Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-dccp-ccid2-02.ps
Sally Floyd
Eddie Kohler
ICIR
10 May 2003
Expires: November 2003

Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control

Status of this Document

Abstract

This document contains the profile for Congestion Control Identifier 2, TCP-like Congestion Control, in the Datagram Congestion Control Protocol (DCCP) [DCCP]. DCCP implements a congestion-controlled, unreliable flow of datagrams suitable for use by applications such as streaming media. The TCP-like Congestion Control CCID is used by senders who are able to adapt to the abrupt changes in the congestion window typical of TCP's AIMD (Additive Increase Multiplicative Decrease) congestion control. TCP-like Congestion Control is particularly useful for senders who would like to take advantage of the available bandwidth in an environment with rapidly changing conditions.

TO BE DELETED BY THE RFC EDITOR UPON PUBLICATION:

Changes from draft-ietf-dccp-ccid2-01.txt:

* Added "Security Considerations" and "IANA Considerations" sections.

* Refer explicitly to SACK-based TCP, and flesh out Section 3 ("Congestion Control on Data Packets").

* When cwnd < ssthresh, increase cwnd by one per newly acknowledged packet up to some limit, in line with TCP Appropriate Byte Counting.

* Refined definition of quiescence.

Changes from draft-ietf-dccp-ccid2-00.txt:

* Said that the Acknowledgement Number reports the largest sequence number, not the most recent packet, for consistency with draft-ietf-dccp-spec.

* Added notes about ECN nonces for acknowledgements, and about dealing with piggybacked acknowledgements.

Table of Contents

1. Introduction

This document contains the profile for Congestion Control Identifier 2, TCP-like Congestion Control, in the Datagram Congestion Control Protocol (DCCP). DCCP uses Congestion Control Identifiers, or CCIDs, to specify the congestion control mechanism in use on a half-connection. (A half-connection might consist of data packets sent from DCCP A to DCCP B, plus acknowledgements sent from DCCP B to DCCP A. DCCP A is the HC-Sender, and DCCP B the HC-Receiver, for this half-connection. In this document, we abbreviate HC-Sender and HC-Receiver as "sender" and "receiver", respectively. These terms are defined more fully in [DCCP].)

The TCP-like Congestion Control CCID sends data using a close variant of TCP's congestion control mechanisms, particularly SACK-based TCP's congestion control mechanisms [RFC 3517]. It is suitable for senders who can adapt to the abrupt changes in congestion window typical of AIMD (Additive Increase Multiplicative Decrease) congestion control in TCP, and particularly useful for senders who would like to take advantage of the available bandwidth in an environment with rapidly changing conditions.

The congestion control mechanisms described here closely follow mechanisms standardized by the IETF for use in SACK-based TCP. We do not define these mechanisms anew; instead, we rely on existing TCP documentation, such as [RFC 793], [RFC 3465], and [RFC 3517]. This is both to avoid respecifying TCP, and to allow our specification to track TCP as it evolves. Conformant CCID 2 implementations MAY track TCP's evolution directly, as updates are standardized in the IETF, rather than waiting for revisions of this document. CCID 2 does define an additional mechanism not currently standardized for use in TCP, namely congestion control on acknowledgements as achieved by the Ack Ratio. Also, DCCP is a datagram protocol, so several parameters whose units are bytes in TCP, such as the congestion window cwnd, have units of packets in DCCP. Unreliability also leads to differences from TCP: DCCP never retransmits a packet, so congestion control mechanisms that distinguish retransmissions from new packets need rethinking in the DCCP context.

For simplicity, we refer to DCCP-Data packets sent by the sender, and DCCP-Ack packets sent by the receiver. Both of these categories are meant to include piggybacked DCCP-DataAck packets.

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 [RFC 2119].

1.1. Usage Scenario

TCP-like Congestion Control is intended to provide congestion control for applications that do not require fully reliable data transmission, or that desire to implement reliability on top of DCCP. It is appropriate for flows that would like to receive as much bandwidth as possible over the long term, consistent with the use of end-to-end congestion control, and that are willing to undergo halving of the congestion window in response to a congestion event.

1.2. Example Half-Connection

This example shows the typical progress of a half-connection using TCP-like Congestion Control specified by CCID 2, not including connection initiation and termination. Again, the "sender" is the HC-Sender, and the "receiver" is the HC-Receiver. (The example is informative, not normative.)

(1) The sender sends DCCP-Data packets, where the number of packets sent is governed by a congestion window, cwnd, as in TCP. Each DCCP-Data packet uses a sequence number. The sender also sends an Ack Ratio feature option specifying the number of data packets to be covered by an Ack packet from the receiver.

Assuming that the half-connection is ECN capable (the ECN Capable feature is turned on---the default), each DCCP-Data packet is sent as ECN-Capable with either the ECT(0) or the ECT(1) codepoint set, as described in [ECN NONCE].

(2) The receiver sends a DCCP-Ack packet acknowledging the data packets for every Ack Ratio data packets transmitted by the sender. Each DCCP-Ack packet uses a sequence number and contains an Ack Vector. The sequence number acknowledged in DCCP-Ack packets is that of the received packet with the highest sequence number, rather than a TCP-like cumulative acknowledgement.

If the half-connection is ECN capable, the receiver returns the sum of received ECN Nonces via Ack Vector options, allowing the sender to probabilistically verify that the receiver is not misbehaving. DCCP-Ack packets from the receiver are also sent as ECN-Capable, but there is no need to verify the nonces.

(3) The sender continues sending DCCP-Data packets as controlled by the congestion window. Upon receiving DCCP-Ack packets, the sender examines their Ack Vectors to learn about marked or dropped data packets, and adjusts its congestion window accordingly. Because this is unreliable transfer, the sender does not retransmit dropped packets.

(4) Because DCCP-Ack packets use sequence numbers, the sender has direct information about the fraction of lost or marked DCCP-Ack packets. The sender responds to lost or marked DCCP-Ack packets by modifying the Ack Ratio sent to the receiver.

(5) The sender acknowledges the receiver's acknowledgements at least once per congestion window. If both half-connections are active, the sender's acknowledgement of the receiver's acknowledgements is included in the sender's acknowledgement of the receiver's data packets. If the reverse-path half-connection is quiescent, the sender sends a DCCP-DataAck packet that includes an Acknowledgement Number in the header.

(6) The sender estimates round-trip times and calculates a TimeOut (TO) value much as the RTO (Retransmit Timeout) is calculated in TCP. The TO is used to determine when a new DCCP-Data packet can be transmitted when the sender has been limited by the congestion window and no feedback has been received from the receiver.

2. Connection Establishment

Use of the Ack Vector is MANDATORY on CCID 2 half-connections, so the sender MUST send a "Change(Use Ack Vector, 1)" option to the receiver as part of connection establishment. The sender SHOULD NOT send data until it has received the corresponding "Confirm(Use Ack Vector, 1)" from the receiver.

3. Congestion Control on Data Packets

CCID 2's congestion control mechanisms are based on those for SACK-based TCP [RFC 3517]. In particular, the Ack Vector provides strictly more information than that transmitted in SACK options.

In particular, a CCID 2 data sender maintains three integer parameters, whose units are in packets:

(1) The congestion window "cwnd", which equals the maximum number of data-carrying packets allowed in the network at any time. ("Data-carrying packet" means any DCCP packet that contains user data: DCCP-Data, DCCP-DataAck, and occasionally DCCP-Request, DCCP-Response, and DCCP-Move.)

(2) The slow-start threshold "ssthresh", which controls adjustments to cwnd.

When halved, cwnd and ssthresh have their values rounded down, except that neither parameter is ever less than one.

(3) The pipe value "pipe", which is the sender's estimate of the number of data-carrying packets outstanding in the network.

These parameters are manipulated, and their initial values determined, according to SACK-based TCP's behavior. The rest of this section provides more specific guidance.

The sender MAY send a data-carrying packet only when pipe < cwnd. In particular, it MUST NOT send a data-carrying packet when pipe >= cwnd. Every data-carrying packet sent increases pipe by 1.

The sender reduces pipe as it infers that data-carrying packets have left the network, either by being received or by being dropped. In particular:

(1) The sender reduces pipe by 1 for each packet newly-acknowledged as received (Ack Vector State 0 or State 1) by some DCCP-Ack.

(2) The sender reduces pipe by 1 for each packet it can infer as lost due to the DCCP equivalent of "duplicate acknowledgements". This depends on TCP's NUMDUPACK parameter, the number of duplicate acknowledgements TCP needs to infer a loss, which currently equals 3. A packet P is inferred to be lost, rather than delayed, when at least NUMDUPACK packets after P have been acknowledged as received (Ack Vector State 0 or 1) by the receiver.

(3) Finally, the sender needs "retransmit" timeouts, handled like TCP's retransmission timeouts, in case an entire window of packets are lost. The sender estimates the round-trip time at most once per window of data, and uses the TCP algorithms for maintaining the average round-trip time, mean deviation, and timeout value. Because DCCP does not retransmit data, DCCP does not require TCP's recommended minimum timeout of one second. The exponential backoff of the timer is exactly as in TCP.

When a "retransmit" timeout occurs, the sender sets pipe to 0.

The sender MUST NOT decrement pipe more than once for any given packet. Duplicate acknowledgements, for example, MUST not affect pipe. Furthermore, the sender MUST NOT decrement pipe for non-data packets, such as DCCP-Acks, even though the Ack Vector will contain information about them.

Congestion events, namely one or more packets lost or marked from a window of data, cause CCID 2 to reduce its congestion window. For each congestion event, either indicated explicitly as an Ack Vector State 1 (ECN-marked) acknowledgement or inferred via "duplicate acknowledgements", cwnd is halved, then ssthresh is set to the new cwnd. Cwnd is never reduced below one packet. After a timeout, the slow-start threshold is set to cwnd/2, then cwnd is set to one packet.

When cwnd < ssthresh, meaning that the sender is in slow-start, the congestion window is increased by one packet for every newly acknowledged (with Ack Vector State 0 or 1) data-carrying packet, up to a maximum of Ack Ratio packets per acknowledgement. This differs from TCP's historical behavior, which (in DCCP terms) would increase cwnd by one per DCCP-Ack received, not by one per packet newly acknowledged by some DCCP-Ack; but it is in line with TCP's behavior with appropriate byte counting [RFC 3465]. When cwnd >= ssthresh, the congestion window is increased by one packet for every window of data acknowledged without lost or marked packets.

4. Acknowledgements

This section describes how the receiver reports acknowledgement information back to the sender. DCCP-Ack packets from the receiver MUST include Ack Vector options, as well as an Acknowledgement Number acknowledging the packet with the largest valid sequence number received from the sender. Acknowledgement data in the Ack Vector options SHOULD generally cover the receiver's entire Unacknowledged Window, as described in [DCCP].

The sender specifies the Ack Ratio to be used by the receiver. In the absence of congestion on the reverse path, the Ack Ratio is set to two if the congestion window is three or more packets, and is set to one otherwise. The receiver sends a DCCP-Ack packet for every Ack Ratio packets sent by the sender.

4.1. Congestion Control on Acknowledgements

In CCID 2, the acknowledgement subflow is loosely congestion-controlled by the Ack Ratio specified by the sender. The receiver sends (cwnd / Ack Ratio) acknowledgement packets for each congestion window of data packets. We note that CCID 2 differs from TCP, which presently has no congestion control for pure acknowledgement traffic. For congestion control for the pure ack stream, DCCP does not try to be TCP-friendly, but just tries to avoid congestion collapse, and to be somewhat better than TCP in explicitly reducing the ack sending rate in the presence of a high packet loss or marking rate on the return path.

If DCCP B, the HC-Receiver, is actively sending data---it is not quiescent---then required acknowledgements may be piggybacked on DCCP B's data packets. In this situation, DCCP B MAY send more piggybacked acknowledgements than the Ack Ratio would allow; but it MUST send at least as many acknowledgements as the Ack Ratio requires. Conceivably, the CCID in use for the B-to-A half-connection might limit DCCP B's sending rate to less than the acknowledgement rate required for the A-to-B half-connection. DCCP B MUST follow both constraints. In practice, this means that DCCP B will not piggyback data on every acknowledgement.

There are three constraints on the Ack Ratio. First, it is always an integer. Second, it is never greater than half the congestion window (with fractions rounded up). Third, it is at least two for a congestion window of four or more packets.

DCCP-Ack packets from the receiver contain sequence numbers, so the sender can infer when DCCP-Ack packets are lost. The sender considers a DCCP-Ack packet lost if at least NUMDUPACK packets with higher sequence numbers have been received from the receiver. (Again, NUMDUPACK equals 3.) If DCCP-Ack packets from the receiver are marked in the network, the sender sees these marks directly.

DCCP responds to congestion events on the return path by modifying the Ack Ratio, loosely emulating TCP. For each congestion window of data with lost or marked DCCP-Ack packets, the Ack Ratio is doubled, subject to the constraints noted above. Similarly, if the Ack Ratio is R, then for each (cwnd/(R^2 - R)) congestion windows of data with no lost or marked DCCP-Ack packets, the Ack Ratio is decreased by 1, again subject to the constraints on the Ack Ratio. See the section below for the derivation. For a constant congestion window, this gives an Ack sending rate that is roughly TCP-friendly. We note that, because the sending rate for the acknowledgement packets changes as a function of both the Ack Ratio and the congestion window, the dynamics will be rather complex, and this Ack congestion control mechanism is intended only to be very roughly TCP-friendly.

As a result of the constraints given earlier in this section, the receiver always sends at least one ack packet for a congestion window of one packet, and the receiver always sends at least two ack packets per window of data otherwise. Thus, 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 that all of the ack packets are dropped, then the sender falls back on a timeout, and the exponential backoff of the timer, as in TCP. 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.

4.1.1. Derivation of Ack Ratio Decrease

The congestion avoidance phase of TCP increases cwnd by one MSS for every congestion-free window. Applying this congestion avoidance behavior to the ack traffic, this would correspond to increasing the number of DCCP-Ack packets per window by one after every congestion-free window of DCCP-Ack packets. We cannot achieve this exactly using the Ack Ratio, since the Ack Ratio is an integer. Instead, we must decrease the Ack Ratio by one after K windows have been sent without a congestion event on the reverse path, where K is chosen so that the long-term number of DCCP-Ack packets per congestion window is roughly TCP-friendly, following AIMD congestion control.

In CCID 2, K = (cwnd/(R^2 - R)), where R is the current Ack Ratio. This result was calculated as follows:

```
          R = Ack Ratio = # data packets / ack packets, and
          W = Congestion Window = # data packets / window, so
        W/R = # ack packets / window.

Requirement: Increase W/R by 1 per congestion-free window.
But can only reduce R by increments of one.

Therefore, find K so that, after K congestion-free windows,
the adjusted W/R would equal W/(R-1).

(W/R) + K = W/(R-1), so
        K = W/(R-1) - W/R = W/(R^2 - R).
```

4.2. Quiescence

This section refers to quiescence in the DCCP sense (see section 8.1 of [DCCP]): How does a CCID 2 receiver determine that the corresponding sender is not sending any data?

Let T equal the greater of 0.2 seconds and two round-trip times. Then the receiver detects that the sender has gone quiescent when at least T seconds have passed without receiving any additional data from the sender, and the sender has acknowledged receiver Ack Vectors that covered all data packets sent. That is, once the sender acknowledges the receiver's

Ack Vectors and the sender has not sent additional data for at least T, the receiver can determine that the sender is quiescent.

4.3. Acknowledgements of Acknowledgements

The sender, DCCP A, must occasionally acknowledge the receiver's acknowledgements, so that the receiver can free up Ack Vector state. The sender can also send acknowledgements to make changes to the Ack Ratio. We assume that DCCP A simply sends Change(Ack Ratio) options whenever required. To let the receiver free Ack Vector state, DCCP A must occasionally acknowledge that it has received one of DCCP B's acknowledgements. When both half-connections are active, this information is automatically contained in A's acknowledgements to B's data. If the B-to-A half-connection goes quiescent, however, DCCP A must do it proactively.

In particular, an active sender MUST occasionally acknowledge the receiver's acknowledgements, probably by encapsulating a datagram in a DCCP-DataAck packet. No acknowledgement options are necessary, just the relevant Acknowledgement Number in the DCCP-DataAck header. The sender SHOULD acknowledge approximately one of the receiver's acknowledgements per congestion window. Of course, the sender's application might fall silent. This is no problem; when neither side is sending data, a sender can wait arbitrarily long before sending an ack.

5. Explicit Congestion Notification

ECN may be used with CCID 2. If ECN is used, then the ECN Nonce will automatically be used for the data packets, following the specification for the ECN Nonce in TCP in [ECN NONCE]. For the data subflow, the sender sets either the ECT(0) or ECT(1) codepoint on DCCP-Data packets. Information about marked packets is returned in the Ack Vector. Because the information in the Ack Vector is reliably transferred, DCCP does not need the TCP flags of ECN-Echo and Congestion Window Reduced.

For unmarked data packets, the receiver computes the ECN Nonce Echo as in [ECN NONCE], and returns the ECN Nonce Echo in DCCP-Ack packets. The sender uses the ECN Nonce to protect against the accidental or malicious concealment of marked packets.

Because the ack subflow is congestion-controlled, ECN can also be used for DCCP-Ack packets. In this case we do not make use of the ECN Nonce, because it would not be easy to provide protection against the concealment of marked ack packets by the sender, and because the sender does not have as much motivation for lying about the mark rate on acknowledgements.

6. Relevant Options and Features

DCCP's Ack Vector option and its Ack Ratio, Use Ack Vector, and ECN Capable features are relevant for CCID 2.

7. Application Requirements

There are no specific application requirements for TCP-like Congestion Control.

8. Thanks

We thank Mark Handley and Jitendra Padhye for their help in defining CCID 2.

9. Normative References

[DCCP] E. Kohler, M. Handley, S. Floyd, and J. Padhye. Datagram Congestion Control Protocol, draft-ietf-dccp-spec-01.txt, work in progress, March 2003.

[ECN NONCE] Neil Spring, David Wetherall, and David Ely. Robust ECN Signaling with Nonces, draft-ietf-tsvwg-tcp-nonce-04.txt, work in progress, October 2002.

[RFC 793] J. Postel, editor. Transmission Control Protocol. RFC 793.

[RFC 2026] S. Bradner. The Internet Standards Process -- Revision 3. RFC 2026.

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

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

[RFC 3465] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465.

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

10. Informative References

11. Security Considerations

Security considerations for DCCP have been discussed in [DCCP], and security considerations for TCP have been discussed in [RFC 2581].

[RFC 2581] discusses ways that an attacker could impair the performance of a TCP connection by dropping packets, or by forging extra duplicate acknowledgements or acknowledgements for new data. We are not aware of any new security considerations created by this document in its use of TCP-like congestion control.

12. IANA Considerations

There are no new IANA considerations created in this document.

13. Authors' Addresses

Sally Floyd <floyd@icir.org>
Eddie Kohler <kohler@icir.org>

ICSI Center for Internet Research,
1947 Center Street, Suite 600
Berkeley, CA 94704.