S. Floyd
ICIR
A. Arcia
D. Ros
ENST Bretagne
J. Iyengar
Connecticut College
18 November 2007

Adding Acknowledgement Congestion Control to TCP
draft-floyd-tcpm-ackcc-02.txt

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. This acknowledgement congestion control mechanism is being proposed as an experimental mechanism for TCP for evaluation by the network community.

Table of Contents

TO BE DELETED BY THE RFC EDITOR UPON PUBLICATION:

Changes from draft-floyd-tcpm-ackcc-01.txt:

* Added a section on "Keep-alive Packets". From a question raised in the working group.

* Added a section on "Possible Complication: TCP Implementations that Skip ACK Packets". Motivated by reports at IETF that many high-bandwidth TCPs don't follow the MUST of sending an ACK for every other packet, if they don't have time.

* Added that receivers might have buffer limitations that require that they ack at least every K packets, for some K. Feedback from Sara Landstrom.

* Added to the discussion of "Possible Complication: Two-Way Traffic". Feedback from Sara Landstrom.

* Added a section on "Possible Complication: Router or Middlebox-based ACK Mechanisms". Feedback from Sara Landstrom.

* Added that SACK is required with ACK congestion control. Feedback from Sara Landstrom.

* Added a discussion of "Reducing the TCP Acknowledgment Frequency" to the related work section.

* Moved the Related Work section to the appendix. Feedback from Alfred Henes.

* General editing from feedback from Alfred Henes.

* Added an appendix on "Design Considerations", with a subsection on "The TCP ACK Ratio Option, or an AckNow bit in data packets?".

Changes from draft-floyd-tcpm-ackcc-00.txt:

* Added a discussion of environments where the reverse path is congested, but the TCP ACK traffic does not significantly contribute to that congestion. In this case, the goal is to minimize the negative impact of AckCC on TCP performance. Feedback from Armando Caro.

* In Section 5.7, added that when ABC is used with Aggregate Congestion Control, and rate-based pacing is also used, the sender MAY increase cwnd by more than 2 MSS. Feedback from Armando Caro.

* Added a section about measurements of ACK traffic and congestion. Feedback from Armando Caro.

* Added a section on the possibility of a TCP receiver-imposed lower bound on the ACK Ratio. Suggested by Mark Allman.

* Added to the discussion of the minimum ACK sending rate. Suggested by Mark Allman.

* Added a note that if the TCP receiver doesn't sent an ACK for every duplicate data packet, the sender's Fast Recovery procedure will have to be modified to take this into account. Feedback from Mark Allman.

* Added a discussion of evaluating ACK Congestion Control. From feedback from Mark Allman.

* Some general editing in response to feedback from Mark Allman.

END OF SECTION TO BE DELETED.

1. Introduction

This document adds an optional congestion control mechanism for acknowledgements (ACKs) to TCP. This mechanism is based on the acknowledgement congestion control in DCCP's CCID 2 [RFC4340], [RFC4341], which is a successor to the TCP acknowledgement congestion control mechanism proposed by Balakrishnan et at. in [BPK97].

In this document we use the terminology 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 uses an ACK Ratio R reported to it by the sender, 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.

The goal of this document is to explore a similar congestion control mechanism for acknowledgement traffic for TCP. The goal is for the TCP sender to monitor the packet drop rate for ACK packets, and to respond to a high ACK packet drop rate by instructing the receiver to reduce the sending rate for ACK packets. The assumption is that in some environments with congestion on the reverse path, reducing the sending rate for ACK traffic traversing the congested path can help to reduce the congestion itself, in turn reducing the packet drop rates for the ACK traffic. For those environments where the reverse path is congested but where TCP ACK traffic does not appreciably contribute to that aggregate congestion, the goal is for TCP's ACK congestion control to have a minimal negative effect on the performance of the TCP connection.

Adding 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 and Terminology

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]. MSS refers to the Maximum Segment Size.

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. Acknowledgement Congestion Control

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. Unlike some of the related work cited in the appendix, 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 either marked or dropped ACK packets.

4.1. 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.

4.2. 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.

4.3. The Receiver: 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.

If the receiver has buffer limitations, the receiver might have to acknowledge K packets, for some K less than the current ACK Ratio R. In this case, the sender could observe from the acknowledgements that the receiver is acknowledging less than R packets.

It is possible for there to be lost or marked ACK packets when there haven't yet been any lost or marked data packets. Thus, the sender could increase the ACK Ratio R even during the initial slow-start.

[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.

4.4. The Sender: 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. Because the use of SACK helps the sender in detecting lost ACK packets, the use of SACK is required with ACK congestion control.

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.

Out-of-order data 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 should be able to detect lost DupACK packets by counting the number of DupACKs that arrive 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). Simulations and/or experiments will be needed to determine whether, in practice, it works for the TCP sender to assess lost ACK packets during loss events, for an ACK Ratio of at most two.

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.

4.5. The Sender: 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 the following three constraints.

(1) The ACK Ratio is an integer.

(2) The minimum ACK sending rate: The ACK Ratio does not exceed max(2, cwnd/(K*MSS)), rounded up, for K=2. This ensures that the TCP receiver sends at least two ACKs for a window of data (for a window of at least four full-sized segments).

(3) If the congestion window is at least as large as four full-sized segments, then the ACK Ratio is at least two. In other words, an ACK Ratio of one is only allowed when the congestion window is at most three 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.) As stated in the previous section, when the ACK Ratio is greater than two the sender does not attempt to detect lost ACK packets during loss events for forward-path traffic.

For a constant congestion window, these modifications to the ACK ratio give 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 the ACK Ratio by one.

The frequency of ACK Ratio negotiations: 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 the ACK Ratio to one for half-connections with persistent congestion windows of 1 or 2 packets.

The minimum ACK sending rate: From rule (2) above, the TCP receiver always sends at least K=2 ACKs for a 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. One possibility would be to use a higher minimum ACK sending rate, adding a constant upper bound on the ACK Ratio. That is, if the ACK Ratio also had an upper bound of J, independent of cwnd, then the receiver would always send at least one ACK for every J data packets, regardless of the level of congestion on the reverse path.

4.6. The Receiver: 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. [We note that the Fast Recovery procedure of the TCP sender might have to be modified to take this change into account.] In addition, a receiver MUST NOT withhold an ACK for more than 500 ms.

4.7. The Sender: 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.

Byte counting at the sender: 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 during congestion avoidance 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 this form of byte counting 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. Because rate-based pacing should be used with Acknowledgement Congestion Control, as recommended earlier in this section, the TCP sender MAY increase the congestion window by more than two MSS for each ACK.

We note that for Appropriate Byte Counting (ABC) as specified in [RFC3465], during Slow-Start the sender is allowed to increase the congestion window by at most two MSS for each ACK. It has not yet been determined whether, with Acknowledgement Congestion Control, the TCP sender could use ABC during Slow-Start. If ABC is used with Acknowledgement Congestion Control, then when the TCP sender is in slow-start and the Ack Ratio is greater than two, the TCP sender MAY increase the congestion window by more that two MSS in response to a single ACK. Section 4.2 of [LL07] explores some of the issues with the use of ABC for TCP connections with an Ack Ratio greater than two.

Inferring lost data packets: 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, 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.

4.8. Possible Additions: Receiver Bounds on the Ack Ratio

It has been suggested that in some environments, the TCP receiver might want to set lower bounds on the ACK Ratio. For example, the TCP receiver might know from configuration or from past experience that the bandwidth on the return path is limited, and might want to set a lower bound (greater than two) on the ACK Ratio R. If this is included, this would require a TCP Option from the TCP receiver to the TCP sender reporting the lower bound on the ACK Ratio. Care would also be needed so that the lower bound on the ACK Ratio was only in effect when the TCP sender's congestion window was sufficiently high.

5. Possible Complications

5.1. Possible Complication: 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 acknowledgement. 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.

5.2. Possible Complication: Duplicate Acknowledgements.

As discussed in Section 4.3, 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 4.6 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.

5.3. Possible Complication: 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. 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.

In a connection with two-way traffic, the TCP sender would not always be able to infer when a pure ACK packet had been lost. For example, the receiver could send a pure ACK packet acknowledging packet K, and soon afterwards, the receiver could send a newly-generated data packet for the reverse-path flow also acknowledging packet K. The pure ACK packet could be dropped in the network, and the sender would not be able to detect this drop.

Fortunately, there are limitations to the potential problems caused by undetected ACK losses in two-way traffic. The sender will only fail to detect the loss of a pure ACK packet if the ACK packet was followed by a data packet with the same acknowledgement number. If the reverse-path traffic for the connection is dominated by data traffic, then the congestion control for the data traffic is more important than the congestion control for the pure ack traffic. If the reverse-path traffic is dominated by pure ACK traffic, then the sender would detect any losses of pure ACK packets followed by other pure ACK packets, and this would include most of the pure ACK packets for that connection. Thus, the sender's failure to detect the loss of a pure ACK packet followed by a data packet with the same acknowledgement number would not disable acknowledgement congestion control for a TCP connection with two-way traffic.

5.4. Possible Complication: 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].

5.5. Possible Complication: 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?

5.6. Possible Complication: 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.

One problem with the interaction of packet corruption and congestion control, for both data and ACK packets, is that it is not always obvious when the packet corruption is related to congestion, and when the packet corruption is independent of the level of congestion on the corrupting link. In environments where packet corruption is independent of the level of congestion on the corrupting link, applying ACK congestion control only makes the connection more sensitive to ACK packet corruption, by reducing the number of ACKs that are sent.

5.7. Possible Complication: ACKs That Don't Contribute to Congestion.

It is possible for the ACK packets in a TCP connection to traverse a congested path where ACK packets are dropped, but where the ACK packets themselves don't significantly contribute to the congestion on the path. In scenarios where ACK packets are dropped but where ACK traffic doesn't make a significant contribution of the congestion on the path, the use of ACK Congestion Control would not contribute to reducing the aggregate congestion on the path. In this case, one goal is to minimize the negative impact of ACK Congestion Control on the overall performance of the TCP connection.

```
       J TCP conns.            link L ->          J TCP conns.
         data ->      |---|                |---|   <- acks
      <-------------> |   |                |   | <------------->
                      |   | <------------->|   |
      <-------------> |   |                |   | <------------->
       K TCP conns.   |---|                |---|  K TCP conns.
        acks ->              <- link L1            <- data
```

A scenario with J forward and K reverse TCP connections.

To explore the relative contribution of ACK traffic on congestion, it is useful to consider a simple scenario with a congested unidirectional link L carrying data traffic from J TCP connections (the forward TCP connections) and ACK traffic from K TCP connections (the reverse TCP connections). We assume that all TCP connections have the same round-trip time R and the same data packet size S of 1500 bytes. We further assume that all of the forward TCP connections have the same data packet drop rate p and the same congestion window W, and that all of the reverse TCP connections have the same congestion window W1 and the same ACK packet drop rate p1. The J TCP connections each use a bandwidth on link L of 1500*W/R bytes per second, and the K TCP connections, without ACK Congestion Control, each use an bandwidth on link L of 40*(W1/2)/R bytes per second. This gives a ratio of 75*(J/K)*(W/W1) for TCP data bandwidth to TCP ACK bandwidth on link L. The ratio J/K is the ratio between the number of forward and reverse TCP connections on link L, and could have a wide range of values (e.g., large for an access link from a web server, and small for an access link to a web server). For this scenario, the ratio W/W1 is largely a function of the different levels of congestion on the forward and reverse paths.

To explore the possibilities, we will consider some of the range of congestion control mechanisms for the congested link. First, we consider scenarios where the limitation on the congested path is in the link bandwidth in bytes per second.

Cases (1), (2), (3), (5), and (7) below represent the best scenarios for ACK Congestion Control, where the fraction of packet drops for TCP ACK packets roughly matches the TCP ACK packets' contribution to congestion. [In several of these cases this is at best a rough match because the data packets are a factor in the bandwidth and in the queue limitations, while the TCP ACK packets are only a factor in the queue limitations.] Cases (4) and (8) below represent problematic scenarios where the fraction of packet drops for TCP ACK packets is much higher than the TCP ACK packets' contribution to congestion. Case (6) below represents scenarios where ACK Congestion Control would not be effective because it would not be invoked. In the scenarios in case (6), the fraction of packet drops for TCP ACK packets would be much smaller than the TCP ACK packets' contribution to congestion.

(1) The Drop-Tail queue for link L is measured in packets. In this case, the congested queue can accommodate N packets, regardless of packet size, there is a limitation of both bandwidth in bytes per second and also in queue space in packets, and large data packets and small TCP ACK packets should see similar packet drop rates. Although TCP ACK packets most likely aren't a major factor in the bandwidth limitation, they can be a significant contribution to the limitation of queue space. So, while the drop rate for ACK packets could be high in times of congestion, the ACK packets are contributing to that congestion somewhat by using scarce buffer space.

(2) The Drop-Tail queue is measured in bytes. In this case, the congested queue can accommodate M bytes of packets, and TCP ACK packets don't make a significant contribution to either the bandwidth limitation or to the limitation in queue space. It is also the case that in this scenario, even if there is heavy congestion, the drop rate for TCP ACK packets should be small (because small ACK packets can often find space on the congested queue when large data packets can't find space). In this case, ACK Congestion Control should not present any problems; the TCP ACK packets aren't contributing significantly to congestion, and aren't experiencing significant packet drop rates.

(3) The RED queue is in packet mode, and is measured in packets. This is similar to case (1) above. Because the queue is measured in packets, small TCP ACK packets contribute to the limitation in queue space, but not to the limitation in link bandwidth. Because the queue is in packet mode, large data packets and small TCP ACK packets

should see similar packet drop rates.

(4) The RED queue is in packet mode, but is measured in bytes. Because the queue is measured in bytes, small TCP ACK packets don't contribute significantly to either the limitation in queue space or to the limitation in link bandwidth. Because the queue is in packet mode, large data packets and small TCP ACK packets should see similar packet drop rates. If it existed, this case would be problematic, because the TCP ACK packets would not be contributing significantly to the congestion, but they would see a similar drop rate as the large data packets that are contributing to congestion.

(5) The RED queue is in byte mode, and is measured in bytes. This is similar to case (2) above. Because the queue is measured in bytes, small TCP ACK packets don't contribute significantly to either the limitation in queue space or to the limitation in link bandwidth. At the same time, because the queue is in byte mode, small TCP ACK packets see much smaller packet drop rates that those of large data packets.

(6) The RED queue is in byte mode, but is measured in packets. Because the queue is measured in packets, small TCP ACK packets contribute to the limitation in queue space, but not to the limitation in link bandwidth. Because the queue is in byte mode, small TCP ACK packets see much smaller packet drop rates that those of large data packets. If this case existed, TCP ACK packets would contribute somewhat to congestion, but would see a much smaller packet drop rate than that of large data packets.

Next, we consider scenarios where the limitation on the congested link is in CPU cycles at the router in packets per second, not in bandwidth in bytes per second.

(7) The CPU load imposed by TCP ACK packets is similar to the load imposed by other packets (e.g., TCP data packets). ACK Congestion Control would be useful in this scenario, particularly if TCP ACK packets saw the same packet drop rates as TCP data packets.

(8) The CPU load imposed by TCP ACK packets is much less than the load imposed by other packets (e.g., TCP data packets). If TCP ACK packets saw a smaller packet drop rate than TCP data packets, then the TCP ACK packet drop rate would roughly match the TCP ACK packets' contribution to congestion, and this would be good. If TCP ACK packets saw the same packet drop rate as TCP data packets, this this case would be problematic, because the TCP ACK packets would not be contributing significantly to the congestion, but they would see a similar drop rate as the large data packets that are contributing to congestion.

5.8. Possible Complication: TCP Implementations that Skip ACK Packets

It has been reported in IETF meetings that current TCP implementations do not always acknowledge at least every other data packet, as required by the TCP specifications. In particular, it has been reported that if a TCP receiver receives many data packets in a burst, before it is able to send an acknowledgement, then it might send a single acknowledgement for the burst of packets. We note that such a behavior would cause complications for a TCP connection that used ACK congestion control, as the sender would not be able to determine when an ACK packet had been dropped in the network, and when the packet had been skipped by the receiver because it was processing a burst of data packet arrivals.

One possibility for addressing this problem would be for TCP receivers using ACK congestion control to be required to send an acknowledgement for each R packets, for ACK Ratio R. A second possibility would be to define a TCP option or flag that the TCP receiver could use, when sending an ACK packet, to inform the sender that the TCP receiver ‘skipped’ some ACK packets, so that the sender should not infer ACK loss if some ACK packets seem to be missing.

5.9. Possible Complication: Router or Middlebox-based ACK Mechanisms

One possible complication would be the interaction of ACK Congestion Control with router-based or middlebox-based ACK mechanisms such as ACK filtering along the reverse path [BPK97]. We are not aware of the deployment of ACK filtering in the Internet, but any testing of ACK congestion control would have to look for interactions with any middlebox-based mechanisms regarding ACK packets.

5.10. Other Issues

Are there any problems caused by the combination of two-way traffic and reordering? Or other issues that have not yet been addressed?

6. Evaluating ACK Congestion Control

Evaluating ACK Congestion Control will have two components: (1) evaluating the effects of ACK Congestion Control on an individual TCP connection; and (2) evaluating the effects of ACK Congestion Control on aggregate traffic (including the effects of ACK Congestion Control on the aggregate congestion of the path).

The first part, evaluating ACK Congestion Control on the performance of an individual TCP connection, will have to examine those scenarios where ACK Congestion Control might help the performance of a TCP connection, and those scenarios where the use of ACK Congestion Control might cause problems.

The second part, evaluating the effects of ACK Congestion Control on aggregate traffic, should consider scenarios where the use of ACK Congestion Control helps all of the connections sharing a path by reducing the aggregate congestion on the path. This part should also see if there are scenarios where ACK Congestion Control causes problems by increasing the burstiness of aggregate traffic, or by otherwise changing traffic dynamics.

6.1. Keep-alive Packets

Some TCP hosts send keep-alive packets when no data or ACK packets have been received over a long period of time [KEEP-ALIVE]. This keep-alive mechanism is not addressed in TCP specifications. However, such keep-alive packets, if used, should not interact with ACK congestion control one way or another. For ACK congestion control, the ACK Ratio is set small enough to allow the receiver to send at least two ACKs for a window of data, In addition, the receiver uses a delayed ACK timer with the Ack Ratio, always sending an acknowledgement if the delayed ACK timer expires. Thus, ACK congestion control will never cause the receiver to delay indefinitely in sending an acknowledgement for a received data packet.

7. Measurements of ACK Traffic and Congestion

There are a number of studies about the traffic composition on various links in the Internet, reporting the fraction of bandwidth used by TCP data and by TCP ACK traffic. [Pointers to be added.]

Are there any studies that show the relative drop rates for TCP data and ACK traffic, for particular links or for particular TCP connections?

Are there any studies of congested links that show the fraction of traffic on the congested link, or in the congested queue, that consist of TCP ACK packets?

8. 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."

9. Security Considerations

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.

10. IANA Considerations

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

11. Conclusions

12. Acknowledgements

Many thanks for feedback from Mark Allman, Armando Caro, Alfred Henes, Sara Landstrom, and Michael Welzl, and for contributed text from Michael Welzl.

A. Appendix: Related Work

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.

Landstrom et al. in [LL07] explore a mechanism where the the receiver sends only four acknowledgements per window of data, along with the sender using a form of Appropriate Byte Counting. In addition, the receiver reverts to a lover acknowledgement frequency after a timeout, to avoid unnecessary Retransmit Timeouts. One conclusion of the paper is that pacing at the sender introduces an additional delay, and might not be necessary. A key result of the paper is that with the use of some form of byte counting at the sender, it is possible for TCP to use a lower acknowledgement frequency than that of delayed acknowledgements.

A.1. ECN-only Mechanisms

Balakrishnan et al. in [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.

A.2. Receiver-only Mechanisms

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.

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 a burst mitigation technique, Rate-Based Pacing, that paces the sending of data segments [AB05] [ASA00] [BPK97].

A.3. Middlebox-based Mechanisms

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.

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.

A. Appendix: Design Considerations

A.1. The TCP ACK Ratio Option, or an AckNow bit in data packets?

In the ACK congestion control mechanism specified in this document, the sender uses the TCP ACK Ratio Option to tell the receiver the ACK Ratio to use. An alternate approach to the TCP ACK Ratio Option could be for the sender to use an AckNow bit in the TCP header of data packets, telling the receiver to acknowledge this data packet. In the discussion below, we call these two approaches the TCP ACK Ratio Option approach and the AckNow approach.

An advantage of an AckNow approach is that it would require less state from the receiver; the receiver would not need to maintain a variable for the current Ack Ratio, and would not need to keep track of the number of data packets unacked to date.

However, a disadvantage of the AckNow approach is that the sender does not know when packets will be reordered, delayed, or dropped on the path to the receiver. In particular, the sender does not have control over whether a data packet with the AckNow bit set is reordered, delayed, or dropped in the network. For this reason, we have chosen the approach of the sender determining the Ack Ratio that should be used, and sending the Ack Ratio to the sender, rather than the sender telling the receiver exactly which data packets to acknowledge.

An additional disadvantage of the AckNow approach is that it would add complications and add difficulties for the default cases of the receiver using an Ack Ratio of one or two, as is done in the absence of ACK congestion control.

For these reasons, we have specified that the sender determines the ACK Ratio to use and tells the sender, rather than the receiver setting an AckNow bit in the TCP Header of selected data packets.

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.

[KEEP-ALIVE] Fabio Busatto, TCP Keepalive HOWTO, May 2007, URL "http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/index.html".

[LL07] Landstrom, S., and Larzon, L.-A., Reducing the TCP Acknowledgement Frequency, CCR, July 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@icir.org

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

Email: AE.ARCIA <at> enst-bretagne.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