How to Improve your Network Performance by Asking your Provider for Worse Service

Radhika Mittal
UC Berkeley
radhika@eecs.berkeley.edu

Justine Sherry
UC Berkeley
justine@eecs.berkeley.edu

Sylvia Ratnasamy
UC Berkeley
sylvia@eecs.berkeley.edu

Scott Shenker
UC Berkeley and ICSI
shenker@icsi.berkeley.edu

ABSTRACT

TCP's congestion control is deliberately "cautious", avoiding overloads by starting with a small initial window and then iteratively ramping up. As a result, it often takes flows several round-trip times to fully utilize the available bandwidth. In this paper we propose using several levels of lower priority service and a modified TCP behavior to achieve significantly improved flow completion times while preserving fairness.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Design, Experimentation, Performance

1. INTRODUCTION

We begin this paper by noting two facts about networks. First, modern ISPs run their networks at relatively low utilization [8, 10, 12]. This is not because ISPs are incapable of achieving higher utilization, but because their network must be prepared for link failures which could, at any time, reduce their available capacity by a significant fraction. Thus, most ISP networks are engineered with substantial headroom, so that ISPs can continue to deliver high-quality service even after failures.

Second, TCP congestion control is designed to be *cautious*, starting from a small window size and then increasing every round-trip time until the flow starts experiencing packet drops. The need for fairness requires that all flows follow the same congestion-control behavior, rather than letting some be cautious and others aggressive. Caution, rather than aggression, is the better choice for a uniform behavior because it can more easily cope with a heavily overloaded network; if every flow started out aggressively, the network could easily reach a congestion-collapsed state with a persistently high packet-drop rate.

These decisions – underloaded networks and cautious congestion control – were arrived at independently, but interact counter-productively. When the network is underloaded, flows will rarely hit congestion at lower speeds. However, the caution of today's congestion control algorithms requires that flows spend significant time ramping up rather than more aggressively assuming that more bandwidth is available. In recent years there have been calls to increase TCP's initial window size to alleviate this problem but, as we shall see later in the paper, this approach brings only limited benefits.

In this paper we propose a new approach called *recursively cautious congestion control* (RC3) that retains the advantages of caution while enabling it to efficiently utilize available bandwidth. The idea builds on a perverse notion of quality-of-service, called WQoS, in which we assume ISPs are willing to offer *worse* service if certain ToS bits are set in the packet header (the mechanisms for doing so – priority queues, are present in almost all currently deployed routers). While traditional calls for QoS – in which better service is available at a higher price – have foundered on worries about equity (should good Internet service only be available to those who can pay the price?), pricing mechanisms (how do you extract payments for the better service?), and peering (how do peering arrangements cope with these higher-priced classes of service?), in our proposal we are only asking ISPs to make several worse classes of service available that would be treated as regular traffic for the purposes of charging and peering. Thus, we see fewer institutional barriers to deploying WQoS. Upgrading an operational network is a significant undertaking, and we do not make this proposal lightly, but our point is that many of the fundamental sources of resistance to traditional QoS do not apply to WQoS.

The RC3 approach is quite simple. RC3 runs, at the highest priority, the same basic congestion control algorithm as normal TCP. However, it also runs congestion control algorithms at each of the k worse levels of service; each of these

Hotnets '13, November 21–22, 2013, College Park, MD, USA.

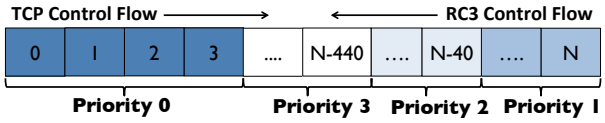


Figure 1: Packet priority assignments.

levels sends only a fixed number of packets, with exponentially larger numbers at lower priority levels. As a result, all RC3 flows compete fairly at every priority level, and the fact that the highest priority level uses the traditional TCP algorithms ensures that RC3 does not increase the chances of congestion collapse. Moreover, RC3 can immediately "fill the pipe" with packets (assuming there are enough priority levels), so it can leverage the bandwidth available in underutilized networks. In simulations that we describe later in the paper, we find that the flow completion times [6] improve by 30-80% in most settings we investigated. We also compare our scheme against various other approaches, such as increasing the initial window size, giving some flows higher priority, and explicit rate negotiation.

The rest of the paper proceeds as follows. In §2, we describe the RC3 design in detail. In §3, we develop a mathematical model to predict the performance gains due to RC3; we validate these results via simulation and compare RC3 to other approaches in §4. In §5, we discuss RC3's deployment and future, and in §6 we present related work.

2. DESIGN

We now discuss RC3's design in detail.

2.1 RC3 in Implementation

RC3 runs two parallel control loops: one transmitting at normal priority and obeying the cautious transmission rate of traditional TCP, and a second "recursive low priority" (RLP) control loop keeping the link saturated with low priority packets.

In the primary control loop, TCP proceeds as normal, sending packets in order from index 0 in the byte stream, starting with slow-start and then progressing to normal congestion-avoidance behavior after the first packet loss. The packets sent by this default TCP are transmitted at 'normal' priority – priority 0 (with lower priorities denoted by higher numbers).

In the RLP control loop, the sender transmits additional traffic from the same buffer as TCP to the NIC[1]. To minimize the overlap between the data sent by the two contol loops, the RLP sender starts from the very *last* byte in the buffer rather than the first, and works its way towards the beginning of the buffer, as illustrated in Figure 1. RLP packets are sent at low priorities (priority 1 or greater): the first 40 packets (from right) are sent at priority 1; the next 400 are sent at priority 2; the next 4000 at priority 3, and so on. [2] The

[1]As end-hosts support priority queueing discipline, this traffic will never pre-empt the primary TCP traffic.

[2]RC3 requires the exponentially spaced priority levels to accomodate large flows within feasible number of priority bits.

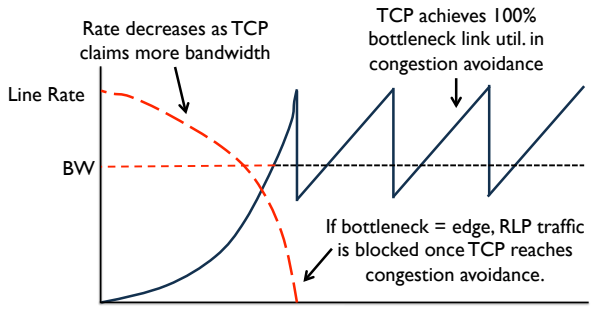


Figure 2: Congestion window and throughput with RC3.

RLP traffic can only be transmitted when the TCP loop is not transmitting, so its transmission rate is the NIC capacity minus the normal TCP transmission rate.

RC3 enables TCP selective ACK (SACK) to keep track of which of low priority (and normal priority) packets have been accepted at the receiver. When ACKs are received for low priority packets, no new traffic is sent and no windows are adjusted. The RLP control loop transmits each low priority packet once and once only; there are no retransmissions. The RLP loop starts sending packets to the NIC as soon as the TCP send buffer is populated with new packets, terminating when its 'last byte sent' crosses with the TCP loop's 'last byte sent'. Performance gains from RC3 are seen only during the slow-start phase; for long flows where TCP enters congestion avoidance, TCP will keep the network maximally utilized with priority 0 traffic. If the bottleneck link is the edge link, high priority packets will pre-empt any packets sourced by the RLP directly at the end host NIC; otherwise the low priority packets will be dropped elsewhere in the network.

Figure 2 illustrates how the two loops interact: as the TCP sender ramps up, the RLP traffic has less and less 'free' bandwidth to take advantage of, until it eventually is fully blocked by the TCP traffic. Since the RLP loop does not perform retransmissions, it can leave behind 'holes' of packets which have been transmitted (at low priority) but never ACKed. Because RC3 enables SACK, the sender knows exactly which segments are missing and the primary control loop retransmits only those segments.[3] Once the TCP 'last byte sent' crosses into traffic that has already been transmitted by the RLP loop, it uses this information to retransmit the missing segments and ensure that all packets have been received. We walk through transmission of a flow with such a 'hole' in the following subsection.

2.2 Example

We now walk through a toy example of a flow with 66 packets transmitted over a link with an edge-limited bandwidth×delay product of 50 packets. Figure 3 illustrates our example.

In the first RTT, TCP sends the first 4 packets at priority 0

[3]Enabling SACK allows fast recovery for dropped low priorty packets. However, RC3 still provides significant performance gains when SACK is disabled, despite some redundant retransmissions.

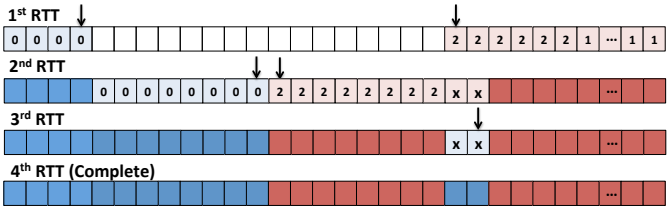


Figure 3: Example RC3 transmission from §2.2.

(from left); after these high priority packets are transmitted, the RLP loop sends the remaining 62 packets to the NIC – 40 packets at priority 1 and 22 packets at priority 2 (from right), of which 46 packets are transmitted by the NIC (filling up the entire BW×RTT product of 50 packets per RTT).

The 21st and 22nd packets from the left (marked as Xs), sent out at priority 2, are dropped. Thus, in the second RTT, ACKs are received for all packets transmitted at priority 0 and for all but packets 21 and 22 sent at lower priorities. The TCP control loop doubles its window and transmits an additional 8 packets; the RLP sender ignores the lost packets and the remaining packets are transmitted by the NIC at priority 2.

In the third RTT, the sender receives ACKs for all packets transmitted in the second RTT and TCP continues to expand its window to 16 under slow start. At this point, the TCP loop sees that all packets except 21st and 22nd have been acked. It, therefore, transmits only these two packets.

Finally, in the fourth RTT the sender receives ACKs for the 21st and 22nd packets as well. As all data acknowledgements have now been received by the sender, the connection completes.

3. PERFORMANCE MODEL

Having described RC3 in §2, we now model our expected improvements in Flow Completion Time (FCT) for a TCP flow using RC3 as compared to a basic TCP implementation. We quantify gains as ((FCT with TCP) - (FCT with RC3)) / (FCT with TCP) – *i.e.* the percentage reduction in FCT. Our model is very loose and ignores issues of queuing, packet drops, or the interaction between flows. Nonetheless, this model does help clarify some of the basic issues and, in the following section (§4), we validate these expected gains via simulation.

Basic Model: Let BW be the capacity of the bottleneck link a flow traverses, and u be the utilization level of that link. We define A, the available capacity remaining in the bottleneck link as $A = (1 - u) \times BW$. Since RC3 utilizes *all* of the available capacity, a simplified expectation for FCTs under RC3 is $RTT + \frac{N}{A}$, where RTT is the round trip time and N is the flow size.

TCP does not utilize all available capacity during its slow start phase; it is only once TCP reaches congestion avoidance that TCP maintains 100% utilization of the available capacity. The slow start phase, during which TCP leaves the

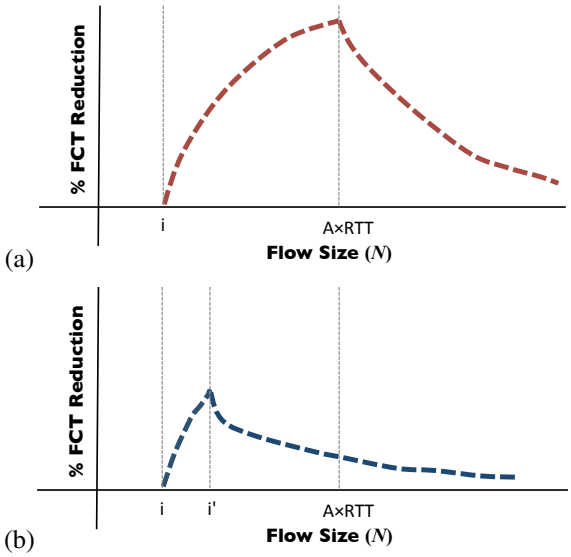


Figure 4: Performance gains as predicted by a simple model for (a) RC3 and (b) an increased initial congestion window.

link partially idle, lasts $log(min(N, A \times RTT)/i)$ RTTs, with i being the initial congestion window of TCP. This is the interval during which RC3 can benefit TCP.

In Figure 4(a), we show our expected gains according to our model. Recall that i denotes the initial congestion window under TCP. For flow sizes $N < i$, RC3 provides no gains over a baseline TCP implementation, as in both scenarios the flow would complete in $RTT + \frac{N}{A}$. For flow sizes $i < N < A \times RTT$, the flow completes in 1 RTT with RC3, and multiple round trip times ($log(N/i)$) with basic TCP in slow start. Consequently, the reduction in FCT increases with N over this interval.

Once flow sizes reach $N > A \times RTT$, basic TCP reaches a state where it can ensure 100% link utilization. After this point, the improvements from RC3 become a smaller fraction of overall FCT with increasingly large flows; this reduction roughly follows $\frac{log(A \times RTT/i) \times RTT \times A}{N}$ (ignoring a few constants in the denominator).

Parameter Sensitivity: The above model illustrates that improvements in FCTs due to RC3 are dependent primarily on three parameters: the flow size (N), the effective bandwidth×delay product ($A \times RTT$), and the choice of the initial congestion window (i). Peak improvements are observed when N is close to the $A \times RTT$, because under these conditions the flow completes in 1 RTT with RC3 and spends its entire life time in slow start without RC3. When the delay-bandwidth product increases, both the optimal flow size (for performance improvement) increases, and the maximum improvement increases.

Adjusting *i*: There are several proposals [4, 7] to adjust the default initial congestion window in TCP to 10 or even more packets. Assume we adjusted a basic TCP implementation to use a new value, some i' as its initial

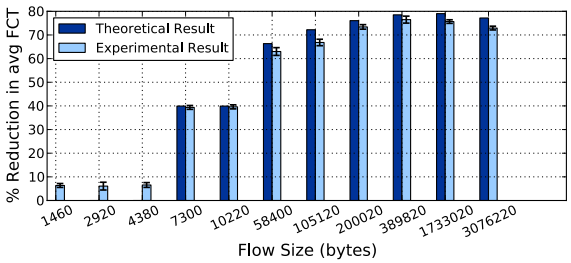


Figure 5: Reduction in FCT as predicted by model vs simulations.

congestion window. Figure 4(b) illustrates the gains from such an i' (compare to RC3 in Figure 4(a)). When i' increases, the amount of time spent in slow start decreases with $log(min(N, A \times RTT)/i') \times RTT$. Flows of up to i' packets complete in a single RTT, but unless $i' = A \times RTT$ (hundreds of packets for today's WAN connections), adjusting the initial congestion window will always underperform when compared to RC3. However, there is good reason not to adjust i' to $A \times RTT$: without the use of low priorities, as in RC3, sending a large amount of traffic without cautious probing can lead to an increase in congestion and overall *worse* performance. Our model does not capture the impact of queueing and drops, however, in §4.2 we show via simulation how increasing the initial congestion window to 10 and 50 packets penalizes small flows in the network.

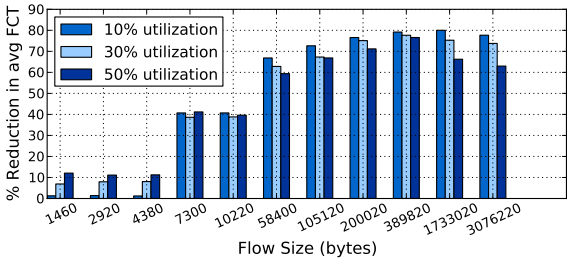
4. EXPERIMENTAL RESULTS

To validate the expected gains according to our theoretical analysis (§3), we now evaluate our proposal in simulation using the NS3 network simulator [1]. Our simulation topology models the Internet-2 network consisting of ten routers, each attached to ten end hosts, with 1Gbps bottleneck bandwidth and 40ms average RTT. Our base simulations are run with 30% average link utilization [8, 10, 12].

The traffic matrix is generated as follows: each end host generates flows with Poisson arrivals. Flow sizes are drawn from an empirical traffic distribution [3]; the Poisson arrival parameter is used to adjust the total load. Each flow is randomly sent to one of the other end hosts. Core link capacities are set to ensure the required average link utilization.

For most experiments we present RC3's performance relative to a baseline TCP implementation. Our baseline TCP implementation is TCP New Reno [13] with SACK enabled [11, 5] and an initial congestion window of 4 [4]; maximum segment size is set to 1460 bytes while slow start threshold and advertised received window are set to infinity. Our RC3 implementation uses the same TCP code as above, with low priority traffic generated as described in §2. All senders transmit using RC3 unless otherwise noted.

For each experiment, we display a bar graph of the average reduction in FCT for each flow size in our distribution. We also show tables with the reduction in FCT averaged over all flows; we display both an unweighted average



		Average Over Flows	Average Over Bytes
10% Load	Regular FCT (s)	0.125	0.423
	RC3 FCT (s)	0.068	0.091
	% Reduction	**45.57**	**78.36**
30% Load	Regular FCT (s)	0.135	0.443
	RC3 FCT (s)	0.076	0.114
	% Reduction	**43.54**	**74.35**
50% Load	Regular FCT (s)	0.15	0.498
	RC3 FCT (s)	0.088	0.176
	% Reduction	**41.44**	**64.88**

Figure 6: Reduction in FCT with load variation.

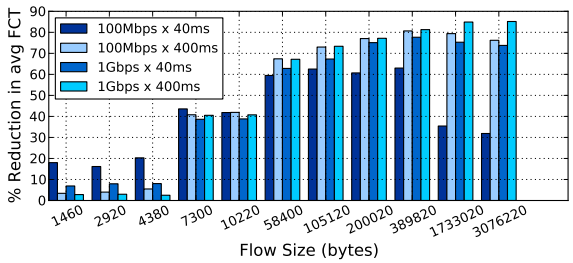
(average over flows) and an average weighted by flow size (average over bytes). Our experiments fall into one of two categories. In §4.1, we evaluate the performance of RC3 as compared to TCP New Reno under varying parameters for bandwidth×delay product, link load, and loss rate. In §4.2, we compare RC3 to other proposals to improve performance: increasing the initial congestion window of TCP New Reno [7], using high priorities for 'important' flows (QoS), and using RCP [6].

4.1 Baseline Performance & Theoretical Model

We now test RC3's performance in simulation under varying parameters for link load, bandwidth-delay product, and loss rate.

Validating the Model: Figure 5 compares the gains predicted by our model (§3) with gains observed in simulation (averaged over 10 runs). The data displayed is for a 1Gbps bottleneck capacity, 40ms average RTT, and 30% load and is representative of other parameters we simulated. For large flows, the simulated gains are slightly lower than predicted; this is the result of queueing delay which is not included in our model. For small flows – four packets or fewer – we actually see actually see *better* results than predicted by the model. This is due to large flows completing sooner than in the base TCP scenario, leaving the high priority network queues more frequently vacant and thus decreasing average queueing delay for short flows. Despite these variations, the simulated and modeled results track each other quite closely.

Link Load: Figure 6 shows FCT performance gains comparing RC3 to the base TCP under uniform link load of 10%, 30%, or 50%. The bandwidth-delay product is fixed at 5MB across all experiments. As expected, performance improvements decrease for higher average link utilization. For large



		Average Over Flows	Average Over Bytes
100Mbps Bottleneck 40ms avg. RTT	Regular FCT (s)	0.167	0.691
	RC3 FCT (s)	0.11	0.442
	% Reduction	**33.98**	**36.05**
100Mbps Bottleneck 400ms avg. RTT	Regular FCT (s)	0.948	3.501
	RC3 FCT (s)	0.567	0.783
	% Reduction	**40.29**	**77.62**
1Gbps Bottleneck 40ms avg. RTT	Regular FCT (s)	0.135	0.443
	RC3 FCT (s)	0.076	0.114
	% Reduction	**43.54**	**74.35**
1Gbps Bottleneck 400ms avg. RTT	Regular FCT (s)	0.971	3.59
	RC3 FCT (s)	0.558	0.569
	% Reduction	**42.45**	**84.17**

Figure 7: Reduction in average FCT with variation in bandwidth×delay product.

flows, this follows from the fact that the available capacity $A = (1 - u) \times BW$, reduces with increase in utilization u. Thus, there is less spare capacity to be taken advantage of in scenarios with higher link load. However, for smaller flows, we actually see the opposite trend. This is once again due to reduced high priority congestion, as large flows complete sooner and the packets they do transmit are mostly transmitted at lower priorities than the packets from the smaller flows.

Bandwidth×Delay: Figure 7 shows the FCT reduction due to RC3 at varying delay-bandwidth products. In this experiment we adjusted RTTs and bandwidth capacities to achieve a RTT×BW product of 500KB (100Mbps×40ms), 5MB (1Gbps×40ms and 100Mbps×400ms) and 50MB (1Gbps×400ms). As discussed in §3, the performance improvement increases with increasing RTT×BW, as the peak of the hill shifts towards right. Note that since RTT×BW products are expected to increase with time – link capacities increasing over time while the delay is essentially fixed due to the speed of light – RC3's performance gains as compared to baseline TCP will likely improve with time.

Loss Rate: All of our experiments simulate scenarios where the sole source of loss is congestion from our simulated traffic. To test RC3's sensitivity to background loss, due to wireless or other factors, we repeated our simulations at 30% load with 1Gbps edge links, with random drops introduced for 0.01% or 0.1% of packets. With 0.01% loss, the average FCT using the baseline TCP increases from 0.135 seconds to over 0.16 seconds; with 0.1% loss, the average FCT further increases to 0.28 seconds. In contrast, using RC3 the average FCT with 0.01% loss is only 0.08 seconds (50% reduction), and with 0.1% loss, the average FCT is just over 0.10 seconds (63% reduction). RC3 provides even stronger gains in such high loss scenarios because each packet essentially has two chances at transmission. Further, since the RLP loop ignores ACKs and losses, low priority losses do not slow the sending rate.

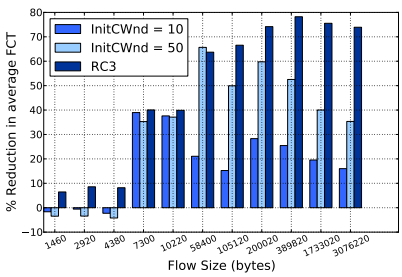
4.2 RC3 in Comparison

We now compare the performance gains of RC3 against three other proposals to reduce TCP flow completion times: increasing TCP's initial congestion window (InitCwnd), assigning critical flows to high priority, and using RCP.

Increasing InitCwnd: Figure 4.2(a) compares the performance gains obtained from RC3 with the performance gains from increasing the baseline TCP's initial congestion window to 10 and 50. For most flow sizes, especially larger flows, RC3 provides stronger improvements than simply increasing the initial congestion window. When averaging across all flows, RC3 provides a 44% reduction in FCT whereas increasing the InitCwnd to 10 reduces the FCT by only 13% and further increasing it to 50 reduces the FCT by just 24%. Further, for small flow sizes ($<$ 4 packets), increasing the InitCwnd actually introduces a performance *penalty* due to increased queueing delays. RC3 never makes flows do worse than they would have under traditional TCP. These results confirm our expectations from §3.

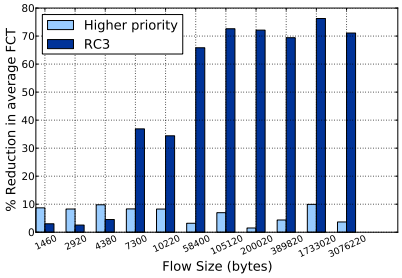
Traditional QoS: An alternate technique to improve FCTs is to designate certain flows as 'critical' and send those flows using unmodified TCP, but at higher priority. We annotated 10% of flows as 'critical'; performance results for the critical flows alone are shown in Fig. 4.2(b). For the 'critical' 10% of flows, while the average FCT reduces from 0.126 seconds to 0.119 seconds; non-critical flows suffered a very slight ($<$2%) penalty. When we repeated the experiment, but assigning the critical flows to use RC3, the average FCT 'reduced from .126 seconds to 0.078 seconds, as shown in Figure 4.2(b) . Furthermore, non-critical flows showed a slight ($<$1%) *improvement*. This suggests that it is better to be able to send an unrestricted amount of traffic, albeit at low priority, than to send at high priority at a rate limited by TCP.

RCP: Finally, we compare against RCP, an alternative transport protocol to TCP. With RCP, routers calculate average fair rate and signal this to flows; this allows flows to start transmitting at an explicitly allocated rate from the first (post-handshake) RTT, overcoming TCP's slow start penalty. We show the performance improvement for RCP and RC3 in Fig. 4.2(c). While for large flows, the two schemes are roughly neck-to-neck, RCP actually imposes a penalty for the very smallest (1-4 packet) flows, in part because RCP's explicit rate allocation enforces *pacing* of packets according to the assigned rate, whereas with traditional TCP (and RC3), all packets are transmitted back to back. These results show that RC3 can provide FCTs which are usually compa-



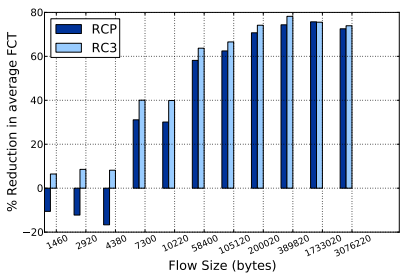
		Average Over Flows	Average Over Bytes
Regular FCT (s)		0.135	0.443
i = 10	FCT	0.117	0.362
	%	**13.21**	**17.87**
i = 50	FCT	0.102	0.272
	%	**24.33**	**38.24**
RC3	FCT	0.076	0.114
	%	**43.54**	**74.35**

(a) Increased InitCwnd



		Average Over Flows	Average Over Bytes
Regular FCT (s)		0.126	0.435
QoS	FCT	0.119	0.411
	%	**5.33**	**5.64**
RC3	FCT	0.078	0.12
	%	**38.31**	**72.43**

(b) Traditional QoS



		Average Over Flows	Average Over Bytes
Regular FCT (s)		0.135	0.443
RCP	FCT	0.088	0.117
	%	**33.86**	**73.58**
RC3	FCT	0.076	0.114
	%	**43.54**	**74.35**

(c) RCP

Figure 8: RC3 as compared to three alternatives. All FCTs are reported in seconds; % shows percent reduction from baseline. All experiments performed with 1Gbps bottleneck and 30% load.

rable or even better than those with RCP. Further, as RC3 can be deployed on legacy hardware and is friendly with existing TCP flows, it is a more deployable path to comparable performance improvements.

5. DISCUSSION

Before concluding, we now briefly discuss RC3's deployability, applicability to non-WAN environments, and compatibility with current technology trends.

Deployment: For RC3 to be widely used requires ISPs to opt-in by enabling the priority queueing that already exists in their routers. As discussed in the introduction, we believe that giving worse service, rather than better service, for these low priority packets alleviates some of the concerns that has made QoS so hard to offer (in the wide area) today. WQoS is safe and backwards compatible because regular traffic will never be penalized and pricing remains unaffected. Moreover, since RC3 makes more efficient use of bandwidth, it allows providers to run their networks at higher utilization, while still providing good performance, resulting in higher return in investment for their network provisioning. We also believe that movement of services to the edge of the network, so that paths to the most popular site traverse only a few ISPs, may make it easier for partial deployment of RC3 to bring end-to-end benefits to users.

Datacenters and Elsewhere: As we've shown via model (§3) and simulation (§4), the benefits of RC3 are strongest in networks with large RTT×BW products. Today's datacenter networks typically do not fit this description: with microsecond latencies, bandwidth×delay products are small and thus flows today can very quickly reach 100% link utilization. Nevertheless, given increasing bandwidth, bandwidth×delay products may not remain small forever. In simulations on a fat-tree datacenter topology with (futuristic) 100Gbps links, we observed average FCT improvements of 45% when averaged over flows, and 66% when averaged over bytes. Thus, while RC3 is not a good fit for datacenters today, it may be in the future.

Future: Outside of the datacenter, bandwidth×delay products are already large – and increasing. While increasing TCP's initial congestion window may mitigate the problem in the short term, given the inevitable expansion of available bandwidth, the problem will return again and again with any new choice of new initial congestion window. Our solution, while posing some deployment hurdles, has the advantage of being able to handle future speeds without further modifications.

6. RELATED WORK

In §4 we compare RC3 directly against RCP, increasing TCP's initial congestion window, and traditional QoS. We now present other related work not discussed previously.

TCP Cubic [9] and Compound TCP [14] are deployed in Linux and Windows respectively. Although they use alternative congestion avoidance algorithms to TCP New Reno, their slow-start behaviors still leave substantial wasted capacity during the first few RTTs – consequently, TCP Cubic or Compound TCP could just as easily be used in RC3's primary control loop as TCP New Reno.

PFabric [2] is a recent proposal for datacenters that similar to RC3 uses many layers of priorities and ensures high utilization. However, PFabric is targeted exclusively at the datacenter environment, and would not work in the wide-area case. Our work is targeted at the wide-area case, and may only be relevant for datacenters when their speeds increase.

7. REFERENCES

[1] ns-3. http://www.nsnam.org.

[2] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. Deconstructing Datacenter Packet Transport. In *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2012.

[3] M. Allman. Comments on bufferbloat. *SIGCOMM Comput. Commun. Rev.*, 2012.

[4] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390.

[5] E. Blanton, M. Allman, K. Fall, and L. Wang. A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP. RFC 3517.

[6] N. Dukkipati and N. McKeown. Why Flow-Completion Time is the Right Metric for Congestion Control. *ACM SIGCOMM Computer Communication Review*, 36(1):59–62, Jan. 2006.

[7] N. Dukkipati, T. Refice, Y. Cheng, J. Chu, T. Herbert, A. Agarwal, A. Jain, and N. Sutin. An Argument for Increasing TCP's Initial Congestion Window. *ACM SIGCOMM Computer Communication Review*, 40(3), June 2010.

[8] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the sprint ip backbone. *IEEE Network*, 17:6–16, 2003.

[9] S. Ha, I. Rhee, and L. Xu. CUBIC: a New TCP-friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, July 2008.

[10] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot. An approach to alleviate link overload as observed on an ip backbone, 2003.

[11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018.

[12] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. NSDI'08, pages 323–336. USENIX Association.

[13] Sally Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582.

[14] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *IEEE INFOCOM*, 2006.