RCP-AC: Congestion control to make flows complete quickly in any environment (Extended Abstract)

Nandita Dukkipati, Nick McKeown
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9030, USA
{nanditad, nickm}@stanford.edu

Alexander G. Fraser
Fraser Research
182 Nassau St. #301
Princeton, NJ 08542, USA
agf@fraserresearch.org

***Abstract*— We believe that a congestion control algorithm should make flows finish quickly - as quickly as possible, while staying stable and fair among flows. Recently, we proposed RCP (Rate Control Protocol) which enables typical Internet-sized flows to complete one to two orders of magnitude faster than the existing (TCP Reno) and the proposed (XCP) congestion control algorithm. Like XCP, RCP uses explicit feedback from routers, but doesn't require per-packet calculations. A router maintains just one rate that it gives to all flows, making it simple and inherently fair. Flows finish quickly because RCP aggressively gives excess bandwidth to flows, making it work well in the common case. However - and this is a design tradeoff - RCP will experience short-term transient overflows when network conditions change quickly (e.g. a route change or flash crowds). In this paper we extend RCP and propose RCP-AC (Rate Control Protocol with Acceleration Control) that allows the aggressiveness of RCP to be tuned, enabling fast completion of flows over a broad set of operating conditions.**

I. Introduction and Motivation

Users want their downloads and file-transfers to complete as fast as possible, making flow-completion time arguably the most important performance metric for a congestion control algorithm [1]. We recently proposed a simple congestion control algorithm called the Rate Control Protocol (RCP) [2] that reduces flow-completion times (FCT) for typical Internet size flows by one to two orders of magnitude compared to TCP and eXplicit Control Protocol (XCP) [3]. RCP achieves this by a simple, and perhaps obvious way to explicitly emulate processor sharing at each router. In RCP, a router assigns *the same rate to all flows* ($R(t)$) passing over a link, where $R(t)$ adapts to current link congestion. Every packet carries a field for the smallest $R(t)$ along the path; if a router has a lower value for $R(t)$ than the one in a packet, it overwrites it. The destination sends the value back to the source, and so the source can send traffic according to the fair-share rate of the bottleneck link. This simple algorithm is inherently fair (all flows at a bottleneck receive the same rate), requires no per-packet calculation of rates, and (most imporantly) allows flows to jump-start to the correct rate (because even connection setup packets are stamped with the fair-share rate). It is also provably stable over a broad range of operating conditions [4]. Even short-lived flows that perform badly under TCP (because they never leave slow-start) will finish quickly with RCP.

However, like all feedback algorithms, RCP sets a rate that is out of date when it is used (an RTT later). While most congestion control algorithms act conservatively (or even timidly) to start with (e.g. the slow-start algorithm of TCP Reno that starts out with a window size of just two packets; and the slow additive increase of XCP), RCP starts up immediately at the best-estimate of the current fair-share rate. This means flows finish quickly; but it also runs the risk of overshooting – particularly when there are rapid changes in network traffic, for example because of a sudden flash crowd. In backbone networks with tens of thousands of flows, traffic is so smooth that sudden change is unlikely; but can and will happen occasionally in access networks. And even in backbone networks, sudden traffic changes can happen when links and routers fail, suddenly redirecting many flows. In these cases RCP will recover (it is provably stable in the control theoretic sense), but only after a transient in which buffers overflow and packets are lost. In many networks this might be quite acceptable - particularly those optimized for the common case [5]. In fact, we believe that most future networks and most users will be well-served by RCP.

In this paper we are interested in the design space ranging from aggressive rate control (RCP) to conservative rate control. In particular, we propose an enhanced version of RCP called RCP-AC (RCP with Acceleration Control) that can be tuned to work over a broad range of conditions: At one end, it can be made aggressive so that flows will finish very quickly; at the other extreme, it can be made more conservative. Our goal is to allow an operator to choose the operating point, and control packet loss, while still allowing flows to finish much more quickly than in conventional algorithms. We are not aware of any prior congestion control algorithm that achieves short flow-completion times under a diverse set of traffic profiles.

II. Description of RCP-AC

RCP-AC is an extension of RCP. In the basic RCP algorithm every link maintains a single rate, $R(t)$, that it gives to all flows. The router updates $R(t)$ approximately once per round-trip time (RTT). Intuitively, to emulate processor sharing, the router should offer the same rate to every flow, try to fill the outgoing link with traffic, *and* keep the queue occupancy close to zero. The RCP rate update equation is based on this intuition:

$$R(t) = R(t-T)(1 + \frac{\frac{T}{d}(\alpha(C - y(t)) - \beta\frac{q(t)}{d})}{C}) \qquad (1)$$

where d is a moving average of the RTT measured across all packets, T is the update interval (i.e., how often $R(t)$ is updated) and is less than or equals d, $R(t-T)$ is the last rate, C is the link-capacity, $y(t)$ is the measured aggregate input traffic rate during the last update interval, $q(t)$ is the instantaneous queue size, and α, β are parameters chosen for stability and performance [4].

At a high level RCP is similar to XCP: both assign explicit rates calculated by routers, and RCP uses a similar rate-update equation. However, XCP gives a different rate to each flow, requiring per-packet calculations. Most relevant here, XCP converges slowly towards the fair-share rate in small increments so as to avoid packet loss. For most flows, the flow has finished before the fair-share rate has been reached, which explains why flows take so long to complete with XCP. On the other hand, RCP gives a high starting rate; so new flows find their fair-share rate quickly because a single rate is given to all flows (new and old). Flows achieve low completion times, but there can be sharp peaks in queue length that lead to packet loss when there is a rapid change in network conditions [5].

RCP-AC achieves flow-completion times that are close to processor sharing, with low risk of overshoot and packet loss even when the offered aggregate traffic changes suddenly, such as when traffic is rerouted and for flash crowds. RCP-AC controls traffic in three different ways: a) *Rate Control.* It sets a target flow-rate, just like RCP; b) *Acceleration Control.* It limits the acceleration (increase in flow-rate); and c) *Feedback Control.* It gives priority to jump-starting new flows; only when there is congestion will RCP-AC give priority to limiting queue size. These functions are executed periodically in every router, typically about once per round-trip time.

The Rate Controller uses the same equation as RCP above. The Acceleration Controller computes, ϕ, the maximum increase in aggregate traffic volume permitted in the next time period:

$$\phi = \alpha(C - y(t)) \cdot d + (\gamma B - \beta q(t)) \qquad (2)$$

where B is the buffer size and γ is the user-defined limit on acceleration. Except for the γB term, this equation is like the Efficiency Controller in XCP. γ controls how aggressive the control system is. Intuitively, γ determines how fast a new flow climbs to $R(t)$ as well as the worst-case buffer occupancy. Small γ means relatively small risk because there is buffer available to absorb any surge caused by flows that start quickly, and so packet drop rate will be low. On backbone links with thousands of flows, each flow is a small fraction of the link capacity, so overflow is unlikely and γ can be large.

The Feedback Controller gives per-packet feedback (in terms of window change)[1] to apportion bandwidth among flows. It is similar in mechanism to the Fairness Controller of XCP. It gives negative feedback to flows above the target-rate, and gives positive feedback to flows below $R(t)$ by distributing ϕ among them in proportion to how far away their flow-rates are below $R(t)$. This enables new flows to get a high proportion of the feedback quota, as opposed to older flows which are already close to the target-rate. In each case, the total feedback given to a flow, positive or negative, is divided among its packets in a control interval.

The above intuition leads to the following negative feedback in packet i:

$$feedback_i^- = -\frac{\max(0, \frac{cwnd_i}{rtt_i} - R(t)) \cdot rtt_i}{\frac{cwnd_i}{rtt_i} \cdot d} \qquad (3)$$

where $\max(0, \frac{cwnd_i}{rtt_i} - R(t)) \cdot rtt_i$ is the desired change in window and $\frac{cwnd_i}{rtt_i} \cdot d$ is the total number of packets of this flow seen by the link in a control interval.

The positive feedback is slightly more complex because the total amount of feedback is bounded and is proportionally divided among flows depending on their rate. Omitting the derivation details:

$$feedback_i^+ = \min[\frac{\max(0, R(t) - \frac{cwnd_i}{rtt_i}) \cdot rtt_i}{\frac{cwnd_i}{rtt_i} \cdot d},\ p_i] \qquad (4)$$

$$p_i = \max(\phi, 0) \cdot \frac{\max(0, R(t) - \frac{cwnd_i}{rtt_i}) \cdot \frac{rtt_i}{cwnd_i}}{\sum^K \frac{rtt_i}{cwnd_i} \max(0, R(t) - \frac{cwnd_i}{rtt_i})} \cdot \frac{rtt_i}{d}$$

where K is the total number of packets seen in a control interval. The intuition to take away is that as acceleration, ϕ, gets larger the first term of the $\min(\cdot)$ in Equation (4) will become more important, the dominant behavior will be that of the Rate Controller and so RCP-AC behaves like RCP. On the other hand, as ϕ gets smaller the second term limits the acceleration in rate.

III. Performance Results

We experimented with RCP-AC under many traffic conditions, using *ns-2*. Our simulation scripts are available at [6]. Here, for brevity, we describe how RCP-AC works under two extreme conditions. First, we consider how well it works when there are sudden and unexpected changes in the number of flows – for example, from flash crowds. Second, we consider a more benign network with many multiplexed flows, so that each flow represents a small fraction of the link capacity.

[1]Each packet carries the sender's current congestion window, $cwnd$, estimate of its round trip time, rtt, and a field for feedback in window increase/decrease, $feedback$, initialized by the sender and modified by the routers along the path. When it reaches the receiver, $feedback$ has the minimum positive feedback along its path. The receiver sends this back to the sender in the acknowledgment packet.

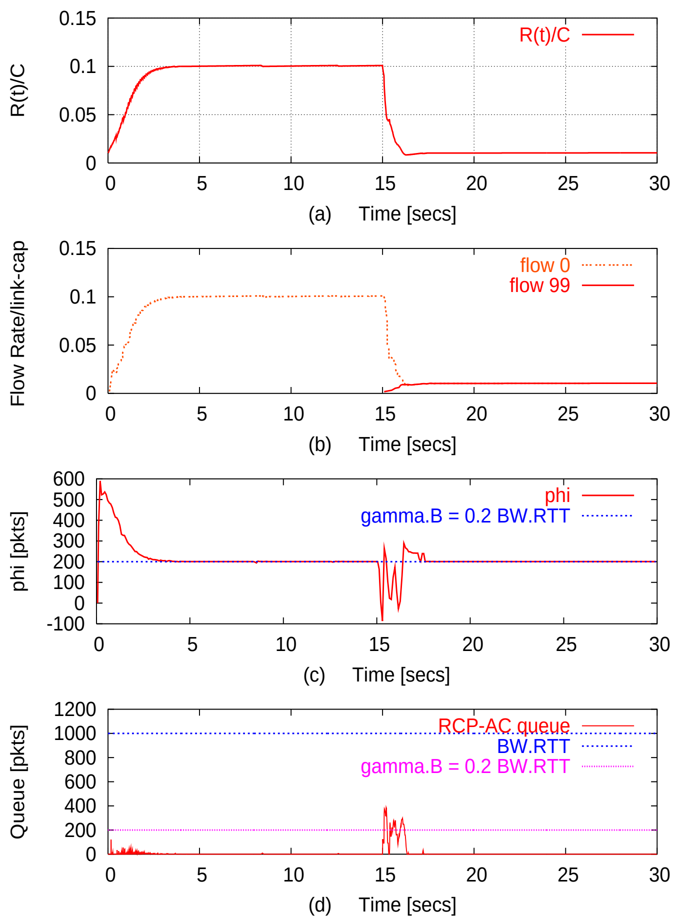

Fig. 1. Example of RCP-AC under bursty traffic: 10 long flows start at $t = 0$ and 90 more flows start at $t = 15$. Link capacity = 100 Mbps, RTT = 80 ms, $B = C \cdot RTT$, $\gamma = 0.2$. (a) the Rate Controller adapts $R(t)$ to find the equilibrium (b) flow-rates of two sample flows - at $t = 15$ flow 0 receives negative feedback while flow 99 gets positive feedback (c) the Acceleration Controller adjusts the positive feedback, ϕ, as per Equation (2) (d) the maximum queue occupancy is limited (approximately) by the maximum positive feedback.

Sudden Unexpected Change: Figure 1 shows the response of RCP-AC when 10 flows start at time 0 and 90 more start at time 15 i.e load is increased by 9 times within a round-trip time. Our goal is to illustrate how RCP-AC can be tuned for good transient state properties under unpredictable traffic changes and it is only for clarity of illustration that we chose a simple set-up with long-lived flows. With RCP, we would have seen a surge in queue occupancy, or lots of dropped packets. Figure 1 shows that RCP-AC doesn't have this problem for small γ. The flows track the target-rate set by the Rate Controller (shown in figures (a) and (b)) without creating a large spike in the queue (shown in figure (d)). The size of the queue spike is a linear function of γB.

The good transient properties of RCP-AC for small γ translate to short flow-completion times under traffic patterns with very sudden changes.

Aggregated Traffic: Figure 2 shows the performance of RCP-AC when lots of flows are multiplexed onto a link, as in a backbone network. Flows arrive in a Poisson process

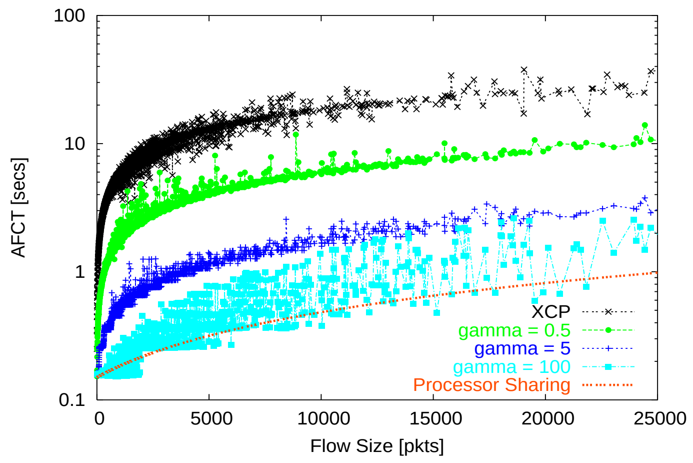

Fig. 2. Example of RCP-AC under aggregated traffic: the plot shows the average flow-completion times (AFCT) for XCP and RCP-AC (under different γ values). Link-capacity = 2.4 Gbps, RTT = 0.1s, offered load = 0.9, flow-sizes are pareto distributed with mean 25 packets and shape 1.2.

and have pareto-distributed flow sizes. The figure compares the average flow-completion time (AFCT) of RCP-AC (for different γ values) and XCP with that of processor sharing. As γ gets larger the AFCT of RCP-AC gets closer to that of RCP[2] and processor sharing. Increasing γ makes flows finish faster because they can jump quickly to the fair-share rate.

IV. Conclusion

RCP-AC (like RCP) is stable over all operating conditions; but represents a tradeoff in aggressiveness between how quickly flows finish, and how much the buffers are allowed to grow. It is promising because it can be tuned for a broad set of traffic conditions. On the other hand, it is much more complex (several per-packet computations) than RCP. It remains an open question if we can achieve similar goals with different mechanisms that require few/no per-packet computation. Ultimately, RCP-AC's usefulness will depend on whether we want to design for the common case (for which RCP works well), or more conservatively for the worst-case where traffic surges occasionally happen.

References

[1] Nandita Dukkipati and Nick McKeown, "Why Flow-Completion Time is the Right Metric for Congestion Control," In *ACM SIGCOMM Computer Communication Review*, Volume 36, Issue 1, January 2006.
[2] Nandita Dukkipati, Masayoshi Kobayashi, Rui Zhang-Shen, and Nick McKeown, "Processor Sharing Flows in the Internet," In *Thirteenth International Workshop on Quality of Service 2005*, Germany, June, 2005.
[3] Dina Katabi, Mark Handley, and Charles Rohrs, "Internet Congestion Control for High Bandwidth-Delay Product Networks," In *Proceedings of ACM Sigcomm 2002*, Pittsburgh, August, 2002.
[4] Hamsa Balakrishnan, Nandita Dukkipati, Nick McKeown and Claire Tomlin "Stability Analysis of Explicit Congestion Control Protocols," In *Stanford University Department of Aeronautics and Astronautics Report, SUDAAR 776*, September 2005.
[5] Nandita Dukkipati, Yashar Ganjali and Rui Zhang-Shen, "Typical versus Worst Case Design in Networking," In *Fourth Workshop on Hot Topics in Networks*, College Park, Maryland, November 2005.
[6] http://yuba.stanford.edu/rcp/

[2]not shown in the figure.