Decomposing RTT-Unfairness in Transport Protocols

Eric Gavaletz and **Jasleen Kaur**
University of North Carolina at Chapel Hill
{gavaletz, jasleen}@cs.unc.edu

Abstract—**In this paper, we consider RTT-unfairness in most existing end-to-end congestion-control protocols, in which transfers with smaller RTTs are allocated a higher share of the bottleneck bandwidth. We consider the congestion-control mechanisms used by TCP NewReno and identify three aspects that introduce an RTT-based bias in different ways. Past attempts at alleviating RTT-unfairness mostly consider only one of these aspects. We use a first-principles approach for deriving a set of simple scaling factors that can be used to remove the RTT-bias. We conduct empirical evaluations to study how fair in practice would be a hypothetical protocol that employs all of these scaling factors. We discuss the practicality of implementing such a protocol.**

I. Introduction

A congestion-control protocol constantly probes the network for the highest rate at which data can be transferred without causing network congestion.[1] All existing end-to-end congestion-control protocols do this by fundamentally operating at an *RTT-long* timescale: (i) they probe for a candidate transfer rate by adopting it for an RTT-duration, and (ii) depending on the feedback received, they increase/decrease the probing rate adopted in the next RTT by some factor.[2] Existing protocols differ mainly in two aspects—the increase/decrease factors used as well as the feedback metric used for detecting congestion. To date, however, *all*[3] end-to-end congestion-control attempts have retained the *RTT-clocked* design framework.

The RTT-clocked design leads to the undesirable property of *RTT-unfairness*. Since all transfers sharing a bottleneck link operate at the scale of their respective RTTs, flows with shorter RTTs have a higher probing frequency—consequently, these are able to grow their sending rate at a faster pace and obtain an unfairly large share of the bottleneck bandwidth. Thus, even when Alice and Bob have purchased the same Internet service plan from their local ISP, if Alice is downloading from a server that is twice as far away as Bob, the throughput of her download may be less than 20% of that of Bob!

[1]This material is based upon work supported by a National Science Foundation CAREER Award CNS-0347814.

[2]The round-trip time (RTT) of a transfer is the total time it takes for a data segment to transfer from the sender to the receiver, and for the corresponding acknowledgment to make its way back.

[3]And we really mean *all* end-to-end protocols—this includes: (i) loss-based protocols ranging from NewReno variants [1], [2] to HighSpeed [3], Scalable [4], CUBIC [5], [6]; (ii) delay-based protocols such as Vegas [7], FAST [8], Illinois [9], Compound-TCP [10]; and (iii) rate-based protocols such as in [11], [12], [13] (which smooth out protocol behavior and operate at even slower timescales).

While some recent protocols attempt to reduce the degree of unfairness by adopting RTT-aware increase-factors [8], [10], [14], [15], [16], [17], empirical evaluations show that most of these simply reduce the degree of unfairness but do not yield truly fair throughput for finite transfers. In this paper, we attempt to understand why.

We carefully examine the congestion-control mechanisms in TCP NewReno in order to better understand the RTT-based bias in the throughput achieved by a transfer. We find that there are at least four different aspects of current protocols that introduce an RTT-bias. Unfortunately, existing proposals for achieving RTT-fairness address only one of these. Specifically, we find that the RTT-based throughput bias manifests itself differently in the Slow-start phase, the Congestion-avoidance phase, as well as in the setting of the Slow-start Threshold and initial congestion-window. Furthermore, the nature of the bias is different depending on whether the bottleneck link is heavily utilized or not. We conclude that in order to address RTT-unfairness for real-world transfers, protocol designers would need mechanisms that (i) introduce an appropriate set of RTT-based anti-bias scaling factors in the window-growth phases as well as parameter settings, as well as (ii) estimate the queuing delay at the bottleneck link—the latter is a challenging requirement and is a somewhat open problem even today.

In order to assess the efficacy of these scaling factors, we consider an ideal protocol that adopts these (we assume that an oracle informs the protocol of the precise state of the bottleneck queue). We run NS-2 simulations with such a protocol and find that incorporating all of these factors does indeed lead to a truly RTT-fair design.

In what follows, we examine the NewReno congestion-control mechanisms in Section II. In Section III, we present the design of an ideal RTT-fair protocol. In Section IV, we implement and evaluate an oracle-assisted prototype of our design. We end with a discussion of future work.

II. How Much Bias Does RTT Introduce?

TCP congestion-control is defined by a collection of several mechanisms, including Slow-start, Congestion-avoidance, Fast Retransmit/Recovery, and Retransmission Timeouts [1]. While each of these mechanisms gets invoked frequently and regularly in real-world TCP transfers [18], most existing attempts at alleviating RTT unfairness focus only on the Congestion-avoidance phase and steady-state behavior. As our simple analysis below shows, other mechanisms can introduce a significant RTT-based throughput bias as well. While this analysis is presented in the specific context of NewReno, the framework and qualitative conclusions apply to most other end-to-end congestion-control protocols.

A. *RTT-bias in Slow-start*

In Slow-start, a TCP sender increments its congestion-window (cwin) by one segment for every ACK received, effectively doubling its cwin every RTT. The window size, w_t, at time t can be approximated as:

$$w_t = w_0 * 2^{\frac{t}{r}} \tag{1}$$

where w_0 is the initial cwin used by the transfer. The throughput attained by a transfer at time t can be approximated as $T_t = \frac{w_t}{r}$, where r is the current RTT of the transfer.

Consider two transfers A and B that start simultaneously and have different path RTTs: r_a and r_b, respectively. Then, the ratio of their throughput, T_a and T_b respectively, at time t is given by:

$$\delta = \frac{w_a 2^{t/r_a}/r_a}{w_b 2^{t/r_b}/r_b} \tag{2}$$

Since the initial cwin is independent of path RTT in TCP, $w_a = w_b$. Hence:

$$\delta = \frac{r_b}{r_a} \cdot 2^{(\frac{t}{r_a} - \frac{t}{r_b})} \tag{3}$$

Note that when $r_a < r_b$, this ratio keeps growing with t. In Figure 1(a), we show how this ratio grows with time and the RTTs of two transfers—here, $r_a = 10ms$, while r_b is varied up to $500ms$. We find that due to the aggressiveness with which cwin is updated in Slow-start, the throughput of the two transfers can differ by several orders of magnitude within a fraction of a second.

Figures 2(a) and 2(b), respectively, illustrate the growth of T_t and w_t for three transfers with RTTs of 20 ms, 30 ms, and 40 ms, respectively. Slow-start governs the initial phases of the three transfers and results in significantly different throughput values before they exit the phase.

B. *RTT-bias in Congestion-avoidance*

In Congestion-avoidance, a TCP sender effectively increments its congestion-window by one segment in every RTT. The window size at time t can be approximated as:

$$w_t = w_0 + \frac{t}{r} \tag{4}$$

where w_0 is the window-size at $t = 0$ (in Congestion-avoidance, assuming that the transfer has crossed the Slow-start Threshold).

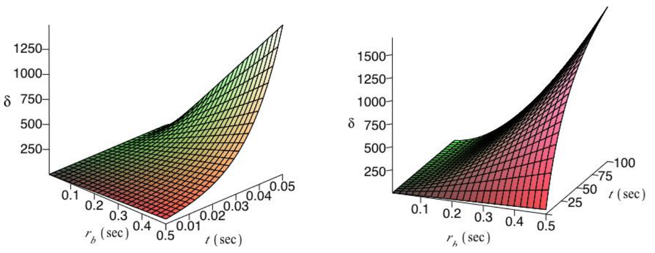
Consider the two transfers A and B. The ratio of their throughput at time t is given by:

$$\begin{aligned} \delta &= \frac{(w_a + \frac{t}{r_a})/r_a}{(w_b + \frac{t}{r_b})/r_b} \\ &= \frac{r_b^2}{r_a^2} \cdot \frac{w_a r_a + t}{w_b r_b + t} \end{aligned} \tag{5}$$

where w_a and w_b are their respective window sizes at $t = 0$ (in Congestion-avoidance).

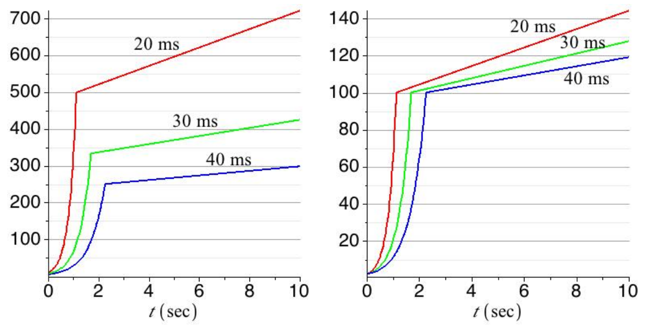
Note that when $r_a < r_b$, this ratio keeps growing with time.[4] Figure 1(b) plots the value of this ratio as a function of time and r_b, when $r_a = 10ms$. We find that even in the milder Congestion-avoidance phase, the throughput of the transfers can differ by several orders of magnitude within a few seconds.

Continuing our example of Figure 2, we find that the throughput of the three transfers continue to diverge in Congestion-avoidance. It is important to also note that since the value of the Slow-start Threshold (SST) is independent of path RTT in TCP, the transfers leave the Slow-start phase after having achieved different levels of throughput—a smaller RTT transfer benefits more from the rapid throughput growth in Slow-start before entering the slower Congestion-avoidance phase.



(a) Slow Start (b) Congestion Avoidance

Fig. 1. $\delta(r_b, t)$, $r_a = 0.01$



(a) T_t (pkts/sec) (b) w_t (pkts)

Fig. 2. Standard AIMD ($w_0 = 1$, and $SST = 100$)

C. *RTT-bias With Persistent Queuing*

All of the above assumes that RTTs remain stable; however, in the case where the bottleneck link has persistent queuing, it impacts the RTTs of A and B disproportionately. More importantly, when the bottleneck link carries long transfers over loss-based congestion-control protocols, it is often driven to a state when its queue keeps building up (till buffers overflow and prompt sources to slow down). When the queue keeps building up persistently, the RTTs of the two transfers grow disproportionately with time. Specifically, if at time t the queuing delay experienced in this queue is q, then the RTTs of the two transfers at time t would have grown to: $r_a + q$ and

[4]Note that when $r_a < r_b$, the right hand side of Equation 5 igrows with t, but eventually flattens out as it approaches r_b^2/r_a^2.

$r_b + q$, respectively. This also implies that:

$$\frac{w_a}{r_a} = \frac{w_b}{r_b} \nrightarrow \frac{w_a}{r_a + q} = \frac{w_b}{r_b + q} \tag{6}$$

Furthermore, since the queuing delay (and consequently, the RTT of a transfer) keeps growing with time, the window growth in Slow-start or Congestion-avoidance can no longer be tracked using the equations in (1) and (4). The nature of the bias introduced by the RTT differences becomes a more complex function that includes the queuing delay and the rate at which it grows—for brevity, we refrain from quantifying this bias here.

Our analysis in this section clearly illustrates that several aspects of TCP congestion-control contribute towards an RTT-based bias in throughput allocation—these include the window-growth in both Slow-start and Congestion-avoidance, as well as the use of a common initial window and a common Slow-start Threshold. Unfortunately, most attempts at alleviating RTT-unfairness focus only on the steady-state behavior of Congestion-avoidance [15], [16]. Clearly, in order to be truly RTT-fair, a congestion-control protocol would need to address each of these sources of bias. In the next section, we attempt to do so.

III. Designing An Ideal RTT-fair Protocol

In this section we derive a set of scalars that can be used to offset the bias in throughput allocation that is introduced by differences in path RTTs. As shown in Section II, the nature of the bias for two given transfers A and B depends on a set of several non-trivial functions of their path RTTs, r_a and r_b. Clearly, the offsets for these bias would also need to be a function of the path RTTs. Since senders are unaware of the state of other transfers sharing the bottleneck link, we instead rely on the notion of a common "reference" transfer, *REF*, which has a path RTT of r_{ref}, a Slow-start Threshold of SST_{ref}, and an initial window of $w_{0,ref}$. Simply put, our goal is to have each transfer achieve the same throughput as what would have been achieved by *REF* if it started at the same time and experienced the same network conditions—each transfer does so by appropriately scaling its window growth functions, as well as the Slow-start Threshold and initial window.

A. Scaling in Slow-start

In all of the derivations in this section, we will rely on an inductive approach—we will start by assuming that at some reference time, the throughput of a given transfer is the same as what would have been achieved by *REF* at that time. We will then derive the manner in which the window should be updated subsequently in order to continue maintaining equality with the throughput of *REF*.

Initialization: In order for the throughput of a transfer to match that of *REF* initially, we first select an initial window w_0 as:

$$w_0 = w_{0,ref} \cdot \frac{r}{r_{ref}} \tag{7}$$

Base-case: We assume that at a given time t, the throughput of the transfer matches that of *REF*. This would imply that:

$$\frac{w}{r} = \frac{w_{ref}}{r_{ref}} \tag{8}$$

Induction-step: After an r worth of time, we would like to update the window-size (to w') so that the updated throughput will continue to match that of *REF*. Note that due to the exponential nature of Slow-start, w_{ref} would increase in an exponential nature even within r time units. That is, we would like:

$$\begin{aligned} \frac{w'}{r} &= \frac{w'_{ref}}{r_{ref}} \\ &= \frac{w_{ref} \cdot 2^{(r/r_{ref})}}{r_{ref}} \\ &= \frac{\frac{w \cdot r_{ref}}{r} \cdot 2^{(r/r_{ref})}}{r_{ref}} \\ w' &= w \cdot 2^{(r/r_{ref})} = w \cdot 2^s \end{aligned} \tag{9}$$

where w' is the updated window-size of the transfer after an RTT worth of delay, and $s = \frac{r}{r_{ref}}$. Instead of Equation (1), the window-size can now be tracked over time with the function:

$$w_t = w_{0,ref} \cdot s \cdot 2^{\left(s \cdot \frac{t}{r}\right)} \tag{10}$$

Further, in order to ensure that the transfer leaves the Slow-start phase (and enters the Congestion-avoidance phase) with the same throughput as that of *REF*, we also scale the Slow-start Threshold as:

$$SST = SST_{ref} \cdot s \tag{11}$$

B. Scaling in Congestion-avoidance

The Congestion-avoidance phase starts with a window size equal to SST. From Equation (11), it follows that a transfer enters this phase with a throughput matching that of *REF*. We proceed with induction as follows:

Base-case: We assume that at a given time t, the throughput of the transfer matches that of *REF*. This would imply that: $\frac{w}{r} = \frac{w_{ref}}{r_{ref}}$.

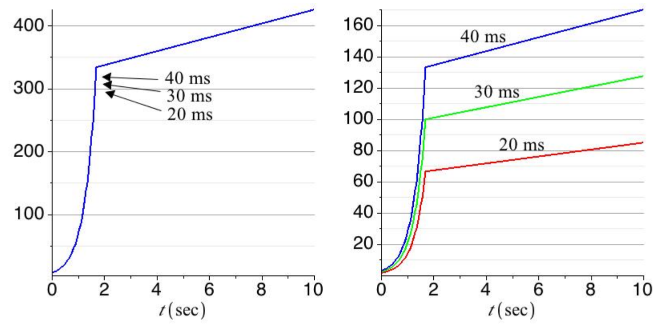
Induction-step: After an r worth of time, we would like to update the window-size (to w') so that the updated throughput will continue to match that of *REF*. That is, we would like:

$$\begin{aligned} \frac{w'}{r} &= \frac{w'_{ref}}{r_{ref}} \\ &= \frac{w_{ref} + \frac{r}{r_{ref}}}{r_{ref}} \\ &= \frac{\frac{w \cdot r_{ref}}{r} + \frac{r}{r_{ref}}}{r_{ref}} \\ w' &= w + \left(\frac{r}{r_{ref}}\right)^2 = w + s^2 \end{aligned} \tag{12}$$

where w' is the updated window-size of the transfer after an RTT worth of delay. Instead of Equation (4), the window-growth over time can now be tracked using:

$$w_t = s \cdot SST_{ref} + s^2 \cdot \frac{t}{r} \tag{13}$$

We make use of Equations 10, 11, and 13 to generate Figure 3—comparison to Figure 2 shows that the scaling factors do help model RTT-independent throughput as desired.



(a) T_t (pkts/sec) (b) w_t (pkts)

Fig. 3. Scaled AIMD ($r_{ref} = 30ms$)

C. *Scaling in the Presence of Packet Losses*

Most literature on TCP fairness analysis works under the assumption that when the bottleneck buffers overflow, *all* transfers using that link experience packet losses [19]. Indeed, if packet loss events are not distributed across transfers in an equitable manner, TCP and its AIMD variants would not result in a fair throughput allocation even among similar RTT transfers! Correspondingly, in this paper we are concerned with the question: when packet losses do occur in all transfers nearly simultaneously, how can one achieve fairness across transfers with different RTTs?

Interestingly, with the above assumption, there is no need to add any additional scaling factors. This is because if the transfers detect the packet losses using Retransmission Timeouts (RTOs), then they will all scale back to their *initial window*—which are already scaled appropriately according to Equation (7). If instead the losses are detected using Fast Retransmit/Recovery (FR/R), the windows are scaled to half of their value before the loss. Since the pre-loss windows are scaled appropriately, applying the constant factor of 0.5 will preserve the scale with respect to *REF*.

D. *Scaling in the Presence of Persistent Queuing*

In case the bottleneck queue grows persistently, the corresponding queuing delay is encountered by all transfers that traverse a bottleneck link, and increases their RTTs "equally". This would also be true for the hypothetical *REF* transfer. We next derive scaling factors that would help maintain RTT fairness even at times when the bottleneck queue is persistently growing (which happens often with loss-based congestion-control protocols). In this section, we use the notation r to denote the round-trip time experienced by a transfer when the bottleneck queue size is zero.

1) Slow-start: We assume that the initial sampled RTT experiences zero-queuing at the bottleneck and simply use the scalar s to set w_0 as before. Then we use induction as follows:
Base-case: We assume that at a given time t, the throughput of the transfer matches that of *REF*. We also assume that each of these transfers experiences the same queuing delay, q, at time t. This would imply that:

$$\frac{w}{r+q} = \frac{w_{ref}}{r_{ref}+q} \tag{14}$$

Induction-step: After an RTT worth of time, the transfer would have received a full cwin of ACKs. Let the new RTT be equal to $r+q'$, where q' is the most recently sampled queuing delay at the bottleneck link. At this point in time, we would like to update our cwin to a value w' such that the throughput of the transfer matches that of *REF*. Thus, we want:

$$\begin{aligned}
\frac{w'}{r+q'} &= \frac{w'_{ref}}{r_{ref}+q'} \\
&= \frac{w_{ref}\cdot 2^{(r+q'/r_{ref}+q')}}{r_{ref}+q'} \\
&= \frac{\frac{w\cdot(r_{ref}+q)}{r+q}\cdot 2^{(r+q'/r_{ref}+q')}}{r_{ref}+q'} \\
w' &= w\cdot\frac{(r+q')(r_{ref}+q)}{(r+q)(r_{ref}+q')}\cdot 2^{\left(\frac{r+q'}{r_{ref}+q'}\right)} \qquad (15)
\end{aligned}$$

The Slow-start Threshold is simply scaled as: $SST = SST_{ref}\cdot\frac{r+q}{r_{ref}+q}$, where q is the most recently sampled queuing delay.

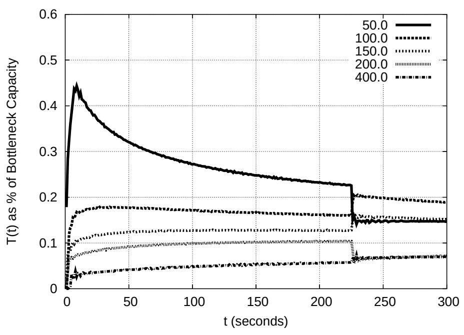
2) Congestion-avoidance: Since the transfer enters Congestion-avoidance with a throughput matching that of *REF*, we start with the inductive derivation as follows:
Base-case: We assume that at a given time t, the throughput of the transfer matches that of *REF*. We also assume that each of these transfers experiences the same queuing delay, q, at time t. Thus, we have: $\frac{w}{r+q} = \frac{w_{ref}}{r_{ref}+q}$.
Induction-step: And after an RTT worth of time, the transfer would have received a full cwin of ACKs. Let the new RTT be equal to $r+q'$, where q' is the most recently sampled queuing delay at the bottleneck link. At this point in time, we would like to update our cwin to a value w' such that the throughput of the transfer matches that of *REF*. Note that after an $r+q't$ worth of time, *REF* would have received $(r+q')/(r_{ref}+q')$ ACKs. Thus, we want to compute a w' such that:

$$\begin{aligned}
\frac{w'}{r+q'} &= \frac{w'_{ref}}{r_{ref}+q'} \\
&= \frac{w_{ref}+\frac{r+q'}{r_{ref}+q'}}{r_{ref}+q'} \\
&= \frac{\frac{w\cdot(r_{ref}+q)}{r+q}+\frac{r+q'}{r_{ref}+q'}}{r_{ref}+q'} \\
w' &= w\cdot\frac{(r+q')(r_{ref}+q)}{(r+q)(r_{ref}+q')}+\left(\frac{r+q'}{r_{ref}+q'}\right)^2 \qquad (16)
\end{aligned}$$

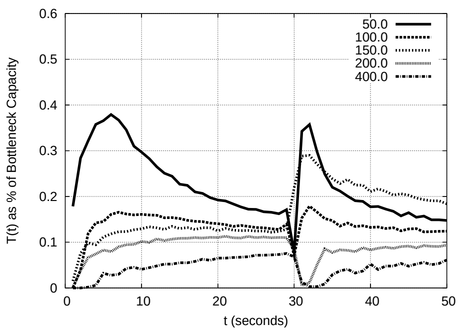
Note that a protocol that adopts the above scaling factors must be able to reliably estimate the queuing delays q and q'.

IV. Prototyping

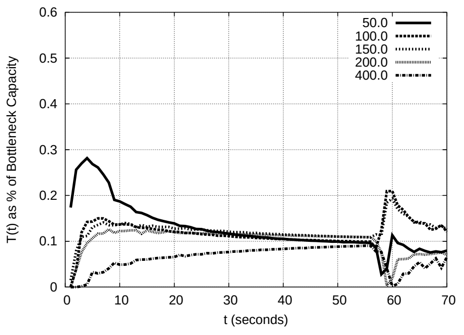
One of the biggest challenges to prototyping the ideal RTT-fair protocol developed in Section III, is estimating the degree of persistent queuing at the bottleneck link. Indeed, it is fair to say that detecting bottleneck queuing using only end-to-end metrics is still an open problem despite having been considered by several end-to-end congestion-control protocols [7], [8], [10]. Addressing this problem is beyond the scope of this paper.



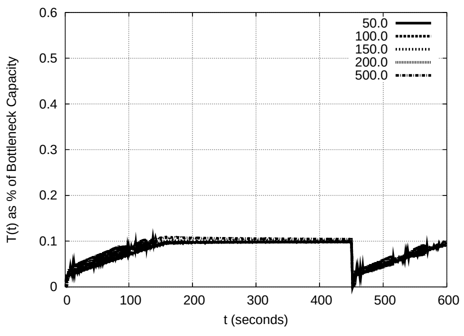
(a) TCP NewReno



(b) SRF TCP

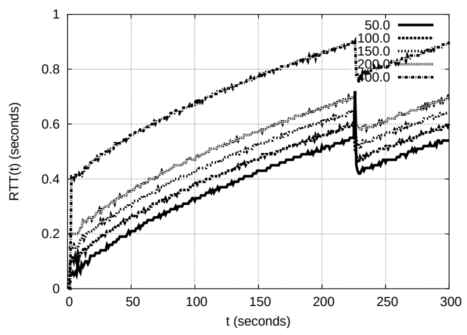


(c) TCP Libra

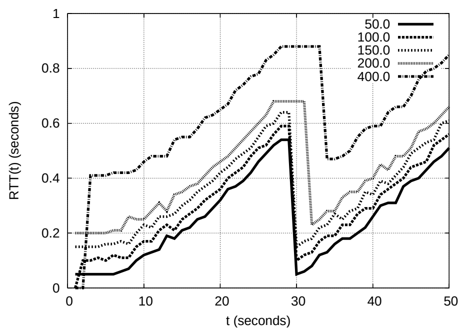


(d) TCP Tarheel

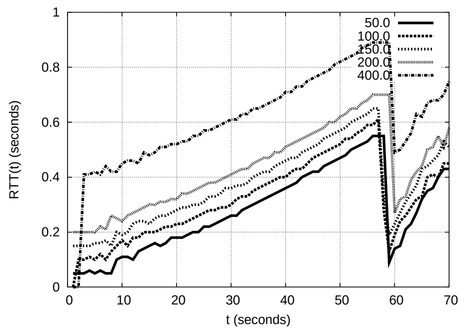
Fig. 4. $T(t)$ as % of Bottleneck Capacity in NS-2



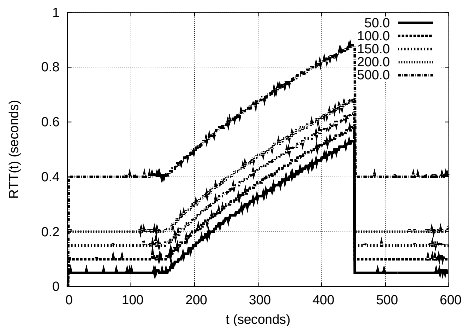
(a) TCP NewReno



(b) SRF TCP



(c) TCP Libra



(d) TCP Tarheel

Fig. 5. $RTT(t)$ (seconds) in NS-2

Our analysis from Section III suggests that *if* a congestion-control protocol could reliably estimate the bottleneck queuing delays, then the derived scalars should help it in achieving RTT-fairness. In this section, we evaluate the validity of this claim by running NS-2 simulations [20]. In order to deal with the issue of reliably estimating persistent queuing, we assume the existence of an *oracle* that informs the protocol of the exact amount of queuing delay it experiences at any given time t.[5] Equations 15, 11, 12, 15, and 16 then can be used to implement an ideal RTT-fair prototype.

We run the resultant prototype, referred to as TCP Tarheel, on a simple dumbbell topology in NS-2 in which 10 senders share a 100 Mbps bottleneck link. All non-bottleneck links have a 1 Gbps capacity, and the values of RTT_{min} for the different senders varies from $50ms$ to $500ms$. We also use this topology for evaluating the fairness yielded by NewReno, SRF [16], and Libra [15]. We ensure that *all* transfers experience packet loss nearly simultaneously, by forcing packet drops in each of the transfers when the bottleneck buffers approach full-occupancy.

Figures 4 and 5 plot as a function of time, respectively, the throughput and the RTTs of the transfers (to avoid clutter, in each plot we include curves for only a selected few of the 10 transfers). Also, for a fair comparison with other protocols, we crop each plot after the first loss event occurs in a given experiment. We find that:

- As expected, NewReno results in significantly different throughput allocations to transfers with different RTTs.
- SRF and Libra (which is fairer of the two) do reduce this degree of unfairness, but still result in throughput allocations that can differ by up to a factor of 4. As mentioned before, these protocols correct for the RTT-bias introduced by only Congestion-avoidance. Figure 4 shows that because of this, these protocols retain a significant penalty for large-RTT transfers in the initial Slow-start phase.
- The oracle-assisted Tarheel is indeed able to achieve RTT-fairness by relying on our analysis of Section III. This fairness is achieved even in the presence of persistent queuing (which starts at around 180 seconds, as illustrated in Figure 5(d)).

V. Concluding Remarks

There are several issues that need to be addressed before an ideal protocol like Tarheel is realizable. First and foremost is the issue of the robust estimation of bottleneck queuing delay. Second, the analysis conducted here considers fairness between transfers that start simultaneously—it is important to define and achieve fairness even with a representative traffic arrival pattern. Finally, we have not considered the impact of the Fast Retransmit/Recovery and Retransmission Timeout mechanisms on RTT-fairness. Each of the above is on our agenda for future work.

[5]In our simple NS-2 simulations on a dumbbell topology, queuing occurs only at the bottleneck link. The bottleneck queuing delay is then indeed given precisely by: $q = RTT - RTT_{min}$, where RTT_{min} is the round-trip time experienced by the transfer initially (in the absence of queuing).

References

[1] W. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison Wesley, 1994.

[2] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no. 3, July 1996.

[3] S. Floyd, "Highspeed TCP and quick-start for fast long-distance networks," *Plenary Talk at the First International Workshop on Protocols for Fast Long-distance Networks*, February 2003.

[4] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," in *First International Workshop on Protocols for Fast Long-distance Networks*, February 2003.

[5] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[6] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast, long distance networks," in *Proceedings of IEEE INFOCOM*, March 2004.

[7] L. Brakmo, S. O'Malley, and L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM*, August 1994.

[8] D. Wei, C. Jin, S. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1246–1259, 2006.

[9] S. Liu, T. Başar, and R. Srikant, "Tcp-illinois: A loss- and delay-based congestion control algorithm for high-speed networks," *Perform. Eval.*, vol. 65, no. 6-7, pp. 417–440, 2008.

[10] K. Tan and J. Song, "A compound tcp approach for high-speed and long distance networks," in *In Proceedings of IEEE Infocom*. Microsoft Press, 2006.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proceedings of ACM SIGCOMM*, August 2000.

[12] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[13] S. Kunniyur and R. Srikant, "End–to–end congestion control schemes: utility functions, random losses and ecn marks," in *Proceedings of IEEE INFOCOM*, March 2000.

[14] Y. Gu, "UDT: A high performance data transport protocol," Ph.D. dissertation, Chicago, IL, USA, 2005, chairperson-Robert L. Grossman.

[15] G. Marfia, C. Palazzi, G. Pau, M. Gerla, M. Sanadidi, and M. Roccetti, "Tcp Libra: exploring rtt-fairness for tcp," *UCLA Computer Science Department Technical Report# UCLA-CSD TR-050037*.

[16] M. Fukuhara, F. Hirose, T. Hatano, H. Shigeno, and K. Okada, "SRF TCP: A TCP-friendly and fair congestion control method for high-speed networks," *Lecture notes in computer science*, vol. 3544, p. 169, 2005.

[17] R. King, R. Baraniuk, and R. Riedi, "TCP-Africa: An adaptive and fair rapid increase rule for scalable TCP," in *IEEE INFOCOM*, vol. 3. Citeseer, 2005, p. 1838.

[18] S. Rewaskar, J. Kaur, and F. Smith, "A performance study of loss detection/recovery in real-world TCP implementations," in *ICNP '07: Proceedings of the 15th IEEE International Conference on Network Protocols*. Beijing, China: IEEE Computer Society, 2007.

[19] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, 1989.

[20] "Network simulator-2 ns2 (http://www.isi.edu/nsnam/ns/)."