Internet Engineering Task Force
INTERNET-DRAFT
Intended status: Experimental
Expires: 13 October 2007

S. Floyd, Editor
ICIR
A. Arcia
D. Ros
ENST Bretagne
J. Iyengar
Connecticut College
13 April 2007

Adding Acknowledgement Congestion Control to TCP draft-floyd-tcpm-ackcc-00.ps

Status of this Memo

Copyright Notice

Abstract

This document adds an optional congestion control mechanism for acknowledgement traffic (ACKs) to TCP. The document specifies an end-to-end acknowledgement congestion control mechanism for TCP that uses participation from both TCP hosts, the TCP data sender and the TCP data receiver. The TCP data sender detects lost and ECN-marked ACK packets, and tells the TCP data receiver the ACK Ratio R to use to respond to the congestion on the reverse path from the data receiver to the data sender. The TCP data receiver sends roughly one ACK packet for every R data packets received. This mechanism is based on the acknowledgement congestion control in DCCP's CCID 2 [RFC4340], [RFC4341]. This acknowledgement congestion control mechanism is being proposed as an experimental mechanism for TCP for evaluation by the network community.

1. Introduction

This documents adds an optional congestion control mechanism to TCP for acknowledgements (ACKs). This mechanism is based on the acknowledgement congestion control in DCCP's CCID 2 [RFC4340], [RFC4341].

In this document we use the termininology of senders and receivers, with the sender sending data traffic, and the receiver sending acknowledgement traffic in response. In CCID 2's acknowledgement congestion control, specified in Section 6.1 of [RFC4341], the receiver maintains an ACK Ratio R, sending roughly one ACK packet for every R data packets received. The CCID 2 sender keeps the acknowledgement rate roughly TCP friendly by monitoring the acknowledgement stream for lost and marked ACK packets and modifying the ACK Ratio accordingly. For every RTT containing an ACK congestion event (that is, a lost or marked ACK packet), the sender halves the acknowledgement rate by doubling the ACK Ratio; for every RTT containing no ACK congestion event, the sender additively increases the acknowledgement rate through gradual decreases in the ACK Ratio.

Adding a similar acknowledgement congestion control as an option in TCP requires the following:

* An agreement from the TCP hosts on the use of ACK congestion control. The TCP hosts use a new TCP option, the ACK-Congestion-Control-Permitted Option.

* A mechanism for the TCP sender to detect lost and ECN-marked pure acknowledgement packets.

* A mechanism for adjusting the ACK Ratio. The TCP sender adjusts the ACK Ratio as specified in Section 6.1.2 of [RFC4341].

* A method for the TCP sender to inform the TCP receiver of a new value for the ACK Ratio. The TCP sender uses a new TCP option, the ACK Ratio Option.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Overview

This section gives a non-normative overview of acknowledgement congestion control for TCP.

[Graphics will be added.]

During connection initiation, TCP host B sends an ACK-Congestion-Control-Permitted option on its SYN or SYN/ACK packet. This allows TCP host A (now called the sender) to send instructions to TCP host B (now called the receiver) about the Ack Ratio to use in responding to data packets.

Also during connection initiation, TCP host A sends an ACK-Congestion-Control-Permitted option on its SYN or SYN/ACK packet. In combination with TCP host B's sending of an ACK-Congestion-Control-Permitted option, this allows TCP host B to send its ACK packets as ECN-Capable.

The TCP receiver starts with an ACK Ratio of two, generally sending one ACK packet for every two data packets received.

The TCP sender detects lost or ECN-marked ACK packets from the TCP receiver, and at some point sends an ACK Ratio option of three to the receiver. The TCP receiver changes to an ACK Ratio of three, generally sending one ACK packet for every three data packets. The TCP sender uses Appropriate Byte Counting and rate-based pacing in responding to these ACK packets.

The TCP sender detects fewer lost ACK packets, and at some point sends an ACK Ratio option of two to the TCP receiver. The TCP receiver changes back to an ACK Ratio of two, generally sending one ACK packet for every two data packets.

4. Related Work

The goal of the mechanism proposed in this document is to control pure ACK traffic on the path from the TCP data receiver to the TCP data sender. Note that the approach outlined here is an end-to-end one (as is the approach followed by DCCP's CCID 2 [RFC4341]), but it may also take advantage of explicit congestion information from the network conveyed by ECN [RFC3168], if available. The ECN specification [RFC3168, section 6.1.4] prohibits a TCP receiver from setting the ECT(0) or ECT(1) codepoints in IP packets carrying pure ACKs, but *only* as long as the receiver does *not* implement any form of ACK congestion control.

There exist several papers dealing with controlling congestion in the reverse path of a TCP connection, especially in the context of networks with bandwidth asymmetry. Some of these proposals require explicit support from routers or middleboxes, whereas others are "pure" end-to-end schemes.

Balakrishnan et al. ([BPK97]) describe the use of ECN to detect congestion in the return path, in order to reduce the sending rate of ACKs. The use of a RED queue in the reverse path allows for marking of ACK packets. The sender echoes back ECN congestion marks to the receiver. The receiver keeps an ACK ratio d (called the "delayed-ACK factor"), specifying the number of data segments that have to be received before the receiver sends a new ACK. The ACK ratio d is managed using multiplicative-increase, additive-decrease; upon reception of a congestion mark, the receiver doubles the value of d (hence dividing the ACK sending rate by two). The ACK ratio decreases linearly for each RTT in which no ECN-marked ACKs are received. Multiple congestion marks received in an RTT are treated as a single congestion event, i.e., d can be doubled at most once per RTT. The TCP timestamp option is used to keep track of the RTT values.

In [TJW00], Tam Ming-Chit et al. propose a receiver-based method for calculating an "appropriate" number of ACKs per congestion window (cwnd) of data, in order to alleviate congestion on the reverse path. The sender's cwnd is estimated at the receiver by counting the number of received packets per RTT (which also has to be estimated by the receiver). From this estimate, a simple algorithm is used to compute the number of ACKs to be sent per cwnd. The algorithm enforces a lower bound on the number of ACKs per cwnd, aiming at minimizing the probability of timeout at the sender due to ACK loss. Similarly, the ACK ratio is upper-bounded so as to avoid excessive ACK delay.

ACK filtering (AF) [BPK97] from Balakrishnan et al. is a router-based technique that tries to reduce the number of ACKs sent over the congested return link. With AF, an arriving ACK may replace preceding, older ACKs at the bottleneck queue. An aggressive replacement policy might guarantee that at most one ACK per connection is waiting in the queue, alleviating congestion. However, as in other proposals, care must be taken to avoid sender timeouts in case the (too few) ACKs resulting from the filtering get lost. The idea of filtering ACKs has been extended in [YMH03] to deal with SACK information.

Blandford et al. [BGG+07] propose an end-to-end, receiver-oriented scheme called "smartacking". The algorithm is based upon the receiver monitoring the inter-segment arrival time for data packets and adapting the ACK sending rate in response. When the bottleneck link is underutilized, ACKs are sent frequently (up to one ACK per received segment) to promote fast growth of the congestion window. On the other hand, when the bottleneck is close to full utilization, the algorithm tries to reduce control traffic overhead and slow congestion window growth by generating ACKs at the minimum rate needed to keep the data pipe full.

Reducing the number of ACKs (or, equivalently, increasing the amount of bytes acknowledged by each ACK) can increase the burstiness of the TCP sender. Hence, any mechanism as those cited above should be coupled with some burst mitigation technique [AB05]. Such a technique may consist in either limiting the size of the bursts or pacing the sending of data segments [ASA00], or a combination of both like the Sender Adaptation proposal in [BPK97].

Aweya et al. [AOM02] present a middlebox-based approach for mitigating data packet bursts and for controlling the uplink ACK congestion. The main idea is to perform pacing on ACK segments on an edge device close to the sender, so as to control the ACK arrival rate at the sender.

Unlike some of the related work cited above, in this document we are proposing an end-to-end ACK congestion control mechanism that controls congestion on the reverse path (the path followed by the ACK traffic) by detecting and responding to marked or dropped ACK packets.

5. Acknowledgement Congestion Control

5.1. Negotiating the Use of ACK Congestion Control

The TCP end-points negotiate the use of ACK Congestion Control (ACKCC) with a TCP option, the ACK-Congestion-Control-Permitted Option. The option number will be allocated by IANA.

The ACK-Congestion-Control-Permitted option can only be sent on packets that have the SYN bit set. If TCP end-point A receives an ACK-Congestion-Control-Permitted option from TCP end-point B, then the TCP end-points MAY use ACK Congestion Control on the pure acknowledgements sent from B to A. This means that TCP end-point A MAY send ACK Ratio values to TCP end-point B, for TCP end-point B to use on pure acknowledgement packets.

Similarly, if TCP end-point B receives an ACK-Congestion-Control-Permitted option from TCP end-point A, then the TCP end-points MAY use ACK Congestion Control on the pure acknowledgements sent from A to B.

If TCP end-point B receives an ACK-Congestion-Control-Permitted option from TCP end-point A and also sent an ACK-Congestion-Control-Permitted option to TCP end-point A, then TCP end-point B can send its ACK packets as ECN-Capable.

```
TCP ACK-Congestion-Control-Permitted Option:

Kind: N

+-----------+-----------+
|  Kind=N   |  Length=2 |
+-----------+-----------+
```

When ACK Congestion Control is used, the default initial ACK Ratio is two, with the receiver acknowledging at least every other data packet.

5.2. The TCP ACK Ratio Option

The sender uses a ACK Ratio TCP Option to communicate the ACK Ratio value from the sender to the receiver.

```
TCP ACK Ratio Option:

Kind: N+1

+-----------+-----------+-----------+
|  Kind=N+1 |  Length=3 | ACK Ratio |
+-----------+-----------+-----------+
```

The ACK Ratio Option is only sent on data packets. Because TCP uses reliable delivery for data packets, the TCP sender can tell if the TCP receiver has received an ACK Ratio Option.

5.3. Implementing the ACK Ratio

With an ACK Ratio of R, the receiver should send one pure ACK for every R newly received data packets unless the delayed ACK timer expires first. A receiver could simply maintain a counter that increments up to R for each new data packet received, and then reset the counter to zero when an ACK is sent, either pure or piggybacked.

[RFC2581] recommends that the receiver SHOULD acknowledge out-of-order data packets immediately, sending an immediate duplicate ACK when it receives a data segment above a gap in the sequence space, and sending an immediate ACK when it receives a data segment that fills in all or part of a gap in the sequence space.

When ACK Congestion Control is being used and the ACK Ratio is at most two, the TCP receiver MUST acknowledge each out-of-order data packet immediately. For an ACK Ratio greater than two, Section 5.6 specifies in detail the receiver's behavior for sending ACKs for out-of-order data packets.

5.4. Determining Lost or Marked ACK Packets

The TCP data sender uses its knowledge of the ACK Ratio in use by the receiver to infer when an ACK packet has been lost.

Because the TCP sender knows the ACK Ratio R in use by the receiver, the TCP sender knows that in the absence of dropped or reordered acknowledgement packets, each new acknowledgement received will acknowledge at most R additional data packets. Thus, if the sender receives an acknowledgement acknowledging more than R data packets, and does not receive a subsequent acknowledgement acknowledging a strict subset (with a smaller cumulative acknowledgement, or with the same cumulative acknowledgement but a strict subset of data acknowledged in SACK blocks), then the sender can infer that an ACK packet has been dropped.

Similarly, the TCP sender knows that in the absence of dropped or delayed data packets from the sender, and in the absence of delayed acknowledgements due to a timer expiring at the receiver, each new pure acknowledgement received will acknowledge at least R additional data packets. In terms of ACK congestion control, the TCP sender does not have to take any actions when it receives an acknowledgement acknowledging less than R additional packets.

If the ACK Ratio is at most two, then the TCP receiver sends a dupACK for every out-of-order data packet. In this case, the TCP sender can detect lost DupACK packets by counting the number of DupACKs that arrived between the beginning of the loss event and the arrival of the first full or partial ACK, and comparing this number with the number of DupACKs that should have arrived (based on the number of packets being ACKed by the full or partial ACK).

If the ACK Ratio is greater than two, the TCP receiver does not send a dupACK for every out-of-order data packet, as specified in Section 5.6. For simplicity, if the ACK Ratio is greater than two, the TCP sender does not attempt to detect lost ACK packets during loss events involving forward-path data traffic. That is, as soon as the sender infers a packet loss for a forward-path data packet, it stops detection of ACK loss on the reverse path. The sender waits until a new cumulative acknowledgement is received that covers the retransmitted data, and then restarts detection of ACK loss for reverse-path traffic.

5.5. Adjusting the ACK Ratio

The TCP sender will adjust the ACK Ratio as specified in Section 6.1.2 of [RFC4341], as follows.

The ACK Ratio always meets three constraints: (1) the ACK Ratio is an integer. (2) the ACK Ratio does not exceed cwnd/(2*MSS), rounded up, except that ACK Ratio 2 is always acceptable. (3) the ACK Ratio is two or more for a congestion window of four or more full-sized segments.

The sender changes the ACK Ratio within those constraints as follows. For each congestion window of data with lost or marked ACK packets, the ACK Ratio R is doubled; and for each cwnd/(MSS*(R^2 - R)) consecutive congestion windows of data with no lost or marked ACK packets, the ACK Ratio is decreased by 1. (See Appendix A of RFC 4341 for the derivation. Note that Appendix A of RFC 4341 assumes a congestion window W in packets, while we use cwnd in bytes.)

For a constant congestion window, this gives an ACK sending rate that is roughly TCP friendly. Of course, cwnd usually varies over time; the dynamics will be rather complex, but roughly TCP friendly. We recommend that the sender use the most recent value of cwnd when determining whether to decrease ACK Ratio by one.

The sender need not keep the ACK Ratio completely up to date. For instance, it MAY rate-limit ACK Ratio renegotiations to once every four or five round-trip times, or to once every second or two. The sender SHOULD NOT attempt to change the ACK Ratio more than once per round-trip time. Additionally, it MAY enforce a minimum ACK Ratio of two, or it MAY set ACK Ratio to one for half-connections with persistent congestion windows of 1 or 2 packets.

With ACK congestion control, the receiver could be sending two ACK packets per window of data even in the face of very heavy congestion on the reverse path. We would note, however, that if congestion is sufficiently heavy, all the ack packets are dropped, and then the sender falls back on an exponentially backed-off timeout. Thus, if congestion is sufficiently heavy on the reverse path, then the sender reduces its sending rate on the forward path, which reduces the rate on the reverse path as well.

5.6. Sending ACKs for Out-of-Order Data Segments

RFC 2581 says that "a TCP receiver SHOULD send an immediate duplicate ACK when an out-of-order segment arrives." After three duplicate ACKs are received, the TCP sender infers a packet loss and implements Fast Retransmit and Fast Recovery, retransmitting the missing packet. When the ACK Ratio is at most two, the TCP receiver SHOULD still send an immediate duplicate ACK when an out-of-order segment arrives.

When the ACK Ratio is greater than two, the TCP receiver still SHOULD send an immediate duplicate ACK for each of the first three out-of-order segments that arrive in a reordering event. (We define a reordering event at the receiver as beginning when an out-of-order segment arrives, and ending when the receiver holds no more out-of-order segments.) However, when the ACK Ratio is greater than two, after the first three duplicate ACKs have been sent, the TCP receiver should perform ACK congestion control on the remaining ACKs to be sent during the current reordering event. That is, after the first three duplicate ACKs have been sent, the TCP receiver SHOULD send an ACK for every R out-of-order segments, instead of sending an ACK for every out-of-order segment. In addition, a receiver MUST NOT withhold an ACK for more than 500 ms.

5.7. The Sender's Response to ACK Packets

The use of a large ACK Ratio can generate line rate data bursts at a TCP sender. When the ACK Ratio is greater than two, the TCP sender SHOULD use some form of burst mitigation, or rate-based pacing for sending data packets in response to a single acknowledgement. The use of rate-based pacing will be limited by the timer granularity at the TCP sender.

We note that the interaction of ACK congestion control and burst mitigation schemes needs further study.

In addition to the impact of a large ACK Ratio on the burstiness of the TCP sender's sending rate, a large ACK Ratio can also affect the data sending rate by slowing down the increase of the congestion window cwnd. As specified in RFC 2581, in slow-start the TCP sender increases cwnd by one full-sized segment for each new ACK received (in this context, a "new ACK" is an ACK that acknowledges new data). RFC 2581 also specifies that in congestion avoidance, the TCP sender increases cwnd by roughly 1/cwnd full-sized segments for each ACK received, resulting in an increase in cwnd of roughly one full-sized segment per round-trip time. In this case, the use of a large ACK Ratio would slow down the increase of the sender's congestion window.

RFC 2581 notes that it is also acceptable to count the number of bytes acknowledged by new ACKs, and to increase cwnd based on the number of bytes acknowledged, rather than on the number of new ACKs received. Thus, the sender SHOULD use Appropriate Byte Counting [RFC3465] with Acknowledgement Congestion Control, so that the Acknowledgement Congestion Control doesn't slow down the window increases for the data traffic sent by the sender.

As cited earlier, RFC 2581 infers that a packet has been lost after it receives three duplicate acknowledgements. Because ACK Congestion Control is only used when there is congestion on the reverse path, after a packet loss one or more of the three duplicate ACKs sent by the receiver could be lost on the reverse path, and the receiver might wait until it has received R more out-of-order segments before sending the next duplicate ACK. All this could slow down Fast Recovery and Fast Retransmit quite a bit. To reduce the potential delay in detecting a lost packet, we add that when SACK is used, a TCP sender SHOULD use the information in the SACK option to detect when the receiver has received at least three out-of-order data packets, and to initiate Fast Retransmit and Fast Recovery in this case, even if the TCP sender has not yet received three dup ACKs.

6. Possible Complications

6.1. Possible Complications: Delayed Acknowledgements

The receiver could send a delayed acknowledgement acknowledging a single packet, even when the ACK Ratio is two or more.

This should not cause false positives (when the TCP sender infers a loss when no loss happened). The TCP sender only infers that a pure ACK packet has been lost when no data packet has been lost, and an ACK packet arrives acknowledging more than R new packets.

Delayed acknowledgements could, however, cause false negatives, with the TCP sender unable to detect the loss of an ack packet sent as a delayed acknowedgement. False negatives seem acceptable; this would result in approximate ACK congestion control, which would be better than no ACK congestion control at all. In particular, when this form of false negative occurs, it is because the receiver is sending acknowledgements at such a low rate that it is sending delayed acknowledgements, rather than acknowledging at least R data packets with each acknowledgement.

6.2. Possible Complications: Duplicate Acknowledgements.

As discussed in Section 5.3, RFC 2581 states that "a TCP receiver SHOULD send an immediate duplicate ACK when an out-of-order segment arrives," and that "a TCP receiver SHOULD send an immediate ACK when the incoming segment fills in all or part of a gap in the sequence space" [RFC2581]. When ACK Congestion Control is used, the TCP receiver instead uses the guidelines from Section 5.6 to govern the sending of duplicate ACKs. More work would be useful to evaluate the advantages and disadvantages of this approach in terms of the potential delay in triggering Fast Retransmit, and to explore alternate possibilities.

6.3. Possible Complications: Two-Way Traffic.

In a TCP connection with two-way traffic, the receiver could send some pure ACK packets, and some acknowledgements piggy-backed on data packets. In this case, how well can the TCP sender infer when pure ACK packets have been lost? The receiver would still follow the rule of only sending a pure ACK packet when there is a need for a delayed ack, or there are R new data packets to acknowledge.

6.4. Possible Complications: Reordering of ACK Packets.

It is possible for ACK packets to be reordered on the reverse path. The TCP sender could either use a parallel mechanism to the dupACK threshold to infer when an ACK packet has been lost, as with TCP, or, more robustly, the TCP sender could wait an entire round-trip time before inferring that an ACK packet has been lost [RFC4653].

6.5. Possible Complications: Abrupt changes in the ACK path.

What happens when there are abrupt changes in the reverse path, such as from vertical handovers? Can there be any problems that would be worse than those experienced by a TCP connection that is not using ACK congestion control?

6.6. Possible Complications: Corruption.

As with data packets, it is possible for ACK packets to be dropped in the network due to corruption rather than congestion. The current assumption of ACK congestion control is that all losses should be taken as indications of congestion. When there is some better answer for corrupted TCP data packets, the same solution hopefully would apply to corrupted ACK packets as well.

6.7. Other Issues

Are there any problems caused by the combination of two-way traffic and reordering?

How well would ACK congestion control work without SACK information? Or would SACK be required with ACK congestion control?

7. Acknowledgement Congestion Control in CCID 2

Rate-based pacing: For CCID 2, RFC 4341 says that "senders MAY use a form of rate-based pacing when sending multiple data packets liberated by a single ACK packet, rather than sending all liberated data packets in a single burst." However, rate-based pacing is not required in CCID 2.

Increasing the congestion window: For CCID 2, RFC 4341 says that "when cwnd < ssthresh, meaning that the sender is in slow-start, the congestion window is increased by one packet for every two newly acknowledged data packets with ACK Vector State 0 (not ECN-marked), up to a maximum of ACK Ratio/2 packets per acknowledgement. This is a modified form of Appropriate Byte Counting [RFC3465] that is consistent with TCP's current standard (which does not include byte counting), but allows CCID 2 to increase as aggressively as TCP when CCID 2's ACK Ratio is greater than the default value of two. When cwnd >= ssthresh, the congestion window is increased by one packet for every window of data acknowledged without lost or marked packets."

8. Security Considerations

[To be finished later.]

What are the sender's incentives to cheat on ACK congestion control? What are the receiver's incentives to cheat? What are the avenues open for cheating?

As long as ACK congestion control is optional, neither host can be forced to use ACK congestion control if it doesn't want to. So ACK congestion control will only be used if the sender or receiver have some chance of receiving some benefit.

As long as ACK congestion control is optional for TCP, there is little incentive for the TCP end nodes to cheat on non-ECN-based ACK congestion control. There is nothing now that requires TCP hosts to use congestion control in response to dropped ACK packets.

What avenues for cheating are opened by the use of ECN-Capable ACK packets? If the end nodes can use ECN to have ACK packets marked rather than dropped, and if the end nodes can then avoid the use of ACK congestion control that goes along with the use of ECN on ACK packets, then the end nodes could have an incentive to cheat. Senders could cheat by not instructing the receiver to use a higher ACK Ratio; the receiver would have a hard time detecting this cheating. Receivers could cheat by not using the ACK Ratio they were instructed to use, but senders could easily detect this cheating. However, receivers could also cheat by not using ACK congestion control and still sending ACK packets as ECN-capable, so ACK congestion control is not a necessary component for receivers to cheat about sending ECN-capable ACK packets. One question would be whether there is any way for receivers to cheat about sending ECN-Capable ACK packets and not using appropriate ACK congestion control without this cheating being easily detected by the sender.

What about the ability of routers or middleboxes to detect TCP receivers that cheat by inappropriately sending ACK packets as ECN-capable? The router will only know if the receiver is authorized to send ACK packets as ECN-Capable if it monitored both the SYN and SYN/ACK packets (and was able to read the TCP options in the packet headers). If ACK congestion control has been negotiated, the router will only know if ACK congestion control is being used correctly by the receiver if it can monitor the ACK Ratio options sent from the sender to the receiver. If ACK congestion control is being used, the router will not necessarily be able to tell if ACK congestion control is being used correctly by the sender, because drops of ACK packets might be occurring after the ACK packets have left the router. However, if the router sees the ACK Ratio options sent from the sender, the router will be able to tell if the sender is correctly accounting for those ACK packets that are dropped or ECN-marked on the path from the receiver to the router.

9. IANA Considerations

IANA will allocate the option numbers for the two TCP options, the ACK-Congestion-Control-Permitted Option, and the ACK Ratio Option.

10. Conclusions

11. Acknowledgements

Many thanks for feedback and contributed text from Michael Welzl.

Normative References

[RFC2119] S. Bradner, Key Words For Use in RFCs to Indicate Requirement Levels, RFC 2119.

[RFC2581] Allman, M., V. Paxson, and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[RFC3465] Allman, M., TCP Congestion Control with Appropriate Byte Counting (ABC), RFC 3465, Experimental, February 2003.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

[RFC4341] Floyd, S., and E. Kohler, Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control, RFC 4341, March 2006.

Informative References

[RFC3168] K. Ramakrishnan, S. Floyd and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, September 2001.

[RFC4653] S. Bhandarkar, A. L. N. Reddy, M. Allman and E. Blanton, Improving the Robustness of TCP to Non-Congestion Events, RFC 4653, August 2006.

[ASA00] A. Aggarwal, S. Savage, and T. Anderson. Understanding the Performance of TCP Pacing. In INFOCOM (3), pages 11571165, 2000.

[AB05] M. Allman and E. Blanton. Notes on Burst Mitigation for Transport Protocols. SIGCOMM Comput. Commun. Rev., 35(2):5360, 2005.

[AOM02] J. Aweya, M. Ouellette, and D. Y. Montuno. A Self-regulating TCP Acknowledgement (ack) Pacing Scheme. Int. J. Netw. Manag., 12(3):145163, 2002.

[BPK97] Balakrishnan, H., V. Padmanabhan, and Katz, R., The Effects of Asymmetry on TCP Performance, Third ACM/IEEE Mobicom Conference, September 1997.

[BGG+07] D.K. Blandford, S.A. Goldman, S. Gorinsky, Y. Zhou, and D.R. Dooly. Smartacking: Improving TCP Performance from the Receiving End. Journal of Internet Engineering, 1(1), 2007.

[TJW00] I. Tam Ming-Chit, D. Jinsong and W. Wang. Improving TCP Performance Over Asymmetric Networks. ACM SIGCOMM Computer Communication Review, 30(3), July 2000.

[YMH03] L. Yu, Y. Minhua, and Z. Huimin. The Improvement of TCP Performance in Bandwidth Asymmetric Network. IEEE PIMRC, 1:482-486, September 2003.

Authors' Addresses

Sally Floyd
ICSI Center for Internet Research
1947 Center Street, Suite 600
Berkeley, CA 94704
USA

EMail: floyd <at> icir <dot> org

Andres Arcia
Networking, Security & Multimedia (RSM) Dpt.
GET / ENST Bretagne
Rue de la Chataigneraie, CS 17607
35576 Cesson Sevigne Cedex
France

Email: AE <dot> ARCIA <at> enst-bretagne <dot> fr

Janardhan R. Iyengar
Connecticut College
270 Mohegan Avenue
New London, CT 06320
USA

Email: iyengar <at> conncoll <dot> edu

David Ros
Networking, Security & Multimedia (RSM) Dpt.
GET / ENST Bretagne
Rue de la Chataigneraie, CS 17607
35576 Cesson Sevigne Cedex
France

Email: David <dot> Ros <at> enst-bretagne <dot> fr

Full Copyright Statement

Intellectual Property