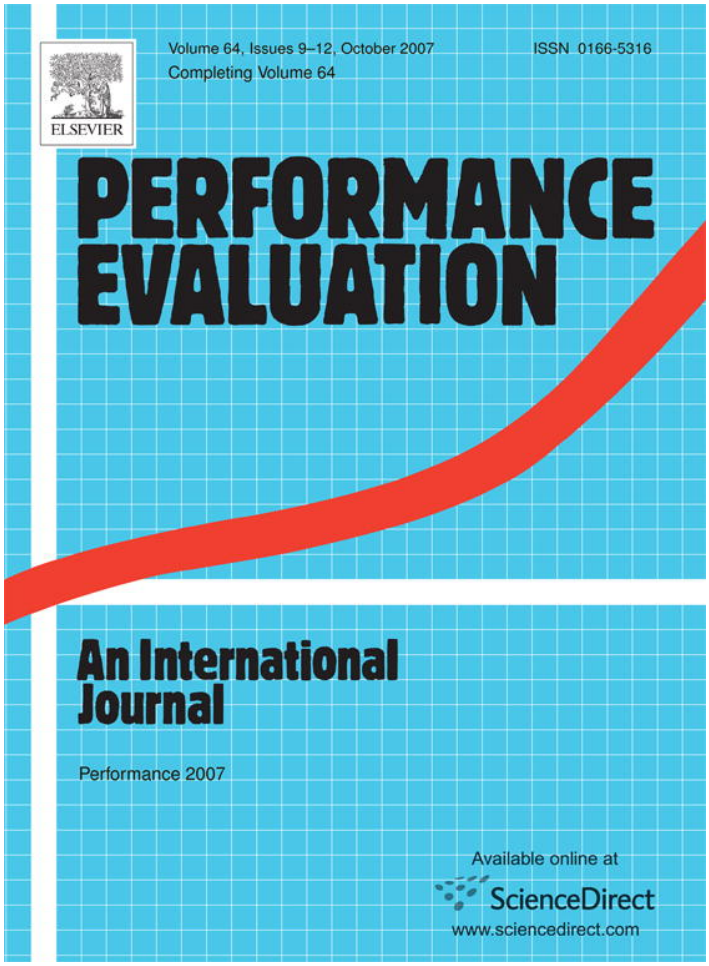


Available online at www.sciencedirect.com
ScienceDirect
Performance Evaluation 64 (2007) 948–964
PERFORMANCE EVALUATION
An International Journal
www.elsevier.com/locate/peva

Target bandwidth sharing using endhost measures

George Kola*, Mary K. Vernon

Computer Sciences Department, University of Wisconsin-Madison, United States

Available online 10 July 2007

Abstract

Recent congestion control protocols such as XCP and RCP achieve fair bandwidth sharing, high utilization, small queue sizes and nearly zero packet loss by implementing an explicit bandwidth share mechanism in the network routers. This paper develops new quantitative techniques for achieving the same results using only end-host measures. We develop new methods of computing bottleneck link characteristics, a new technique for sharing bandwidth fairly with Reno flows, and a new approach for rapidly converging to bandwidth share. A new transport protocol, TCP-Madison, that employs the new bandwidth sharing techniques is also defined in the paper. Experiments comparing TCP-Madison with FAST TCP, BIC-TCP and TCP-Reno over hundreds of PlanetLab and other live Internet paths show that the new protocol achieves the stated bandwidth sharing properties, is easily configured for near-optimal performance over all paths, and significantly outperforms the previous protocols.

Keywords: Performance; Internet transport protocol; Network queuing

1. Introduction

TCP-Reno has been the most widely used Internet transport protocol since the mid 1990s. However, Reno's small additive increase in throughput for each successful packet transmission, and multiplicative decrease in throughput for each packet loss event leads to suboptimal throughput over long high bandwidth network paths. This has led to the design of a number of new transport protocols with improved performance [5,9,10,12,14,15,21]. The key advantages and drawbacks of these recent protocols are outlined in Section 2.

This paper develops a new simple protocol, TCP-Madison, that is designed to meet the following goals over any network path:

(1) high total bottleneck link utilization
(2) low average backlog at the bottleneck link
(3) equal sharing of the bottleneck bandwidth among all Madison flows,
(4) fair sharing of the bottleneck bandwidth with Reno flows,
(5) during flow start up and whenever the cross-traffic load on the bottleneck link changes, converge *as quickly as possible* to the correct bandwidth share while maintaining low average network backlog,
(6) require minimum router support,

* Corresponding author.
E-mail addresses: kola@cs.wisc.edu (G. Kola), vernon@cs.wisc.edu (M.K. Vernon).

0166-5316/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2007.07.001

(7) use a simple set of protocol parameters that are easy to configure
(8) interoperate with existing widely deployed TCP protocols.

To achieve the goals, we first develop a new method for computing a target average backlog from the measured packet loss rate (p) so as to achieve the stated high bottleneck utilization, low average backlog, and fair bandwidth sharing goals.

We then develop new quantitative methods for computing several key bottleneck link characteristics from the basic sending host measures of minimum and average round trip time and estimated bottleneck link capacity. The bottleneck measures, computed adaptively as average RTT changes, include: average total backlog, the bottleneck buffer size, link utilization, cross traffic throughput, and the number of equivalent flows sharing the link. These measures may also be useful in other network applications, and are one key contribution of this paper.

Finally, we develop a method for computing the packet sending rate (S) that will achieve the target average network backlog.

We develop the new TCP-Madison protocol, which implements the fair bandwidth sharing using the new quantitative methods and requires five simple parameters. Since target average backlog is small under zero loss, and since increase in packet loss rate leads to a decrease in target average backlog which in turn leads to lower target sending rate, the new Madison protocol avoids congestion changes in target backlog without the need for additive increase or multiplicative decrease functions.

We have implemented TCP-Madison, as well as FAST TCP [12], BIC-TCP [21] and TCP-Africa [16]. We provide experimental performance comparisons of these protocols and TCP-Reno over a large number of live Internet paths with widely varying characteristics. The performance comparisons include protocol throughput as a function of time, average backlog, and loss rate. To our knowledge, these are the first experimental protocol comparisons to date that: (1) compare Reno, BIC-TCP, and TCP-Africa as well as our new protocol over actual Internet paths, (2) use wide-area dumbbell topologies to examine protocol bandwidth sharing in the presence of live Internet cross traffic, and (3) provide measures of live cross-traffic load as well as the protocol throughput as a function of time.

Experiments over congested as well as lightly loaded wide-area dumbbell topologies validate that the Madison protocol achieves rapid convergence to bandwidth share, high bottleneck link utilization, and low network backlog. The results also show that Madison greatly outperforms FAST TCP as well as Reno on congested and lightly loaded paths with RTT >70 ms, due to Madison's quick convergence to bandwidth share, rapid adaptation to changes in cross-traffic, and tolerance of packet loss while transmitting at bandwidth share. Regarding network backlog, the results show that with parameter settings that ensure fair sharing with Reno, the Madison flows have lower average backlog than Reno and FAST TCP on congested paths, and average network backlog up to seven or eight packets on lightly loaded links.

The remainder of the paper is organized as follows. Section 2 reviews related previous work in transport protocol design. Section 3 develops the new quantitative techniques, Section 4 defines the new TCP-Madison protocol, and Section 5 presents the experimental evaluations and comparisons of Madison, TCP-Reno, FAST TCP, and BIC-TCP. Section 6 concludes the paper.

2. Related work

In this section we review the previous protocols, and point out the key similarities and differences between those protocols and the approach taken in this paper.

Unlike Reno and other widely deployed TCP protocols, TCP-Vegas [3] and FAST TCP [12] detect new network congestion when network delay increases, and reduce throughput before inducing packet loss. Vegas computes the average network backlog for a flow from the average and minimum RTT or the path, and halts the additive increase in window size when the average backlog exceeds a given specified threshold (α). A recent analytic throughput model for Vegas [19] quantifies the RTT unfairness in Vegas and provides the path conditions under which the Vegas protocol cannot attain average backlog equal to α. FAST TCP further improves on Reno by implementing a more aggressive additive increase mechanism when the average network backlog for the flow is below the target, and by using paced packets (which are also considered in recent improvements to Reno [7]). The more aggressive additive increase in FAST TCP also reduces the RTT unfairness compared with Reno and Vegas. FAST TCP also changes the window size every other RTT, thus obtaining smoother estimates of RTT_{avg}. These features lead to improved performance over high speed long delay paths. On the other hand, FAST TCP's additive increase function is at most four-fold

more aggressive than Reno (for recommended protocol settings) and thus FAST TCP has relatively slow convergence to bandwidth share over lightly loaded paths with long RTT. Due to its use of the Reno multiplicative decrease on packet loss, FAST TCP also may not reach bandwidth share on a long path with moderate to high congestion. We show evidence of these problems in the experimental protocol comparisons in Section 5 of this paper. A further significant drawback of the Vegas and FAST TCP protocols is that the target backlog parameter α is set independently for each protocol implementation. Two flows sharing the same bottleneck will attain unequal shares of the bottleneck bandwidth if they have different values of the target backlog, even if they each have a similar RTT.

Like FAST TCP, the Madison protocol makes use of the Vegas method of computing the observed average backlog for the flow. However, unlike FAST TCP and Vegas which perform additive throughput adjustments when average backlog is below or above a fixed threshold, and multiplicative throughput decrease when a packet loss event occurs, Madison (a) responds to increase or decrease in packet loss rate by adjusting the target average backlog, and (b) directly computes the sending rate needed to achieve the target average backlog. New quantitative methods, not defined in the Vegas protocol, are needed to implement the new Madison mechanisms.

Another class of protocols, including BIC-TCP [21], STCP [15], HSTCP [9], and the recent E-TCP [10], use a significantly more aggressive additive increase function than Reno, as well as a significantly less aggressive decrease function. These protocols tolerate relatively high loss rate (e.g., 1%) in order to gain significantly better utilization of high speed bottleneck links. The major drawback of these protocols is that they induce significant loss for other flows, leading for example to significant throughput degradation for TCP-Reno flows that share the same bottleneck. Results in Section 5 illustrate this problem for the BIC-TCP protocol.

Unlike these more aggressive protocols, Madison does not induce packet loss when efficiently utilizing a high speed link. Also unlike these protocols, Madison computes a sending rate that will achieve fair and efficient bandwidth sharing without inducing packet loss. Like the E-TCP protocol, Madison can implement fair bandwidth sharing for flows that don't need reliable packet delivery by not retransmitting lost packets.

XCP [14] and RCP [5] use router feedback to achieve high performance bottleneck bandwidth sharing. In particular, each router estimates spare link capacity as well as the persistent queue size of the outgoing link. The protocol uses these values to compute desired change in aggregate throughput in order to maintain both high link utilization and low persistent queue size. RCP also estimates the number of active flows that are sharing the link (n), divides desired change in aggregate throughput by n, and sends the desired rate change back to the sending host. XCP instead translates the desired change in aggregate throughput to a desired additive increase or multiplicative decrease in the window for each flow. In both RCP and XCP, the rate-limiting link provides feedback to the sending host via fields in the packet header. Katabi et al. [14] have shown that for particular settings of the XCP parameters, XCP has stable convergence for any number of sources.

A key disadvantage of RCP and XCP is that more complex routers must be developed and widely deployed. Interoperability with existing widely deployed TCP protocols further increases the complexity of the router algorithms [14]. The protocol developed in this paper achieves the same goals as XCP and RCP. However, unlike RCP and XCP, we devise methods to achieve the fair bandwidth sharing, high bottleneck utilization, and low persistent queue size based on sender-based calculations rather than new network router support. Furthermore, XCP implements Reno multiplicative decrease packet loss recovery, whereas TCP-Madison implements congestion avoidance by reducing the target average network backlog when loss rate increases. Like XCP and RCP, Madison has a parameter that defines the control interval, but Madison does not require weights to determine the relative influence of persistent queue size and spare link capacity on changes in flow sending rate. Like XCP and RCP, Madison can also be modified to implement differentiated service instead of equal bandwidth sharing. Such modifications are left to future work.

3. New quantitative techniques

In this section, we develop the quantitative techniques that are used by the new TCP-Madison protocol to achieve the goals of fair bandwidth sharing, high bottleneck link utilization, low network backlog, and rapid convergence to bandwidth share. The input parameters for these techniques are the first four measures in Table 1. We comment further on how we obtain these four measures, adaptively as cross-traffic load changes, in Section 4.

Section 3.1 develops a new technique for computing a target average backlog (b_{target}) as a function of packet loss event rate (p) that will achieve high bottleneck link utilization, low network backlog, and fair bandwidth sharing with Reno. Section 3.2 discusses two key assumptions that are used to develop a few of the further calculations in

Table 1
Notation for path characteristics

Measure	Definition
C	Capacity of the bottleneck link (in Mb/s)
RTT_{avg}	Average round trip time for packets in a flow (in ms)
RTT_{min}	Minimum observed round trip time (in ms)
p	Packet loss event rate
$\bar{d}$	Average network queuing delay ($\text{RTT}_{\text{avg}} - \text{RTT}_{\text{min}}$) (in ms)
$\bar{b}$	Average network backlog (in 1500-byte packets)
$\bar{B}$	Average total backlog at the bottleneck link (in 1500-byte packets)
B_{size}	Router buffer size for the bottleneck link (in 1500-byte packets)
U	Utilization of the bottleneck link
$X_{x\text{traffic}}$	Cross-traffic throughput at the bottleneck link
n	Number of "equivalent flows" sharing the bottleneck link

the section. Section 3.3 develops new methods for computing the four bottleneck link characteristics at the bottom of Table 1. Finally, Section 3.4 develops new techniques for computing the sending rate that will achieve the target backlog. A key theme throughout this section is the use of quantitative methods to achieve near-optimal system design and operation.

3.1. Computing b_{target}

The goal is to derive a target average backlog $\bar{b}_{\text{target}}$ that (a) can be computed from packet loss rate 'p' and thus by all flows that share the bottleneck, and (b) shares bottleneck capacity fairly with Reno cross-traffic.

We begin by deriving the average backlog that a TCP-Reno flow achieves as a function of RTT_{avg}, RTT_{min}, and p. We derive this result based on a key insight about a recent TCP-Vegas throughput model [19], described below.

The Vegas model provides the following upper bound on the packet loss rate (p) in order for Vegas to be able to attain average backlog equal to α (see Eq. (12) in [19]):

$$p \leq \frac{32}{7W^2 - 36W + 32}, \tag{1}$$

where $W = \alpha \text{RTT}_{\text{avg}}/\bar{d}$.

By solving inequality (1) for α we obtain an upper bound on the α that is attainable for the path. For larger α, packet loss occurs before the average backlog grows to α, in which case Vegas reverts to Reno.

We make the following key observation. Since the upper bound on α is the point at which Vegas reverts to Reno behavior, the average backlog of a Reno flow over the same path is this upper bound on the feasible α. We obtain the average Reno backlog by solving inequality (1) for α:

$$\alpha_{\text{Reno}} = \frac{2}{7}\frac{\bar{d}}{\text{RTT}_{\text{avg}}}(9 + \sqrt{25 + 56/p}). \tag{2}$$

Note that the equation quantifies how the path characteristics impact the average Reno backlog, which in turn impacts the target average backlog that will achieve fair sharing.

We have experimentally verified that Reno flows achieve average backlog as given by the above equation. Those results (omitted to conserve space) also show that the average backlog of the Reno flows can vary from less than 0.25 packets to over 64 packets depending on the path characteristics.

We use Eq. (2) as a starting point for deriving an adaptive target backlog for a flow, b_{target} that (a) is a function only of p and $\bar{d}$, so that all flows sharing the bottleneck can independently compute the same value, (b) will achieve fair bandwidth sharing with TCP-Reno flows that share the same bottleneck link, and (c) will achieve low persistent bottleneck queue size yet high bottleneck utilization (U). To our knowledge, there are no other previously proposed adaptive values of target backlog that achieve these properties.

One drawback of Eq. (2) is that α_{Reno} is dependent on RTT_{avg}. That is, Reno's RTT average backlog, which equals throughput times $\bar{d}$, is inversely proportional to average RTT.

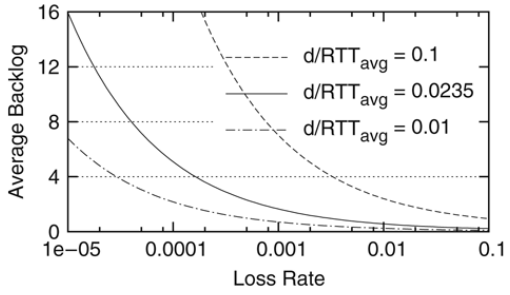


Fig. 1. Average Reno backlog vs p.

A key further observation is that the cause of the problem is also the solution to the problem. That is, since RTT unfairness is inherent in Reno flows, a flow that computes its own target backlog will be fair to Reno if it uses Eq. (2) with any feasible value of average RTT. In particular, a normalized RTT value that can be computed by all flows sharing a bottleneck, and that yields a suitably low value of average backlog, will provide all of the needed functionality. We further observe that if we let the normalized RTT value be proportional to $\bar{d}$, then $\bar{d}$ divided by normalized RTT becomes a constant. This has the advantage that the average backlog calculation in Eq. (2) depends only on this constant and p.

We further observe that letting the normalized RTT be proportional to $\bar{d}$, for a suitable proportionality constant can yield a small target backlog. The approach is intuitive because the normalized RTT will be smaller when the measured network delay $\bar{d}$ is smaller. This approach is also simple because for a suitable proportionality constant $\bar{d}/\mathrm{RTT}_{\mathrm{norm}}$, Eq. (2) shows that the average Reno backlog will be a function of the proportionality constant and p. What remains is to define the constant.

Fig. 1 plots the average Reno backlog as a function of packet loss rate for three possible fixed values of $\bar{d}/\mathrm{RTT}_{\mathrm{avg}}$. Note that the normalized ratio of 0.0235 yields a target backlog of 16 when the loss rate is 10^{-5}, 4.33 when the loss rate is 10^{-4}, and decreasing target average backlog as loss rate increases above 10^{-4}. Since average total backlog of 4.33 leads to high bottleneck link utilization for a paced flow, per-flow average backlog should only grow beyond that if loss rate is extremely low. For these reasons, 0.0235 is a normalized ratio that achieves a desirable trade-off between high link utilization and small persistent per-flow backlog. For this ratio, if average bottleneck delay $\bar{d}$ is 0.235 ms, the normalized $\mathrm{RTT}_{\mathrm{avg}}$ is 10 ms, and so forth.

We note that this is not the only possible choice for normalized RTT. Other functional dependences between $\bar{d}$ and normalized RTT are possible or other values of $\bar{d}/\mathrm{RTT}_{\mathrm{avg}}$ could be selected. However, the curve for $\bar{d}/\mathrm{RTT}_{\mathrm{avg}} = 0.0235$ also reflects a range of average backlog as a function of p that corresponds well with the range of average backlog measured for Reno flows over a wide range of paths on today's Internet, noted above. This provides further motivation that this choice of normalized RTT will have an appropriate level of aggressiveness for incremental deployment in networks that use the widely deployed Reno protocol. The target average backlog calculation for this choice of normalized average RTT is thus given by:

$$b_{\mathrm{target}} = \frac{2}{7}0.0235(9 + \sqrt{25 + 56/p}). \tag{3}$$

All flows sharing a given bottleneck link should have the same value of the $\bar{d}/\mathrm{RTT}_{\mathrm{avg}}$ ratio. When a flow achieves the computed b_{target} it will (a) interfere with other Reno flows as if it were a Reno flow with a normalized average RTT equal to $\bar{d}/0.0235 = 42.55\bar{d}$, and also (b) share bottleneck bandwidth equally with other flows that have computed the same target average backlog.

3.2. Key assumptions

To develop a few of the further calculations of interest in the next two sections, we make two key assumptions.

One key assumption is that a non-negligible average network delay or backlog for an Internet flow is primarily due to queuing at the most congested link in the path. We make this assumption for three reasons. First, non-negligible queuing delays generally occur at lower-bandwidth access links and peering links, rather than in high speed LANs at the endpoints or at high speed backbone links. Second, if the two or more lowest bandwidth links in the path have

the same capacity (C), it is unlikely that those links have exactly the same cross-traffic load. If there is even a modest inequity in the cross-traffic load at the lowest capacity links, most of the average total queuing delay in the path will occur at the most congested node. Finally, if the average network delay is due to queuing at more than one node in the network, the impact of assuming that it occurs primarily at the bottleneck link is that the flow will conservatively estimate somewhat higher than actual congestion at the bottleneck, and thus the flow may use somewhat less than its fair share of the bandwidth. We have not observed that problem over any of the paths used in our experimental protocol comparisons.

In a few calculations, noted in the sections below, we make the assumption that the bottleneck link can be modeled approximately as an $M/D/1$ queue. We note that the goal is to enable design of the system from first principles as well as near-optimal functionality during system operation, yet also to develop a protocol that is tolerant of the error that may exist in the model calculations. Key functionality enabled by the model includes the capability to compute a target sending rate for rapid convergence to the computed target average backlog.

In support of the approximate model, we note that prior work has provided experimental evidence as well as theoretical justification that as network links carry flows from larger and larger aggregations of sources, the packet arrival process for the link tends toward the Poisson [4,13]. Bottleneck links with high aggregation are increasingly common as Internet bandwidths increase. We also note that large flows dominate the load on network paths, that such flows have large packet sizes, and hence the overall coefficient of variation in the packet sizes that arrive to the link is very small. We thus model the bottleneck link as an $M/D/1$ queue with customers that are packet of size 1500 bytes. We thus assume that a packet that measure $d = \mathrm{RTT} - \mathrm{RTT}_{\min}$ is measuring delay due to 1500-byte service times that generally do not coincide with packet boundaries.

Links with lower flow aggregations are known to have packet arrivals that are more bursty than Poisson. On the other hand, if protocols increasingly use packet pacing, packet arrivals in some network paths could be somewhat less bursty than Poisson. We keep in mind the qualitative direction of the error in the estimates calculated for the $M/D/1$ queue, and take suitable precautions when applying the results of the model in the protocol.

We apply the $M/D/1$ model, for example, in Section 3.4 to compute an estimate of the sending rate that will achieve a given target average backlog. We provide experimental results in that section, and again in Section 5 that show the computed sending rate achieves very close to the intended target average backlog, which we found over the wide range of Internet paths we've examined.

3.3. Computing bottleneck measures

Previous work [3] has shown that it is possible to compute the average network delay for the packets in a flow, $\bar{d} = \mathrm{RTT}_{\mathrm{avg}} - \mathrm{RTT}_{\min}$. Assuming the flow has been sending at rate S, that work also uses Little's result to compute the average network backlog for the flow, in units of 1500-byte packets, as follows:

$$\bar{b} = S\bar{d}. \tag{4}$$

We compute the average total backlog for all flows that share the bottleneck link, $\bar{B}$ measured in units of 1500-byte packets, by first observing that the time to transmit an MTU on a link with bandwidth C Mb/s or kb/ms is equal to:

$$\frac{1.5 \text{ kb/packet} \times 8 \text{ bits/byte}}{C \text{ kb/ms}} = \frac{12}{C} \text{ ms/packet}. \tag{5}$$

If $\mathrm{RTT}_{\mathrm{avg}}$ is measured using randomly arriving probe packets (as defined in Section 4), then the total average queue size in units of 1500-byte packets is

$$\bar{B} = \frac{\bar{d}}{\frac{12}{C}} = \bar{d}\frac{C}{12}. \tag{6}$$

To obtain the bottleneck link buffer size, B_{size}, we note that the Madison protocol will attain high link utilization (e.g., greater than 90%) in order to maximize flow throughput and thus minimize average flow transmission time. In case of $p > 0$ and cross-traffic that induces loss, we can expect that some fraction of the arriving packets will observe queue length close to the maximum possible, and thus we use the maximum observed value of $\bar{B}$ as an estimate of B_{size}. We also obtain this value for the BIC protocol, which induces loss on most paths. Note that TCP-Madison does

not use B_{size} but we report this value as a useful path characteristic in our experimental results in Section 5. We have found that the BIC estimate in particular is reproducible in experiments over the same path.

The next goals are to compute bottleneck link utilization, cross-traffic throughput, and number of equivalent flows that share the bottleneck.

To compute bottleneck utilization, we use the approximate $M/D/1$ queue for the bottleneck, in which case we can compute the average total delay in the queue as follows:

$$\bar{d} \approx \frac{U12/C}{2(1-U)}. \tag{7}$$

Solving this equation for U yields

$$U \approx \frac{2\bar{B}}{2\bar{B}+1}. \tag{8}$$

We note that the above is an overestimate of U if the actual packet arrival process is more bursty than the Poisson. That is, a higher U is needed to observe the measured $\bar{d}$ if the arrivals are Poisson than if they are more heavy tailed, for the same mean interarrival time.

By Little's result, the link utilization by a flow that has been sending at rate S, where S is in units of 1500-byte packets/ms is

$$U_{\text{flow}} = S \times 12/C. \tag{9}$$

Finally, we compute the desired cross-traffic throughput by Little's result,

$$X_{xt} = \frac{U_{xt}}{12/C} = [U - U_{\text{flow}}] \times C/12. \tag{10}$$

Using the average network backlog for the flow ($\bar{b}$) and the average total bottleneck backlog, we define the following average number of "equivalent flows" that share the bottleneck link:

$$n = \frac{\bar{B}}{\bar{b}} = \frac{C \times U}{12S}. \tag{11}$$

Thus, by "equivalent flow" we mean another flow, or portion of another flow, or possibly an aggregation of flows that produces the same average backlog at the bottleneck. We use this and other quantities derived above in the next section. These formulas might also prove useful in other network applications.

3.4. Target sending rate

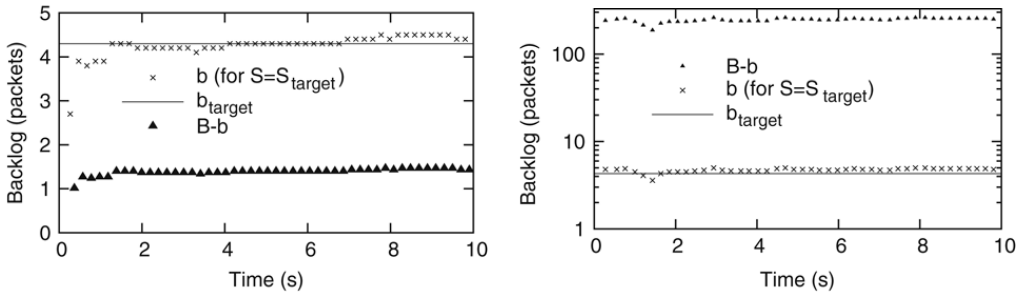
In this section we use the $M/D/1$ queue approximation for the bottleneck link to derive a target sending rate for rapidly converging to a specified new target average backlog (b_{target}) during flow start up or when cross-traffic load changes. After deriving the new S_{target} we point out that it is an underestimate of the actual sending rate that will achieve the target backlog if packet arrivals are more bursty than Poisson. We note that computing the sending rate without router support and in this manner is a new approach. Section 4 provides the reasons to believe that the approach is sound, including the key idea that the calculated sending rate will be used in a way that tolerates error in the estimate.

By Little's result, the desired S_{target} times the new average delay at the bottleneck link after one or more flows transmit at the new target rate, will be equal to b_{target}. Using the $M/D/1$ formula for the average delay, we have:

$$S_{\text{target}} \frac{12/C}{2(1-U_{\text{new}})} = b_{\text{target}} \tag{12}$$

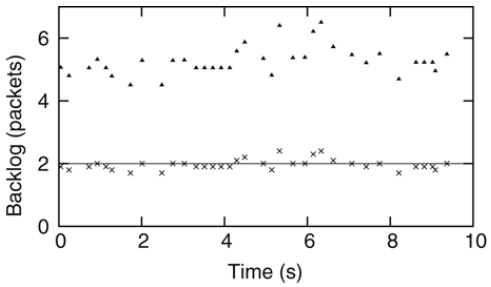
where U_{new} is the predicted new utilization of the link after the changes in sending rate, derived below.

Note that to simplify the calculation of S_{target}, we have omitted the factor of U_{new} in the numerator of the formula for average delay. If the target link utilization is on the order of 0.9 or higher, omitting this factor will not greatly impact the accuracy of the approximate value for S_{target}. On the other hand, the factor of U_{new} can easily be included



(a) ($U_{xt} < 20\%$) $C = 98$ Mb/s, RTT = 144 ms WI to Dortmund, Germany.

(b) $U_{xt} \approx 99\%$ $C = 155$ Mb/s, RTT = 127 ms WI to Tubingen, Germany.



(c) $U_{xt} \approx 80\%$ $C = 98$ Mb/s, RTT = 103 ms WI to Paris, France.

Fig. 2. Average backlog achieved by S_{target}.

in the future if the additional accuracy is desired. For the results obtained in over a wide range of Internet paths, reported in Section 5, this simpler formula has proved to be sufficient.

To compute the predicted value of U_{new}, we consider two cases. First, when a flow starts and has not yet achieved within 20% of its first target backlog (computed for an initial loss rate parameter as defined in the next section), we assume that no other flows have yet measured the impact of the new flow and thus the only flow changing its sending rate is the new flow. Assume this flow has been sending packets at rate S, where $S = 0$ at the very beginning of the flow. In this case,

$$U_{\text{new}} = \max\left(0, U - S\frac{12}{C}\right) + S_{\text{target}}\frac{12}{C}. \tag{13}$$

Substituting the above equation in Eq. (12) and solving for S_{target} yields the desired estimate of the target sending rate. Note that since U is overestimated when the actual packet arrivals are more bursty than Poisson, U_{new} will also be overestimated in that case, and S_{target} is likely to be a conservative value.

The second case occurs after a flow has achieved within 20% of its first target backlog. In this case, it computes the number of "equivalent flows" sharing the bottleneck link, n, as defined in Eq. (11).

Let n_D, initially equal to n, be the number of those flows that will compute a new target backlog and/or a new target sending rate if p or $\bar{d}$ changes. We assume the other $n - n_D$ flows make gradual changes in their sending rate, and thus we estimate U_{new} as follows:

$$U_{\text{new}} = \left[\max(0, X_{xt} - (n_D - 1)S) + n_D S_{\text{target}}\right] 12/C. \tag{14}$$

When Eq. (14) for U_{new} is substituted in Eq. (12), the sender can solve for S_{target} and get the new rate. If the desired backlog is not achieved, it could be that the value of n_D is incorrect. The sending host can compute a new value of n_D by (a) substituting the achieved target backlog in place of b_{target} in Eq. (12) and (b) back calculate the value of n_D to determine how many flows changed in order to achieve the observed backlog.

Fig. 2 provide typical results of actual backlog achieved by a new flow that measures the path characteristics ($\bar{d}$, RTT_{avg}, and C), computes the sending rate to achieve a target average backlog using the above formulas, and transmits packets at that calculated rate for ten seconds. The average backlog achieved by the computed sending rate is plotted

Table 2
TCP-Madison protocol parameters

Parameter	Definition	Value
$\frac{\bar{d}}{\text{RTT}_{\text{norm}}}$	Normalized $\bar{d}/\text{RTT}_{\text{avg}}$	0.0235
$p_{\min}$	Minimum loss rate for b_{target} calculation	2.55×10^{-5}
p_{init}	Initial loss rate estimate	10^{-4}
T	Minimum time between rate adjustment	200 ms
$tProbe^*$	Minimum spacing between probe packets	32 packet service times

as a function of time, as is the target average backlog and the average backlog due to the cross traffic. Note the different scales on the y-axis in each figure, and note that the path in Fig. 2(c) has high variability in the average cross-traffic backlog over time. As can be seen in the figure, over live Internet paths with substantial RTTs and different cross-traffic loads, the $M/D/1$ formula computes a value of the sending rate that enables the new flow to achieve the desired average backlog.

4. The TCP-Madison protocol

In this section, we develop the TCP-Madison protocol that uses the methods developed in the previous section to achieve rapid and efficient convergence to fair bandwidth share as well as low average backlog and high bottleneck link utilization.

We first describe the protocol parameters and measures (Section 4.1) and then outline the protocol operation during Flow Start-up (Section 4.2) and the remainder of the flow (Section 4.3).

4.1. Protocol parameters and measures

Table 2 summarizes the five parameters of the TCP-Madison protocol. Recall from Eq. (3) and Fig. 1 that the first parameter ($\bar{d}/\text{RTT}_{\text{norm}} = 0.0235$) determines a desirable target average backlog (b_{target}) as a function of measured packet loss rate (p). The second parameter, $p_{\min}$, is the minimum loss rate that will be used to compute b_{target}. Note that $b_{\text{target}} = 10$ for $p_{\min} = 2.555 \times 10^{-5}$. These parameters should have the same values in all implementations of TCP-Madison, in order to achieve equal bandwidth sharing among Madison flows that share the same bottleneck link.

The third parameter p_{init} is the initial value of loss rate assumed for the path. Note that $b_{\text{target}} = 4.33$ for $p_{\text{init}} = 10^{-4}$. This value has the desirable properties of relatively high initial utilization of a lightly loaded bottleneck link, relatively low increase in total backlog for a congested link, and enough perturbation in the total backlog by the new flow for other Madison flows to observe the impact of the new flow and recompute their bandwidth share, as described in Section 4.3 below. Hence, this parameter should also be set to the specified value in all implementations of TCP-Madison.

The fourth parameter, $T = 200$ ms, is the minimum spacing between changes in the sending rate. This parameter achieves a trade-off between responsiveness to changing path conditions on the one hand, and ensuring that the flow has enough opportunity to observe the new bottleneck characteristics including impact due to changes in sending rate by other Madison flows before making another change in its own sending rate.

The fifth and final parameter, $tProbe^*$ is the desired minimum average time between "probe packets", where probe packets are data packets that will be used to measure RTT. These packets have a new "ACK request" bit in the header set to one. All other data packets have the ACK request bit set to zero.

The specified value of $tProbe^*$ is 32, in units of a packet service time on the bottleneck link. The principal reason for the desired minimum average probe packet spacing is to obtain relatively independent samples of $\text{RTT} - \text{RTT}_{\min}$. We have derived the distribution of the $M/D/1$ busy period with server utilization equal to 90% from results in [2], and assuming (approximate) exponential spacing between probe packets, we have calculated that if average probe spacing is equal to $tProbe^* = 32$, then 84% of the probes will arrive in different busy periods on the bottleneck link. Such probes will generate relatively independent samples of observed backlog. We have further experimentally verified that such probe spacings work well in the protocol specified below. A more detailed study of probe packet spacing is left for future work.

Since packet loss events are relatively rare and aren't detected until the ACK for a subsequent packet reaches the sender, it is also safe to use only the ACKs from probe packets to determine whether a packet loss event has occurred. In other words, for recomputing p, b_{target}, and retransmitting packets for reliable flows, the delay due to probe packet spacing has the same impact on loss responsiveness as increasing the round trip time by the probe packet spacing. A TCP-Madison sender can interoperate with a TCP-Reno receiver, but the sender will operate more efficiently if the receiver TCP is modified to return a selective ACK only when the ACK request bit is set. If the receiver sends ACKs more often, the sender will selectively use the probe packet ACKs for RTT_{avg} calculation.

We have found that the relatively intuitive parameters values given in the table are very robust, and thus that the protocol is easy to configure for near-optimal operation over a very wide range of Internet paths. Further experimentation is needed, but the results to date are encouraging.

4.2. Flow start-up

The sending host performs four principal steps to obtain the four basic path measures needed for protocol operation during Flow Start-up. First, a 3-way handshake for connection setup. Second, using a fixed spacing of 15 ms, transmit a packet pair, followed by four probe packets with the ACK request flag set, followed by another packet pair and another four probe packets. Third, when the ACKs for the first four packets have been received, compute initial estimates of RTT_{min}, RTT_{avg}, and C. Finally, if 15 ms is less than $20 \times 12/C$ or 20 packet service times on the bottleneck link, wait for the ACKS of the last four probes and use every other probe ACK (i.e., four of the eight probe ACKs) as well as both packet pairs to compute initial estimates of RTT_{min}, RTT_{avg}, and C.

We note that the initial estimates of RTT_{avg}, RTT_{min} and C are very approximate due to a small number of samples with spacing that may violate $tProbe^*$. However, the goal is to initiate the flow quickly using these crude measures, and then to refine the measured values during the Congestion Avoidance phase. To that end, the Congestion Avoidance phase sends another packet pair during each RTT for the flow until 20 packet pairs have been sent, and the sender also continues to recompute RTT_{min}.

Set $p = p_{\text{init}}$. Compute b_{target} and S_{target}.

Compute $tProbe^* \times 12/C$ and set the desired average time between probe packets to the minimum of this value and 100 ms. If the minimum time between probe packets is under 15 ms, set it to 15 ms. Note that these rules sacrifice the desired independence of the probe RTT samples if $tProbe^* \times 12/C$ is too long, and or improve the quality of the samples if $tProbe^* \times 12/C$ is very small. Once these steps are complete, the protocol moves to the congestion avoidance phase.

4.3. Congestion avoidance

In this phase, the sender performs the following operations until all data is sent:

(1) Transmit packets with equal spacing at rate S_{target}, but shift sending time for probe packets that have the ACK request flag set, so as to achieve exponential spacing of those packets with average that is the maximum of $1/S_{\text{target}}$ and the desired spacing computed during flow start-up.
(2) On each probe ACK received,
   (a) compute the RTT. Update RTT_{min} if this new RTT is smaller. Computer RTT_{avg} by taking the average of the six most recent RTT samples.
   (b) If a packet loss event is detected (for example by two consecutive selective ACKs that contain the same packet missing) recompute p as follows: $p = \frac{\#\text{ packets lost}+1}{\#\text{ packets sent since last loss}+1/p}$. Recompute b_{target} and S_{target}.
   (c) Otherwise, if the number of packets sent since the last loss event (q) is $>10/p$ let $p = 1/q$. Recompute b_{target} and S_{target}.
   (d) Otherwise, compute $\bar{b}$ from RTT_{avg} and store the most recent two values of $\bar{b}$. If $\bar{b}$ is within 20% of b_{target}, compute n and set $n_D = n$. Otherwise, if two consecutive values of $\bar{b}$ are more than 30% higher or lower than b_{target}, compute a new S_{target}.

Note that ACK requests occur at most once every 15 ms and at least once every 100 ms, depending on bottleneck bandwidth. Such spacing enables Madison to respond to congestion and packet loss, but greatly reduces overhead for ACK processing, especially in high bandwidth paths. If a requested ACK is delayed (or lost), the sender quiesces until the next ACK arrives of time-out occurs. This implements a coarse-grained ACK clocking in TCP-Madison.

Table 3
Numbers of protocol evaluation experiments

Available bandwidth (Mb/s)	Round trip time (ms)			
	[0.2, 10]	(10, 100]	(100, 200]	(200, 823]
[0.1, 1]		379	641	732
(1, 10]	155	2232	2217	1830
(10, 50]	323	2291	2268	1821
(50, 550]	861	8239	3037	2645

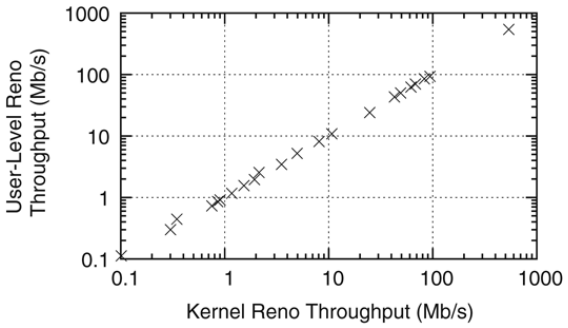


Fig. 3. Accuracy of user-level Reno performance.

In the next section we show that the TCP-Madison protocol achieves the goals it is designed to achieve, and outperforms previous protocols over many Internet paths including those between PlanetLab nodes.

5. Protocol performance comparisons

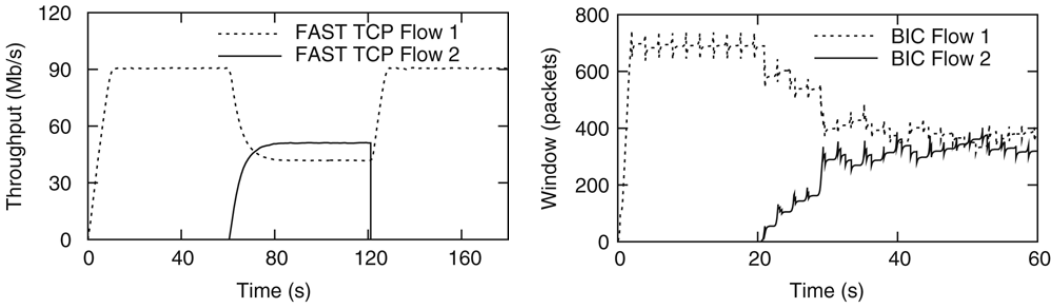
In this section we provide experimental results that (1) demonstrate that TCP-Madison meets its design goals over a wide range of Internet paths, (2) compare Madison performance against Reno, FAST TCP, BIC-TCP and TCP-Africa and (3) gain new insights and understanding of the performance of previous protocols that do not require router modification.

We have experimented with Madison and the previous protocols on over 400 Internet paths with a wide range of path characteristics over a period of many months. These experiments were performed at a variety of times to observe different amounts of cross-traffic load on a given path. (In some cases, the cross-traffic load varied from 10% of capacity to greater than 90% of capacity.) We used PlanetLab nodes as well as some additional nodes that increase the variety of paths explored. Table 3 shows the number of experiments performed broken down by available bandwidth and RTT.

Section 5.1 describes the experimental setup as well as the variety of paths explored. Section 5.2 presents representative results for how parallel protocol transfers over wide-area Internet dumbbell topologies share the bandwidth of a common bottleneck link. Section 5.3 presents results from back-to-back protocol transfer over further Internet paths.

5.1. Experimental setup

The PlanetLab (and other) experiments require user-level protocol implementations that have been optimized to achieve kernel-level performance measures, similar to previous work [6,18,20]. Our implementation while being similar to previous work has two important features. First, it has a measurement mode where the receiver copies only the packet header from the kernel thereby reducing overhead and enables us to use moderately loaded PlanetLab nodes as receivers. Second, we detect context switches and determine the amount of time context switched out and discard experiments affected by context switches. Fig. 3 shows that our steady state user level Reno throughput (y-axis) is nearly identical to steady state kernel level Reno throughput (x-axis), for Internet paths with a wide range of Reno throughputs.



(a) FAST TCP ($\alpha = 8$, $\gamma = 1$) $A/C = 0.96$, $C = 94$ Mb/s.

(b) BIC-TCP ($S_{\max} = 32$, $S_{\min} = 0.01$) $A/C = 0.85$, $C = 100$ Mb/s.

Fig. 4. Behavior of the protocol implementations (RTT = 61 ms, 14 shared hops).

Fig. 5. Wide-area Internet dumbbell topology.

Table 4
Baseline and other experimental paths

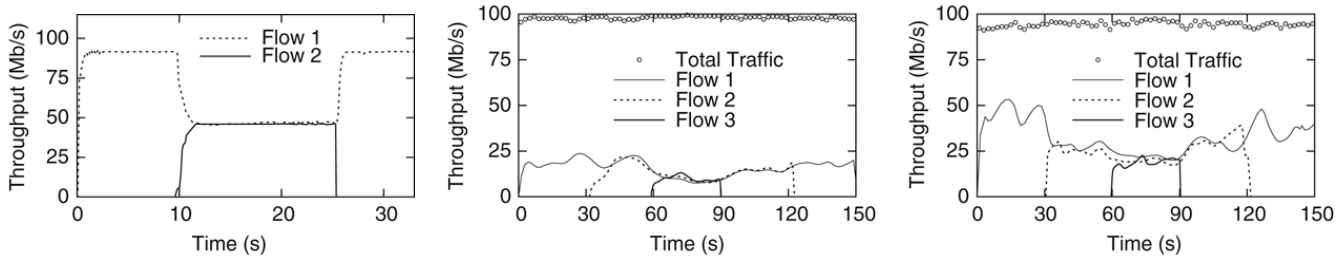
	Baseline paths	Other key paths	Total range
RTT_{min} (ms)	≥200		0.1–823
C (Mb/s)	≥100	1–10	0.128–1000
$U_{x\text{traffic}}$ (%)	70–95	<1, 99	1–99
Loss rate	0, 10^{-4}, 10^{-3}, 10^{-2}	10^{-4}, 10^{-3}, 10^{-2}	0–0.08
B_{size} (packets)	150, 1000		40–4800
X_r (5 s)	≤1.1	>1.5	1–2.6

We have also validated each of our protocol implementations against previously published results. For example, Fig. 4(a) contains results similar to Fig. 8 in [12], showing that two parallel FAST TCP flows sharing the same bottleneck link quickly converge to nearly equal bandwidth share over paths with moderate RTT. Fig. 4(b) shows convergence behavior of two parallel BIC flows sharing a bottleneck similar to the results in Fig. 5 of [21]. Our results similar to Fig. 5 of [16] for TCP-Africa transfer and other additional validation results are omitted to conserve space. The results of these and other validation experiments indicate that our protocol implementations are operating correctly.

We use the protocol implementations to measure throughput and packet loss rate in back-to-back transfer as well as for parallel transfers over paths that share a common bottleneck link. In the latter case, we use new wide-area Internet dumbbell topologies as shown in Fig. 5. In such topologies, each host at site A is paired with a host at site B and flows are run in parallel between each host pair starting at different times.

In all results reported in this section, we set $W_{\max} = 172$ kb for Reno. We set FAST TCP α as mentioned in [8] to two times the maximum observed FAST throughput in pkts/ms measured over 200 s with a minimum value of 8. We set BIC $S_{\min}$ to 0.01 and $S_{\max} = 32$ as mentioned in [21].

To organize the variety of paths used in the experiments, we group the paths as follows. First, we consider a large subset of the paths having the "baseline" parameter values shown in Table 4 that provide among the most challenging conditions for transport protocols. Those parameters include large RTT (≥200 ms), the high end of current bottleneck link capacity (C = 100–500 Mb/s) and significant cross-traffic load (>70% of C). Those paths include a wide range



(a) $U_{xt} < 10\%$,
no loss, $B_{\text{size}} = 318$ KB
RTT = 59 ms, 14 hops, WI to CA.

(b) $U_{xt} \approx 85\%$,
no loss, $B_{\text{size}} = 484$ KB
RTT = 246 ms, 16 hops, WI to Brazil.

(c) $U_{xt} \approx 60\%$,
$p = 4 \times 10^{-4}$ $B_{\text{size}} = 142$ KB
RTT = 27 ms, 11 hops, WI to NY.

Fig. 6. Bandwidth sharing among Madison flows ($C = 100$ Mb/s).

of loss rates and bottleneck link buffer capacities as shown in Table 4. The baseline paths also include commercial as well as high-performance (futuristic) research network paths. Finally, the baseline paths have low variation in the average cross-traffic load ($X_r \leq 1.1$, where X_r is the ratio of the maximum cross-traffic load to minimum cross-traffic load observed during any given 5 s period of a 30 s data transfer on the path). This is because very few paths available (<3%) have significant variation in cross-traffic load ($X_r > 1.5$) as observed over periods of several minutes. In addition to the baseline cases, we also examine the performance of the protocols with paths that have other key characteristics as given in the table.

In the case of back-to-back experiments, we observe as in previous work [1,17,22] that the path characteristics remain stable over time horizons of several minutes. This is true for cross-traffic utilization above 90% as well as for low to moderate cross traffic utilisation. We run two 30 s transfers (one Reno and one Madison) at the beginning and at the end of each back-to-back experiment for all protocols. We use the results only if there is less than 10% difference between the two Reno throughputs and less than 10% difference between the two Madison throughputs, to ensure that the protocols are compared under very similar conditions. Note that throughput within each transfer may vary if cross-traffic load varies on smaller time scales but the average cross-traffic load remains steady in each back-to-back experiment.

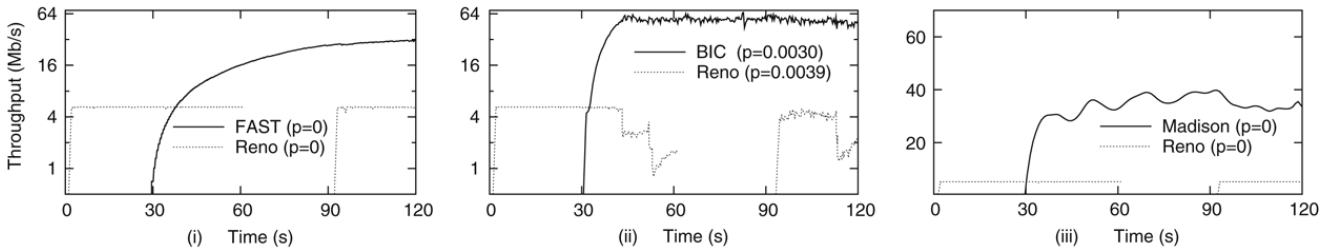
Results for the baseline and other key path characteristics are shown in the next two sections.

5.2. Protocol bandwidth sharing performance

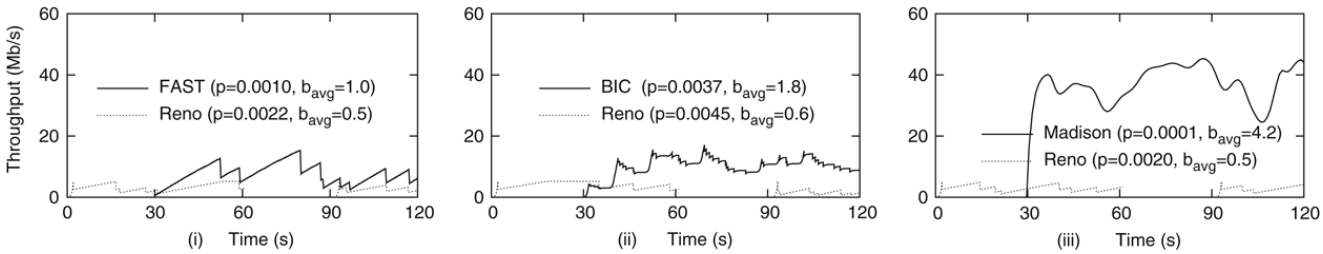
In this section we provide bandwidth sharing results for parallel flows that share a common bottleneck link. As in previous work, we start the flows at different times and observe: (1) how quickly new flows converge to their bandwidth share, (2) how equally the flows share the bottleneck capacity, and (3) the impact of loss rate on sharing. We also use a wide-area dumbbell topology (Fig. 5) so that the flows are transmitted over actual Internet paths and share an actual Internet bottleneck link.

We first evaluate bandwidth sharing among parallel Madison flows, with representative results provided in Fig. 6. Fig. 6(a) considers the case where the bottleneck link has very low load (under 10%) from the normal Internet traffic (at the time of the experiment). Under such conditions, protocol throughput is very stable and convergence to equal bandwidth share is rapid, similar to and slightly better than the results for FAST TCP in Fig. 4(a). Fig. 6(b) consider a more challenging "baseline" path with higher RTT (246 ms) and significant fluctuating load due to the background Internet traffic. Again we observe rapid convergence to bandwidth share for each of three parallel Madison flows (starting at time 0, 32 s, and 59 s, respectively). Madison throughputs fluctuate as the background traffic load fluctuates, with total throughput (for Madison flows plus background flows) remaining steady and close to the bottleneck capacity (100 Mb/s). No packet loss is observed, likely due to the fairly large bottleneck buffer capacity (484 KB) on this hybrid commercial and high speed research network path. Results in Fig. 6(c) are similar to Fig. 6(b) for a commercial Internet path with lower RTT and lower bottleneck buffer capacity (142 KB). In this case packet loss also contributes to Madison flow throughput fluctuations, yet Madison still achieves near-equal bandwidth sharing and fairly steady total bottleneck link load.

We next evaluate how FAST TCP, BIC, and Madison each share bandwidth with parallel Reno flows that share a common bottleneck. Representative results for the challenging "baseline" paths are provided in Fig. 7(a)



(a) $B_{\text{size}} = 1320$ KB, RTT = 260 ms, 16 hops (WI to Brazil).



(b) $B_{\text{size}} = 140$ KB, RTT = 261 ms, 15 hops (WI to central China).

Fig. 7. Bandwidth sharing with Reno ($U_{xt} \approx 70\%$, $C = 100$ Mb/s, Reno $W_{\max} = 172$ KB).

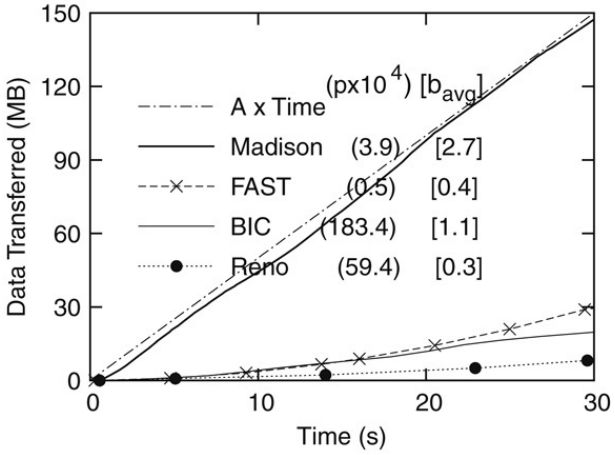
($B_{\text{size}} = 1320$ KB) and Fig. 7(b) ($B_{\text{size}} = 140$ KB). In both figures there is significant bottleneck load (60%–70%) due to other Internet cross-traffic. Note that in each figure, a Reno flow is started at time 0. At 30 s, another flow using the given newer protocol, such as FAST TCP in Fig. 7(a)(i), starts. At 60 s, the Reno flow terminates, and at time 92 s another Reno flow starts. In Fig. 7(a)(i), there is no observed packet loss. The FAST TCP flow shares bandwidth fairly with the on-going and new Reno flow, and gradually converges to a significantly higher throughput that fully utilizes the bottleneck. The gradual convergence, due to the longer RTT than in Fig. 4(b), has also been observed in previous work [11]. In Fig. 7(a)(ii), BIC rapidly achieves its bandwidth share, but the BIC flow causes degradation in the throughput and loss rate for the on-going as well as new Reno flow. Fig. 7(a)(iii) shows that the Madison flow rapidly converges to its bandwidth share while not degrading the performance of the Reno flows. FAST and Madison cause no degradation in the Reno flows because they maintain a low average backlog in the network. Fig. 7(b) provides similar results for the case where packet loss occurs due to smaller bottleneck buffer capacity. Note that FAST and BIC achieve lower throughputs due to the packet losses at a bottleneck with fairly high load from existing Internet traffic. To our knowledge, the impact of existing Internet traffic on FAST TCP and BIC performance has not previously been published. Fig. 7(b)(iii) shows that Madison throughput fluctuates more due to greater fluctuations in the bottleneck queue length; on the other hand, overall bottleneck throughput (not shown in the figure) remains steady and close to C. For paths with lower RTT, lower C, and/or less cross-traffic, the differences among FAST, BIC, and Madison are less pronounced. From the large number of experiments, we find that on long RTT paths with high bandwidth, FAST TCP takes over 30 s to reach its bandwidth share [1] and BIC TCP creates packet loss and has trouble maintaining high throughput on paths with low B_{size}. In all experiments we have run Madison flows share bandwidth equally with other Madison flows and fairly with Reno flows, as it is designed to do.

5.3. Protocol throughput comparisons

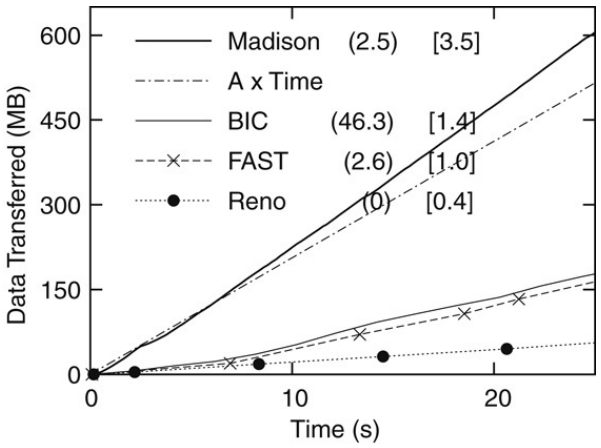
In this section we discuss results of back to back experiments over a variety of Internet paths. As explained in Section 5.1, we only retain results if the average load due to background Internet traffic is the same throughout the back to back transfers.

Fig. 8 provides representative results for Internet paths that present a challenge for transport protocol performance. For each protocol in each figure, the curve shows the cumulative data transferred vs time, the number in parentheses

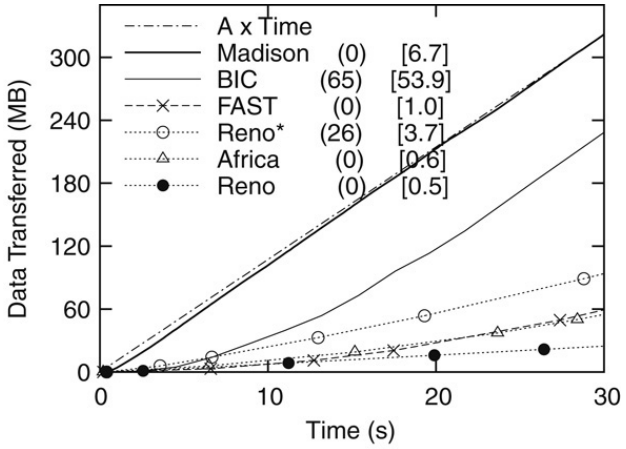
[1] This has been previously observed [11].



(a) Baseline path, $B_{\text{size}} = 158$ KB
$U_{xt} \approx 42\%$, $C = 100$ Mb/s
RTT = 278 ms, WI to North-Eastern China.



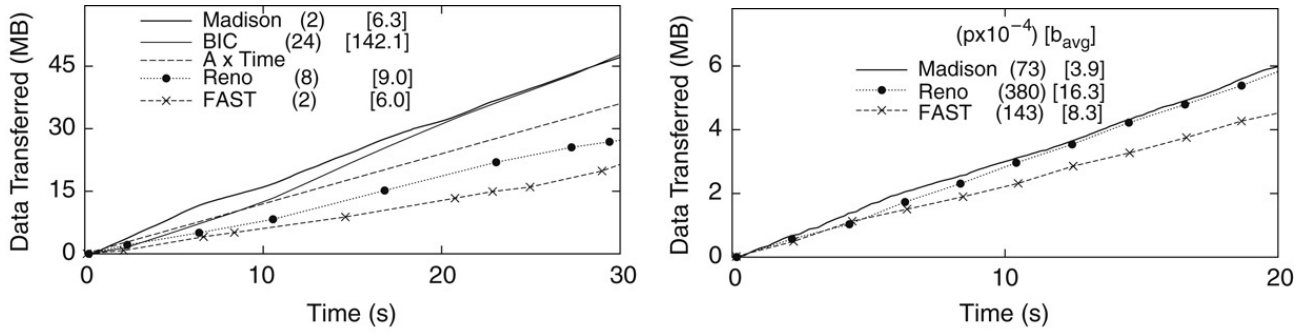
(b) Moderate RTT, $B_{\text{size}} = 172$ KB
$U_{xt} \approx 70\%$, $C = 500$ Mb/s
RTT = 78 ms, WI to San Diego.



(c) Negligible Cross-Traffic, $B_{\text{size}} = 2000$ KB
$U_{xt} < 10\%$, $C = 100$ Mb/s
RTT = 196 ms, WI to Taiwan.

Fig. 8. Protocol performance comparison: $C \geq 100$ Mb/s, RTT >70 ms.

in the legend provides the number of packets lost per 10 000, and the number in square brackets provides the average network backlog for the flow. The curve labeled A × Time shows the amount of data that would be transferred if the protocol throughput is equal to the unused bottleneck link capacity before and after the back-to-back protocol transfers. Note that in Fig. 8(a), Madison has throughput very close to A × Time, with loss rate 3.9×10^{-4} and average backlog of 2.7 packets. This figure is for a path with the challenging "baseline" characteristics and with small (158 KB) bottleneck buffer (B_{size}). Fig. 8(b) is for a path with similar characteristics but higher C (500 Mb/s) and more moderate RTT (78 ms). FAST TCP and BIC outperform Reno in both of those paths. On the other hand, BIC throughput is hindered by the small B_{size} and FAST TCP is only slowly converging to higher throughput (due to large RTT and non-negligible packet loss caused by the existing Internet cross-traffic). Hence, Madison outperforms FAST and BIC for such paths. Fig. 8(c) is for a path with similar characteristics to the one in Fig. 8(a) but with lower cross-traffic load U_{xt} and a large B_{size} (2000 KB). Lower U_{xt} improves the throughput of each protocol. The key change is the larger B_{size}, which improves BIC throughput and lowers the loss rate for the other protocols. In this case, Madison still outperforms BIC because the packet losses decrease throughput in the BIC flow. Note also that Africa has slightly better performance than Reno. We found that in most of our experiments, Africa only performed slightly better than Reno. Hence, we omit the Africa results from the rest of the graphs in this paper.



(a) $C = 24$ Mb/s $A/C = 0.4$, RTT $= 113$ ms, $B_{size} = 392$ KB WI to Ireland.

(b) $C = 10$ Mb/s, Reno $p = 3.8\%$ $A/C = 0.18$, RTT $= 72$ ms, $B_{size} = 160$ KB WI to CA.

Fig. 9. Protocol performance comparison: Lower C.

Fig. 9 provides results for paths with lower bottleneck link bandwidths and more moderate RTT than in Fig. 8. In this case, there are smaller differences in the protocol throughputs and loss rates. We note however, that FAST throughput is still somewhat hindered by the loss rate due to cross-traffic on the bottleneck. Also note that for these lower bottleneck link bandwidths, the slope of the curves changes during the 20–30 s data transfers, indicating that the aggregate load due to existing Internet cross-traffic is varying more with time than in the paths in Fig. 8. This is likely due to lower numbers of flows traversing these lower capacity links.

6. Conclusions

This paper has developed new methods for computing bottleneck link characteristics in a network path, for sharing bandwidth fairly with Reno flows, and for achieving rapid convergence to bandwidth share, all from simple path characteristics (i.e., average and minimum RTT, loss rate, and bottleneck link capacity). We have also developed a new protocol, TCP-Madison, that uses the new quantitative methods to implement fair sharing with Reno flows, equal bandwidth sharing with other Madison flows regardless of their RTT values, rapid convergence to bandwidth share, high bottleneck link utilization, and low network backlog. The key congestion avoidance mechanism in TCP-Madison is to use a low average network backlog when there is no packet loss, and to reduce the average network backlog of the flow when packet loss rate increases. The protocol also has five simple parameters, four of which have fixed values for all hosts, and the fifth is the minimum average time between RTT samples.

We implemented the Madison protocol, as well as FAST TCP, BIC-TCP, and Reno. Experiments over hundreds of PlanetLab and other live Internet paths show that the Madison protocol is simple to implement and achieves rapid convergence to bandwidth share without inducing packet loss, at flow start-up as well as in response to changes in cross-traffic congestion. The results show that TCP-Madison (a) shares bandwidth fairly with Reno flows, (b) greatly outperforms FAST TCP as well as Reno on congested as well as lightly loaded paths with moderate to high RTT, and (c) outperforms BIC on paths with relatively small bottleneck link buffers. For example, on a congested path from Wisconsin to San Diego, Madison achieves four times higher throughput than FAST TCP and over ten times higher throughput than Reno, while maintaining a total average network backlog of only 3.5 packets and a loss rate of only 2.5 per 10 000 packets sent. Madison outperforms FAST TCP due to its quick convergence to bandwidth share, rapid adaptation to changes in cross-traffic load, and tolerance of packet loss when transmitting at bandwidth share.

On-going research includes extensions to the TCP-Madison protocol for differentiated service bandwidth allocations, optimizing the protocol for wireless networks, and technology transfer to industry.

Acknowledgments

We thank Leana Golubchik, Christoph Lindemann and John C.S. Lui for granting us access to machines in their research laboratories, enabling us to increase the diversity of paths explored in our experimental protocol evaluation. This work was partially supported by the National Science Foundation under Grants 0117810 and 0435437.

References

[1] H. Balakrishnan, M. Stemm, S. Seshan, R.H. Katz, Analyzing stability in wide-area network performance, Measurement and Modeling of Computer Systems (1997).
[2] E. Borel, Sur l'emploi du théorème de bernoulli pour faciliter le calcul d'un infinité de coefficients, Application au probleme de l'attente à un guichet, Comptes Rendus Mathématique. Académie des Sciences. Paris 214 (1942) 452–456.
[3] L.S. Brakmo, S.W. O'Malley, L.L. Peterson, TCP vegas: New techniques for congestion detection and avoidance, in: SIGCOMM, 1994.
[4] J. Cao, W. Cleveland, D. Lin, D. Sun, Internet traffic tends toward poisson and independent as the load increases, in: C. Holmes, D. Denison, M. Hansen, B. Yu, B. Mallick (Eds.), Nonlinear Estimation and Classification, 2002.
[5] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, N. McKeown, Minimizing the duration of flows, Tech. Rep. TR04-HPNG-061604, Stanford University, June 2004.
[6] A. Edwards, S. Muir, Experiences implementing a high performance TCP in user-space, in: SIGCOMM, 1995.
[7] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, Part iii: Routers with very small buffers, Computer Communication Review 35 (3) (2005) 83–90.
[8] FAST TCP, Frequently Asked Questions. http://netlab.caltech.edu/FAST/, Jan. 2007.
[9] S. Floyd, High speed TCP for large congestion windows, RFC 3649 (Experimental), Dec. 2003.
[10] Y. Gu, D. Towsley, C.V. Hollot, H. Zhang, Congestion control for small buffer high speed networks, in: IEEE INFOCOM, 2007.
[11] C. Jin, D.X. Wei, S.H. Low, FAST TCP: Motivation, architecture, algorithms, performance, Tech. Rep. CSTR:2003:010, Caltech, December 2003.
[12] C. Jin, D.X. Wei, S.H. Low, FAST TCP: Motivation, architecture, algorithms, performance, in: IEEE INFOCOM, 2004.
[13] T. Karagiannis, M. Molle, M. Faloutsos, A. Broido, A nonstationary poisson view of internet traffic, in: IEEE INFOCOM, 2004.
[14] D. Katabi, M. Handley, C. Rohrs, Congestion control for high bandwidth-delay product networks, in: ACM SIGCOMM, 2002.
[15] T. Kelly, Scalable TCP: Improving performance in highspeed wide area networks. http://www-lce.eng.cam.ac.uk/ctk21/scalable/, February 2003.
[16] R. King, R. Baraniuk, R. Riedi, TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP, in: IEEE INFOCOM, 2005.
[17] V. Paxson, End-to-end internet packet dynamics, IEEE/ACM Transactions on Networking 7 (3) (1999) 277–292.
[18] P. Pradhan, S. Kandula, W. Xu, A. Shaikh, E. Nahum, Daytona: A user-level TCP stack. http://nms.lcs.mit.edu/kandula/data/daytona.pdf, 2002.
[19] C. Samios, M. Vernon, Modeling the throughput of tcp vegas, in: 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2003.
[20] C.A. Thekkath, T.D. Nguyen, E. Moy, E.D. Lazowska, Implementing network protocols at user level, IEEE/ACM Transactions on Networking 1 (5) (1993).
[21] L. Xu, K. Harfoush, I. Rhee, Binary increase congestion control for fast, long distance networks, in: IEEE INFOCOM, 2004.
[22] Y. Zhang, N. Duffield, V. Paxson, S. Shenker, On the constancy of internet path properties, in: ACM SIGCOMM Internet Measurement Workshop, 2001.

George Kola received the B.E. (Computer Science and Engineering) degree from College of Engineering, Guindy (Anna University), India in 2001 and M.S. (Computer Science) degree from University of Wisconsin-Madison in 2003. Currently, he is a Ph.D. candidate in computer science at the University of Wisconsin-Madison. His thesis advisor is Prof. Mary Vernon. His research interests include networking, operating systems, distributed systems, systems design, performance analysis and modeling.

Mary K. Vernon received a B.S. degree with Departmental Honors in chemistry and the M.S. and Ph.D. degrees in computer science from the University of California at Los Angeles.

In 1983 she joined the Computer Science Department at the University of Wisconsin-Madison, where she is currently Professor of Computer Science and Industrial Engineering.

Her research interests include performance analysis techniques for evaluating computer/communication system design tradeoffs, network protocols, computer system architectures, parallel applications, optimized CMP hardware/software co-design, storage systems, and flexible manufacturing systems.

Prof. Vernon is the co-inventor on two US patents for bus arbitration protocols and four US patents for scalable streaming content distribution. Her most recent paper awards include papers in Sigcomm 2001, Infocom 2004, and the 2005 Usenix Security Symposium. She has served on the editorial board of the IEEE Transactions on Parallel and Distributed Systems, the NSF CISE Advisory Committee, the NSF Blue Ribbon Panel for High Performance Computing, the Board of Directors of the Computing Research Association, the Executive Committee of the National Computational Science Alliance, and as Chair of the ACM SIGMETRICS.

She received the NSF Presidential Young Investigator Award (1985), the ACM Fellow award (1996), and the University of Wisconsin Kellett Mid-career Award (2006). She is a member of the ACM and the IFIP WG 7.3 on Information Processing System Modeling, Measurement and Evaluation.