. Section 2 describes the generalized DAIMD algorithm. Section 3 briefly reviews the UDT control algorithm and models its performance. Section 4 analyzes the fairness equilibrium of UDT. Section 5 evaluates the TCP friendliness of UDT. Section 6 discusses the impact of network delay on the performance of UDT. Section 7 discusses how to estimate the link capacity parameter and the impact of estimation error. Section 8 lists some related work. Section 9 concludes this paper.

II. Decreasing AIMD Rate Control Algorithm

We consider a general class of the following AIMD rate control algorithm:

For every rate control interval, if there is no negative feedback from the receiver (loss, increasing delay, etc.), but there are positive feedbacks (acknowledgements), then the packet-sending rate (x) is increased by $\alpha(x)$.

$$x \leftarrow x + \alpha(x) \tag{1}$$

$\alpha(x)$ is non-increasing and it approaches 0 as x increases, i.e., $\lim_{x->+\infty} \alpha(x) = 0$.

For any negative feedback, the sending rate is decreased by a constant factor β ($0 < \beta < 1$):

$$x \leftarrow (1-\beta) \cdot x \tag{2}$$

This work was supported in part by the National Science Foundation under grants number 0129609 and 9977868.

The authors are with the Laboratory for Advanced Computing, University of Illinois at Chicago. ADDR: 700 SEO, MC 249, 851 S Morgan St, Chicago, IL 60607. EMAIL: ygu@cs.uic.edu, xwhong@lac.uic.edu, grossman@uic.edu. Robert Grossman is also with Open Data Partners.

Note that formula (1) is based on a fixed control interval, e.g., the network round trip time (RTT). This is different from TCP control, in which every acknowledgement triggers an increase.

By varying $\alpha(x)$, we can get a class of rate control algorithm that we name the DAIMD algorithm, because the additive parameter is decreasing.

If we use the rate control interval as a unit of time, then from time t to t+1, the increase to the sending rate from (1) is:

$$x(t+1) = x(t) + \alpha(x(t))$$

the decrease from (2) is:

$$x(t+1) = (1-\beta)^n \cdot x(t)$$

where n is the number of negative feedbacks.

Thus, the net change (contributed by both the increase and the decrease) of the sending rate x is approximated by:

$$x(t+1) - x(t) = P(0) \cdot \alpha(x(t)) - \sum_{i=1}^{x(t-D)} (P(i) \cdot (1-(1-\beta)^i)) \cdot x(t) \quad (3)$$

where $P(i)$ is the probability that i packets are lost during the period of (t, t+1), and D is the network round trip delay. In equation (3), $x(t\text{-}D)$ is the number of packets that can be fed back at period (t, t+1), and $P(i) \cdot (1-(1-\beta)^i)$ means the possible decrease when i packets are lost.

To simplify the analysis, we assume that the loss rate $p(t)$ is very small and there is, at most, one negative feedback during one unit of time ($P(0) + P(1) = 1$).

$$P(0) = (1-p(t))^{x(t-D)} \approx 1 - p(t) \cdot x(t-D)$$

$$P(1) = 1 - P(0) = p(t) \cdot x(t-D)$$

In addition, at the stable state, the difference between $x(t)$ and $x(t\text{-}D)$ is small and we assume $x(t) = x(t\text{-}D)$. (The UDT flow control limits the difference between these two values. See Section 6 for details.) Equations (3) can be simplified as:

$$\dot{x} = (1 - x(t) \cdot p(t)) \cdot \alpha(x(t)) - x(t) \cdot p(t) \cdot \beta \cdot x(t) \quad (4)$$

The differential function (4) can be written in the form of:

$$\dot{x} = k(x)(U'(x) - p(t)) \quad (5)$$

where $k(x) = x \cdot \alpha(x) + \beta \cdot x^2$ is positive and non-decreasing for any x (x>0)[1], and

$$U(x) = \int \frac{\alpha(x)}{x \cdot \alpha(x) + \beta \cdot x^2} dx \quad (6)$$

is called the utility function [1] of the above congestion control algorithm.

$U(x)$ is concave because $U'(x)$ is strictly decreasing and hence $U''(x) < 0$. According to Srikant [16] (page 26, theorem 3.4), the congestion control algorithm (5) (hence the DAIMD algorithm) is globally asymptotically stable and will converge to an equilibrium.

We further show that the equilibrium of the DAIMD algorithm described above satisfies max-min fairness. We use Jain's fairness index (7) to evaluate the max-min fairness among multiple flows.

$$FI = \frac{\left(\sum x_i\right)^2}{n \sum x_i^2} \quad (7)$$

where n is the number of concurrent flows and x_i is the sending rate of the ith flow at equilibrium. FI is a value between 0 and 1, and FI = 1 is perfectly fair. Following the methods used in [9], it can be easily seen that a decrease of x according to (2) will not affect the value of FI, but an increase of x according to (1) increases FI.

Fig. 1 illustrates the increase function in TCP Reno, Scalable TCP, HighSpeed TCP, and the DAIMD algorithm. If $\alpha(x) \equiv \alpha$, DAIMD turns into AIMD.

There is one important difference between DAIMD and some TCP variants that use loss as a congestion signal: as the window size becomes larger, both Scalable TCP and HighSpeed TCP increase faster, whereas the increase of Bic TCP may be independent of the absolute sending rate but it is determined by the distance between the current sending rate and a target rate.

In fact, the increment of an XCP flow may decrease as its sending rate increases, depending on the entering or leaving of coexisting flows, because XCP uses available bandwidth to determine the overall increment. If there is no flow enters or leaves, this is always true.

A detailed description of other AIMD algorithms and XCP can be found in Section 8.

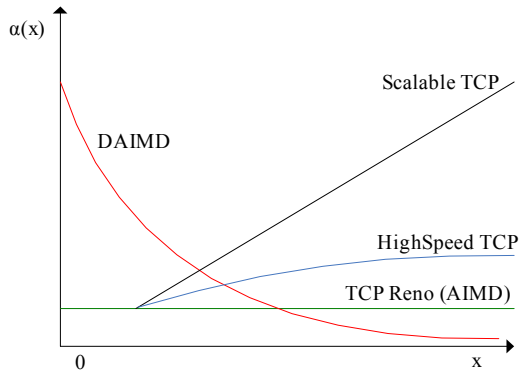

Figure 1: Function of increase parameter of DAIMD and several TCP variants.

In addition to stability and fairness, the function of $\alpha(x)$ has to be large around $\alpha(0)$ to be efficient and it has to decrease quickly to reduce oscillations. An important special case is provided by an $\alpha(x)$ of the following form (Fig. 2).

[1] Strictly speaking, $k'(x) = \alpha(x) + x\,\alpha'(x) + 2\beta x$ may be less than 0, so $k(x)$ may not be strictly non-decreasing. However, because $\alpha(x)$ is non-increasing and it is infinitely close to 0, there exists a const c, such that $k'(x) > 0$ for any x (x>c). Therefore, we can construct a new variable: $y = x - c$, and $k(y)$ is non-decreasing for any y (y>0). We can replace x using y in formula (4) - (6).

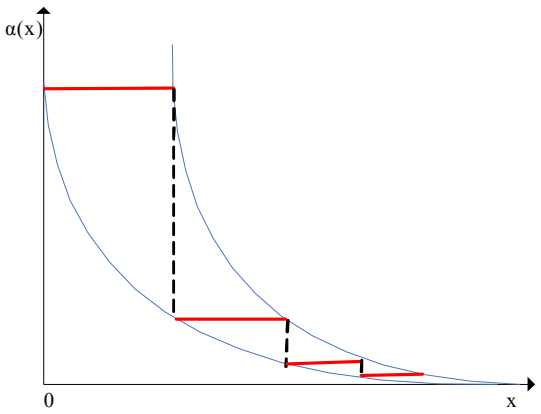

Figure 2. A piecewise α(x) with breakpoints.

The first stage in the piecewise function in Fig. 2 decides how quickly a DAIMD flow can probe the available bandwidth at the beginning, and the length of the stage determines its aggressiveness. The longer the stage is, the more aggressive it will be. Each later stage has a smaller increment as the flow approaches available bandwidth. This will reduce the oscillations at the equilibrium.

Specifically, to achieve efficiency, the increment at each stage should be proportional to the available bandwidth (similar to the mechanism of the XCP efficiency controller [6]).

UDT adopts this efficiency idea and specifies a piecewise $a(x)$ that is related to the link capacity.

III. UDT

We first briefly review the UDT congestion control mechanism. In UDT, a periodical timer triggers a sending event, and a data packet is sent out if and only if the number of unacknowledged packets does not exceed a congestion window.

The period of the sending timer is updated by rate control and the congestion window size is updated by the flow control, respectively.

The rate control algorithm is the major mechanism in UDT and in this section we only model the UDT throughput according to the rate control. We will describe flow control in Section 6.

The UDT rate control directly tunes the packet-sending period (T), which indirectly determines the packet-sending rate (x):

$$T \times x = 1$$

We therefore can write the rate control formula in the form of the sending rate.

The fixed rate control interval of UDT is SYN, which is 0.01 seconds.

UDT rate control is a special DAIMD algorithm by specifying $a(x)$ as:

$$\alpha(x) = 10^{\lceil \log(L - C(x)) \rceil - \tau} \times \frac{1500}{S} \cdot \frac{1}{SYN} \quad (8)$$

In formula (8), x has the unit of packets/second. L is the link capacity measured by bits/second. S is the UDT packet size (in terms of IP payload) in bytes. $C(x)$ is a function that converts the unit of the current sending rate x from packets/second to bits/second ($C(x) = x * S * 8$). τ is a protocol parameter, which is 9 in the current protocol specification.

The factor of $(1500/S)$ in function (8) is to balance the impact of flows with different packet sizes. UDT treats 1500 bytes as a standard packet size.

Due to the ceiling function in (8), the UDT congestion control has multiple stages, as shown in Fig. 3. UDT increases its sending rate quickly at the beginning and slows down as it is approaching the link capacity. In addition, every stage has the same time span, except for the first stage, if L is not an integer power of 10.

To simplify, we suppose S = 1500 and use packets/SYN as the time unit of $x(t)$. Equation (8) can be rewritten as

$$\alpha(x) = 10^{\lceil \log(L - C(x)) \rceil - \tau} \quad (9)$$

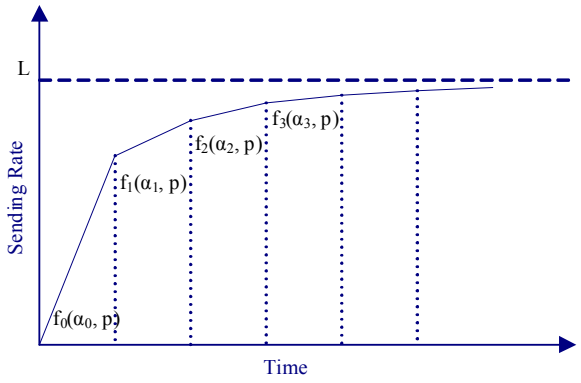

Figure 3. Sending rate changes over time. This is the situation when there is no loss in the system; otherwise there will be oscillations in the sending rate.

Thus, UDT implements a piecewise $a(x)$ and according to Section 2, it is stable and fair (given that the value of L is the same for all flows. We will discuss this further in Section 4). We now discuss its efficiency characteristic.

Suppose in stage k (k = 0, 1, 2, …), the throughput function is f_k, the increase parameter is α_k, and the loss rate is p. Let e be an integer that satisfies $10^{e-1} < L \le 10^e$.

According to the rate differential function (4), the equilibrium solution ($\dot{x} = 0$) of UDT for any stage k (x_k*) is:

$$x_k^* = \frac{1}{2\beta}\left(-\alpha_k + \sqrt{\alpha_k^2 + \frac{4\beta\alpha_k}{p}}\right) \quad (10)$$

$$\approx \sqrt{\frac{\alpha_k}{\beta \cdot p}}$$

The approximation is due to the fact that a_k is very small compared to $1/p$. The result of (10) shows that at each stage UDT acts as an AIMD control (the response function is

proportional to $p^{-0.5}$), and its increase parameter decreases as the sending rate increases, whereas its decrease factor is a constant.

The increase parameter of each stage decreases by 1/10, and $\alpha_0 = \alpha(0)$, therefore,

$$\alpha_k = 10^{-k} \cdot \alpha_0 = 10^{-k+\lceil \log L \rceil - \tau} \tag{11}$$

Recall that

$\tau = 9$,

and UDT defines the decrease factor as[2]

$\beta = 1/9$

We finally reach

$$x_k^* = \frac{3}{\sqrt{p}} \cdot 10^{\frac{-k+e-9}{2}} \tag{12}$$

Note that x_k has units that are measured by packets/SYN. Suppose X_k is the throughput function whose units are bits/second, then

$$\begin{aligned} X_k^* &= \frac{1}{SYN} \cdot \frac{1500}{S} \cdot x_k^* \cdot S \cdot 8 \\ &= \frac{3.6}{SYN} \cdot \frac{1}{\sqrt{p}} \cdot 10^{\frac{e-k+1}{2}} \end{aligned} \tag{13}$$

Once the sending rate increases to a certain value such that (L-C) falls into the next class of the power of 10, i.e., L-C<10^{e-k-1}, the UDT congestion control enters the next stage. However, as k increases, the throughput at stable state (x_k*) decreases, and k will stop increasing when

$$L - \frac{3.6}{SYN} \cdot \frac{1}{\sqrt{p}} \cdot 10^{\frac{e-k+1}{2}} \geq 10^{e-k-1}$$

The minimum k that satisfies the above condition is the stable stage of a UDT flow. (The operator $[op]^+$ is equivalent to $\max\{op, 0\}$.)

$$\begin{aligned} k^* &= \left\lceil \left\lfloor e - 1 - 2\log\left(\sqrt{d^2 + L} - d\right) \right\rfloor \right\rceil^+ \\ d &= \frac{18}{SYN \cdot \sqrt{p}} \end{aligned} \tag{14}$$

When $p = 0$, equation (3) turns into:

$$x_k(t+1) = x_k(t) + \alpha_k \tag{15}$$

This linear increase shows that each stage will need a fixed time interval to increase to the next stage. Specifically, there is a fixed time interval for a UDT flow to increase from 0 to 90% of the link capacity.

Suppose at the end of the first stage, UDT reaches rate R_0 ($R_0 \leq 0.9L$), then to reach $0.9L$ it takes

$$\left(\frac{R_0}{\alpha_0} + \frac{0.9L - R_0}{\alpha_1} \right) \cdot \frac{SYN}{1500 \times 8} = \frac{9(L - R_0)}{\alpha_0} \cdot \frac{SYN}{1500 \times 8}$$

[2] We actually increase the packet-sending period by 1/8 in UDT, and it is a decreases factor of 1/9 on the packet sending rate.

Since L-R_0=10^{e-1}, $\alpha_0 = 10^{e-9}$, and SYN = 0.01, the above formula yields 750 SYN, which is 7.5 seconds.

In contrast, at 200ms RTT, TCP needs 28 minutes to recover from a single loss to 1Gbits/s, or 4 hours 43 minutes to recover to 10Gbits/s, etc.

IV. Fairness of UDT

A. Max-min Fairness

If all concurrent flows have the same L, then the increment function of each flow will satisfy the condition of the $\alpha(x)$ in the DAIMD algorithm. Therefore, in this situation UDT satisfies max-min fairness. In addition, this fairness is independent of RTT, since UDT uses a constant rate control interval.

We now discuss the situation when two flows F1 and F2 have different bottleneck link capacities. Suppose the bottleneck link capacities for F1 and F2 are L_1 and L_2 (L_1>L_2), respectively. The equilibrium bandwidth allocation is (x_1, x_2). The following condition should stand:

$$L_1 - x_1 \geq L_2 - x_2 \tag{16}$$

Otherwise F2 has smaller decrements but has higher increments so that (x_1, x_2) cannot be the equilibrium. If the loss rate is small and

$$x_1 + x_2 \approx L_1 \tag{17}$$

then according to the equation (16) and (17) we can conclude that F2 will take at least half of L_2 ($x_2 \geq L_2/2$).

Fig. 4 illustrates the details of the competition between the two flows. Suppose at equilibrium, the two flows stay at stage k_1 and k_2, respectively. According to (9), $\alpha_{k1} \geq \alpha_{k2}$, otherwise F2 will occupy more bandwidth, which is impossible. If $k_2 \geq 2$, then F2 has already occupied more than 90% of L_2, so it is approximately fair. If $k_2 = 1$, then either F1 stays at the same stage such that $\alpha_{k1} = \alpha_{k2}$, which means the two flows share the bandwidth equally, or it stays below (L_1 - L_2), which means all the bandwidth of L_2 is left for F2.

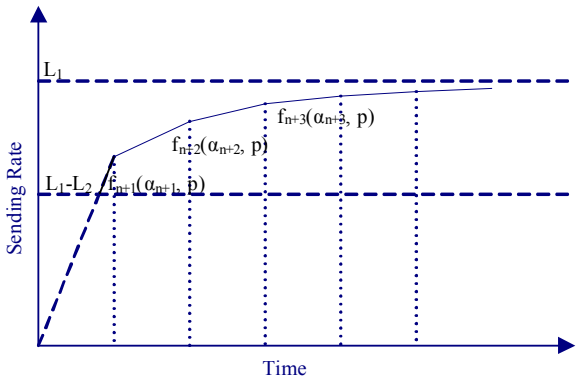

Figure 4. Two UDT flows with different link capacities.

The only situation that can cause unfairness is that F2 stays at stage 2, and F1 stays at the first stage above (L_1 - L_2).

In this case, F2 is still more competitive than F1 and will obtain more bandwidth of L_2. Therefore, the lower bound of the throughput of F2 is $L_2/2$.

B. Experiment

We set up an experiment to check the fairness of UDT. The network configuration is shown in Fig. 5. Two sites, StarLight (Chicago) and SARA (Amsterdam), are connected with 1 Gbits/s link. At each site, four nodes are connected to the gateway switch through 1GigE NIC. The RTT between the two sites is 104ms. All nodes run Linux 2.2.19 SMP on dual Intel Xeon 2.4GHz CPU.

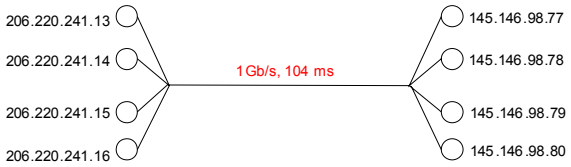

Figure 5. Fairness testing configuration. Four pairs of nodes share 1 Gbits/s, 104 ms RTT link.

For the four pairs of nodes, we start a UDT flow every 100 seconds, and stop each of them in the reverse order every 100 seconds, as depicted in Fig. 6.

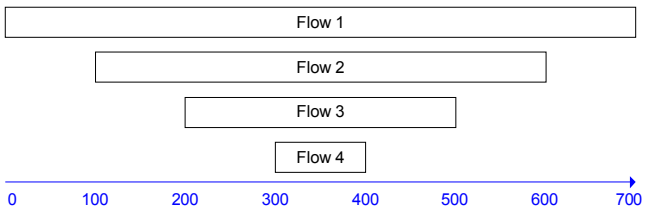

Figure 6. Flow start and stop configuration.

The results are shown in Fig. 7 and Table 1. Fig. 7 shows the detailed performance of each flow and the aggregate throughput. Table 1 lists the average throughput of each flow, the average RTT and loss rate at each stage, the efficiency index (*EI*), the fairness index (*FI*), and the stability index (*SI*).

Here the efficiency index is defined as the aggregate throughput, the fairness index is defined as Jain's fairness index (7), and the stability index uses the one defined by Jin, et al. [2].

$$SI = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{\bar{x}_i}\sqrt{\frac{1}{m-1}\sum_{k=1}^{m}(x_i(k)-\bar{x}_i)^2}\right) \qquad (18)$$

where n is the number of concurrent flows, m is the number of samples of each flow, $x_i(k)$ is the sending rate of kth sample. In this experiment $n = 4$ and $m = 100$ for each stage.

All stages achieve good bandwidth utilization. The maximum possible bandwidth is about 940Mbits/s on the link, measured by other benchmark software. The fairness among concurrent UDT flows is very close to 1. The stability index reflects the oscillations of sending rates and a smaller value means the sending rate is more stable. Furthermore, UDT causes little increase in the RTT (107 ms vs. 104 ms) and a very small loss rate.

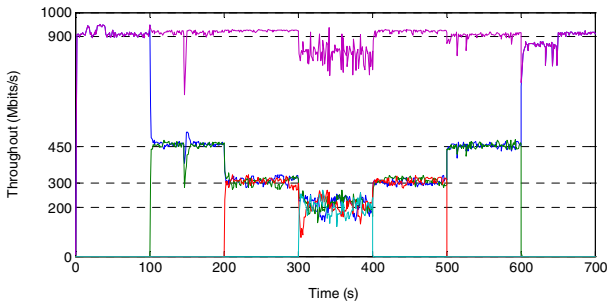

Figure 7. UDT Fairness. This figure shows 4 UDT flows shares 1 Gbits/s, 104 ms RTT link. The highest line is the aggregate throughput.

TABLE 1.
CONCURRENT UDT FLOW EXPERIMENT RESULTS

Time (sec)	1 - 100	101 - 200	201 - 300	301 - 400	401 - 500	501 - 600	601 - 700
Flow1	902	466	313	215	301	452	885
Flow2		446	308	216	310	452	
Flow3			302	202	307		
Flow4				197			
RTT	106	106	106	106	107	105	105
Loss	0	10^{-6}	10^{-4}	10^{-3}	10^{-3}	0	10^{-6}
EI	902	912	923	830	918	904	885
FI	1	.999	.999	.998	.999	1	1
SI	0.11	0.11	0.08	0.16	0.04	0.02	0.04

C. Simulation

We use a simulation[3] to check the fairness characteristic for flows having different link capacities. The simulation topology is shown in Fig. 8. The link capacity of AB is 200 Mbits/s, and that of AC is y ($y < 200$).

Two flows are started at the same time and each sends data from A to B and C, respectively.

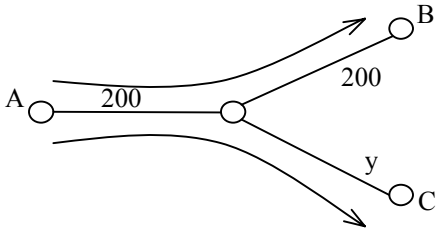

Fig. 8. UDT performance in multi-bottleneck topology network. In this topology, the end-to-end capacity of AB is 200Mbps, whereas AC is y (y < 200). DropTail queue is used in the network.

The results are shown in Table 2. In all cases flow AC obtained at least 90% of its fair share.

[3] All simulations in this paper are performed on NS-2 simulator.

TABLE 2
UDT PERFORMANCE (IN MBITS/S) OF FIG. 8

y	0.1	1	10	20	40	60
AB	198.8	189.2	180.1	170.9	152.5	137.6
AC	0.098	0.979	9.955	19.88	39.46	57.70
y	80	100	120	140	160	180
AB	108.4	104.6	100.8	101.3	100.7	100.3
AC	73.49	92.42	98.47	98.04	98.65	99.00

We set up another simulation to check RTT fairness. Fig. 9 is the network configuration of this simulation: five flows share a 100 Mbits/s bottleneck, with each having RTT of 10 µs, 100 µs, 1 ms, 10 ms, and 100 ms.

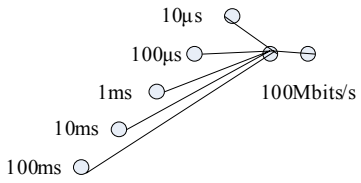

Figure 9. RTT fairness simulation topology. Five flows share a 100 Mbits/s link, with each having RTT from 10 µs to 100 ms.

The result can be seen in Fig. 10. The average throughput for the five flows is 25.37, 19.93, 20.17, 19.70, and 13.72 Mbits/s, respectively. The aggregate throughput is 98.89 Mbits/s, the Jain's fairness index is 0.966, and the stability index is 0.19.

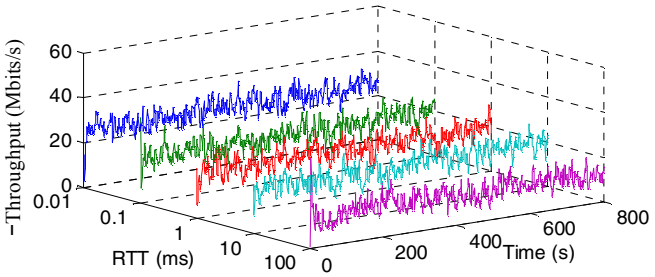

Figure 10. RTT fairness of UDT. The figure shows the throughputs of 5 concurrent UDT flows with different RTTs.

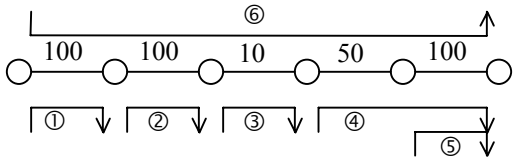

Fig. 11. UDT Performance in complex topology network. The topology consists of 6 nodes, and the capacity is noted above each link. The RTT between any 2 adjacent nodes is 10ms. There are 6 flows in the network and are noted as arrowed lines in the figure.

TABLE 3
UDT PERFORMANCE (IN MBITS/S) OF FIG. 11.

Flow ID	1	2	3	4	5	6
	89.3	90.0	5.18	41.7	50.8	4.78

A more complex simulation is set up as a parking lot topology (Fig. 11). Six flows pass through a 5-node network with different bottleneck capacities. The result is listed in Table 3, where we can see that the max-min fairness is still observed.

V. TCP FRIENDLINESS

Because UDT uses a fixed rate control interval, when it competes with TCP, the network RTT will play an important role in the bandwidth sharing. Meanwhile, the increase parameter is decided by the parameter of link capacity L, which also affects the TCP friendliness.

If SYN = RTT, UDT increases no less than 1 packet per RTT ($\alpha_0 >= 1$) only at L > 100Mbps; If SYN < RTT, UDT increases at a lower frequency than TCP.

Specifically, according to (12) (note that it has to be converted to use the units of packets/second) and the simple version of the TCP throughput model ($\sqrt{1.5/p}/RTT$) [21, 22], the relationship between UDT and TCP (TF) can be written in the equation (19):

$$TF = \left(\frac{3}{SYN}\cdot\frac{1}{\sqrt{p}}\cdot 10^{\frac{-k+e-9}{2}}\right)\Big/\left(\frac{1}{RTT}\cdot\sqrt{\frac{1.5}{p}}\right) = \frac{RTT}{SYN}\cdot 10^{\frac{-k+e-9}{2}}\cdot\sqrt{6} \quad (19)$$

UDT will obtain less bandwidth than coexisting TCP if $TF \le 1$. The first stage of UDT is the most aggressive one, so k=0 yields a sufficient condition for TCP friendliness, i.e., any UDT flow that satisfies the following condition must be friendly to TCP:

$$\frac{RTT}{SYN}\cdot 10^{\frac{e-9}{2}}\cdot\sqrt{6} \le 1$$

Since $10^{e-1} < L \le 10^e$, the above equation is satisfied if

$$RTT^2 \cdot L \le SYN^2 \cdot 10^8 / 6 \quad (20)$$

Condition (20) is sufficient to guarantee that UDT is less aggressive than TCP and it shows that UDT is very friendly to TCP in low BDP environments.

Fig. 14 is the simulation result of UDT/TCP bandwidth allocation under different bandwidth and RTT. The figure shows the ratio between UDT throughput and TCP throughput. As the RTT increases, UDT obtain more bandwidth; however, at 1 Gbits/s and 100ms RTT, a UDT flow still only obtains about 5 times the bandwidth as that of the coexisting TCP flow.

We noticed that at very low link capacity, UDT may take more bandwidth than in high link capacity under the same RTT, this is because in such environments the queue size is relatively large (much larger than BDP) and it can have a negative impact on the TCP bursting flow [17].

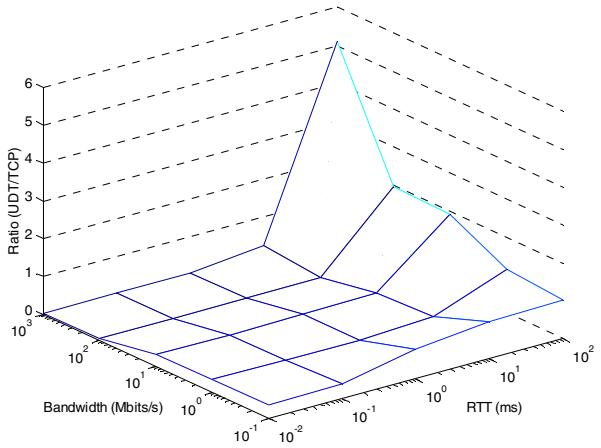

Figure 14: Bandwidth allocation between UDT and TCP. This is the simulation of one UDT flow and one TCP flow under different bandwidth and RTT. This figure shows the ratio between the throughputs of the UDT flow verses the throughput of the TCP flows.

VI. Impact of Network Delay on UDT Performance

Our analysis in the previous sections has simplified the impact of the network delay. As the RTT increases, the increment per RTT may also become larger. This may cause a stability concern because large increments can cause oscillations (because the congestion, if caused by the rate increase, will need one RTT to feed back).

The UDT window-based flow control is to prevent such stability problems from occurring. The window control limits the number of unacknowledged packets (w). It is done at the receiver side and the window size is sent back in acknowledgements. Every SYN time, the window size is updated by:

$$w = w \times \lambda + AS \times (SYN + RTT) \times (1 - \lambda) \qquad (21)$$

where w has the unit of packets, AS is the packet arrival speed since last time w is updated (w will not be updated if no packets arrive or there are too few packets to estimate the arrival speed), and λ ($0 < \lambda < 1$) is a factor for the moving average.

The acknowledgment feeds back the minimum value between w and the available receiver buffer size, but we assume there is always a large enough receiving buffer and the feedback is always w.

According to the window control (21), a UDT sender can send out no more packets than (1+ SYN/RTT) of that which the receiver was able to handle one RTT ago[4]. (However, the packet-sending period may still be decreasing, independent of the number of sent packets.)

This indicates that RTT does have a slightly negative impact on the performance of UDT by delaying the increase effect: the increase on the sending rate cannot have an effect immediately. It must wait until the next RTT. (In Fig. 10, flows with longer RTT have lower throughputs.) This eliminates the stability problem of using constant rate control intervals.

We set up 10 UDT flows on a similar network topology as Fig. 5 but with each side having 10 nodes. The route has a fixed 2000-packet DropTail queue. We vary the RTT between 10 microseconds (10^{-5} second) to 1 second, and compute the aggregate throughput, fairness index, and stability index.

The result is shown in Table 4. For comparison, we also list the same experimental data for TCP in Table 5. In these two tables, EI is the efficiency index, FI is the fairness index, and SI is the stability index, which have the same definition as those in Table 1. At lower RTT, both UDT and TCP work well; however, as the RTT increases, UDT is more efficient and fair than TCP. It still obtains an acceptable performance, even at 1 second RTT.

TABLE 4
UDT Performance against RTT
This table lists the performance of 10 UDT flows sharing a single 1 Gbits/s link. In this table, RTT uses the unit of seconds, EI is the efficiency index (i.e., aggregate throughput), FI is the fairness index, and SI is the stability index.

RTT	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
EI	998	997	997	989	929	719
FI	.999	.999	.999	.999	.997	.993
SI	0.06	0.07	0.09	0.11	0.23	0.64

TABLE 5
Impact of RTT on TCP
This table lists the performance of 10 TCP flows sharing a single 1 Gbits/s link. In this table, RTT uses the unit of seconds, EI is the efficiency index (i.e., aggregate throughput), FI is the fairness index, and SI is the stability index.

RTT	10^{-5}	10^{-4}	10^{-3}	10^{-2}	10^{-1}	1
EI	996	994	998	997	839	145
FI	.999	.999	.999	.997	.681	.537
SI	0.08	0.09	0.07	0.07	0.30	0.28

VII. Estimating Link Capacity "L"

The parameters of link capacity L can be manually configured by applications if the network topology is known or it can be set up to be the upper limit of the sending rate of a certain UDT flow. In this section, we discuss how to estimate L automatically.

UDT uses receiver-based packet pairs to estimate the link capacity L. The UDT sender sends out a packet pair (by omitting the inter-packet waiting time) every 16 data packets. The receiver records the inter-arrival time of each packet pair and uses a median filter (more complex

[4] UDT Flow control simulates the self-clocking mechanism of TCP. It prevents the sending rate from exceeding the receiving rate.

mechanisms can be found in [23, 24]) on them to compute link capacity.

There are two major concerns in using packet pairs to estimate link capacity. One is the impact of cross traffic. The existence of cross traffic can cause the capacity be under estimated. Dovrolis, et al. point out that using packet pairs leads to a value referred to as Asymptotic Dispersion Rate [10], which is a value between available bandwidth and link capacity.

The other concern is the NIC interrupt coalescence. High speed NIC often has the functionality to interrupt coalescence to avoid too frequent interrupts. This can cause multiple packet arrivals to be notified by one single interrupt and hence the link capacity may be overestimated. This error can be eliminated by using the average inter-arrival time of multiple packet pairs. Prasad, et al. have a detailed discuss about the impact of interrupt coalescence on bandwidth measurement in [26].

We have seen that UDT may overestimate the capacity when there is only one flow in the network, whereas it tends to underestimate the capacity when there are multiple flows.

For a single flow, capacity estimation error only affects the convergence time. For multiple flows, it can also affect the fairness. Note that if all flows have the same estimation error, they can still reach fairness.

Consider a simple situation where we suppose L=10^e, L' is the estimated value, and the estimation error is ε, i.e., $L' = (1+\varepsilon)*L$. We can safely assume that $-0.9 < \varepsilon < 9$, because such a large error is very unlikely and we can even use the sending rate history record to eliminate certain extreme error[5].

When competing with a flow with accurate L estimation, the bandwidth sharing between the two flows will be at most:

$$\begin{cases} \dfrac{11+2\sqrt{10}+9\varepsilon}{11+2\sqrt{10}-9\varepsilon} & \text{if } (0<\varepsilon<1) \\ \sqrt{10} & \text{if } (1\le\varepsilon<9) \\ 1 & \text{if } (-0.5<\varepsilon<0) \\ -\varepsilon/(1+\varepsilon) & \text{if } (-0.9<\varepsilon\le 0.5) \end{cases}$$

We omit the detailed deduction process for these results and only give the following intuitive analysis. Suppose flow 1 has the right estimation L and flow 2 has the error estimation L'. For the first case, before flow 2 reaches $L' - L$, it increases $\sqrt{10}$ (according to equation (10)) faster than flow 1, after which they have the same increments to compete for the rest of the bandwidth. In the second case, flow 2 is always $\sqrt{10}$ faster than flow 1. In the third case, the two flows will reach equal shares because they have the same increments. Finally, if $L' < L/2$, the throughput of flow 2 will be limited by L'.

As a simple example, if two flows share one 100 Mbits/s link, flow 1 measures the link capacity as 101 Mbits/s and flow 2 measures 99 Mbits/s, then the two flows will still share the bandwidth almost equally. After flow 1 reaches 1 Mbits/s, it will enter the same stage as flow 2, and both of the two flows will have the same increments and decrements.

[5] For example, if a UDT flow does not reaches 100 Mbits/s for some time, say, the last 100 RTTs, but the estimation result is 1 Gbits/s, such a result is either wrong or there are other limitations such that the flow will not reach 1 Gbits/s in the next several RTTs. At this case, UDT can conclude that this estimation is invalid.

VIII. Related Work

Recently there have been several other new end-to-end congestion control algorithms proposed for grid networks. They can be roughly classified into three types.

The first type is to modify TCP by using large increase parameters (especially at large windows). Scalable TCP, High Speed TCP, and Bic TCP belong to this category. They all use binary indication of congestion and either increase or decrease the sending rate (congestion window size). Protocols in this category differ from each other by using different increase/decrease functions.

Scalable TCP [5] uses an MIMD approach to increase the increase parameter in proportion to the current window size: $\alpha(x) = 0.1x$. Its decrease factor is a constant of 1/8. Scalable TCP does not satisfy intra-protocol fairness due to its MIMD nature.

HighSpeed TCP [4] redefines the response function of TCP, according to which it computes a series of increase and decrease parameters. Its increase parameter is an increasing function of the current window size, whereas the decrease factor is a decreasing function of the window size.

Bic TCP [3] introduces a binary increase stage and it approximately approaches to AIMD(32, 1/8) at large window size.

Table 6 lists the increase/decrease function and response function of TCP Reno, Scalable TCP, High Speed TCP, Bic TCP, and UDT. Specifically, the Bic TCP parameter we use in this table is (32, 1/8, 0.01), and *target_win* is the window size at the midway between the current window size and the maximum window size, which is approximately the window size when last loss occurs or is infinitely large if the current window size exceeds the old maximum window size. The response function of Bic TCP is according to equation (1.4) in [3]. In Table 6, p is the loss rate, w is the congestion window size, x is the sending rate, and $w = x*RTT$. The symbols in the UDT formula have the same meanings as before.

The second type can be seen in FAST TCP [2], which uses queuing delay as a multi-bit congestion flag to tune the congestion window size with an equation-based method. FAST TCP extends TCP Vegas. The analysis of FAST and Vegas can be found in [2, 20], and more general analysis on delay-based approaches can be found in [19, 25].

TABLE 6
INCREASE/DECREASE AND RESPONSE FUNCTIONS OF RENO/SCALABLE/HIGHSPEED/BIC TCP

	Increase α	Decrease β	Response Function
TCP Reno	1	0.5	$1.22 \cdot p^{-0.5}$
Scalable TCP	$0.1w$	0.125	$0.8 \cdot p^{-1}$
HighSpeed TCP	$0.1578 \cdot w^{0.8024} \cdot \beta(w)/(2-\beta(w))$	$-0.0520 \cdot \ln w + 0.6892$	$0.12 \cdot p^{-0.835}$
Bic TCP	$min(target\_win - w, 32)$	0.125	$24.65 \cdot p^{-0.5}$
UDT	$10^{\lceil \log(L-C(x)) \rceil - 9}$	0.111	$3 \cdot 10^{\frac{-k+\lceil \log L \rceil - 9}{2}} \cdot p^{-0.5}$

According to [2], FAST TCP tunes the congestion window size every two RTTs, according to the ratio of *BaseRTT/RTT*, where *BaseRTT* is the minimum RTT observed so far.

However, on each packet loss event, FAST still decrease s its window size by 1/8.

The FAST algorithm converges to weighted proportional fairness [2].

The third type of congestion control algorithm for high BDP networks is to use explicit router feedback. XCP [6] is such a window-based protocol. In XCP, each router computes an increment or a decrement, which can be updated as it passes a successive router. The increment and decrement information is carried back by acknowledgements.

At each router, the XCP efficiency controller computes the aggregate feedback according to the available bandwidth and the persistent queue size.

The XCP fairness controller then distributes the aggregate feedback to all flows. If the aggregate feedback is positive, all the flows will have the same increase; if it is negative, each flow decreases in proportion to its own sending rate. The objective of this AIMD fairness controller is to make XCP satisfy max-min fairness.

In particular, XCP uses a control interval of the average RTTs of all flows.

The DAIMD algorithm can be classified into the first type. It uses a decreasing function of the increase parameter, and a constant decease parameter. However, the difference is that DAIMD tunes the sending rate based on time interval, which is similar to the second and the third approaches. UDT uses a constant control interval.

A lot of previous work has focused on the analysis of distributed congestion control algorithms. For example, Low's duality model has been used in analyzing TCP and AQM [1]. Kelly introduced a series of analysis on rate control and proportional fairness [8, 18]. Ott used a fluid model to describe the binary-based congestion control [32]. Bansal and Balakrishnan analyzed a group of binomial algorithms [30]. Gorinsky and Vin pointed out the limitations of Chiu and Jain's AIMD model and provided an extended analysis [31]. Srikant summarized the stability and fairness of Internet congestion control in [16].

IX. CONCLUSION

In this paper, we described a general type of AIMD congestion control algorithm, named DAIMD, whose increment decreases as the sending rate increases. This is different from other AIMD-based algorithms recently proposed to improve the performance of TCP at high BDP environments, which generally use large increase parameters.

DAIMD is stable and converges to max-min fairness equilibrium.

UDT is a special case of such algorithms. We showed that UDT can converge to 90% of the link capacity in a fixed time interval, independent of the network BDP. This makes UDT very scalable and efficient. UDT is fair when multiple flows have the same bottleneck capacities, and the unfairness is lower bounded when flows have different capacities. It is friendly to TCP in low BDP networks.

Because UDT uses a constant rate control interval independent of RTT, we also discussed the impact of network delay on the performance and used simulations to show that UDT can still work well even if the RTT is very large.

In addition, we described how to estimate the parameter of link capacity in UDT and discussed the impact of estimation error.

Finally, we listed some recent progress on congestion control algorithms in dealing with the high BDP environments of grid networks, and compared their mechanisms to DAIMD.

REFERENCES

[1] Steven H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Transactions on Networking (TON)*, Volume 11, Issue 4, (August 2003), Pages: 525 - 536.

[2] Cheng Jin, David X. Wei and Steven H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in *IEEE Infocom*, March 2004.

[3] Lisong Xu, Khaled Harfoush, and Injong Rhee, "Binary Increase Congestion Control for Fast Long-Distance Networks", in *IEEE Infocom*, March 2004.

[4] Sally Floyd, "HighSpeed TCP for Large Congestion Windows," RFC 3649, Experimental, December 2003.

[5] Tom Kelly, "Scalable TCP: Improving Performance in Highspeed Wide Area Networks," *Computer Communications Review*, April 2003.

[6] Dina Katabi, Mark Handley, and Chalrie Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks," in *the proceedings on ACM Sigcomm 2002*.

[7] Van Jacobson, Michael J. Karels, "Congestion Avoidance and Control," in *Proceedings of the Sigcomm '88,* Stanford, CA, August, 1988.

[8] Frank Kelly, "Fairness and stability of end-to-end congestion control," *European Journal of Control*, 9 (2003) 159-176.

[9] D. Chiu and R. Jain, "Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Networks," *Journal of Computer Networks and ISDN*, Vol. 17, No. 1, June 1989, pp. 1-14.

[10] C. Dovrolis, P. Ramanathan, D. Moore, "What do Packet Dispersion Techniques Measure?", in *Proceedings of IEEE Infocom*, April 2001.

[11] D. Loguinov and H. Radha, "End-to-End Rate-Based Congestion Control: Convergence Properties and Scalability Analysis," *IEEE/ACM Transactions on Networking*, vol. 11, no. 4, August 2003.

[12] Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", *Proc. ACM SIGCOMM'98* (Vancouver, CA, September 1998).

[13] Tom DeFanti, Cees de Laat, Joe Mambretti, Kees Neggers, Bill St. Arnaud, "TransLight: a global-scale LambdaGrid for e-science", *Communications of the ACM*, Volume 46, Issue 11, (November 2003), Pages: 34 - 41.

[14] UDT, *http://sourceforge.net/projects/dataspace*.

[15] W. Feng and P. Tinnakornsrisuphap, "The Failure of TCP in High-Performance Computational Grids," in *Proc. of SC 2000: High-Performance Networking and Computing Conf.*, November 2000.

[16] R. Srikant. "The Mathematics of Internet Congestion Control." *Birkhauser*, 2004.

[17] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, 3:115-156, 1992.

[18] F. P. Kelly, A.K. Maulloo, and D.K.H. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, 49 (1998), 237-252.

[19] J. Martin, A. Nilsson, and I. Rhee, *Delay-based congestion avoidance for TCP*. ACM/IEEE Transactions on Networks, June 2003.

[20] Steven H. Low, Larry L. Peterson, Limin Wang, "Understanding TCP vegas: a duality model," SIGMETRICS/Performance 2001: 226-235

[21] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *Proceedings of SIGCOMM* '98.

[22] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott, "The Macroscopic Behavior of the Congestion Avoidance Algorithm", *Computer Communications Review*, volume 27, number 3, July 1997.

[23] V. Paxson, "End-to-end Internet packet dynamics," in *Proc. ACM SIGCOMM*, pp. 139--152, September 1997.

[24] Kevin Lai, Mary Baker, "Measuring link bandwidths using a deterministic model of packet delay," *SIGCOMM 2000*: 283-294.

[25] R. Jain, "A Delay Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks," *Computer Communications Review*, ACM SIGCOMM, October 1989, pp. 56-71.

[26] Ravi Prasad, Manish Jain and Constantinos Dovrolis, "Effects of Interrupt Coalescence on Network Measurements," *PAM2004*, Antibes Juan-les-Pins, France, April 19-20, 2004.

[27] Yunhong Gu and Robert L. Grossman, "SABUL: A Transport Protocol for Grid Computing," *Journal of Grid Computing*, to appear.

[28] H. Sivakumar, R. L. Grossman, M. Mazzucco, Y. Pan, Q. Zhang, "Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks," *Journal of Supercomputing*, 2004, to appear.

[29] A. Chien, T. Faber, A. Falk, J. Bannister, R. Grossman, J. Leigh, "Transport Protocols for High Performance: Whither TCP?," *Communications of the ACM*, Volume 46, Issue 11, November, 2003, pages 42-49.

[30] Deepak Bansal and Hari Balakrishnan, "Binomial Congestion Control Algorithm," Proc. *IEEE INFOCOM* Conf., Anchorage, AK, April 2001.

[31] S. Gorinsky and H. Vin, "Extended Analysis of Binary Adjustment Algorithms," Technical Report TR2002-39, Department of Computer Sciences, The University of Texas at Austin, August 2002.

[32] T. J. Ott, J. H. B. Kemperman, and M. Mathis, "The stationary behavior of ideal TCP congestion avoidance," in Proceedings of IEEE INFOCOM'99, New York, 1999.