Internet Engineering Task Force
INTERNET-DRAFT
draft-floyd-dcp-ccid2-03.txt

Sally Floyd
Eddie Kohler
ICIR

24 May 2002
Expires: November 2002

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 the AIMD (Additive Increase Multiplicative Decrease) congestion control in TCP. 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.

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.)

The TCP-like Congestion Control CCID sends data using a close variant of TCP's congestion control mechanisms. It is suitable for senders who can adapt to the abrupt changes in the 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 TCP. We do not define these mechanisms anew; instead, we rely on existing TCP documentation. This is both to avoid respecifying TCP, and to allow our specification to track TCP as it evolves. Conformant CCID 2 implementations may actually 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.

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.

1.1. Usage Scenario

TCP-like Congestion Control is intended to provide congestion control for the flow of data packets from the server to the client for applications that do not require fully reliable data transmission, or that desire to implement reliability on top of DCCP. TCP-like Congestion Control 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 the halving of the congestion window in response to a congestion event.

1.2. Example Half-Connection

This example, taken from the main DCCP draft, is of a half-connection using TCP-like Congestion Control specified by CCID 2. Again, the "sender" is the HC-Sender, and the "receiver" is the HC-Receiver.

(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.

(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. Because DCCP does not use reliable transfer, the DCCP-ACK packet does not have a Cumulative Acknowledgement field.

(3) The sender continues sending DCCP-Data packets as controlled by the congestion window. Upon receiving DCCP-Ack packets, the sender examines the Ack Vector 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 loss 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.

(7) 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 DRAFT]. For DCCP-Data packets from the sender, the receiver returns the ECN Nonce in the DCCP-Ack packet. The DCCP-Ack packets from the receiver are sent as ECN-Capable with ECT(0). For DCCP-Ack packets from the receiver, the sender observes directly if the CE codepoint is set in the received DCCP-Ack packet.

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

The data sender uses the congestion window cwnd to control the sending of packets, and uses the slow-start threshold ssthresh to control adjustments to cwnd. These integer parameters have units measured in packets. When halved, their values are rounded down, except that neither parameter is ever less than one. The cwnd and ssthresh variables are modified as in TCP. The initial window is determined using the specification for TCP. The equivalent of a TCP MSS is simply one packet.

The sender uses the information in Ack Vectors to infer a lost packet. Ack Vectors explicitly declare which packets have not yet been received. One of these packets, P, is inferred to be lost (rather than delayed) when at least NUMDUPACK packets after packet P have been acknowledged by the receiver. The NUMDUPACK parameter equals 3, the number of duplicate acknowledgements TCP requires to infer a loss. A congestion event is defined as one or more packets lost or marked from a window of data. For each congestion event, cwnd is halved, then ssthresh is set to the new cwnd. Cwnd is never reduced below one packet.

When cwnd < ssthresh, meaning that the sender is in slow-start, the congestion window is increased by one packet for every DCCP-Ack packet received acknowledging a new DCCP-Data packet from the sender. Note that cwnd is increased by one per DCCP-Ack received, not by one per packet acknowledged by the DCCP-Ack; this follows TCP's behavior. When cwnd >= ssthresh, the congestion window is

increased by one packet for every window of data acknowledged without lost or marked packets.

If all of the data packets from a window of data are lost, the sender needs timeouts to know when to send a new data packet. 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. After a timeout, the slow-start threshold is set to cwnd/2, then cwnd is set to one packet, and a new packet is transmitted (thus using up cwnd). The exponential backoff of the timer is used exactly as in TCP.

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 most recent packet received from the sender. Acknowledgement data in the Ack Vector options SHOULD generally cover the receiver's entire Unacknowledged Window, as described in the DCCP draft.

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 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 terms of reducing the ack sending rate in the presence of a high packet loss or marking rate on the return path.

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 6.1 of [DCCP]): How does a CCID 2 receiver determine that the corresponding sender is not sending any data?

The receiver detects that the sender has gone quiescent after two of its Ack Vectors are acknowledged without receiving any additional data. That is, once the sender acknowledges two of the receiver's Ack Vectors without sending additional data, 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 manages the Ack Ratio proactively, sending 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, the sender must acknowledge at least one of the receiver's acknowledgements per congestion window, probably by sending a DCCP-DataAck packet for the next datagram it sends. No acknowledgement options are necessary, just the relevant Acknowledgement Number in the DCCP-DataAck header. Of course, the sender's application might fall silent before DCCP A can send an

ack. This is no problem; A can wait arbitrarily long before sending the 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 [SWE01]. 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 as in [SWE01], and returns the ECN Nonce 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 use the ECN Nonce, because it would not be easy to provide protection against the concealment of marked ack packets by the sender.

6. Relevant Options and Features

DCCP's Ack Vector option and Ack Ratio and Use Ack Vector 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. References

[DCCP] Eddie Kohler, Mark Handley, Sally Floyd, and Jitendra Padhye. Datagram Congestion Control Protocol (DCCP). Work in progress.

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

[RFC 2861] M. Handley, J. Padhye, and S. Floyd. TCP Congestion Window Validation. RFC 2861.

[SWE01] Neil Spring, David Wetherall, and David Ely. Robust ECN Signaling with Nonces. draft-ietf-tsvwg-tcp-nonce-02.txt, work in progress, October 2001.

10. 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.