Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-dccp-ccid2-09.ps
Expires: 7 September 2005

Sally Floyd
ICIR
Eddie Kohler
UCLA
7 March 2005

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

Status of this Memo

Copyright Notice

Abstract

This document contains the profile for Congestion Control Identifier 2, TCP-like Congestion Control, in the Datagram Congestion Control Protocol (DCCP). CCID 2 should be used by senders who would like to take advantage of the available bandwidth in an environment with rapidly changing conditions, and who are able to adapt to the abrupt changes in the congestion window typical of TCP's Additive Increase Multiplicative Decrease (AIMD) congestion control.

TO BE DELETED BY THE RFC EDITOR UPON PUBLICATION:

Changes from draft-ietf-dccp-ccid2-07.txt:

* Restrict the use of byte-counting to be at most as aggressive as the current TCP (without byte-counting).

Changes from draft-ietf-dccp-ccid2-06.txt:

* Moved three citations to Informational.

* Added that "The sender SHOULD not attempt Ack Ratio renegotiations more than once per round-trip time."

* Specified that ssthresh is never less than two, instead of one.

* Added references to RFC 2988 and RFC 2018.

* Specify that the congestion window is only increased for packets that aren't ECN-marked.

Changes from draft-ietf-dccp-ccid2-05.txt:

* Changes to the discussion about how the sender infers that DCCP-Ack packets are lost. The sender does not know for sure whether a missing sequence number is for a dropped ACK packet or a dropped data packet. Our changes include a new appendix on "The Costs of Inferring Lost Ack Packets".

* Minor editing for clarity, including some reordering of sections.

* Added a section on response to idle and application-limited periods.

* Clarifications on changing the Ack Ratio, based on feedback from Nils-Erik Mattsson.

Changes from draft-ietf-dccp-ccid2-04.txt:

* Minor editing, as follows:
  - Added a note that CCID2 implementations MAY check for apps that are gaming with regard to the packet size.
  - Deleted a statement that the maximum packet size is 1500 bytes.
  - Added that the receiver MAY know the round-trip time from its role as
  - Added a note that the initial cwnd is up to four packets.

* Added Intellectual Property Notice.

Changes from draft-ietf-dccp-ccid2-03.txt:

* Disallow direct tracking of TCP standards.

Changes from draft-ietf-dccp-ccid2-02.txt:

* Added to the section on application requirements.

* Changed the default Ack Ratio to be two, as recommended for TCP.

* Added a paragraph about packet sizes.

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]. DCCP uses Congestion Control Identifiers, or CCIDs, to specify the congestion control mechanism in use on a half-connection.

The TCP-like Congestion Control CCID sends data using a close variant of TCP's congestion control mechanisms, incorporating selective acknowledgements (SACK) [RFC 2018, RFC 3517]. CCID 2 is suitable for senders who can adapt to the abrupt changes in congestion window typical of TCP's Additive Increase Multiplicative Decrease (AIMD) congestion control, and particularly useful for senders who would like to take advantage of the available bandwidth in an environment with rapidly changing conditions. See Section 3 for more on application requirements.

2. Conventions and Notation

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.

A DCCP half-connection consists of the application data sent by one endpoint and the corresponding acknowledgements sent by the other endpoint. The terms "HC-Sender" and "HC-Receiver" denote the endpoints sending application data and acknowledgements, respectively. Since CCIDs apply at the level of half-connections, we abbreviate HC-Sender to "sender" and HC-Receiver to "receiver" in this document. See [DCCP] for more discussion.

For simplicity, we say that senders send DCCP-Data packets and receivers send DCCP-Ack packets. Both of these categories are meant to include DCCP-DataAck packets.

The phrases "ECN-marked" and "marked" refer to packets marked ECN Congestion Experienced unless otherwise noted.

3. Usage

CCID 2, TCP-like Congestion Control, is appropriate for DCCP 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 can tolerate the large sending rate variations characteristic of AIMD congestion control, including halving of the congestion window in response to a congestion event.

Applications that simply need to transfer as much data as possible in as short a time as possible should use CCID 2. This contrasts with CCID 3, TCP-Friendly Rate Control (TFRC) Congestion Control [CCID 3 PROFILE], which is appropriate for flows that would prefer to minimize abrupt changes in the sending rate. For example, CCID 2 is recommended over CCID 3 for streaming media applications that buffer a considerable amount of data at the application receiver before playback time, insulating the application somewhat from abrupt changes in the sending rate. Such applications could easily choose DCCP's CCID 2 over TCP itself, possibly adding some form of selective reliability at the application layer. CCID 2 is also recommended over CCID 3 for applications where halving the sending rate in response to congestion is not likely to interfere with application-level performance.

An additional advantage of CCID 2 is that its TCP-like congestion control mechanisms are reasonably well-understood, with traffic dynamics quite similar to those of TCP. While the network research community is still learning about the dynamics of TCP after 15 years of its being the dominant transport protocol in the Internet, some applications might prefer the more well-known dynamics of TCP-like congestion control over that of newer congestion control mechanisms, which haven't yet met the test of widespread Internet deployment.

3.1. Relationship with TCP

The congestion control mechanisms described here closely follow mechanisms standardized by the IETF for use in SACK-based TCP, and we rely partially on existing TCP documentation, such as RFC 793, RFC 2581, RFC 3465, and RFC 3517. TCP congestion control continues to evolve, but CCID 2 implementations SHOULD wait for explicit updates to CCID 2 rather than track TCP's evolution directly. The differences between CCID 2 and straight TCP include: CCID 2 applies congestion control to acknowledgements, a mechanism not currently standardized for use in TCP. DCCP is a datagram protocol, so several parameters whose units are specified in 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 have been redesigned for the DCCP context.

3.2. Example Half-Connection

This example shows the typical progress of a half-connection using CCID 2's TCP-like Congestion Control, not including connection initiation and termination. 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; Ack Ratio defaults to two. The DCCP header's CCVal field is set to zero.

   Assuming that the half-connection is Explicit Congestion Notification (ECN) capable (the ECN Incapable feature is zero -- 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 RFC 3540.

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 a DCCP-Ack packet is that of the received packet with the highest sequence number, rather than a TCP-like cumulative acknowledgement.

   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, since the sender will control the acknowledgement rate in a roughly TCP-friendly way using the Ack Ratio feature. There is little need for the receiver to verify the nonces of its DCCP-Ack packets, since the sender cannot get significant benefit from misreporting the ack mark rate.

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 some information about 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, either through keeping track of acknowledgement round-trip times as TCP does or through explicit Timestamp options, 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.

4. Connection Establishment

Use of the Ack Vector is MANDATORY on CCID 2 half-connections, so the sender MUST send a "Change R(Send 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 L(Send Ack Vector, 1)" from the receiver, except possibly for data included on the initial DCCP-Request packet.

5. Congestion Control on Data Packets

CCID 2's congestion control mechanisms are based on those for SACK-based TCP [RFC 3517], since the Ack Vector provides all the information that might be transmitted in SACK options.

A CCID 2 data sender maintains three integer parameters measured in packets.

1. The congestion window "cwnd", which equals the maximum number of data packets allowed in the network at any time. ("Data packet" means any DCCP packet that contains user data: DCCP-Data, DCCP-DataAck, and occasionally DCCP-Request and DCCP-Response.)
2. The slow-start threshold "ssthresh", which controls adjustments to cwnd.
3. The pipe value "pipe", which is the sender's estimate of the number of data packets outstanding in the network.

These parameters are manipulated, and their initial values determined, according to SACK-based TCP's behavior, except that they are measured in packets, not bytes. The rest of this section provides more specific guidance.

The sender MAY send a data packet when pipe < cwnd, but MUST NOT send a data packet when pipe >= cwnd. Every data packet sent increases pipe by 1.

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

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

2. **Dropped data packets.** The sender reduces pipe by 1 for each data packet it can infer as lost due to the DCCP equivalent of TCP's "duplicate acknowledgements". This depends on the NUMDUPACK parameter, the number of duplicate acknowledgements needed to infer a loss. The NUMDUPACK parameter is set to three, as is currently the case in TCP. A packet P is inferred to be lost, rather than delayed, when at least NUMDUPACK packets transmitted after P have been acknowledged as received (Ack Vector State 0 or 1) by the receiver. Note that the acknowledged packets following the hole may be DCCP-Acks or other non-data packets.
3. **Transmit timeouts.** Finally, the sender needs transmit timeouts, handled like TCP's retransmission timeouts, in case an entire window of packets is 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 [RFC 2988]. (If more than one measurement per round-trip time was used for these calculations, then the weights of the averagers would have to be adjusted, so that the average round-trip time is effectively derived from measurements over multiple round-trip times.) 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 transmit timeout occurs, the sender sets pipe to zero. The adjustments to cwnd and ssthresh are described below.

The sender MUST NOT decrement pipe more than once per data packet. True 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 cause CCID 2 to reduce its congestion window. A congestion event contains at least one lost or marked packet. As in TCP, two losses or marks are considered to be part of a single congestion event when the second packet was sent before the loss or mark of the first packet was detected. As an approximation, a sender can consider two losses or marks to be part of a single congestion event when the packets were sent within one RTT estimate of one another, using an RTT estimate current at the time the packets were sent. 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 halved, cwnd and ssthresh have their values rounded down, except that cwnd is never less than one and ssthresh is never less than two.

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 [RFC 3465] 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. The cwnd parameter is initialized to at most four packets for new connections, following the rules from RFC 3390; the ssthresh parameter is initialized to an arbitrarily high value.

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.

5.1. Response to Idle and Application-limited Periods

CCID 2 is designed to follow TCP's congestion control mechanisms to the extent possible, but TCP does not have complete standardization for its congestion control response to idle periods (when no data packets are sent) or to application-limited periods (when the sending rate is less than that allowed by cwnd). This section is a brief guide to the standards for TCP in this area.

For idle periods, RFC 2581 recommends that the TCP sender SHOULD slow-start after an idle period, where an idle period is defined as a period exceeding the timeout interval. RFC 2861, currently Experimental, suggests a slightly more moderate mechanism where the congestion window is halved for every round-trip time that the sender has remained idle.

There are currently no standards governing TCP's use of the congestion window during an application-limited period. In particular, it is possible for TCP's congestion window to grow quite large during a long uncongested period when the sender is application-limited, sending at a low rate. RFC 2861 essentially suggests that TCP's congestion window not be increased during application-limited periods, when the congestion window is not being fully utilized.

5.2. Response to Data Dropped and Slow Receiver

As described in [DCCP], the Data Dropped option lets an endpoint declare that a packet was dropped at the end host before delivery to the application -- for instance, because of corruption or receive buffer overflow. CCID 2 senders respond to these options as described in [DCCP], with the following further clarifications.

- **Drop Code 2 ("receive buffer drop").** The congestion window "cwnd" is reduced by one for each packet newly acknowledged as Drop Code 2, except that it is never reduced below one.
- **Exiting slow-start.** The sender MUST exit slow start whenever it receives a relevant Data Dropped or Slow Receiver option.

5.3. Packet Size

CCID 2 is optimized for applications that generally use a fixed packet size, and that vary their sending rate in packets per second in response to congestion. CCID 2 is not appropriate for applications that require a fixed interval of time between packets, and vary their packet size instead of their packet rate in response to congestion. CCID 2 maintains a congestion window in packets, and does not increase the congestion window in response to a decrease in the packet size. However, some attention might be required for applications using CCID 2 that vary their packet size not in response to congestion, but in response to other application-level requirements.

CCID 2 implementations MAY check for applications that appear to be manipulating the packet size inappropriately. For example, an application might send small packets for a while, building up a fast rate, then switch to large packets to take advantage of the fast rate. (Preliminary simulations indicate that applications may not be able to increase their overall transfer rates this way, so it is not clear this manipulation will occur in practice [V03].)

6. Acknowledgements

CCID 2 acknowledgements are generally paced by the sender's data packets. Each required acknowledgement MUST contain Ack Vector options that declare exactly which packets arrived, and whether those packets were ECN-marked. Acknowledgement data in the Ack Vector options SHOULD generally cover the receiver's entire Acknowledgement Window; see [DCCP] (Section 11.4.2).

CCID 2 senders use DCCP's Ack Ratio feature to influence the rate at which DCCP-Ack packets are generated, thus controlling reverse-path congestion. This differs from TCP, which presently has no congestion control for pure acknowledgement traffic. CCID 2's reverse-path congestion control does not try to be TCP-friendly; it just tries to avoid congestion collapse, and to be somewhat better than TCP in the presence of a high packet loss or mark rate on the reverse path. The default Ack Ratio is two, and CCID 2 with this Ack Ratio behaves like TCP with delayed acks. [DCCP] (Section 11.3) describes the Ack Ratio in more detail, including its relationship to acknowledgement pacing and DCCP-DataAck packets. Section 6.1.1 below describes the sender's detection of lost or marked acknowledgements, and Section 6.1.2 gives the sender's rules for changing the Ack Ratio.

6.1. Congestion Control on Acknowledgements

When Ack Ratio is R, the receiver sends one DCCP-Ack packet per R data packets, more or less. Since the sender sends cwnd data packets per round-trip time, the acknowledgement rate equals cwnd/R DCCP-Acks per round-trip time. The sender keeps the acknowledgement rate roughly TCP-friendly by monitoring the acknowledgement stream for lost and marked DCCP-Ack packets, and modifying R accordingly. For every RTT containing a DCCP-Ack congestion event (that is, a lost or marked DCCP-Ack), the sender halves the acknowledgement rate by doubling Ack Ratio; for every RTT containing no DCCP-Ack congestion event, it additively increases the acknowledgement rate through gradual decreases in Ack Ratio.

6.1.1. Detecting Lost and Marked Acknowledgements

All packets from the receiver contain sequence numbers, so the sender can detect both losses and marks on the receiver's packets. The sender infers receiver packet loss in the same way as it infers losses of its data packets: a packet from the receiver is considered lost after at least NUMDUPACK packets with greater sequence numbers have been received.

DCCP-Ack packets are generally small, so they might impose less load on congested network links than DCCP-Data and DCCP-DataAck packets. For this reason, Ack Ratio depends on losses and marks on the receiver's non-data packets, not on aggregate losses and marks on all of the receiver's packets. The non-data packet category consists of those packet types that cannot carry application data: DCCP-Ack, DCCP-Close, DCCP-CloseReq, DCCP-Reset, DCCP-Sync, and DCCP-SyncAck. The sender can easily distinguish non-data marks from other marks. This is harder for losses, though, since the sender can't always know whether a lost packet carried data. Unless it has better information, the sender SHOULD assume, for the purpose of Ack Ratio calculation, that every lost packet was a non-data packet. Better information is available via DCCP's NDP Count option, if necessary. (Appendix B discusses the costs of mistaking data packet loss for non-data packet loss.)

A receiver that implements its own acknowledgement congestion control SHOULD NOT reduce its DCCP-Ack acknowledgement rate due to losses or marks on its data packets.

6.1.2. Changing Ack Ratio

Ack Ratio always meets three constraints: (1) Ack Ratio is an integer. (2) Ack Ratio does not exceed cwnd/2, rounded up, except that Ack Ratio 2 is always acceptable. (3) Ack Ratio is two or more for a congestion window of four or more packets.

The sender changes Ack Ratio within those constraints as follows. For each congestion window of data with lost or marked DCCP-Ack packets, Ack Ratio is doubled; and for each cwnd/(R^2 - R) consecutive congestion windows of data with no lost or marked DCCP-Ack packets, Ack Ratio is decreased by 1. (See Appendix A for the derivation.) Changes in Ack Ratio are signalled through feature negotiation; see [DCCP] (Section 11.3).

For a constant congestion window, this gives an Ack sending rate that is roughly TCP-friendly. Of course, cwnd usually varies over time; the dynamics will be rather complex, but roughly TCP-friendly. We recommend that the sender use the most recent value of cwnd when determining whether to decrease Ack Ratio by 1.

The sender need not keep 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 renegotiate the Ack Ratio more than once per round-trip time. Additionally, it MAY bound Ack Ratio below by two, or it MAY set Ack Ratio to one for half-connections with persistent congestion windows of 1 or 2 packets.

Putting it all together, the receiver always sends at least one acknowledgement per window of data when cwnd = 1, and at least two acknowledgements 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 an exponentially-backed-off timeout, 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.

6.2. Acknowledgements of Acknowledgements

An active sender DCCP A MUST occasionally acknowledge its peer DCCP B's acknowledgements, so that DCCP B can free up Ack Vector state. When both half-connections are active, A's acknowledgements of B's acknowledgements are automatically contained in A's acknowledgements of B's data. If the B-to-A half-connection is quiescent, however, DCCP A must occasionally send acknowledgements proactively, such as by sending a DCCP-DataAck packet that includes an Acknowledgement Number in the header.

An active sender SHOULD acknowledge the receiver's acknowledgements at least once 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.

6.2.1. Determining Quiescence

This section describes how a CCID 2 receiver determines that the corresponding sender is not sending any data, and therefore has gone quiescent. See [DCCP] (Section 11.1) for general information on quiescence.

Let T equal the greater of 0.2 seconds and two round-trip times. (The receiver may know the round-trip time in its role as the sender for the other half-connection. If it does not, it should use a default RTT of 0.2 seconds, as described in [DCCP] (Section 3.4).) Once the sender acknowledges the receiver's Ack Vectors, and the sender has not sent additional data for at least T seconds, the receiver can infer that the sender is quiescent. More precisely, the receiver infers that the sender has gone quiescent when at least T seconds have passed without receiving any data from the sender, and the sender has acknowledged receiver Ack Vectors covering all data packets received at the receiver.

7. Explicit Congestion Notification

CCID 2 supports Explicit Congestion Notification (ECN) [RFC 3168]. The sender will use the ECN Nonce for data packets, and the receiver will echo those nonces in its Ack Vectors, as specified in [DCCP] (Section 12.2). Information about marked packets is also 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 RFC 3540, and returns it as part of its Ack Vector options. The sender SHOULD check these ECN Nonce Echoes against the expected values, thus protecting against the accidental or malicious concealment of marked packets.

Because CCID 2 acknowledgements are congestion-controlled, ECN may also be used for its acknowledgements. 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 much motivation for lying about the mark rate on acknowledgements.

8. Options and Features

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

9. 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.

10. IANA Considerations

This specification defines the value 2 in the DCCP CCID namespace managed by IANA. This assignment is also mentioned in [DCCP].

CCID 2 also introduces three sets of numbers whose values should be allocated by IANA, namely CCID 2-specific Reset Codes, option types, and feature numbers. These ranges will prevent any future CCID 2-specific allocations from polluting DCCP's corresponding global namespaces; see [DCCP] (Section 10.3). However, this document makes no particular allocations from any range, except for experimental and testing use [RFC 3692]. We refer to the Standards Action policy outlined in RFC 2434.

10.1. Reset Codes

Each entry in the DCCP CCID 2 Reset Code registry contains a CCID 2-specific Reset Code, which is a number in the range 128-255; a short description of the Reset Code; and a reference to the RFC defining the Reset Code. Reset Codes 184-190 and 248-254 are permanently reserved for experimental and testing use. The remaining Reset Codes -- 128-183, 191-247, and 255 -- are currently reserved, and should be allocated with the Standards Action policy, which requires IESG review and approval and standards-track IETF RFC publication.

10.2. Option Types

Each entry in the DCCP CCID 2 option type registry contains a CCID 2-specific option type, which is a number in the range 128-255; the name of the option; and a reference to the RFC defining the option type. Option types 184-190 and 248-254 are permanently reserved for experimental and testing use. The remaining option types -- 128-183, 191-247, and 255 -- are currently reserved, and should be allocated with the Standards Action policy, which requires IESG review and approval and standards-track IETF RFC publication.

10.3. Feature Numbers

Each entry in the DCCP CCID 2 feature number registry contains a CCID 2-specific feature number, which is a number in the range 128-255; the name of the feature; and a reference to the RFC defining the feature number. Feature numbers 184-190 and 248-254 are permanently reserved for experimental and testing use. The remaining feature numbers -- 128-183, 191-247, and 255 -- are currently reserved, and should be allocated with the Standards Action policy, which requires IESG review and approval and standards-track IETF RFC publication.

11. Thanks

We thank Mark Handley and Jitendra Padhye for their help in defining CCID 2. We also thank Mark Allman, Aaron Falk, Nils-Erik Mattsson, Greg Minshall, Arun Venkataramani, Magnus Westerlund, and members of the DCCP Working Group for feedback on this document.

A. Appendix: Derivation of Ack Ratio Decrease

This section justifies the algorithm for increasing and decreasing the Ack Ratio given in Section 6.1.2.

The congestion avoidance phase of TCP halves the cwnd for every window with congestion. Similarly, CCID 2 doubles Ack Ratio for every window with congestion on the return path, roughly halving the DCCP-Ack sending rate.

The congestion avoidance phase of TCP increases cwnd by one MSS for every congestion-free window. Applying this congestion avoidance behavior to acknowledgement 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 Ack Ratio, since it is an integer. Instead, we must decrease 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, rough TCP-friendliness for the ack traffic can be accomplished by setting K to 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.
   Since we can only reduce R by increments of one, we find K
   so that, after K congestion-free windows,
   W/R + K would equal W/(R-1).

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

B. Appendix: Cost of Loss Inference Mistakes to Ack Ratio

As discussed in Section 6.1.1, the sender often cannot determine whether lost packets carried data. This hinders its ability to separate non-data loss events from other loss events. In the absence of better information, the sender assumes, for the purpose of Ack Ratio calculation, that all lost packets were non-data packets. This may overestimate the non-data loss event rate, which can lead to a too-high Ack Ratio, and thus a too-slow acknowledgement rate. All acknowledgement information will still get through -- DCCP acknowledgements are reliable -- but acknowledgement information will arrive in a burstier fashion. Absent some form of rate-based pacing, this could lead to increased burstiness for the sender's data traffic.

There are several cases when the problem of an overly-high Ack Ratio, and the resulting increased burstiness of the data traffic, will not arise. In particular, call the receiver DCCP B and the sender DCCP A. Then:

- The problem won't arise unless DCCP B is sending a significant amount of data itself. When the B-to-A half-connection is quiescent or low-rate, most packets sent by DCCP B will, in fact, be pure acknowledgements, and DCCP A's estimate of the DCCP-Ack loss rate will be reasonably accurate.
- The problem won't arise if DCCP B habitually piggybacks acknowledgement information on its data packets. The piggybacked acknowledgements are not limited by Ack Ratio, so they can arrive frequently enough to prevent burstiness.
- The problem won't arise if DCCP A's sending rate is low, since burstiness isn't a problem at low rates.
- The problem won't arise if DCCP B's sending rate is high relative to DCCP A's sending rate, since the B-to-A loss rate must be low to support DCCP B's sending rate. This bounds the Ack Ratio to reasonable values even when DCCP A labels every loss as a DCCP-Ack loss.
- The problem won't arise if DCCP B sends NDP Count options when appropriate (the Send NDP Count/B feature is true). Then the sender can use the receiver's NDP Count options to detect, in most cases, whether lost packets were data packets or DCCP-Acks.
- Finally, the problem won't arise if DCCP A rate-paces its data packets.

This leaves the case when DCCP B is sending roughly the same amount of data packets and non-data packets, without NDP Count options, and with all acknowledgement information in DCCP-Ack packets. We now quantify the potential cost, in terms of a too-large Ack Ratio, due to the sender's misclassifying data packet losses as DCCP-Ack losses. For simplicity, we assume an environment of large-scale statistical multiplexing, where the packet drop rate is independent of the sending rate of any individual connection.

Assume that when DCCP A correctly counts non-data losses, Ack Ratio is set so that B-to-A data and acknowledgement traffic both have a sending rate of D packets per second. Then when DCCP A incorrectly counts data losses as non-data losses, the sending rate for the B-to-A data traffic is still D pps, but the reduced sending rate for the B-to-A acknowledgement traffic is f*D pps, with f < 1. Let the packet loss rate be p. The sender incorrectly estimates the non-data loss rate as (pD+pfD)/fD, or, equivalently, as p(1 + 1/f). Because the congestion control mechanism for acknowledgement traffic is roughly TCP-friendly, and therefore the non-data sending rate and the data sending rate both grow as 1/sqrt(x) for x the packet drop rate, we have

    fD/D = sqrt(p)/sqrt(p(1 + 1/f)),

so

    f^2 = 1/(1 + 1/f).

Solving, we get f = 0.62. If the sender incorrectly counts lost data packets as non-data in this scenario, the acknowledgement rate is decreased by a factor of 0.62. This would result in a moderate increase in burstiness for the A-to-B data traffic, which could be mitigated by sending NDP Count options or piggybacked acknowledgements, or by rate-pacing out the data.

Normative References

[DCCP] E. Kohler, M. Handley, and S. Floyd. Datagram Congestion Control Protocol, draft-ietf-dccp-spec-09.txt, work in progress, November 2004.

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

[RFC 2018] M. Mathis, J. Mahdavi, A. Floyd, and A. Romanow. TCP Selective Acknowledgement Options, RFC 2018, October 1996.

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

[RFC 2434] T. Narten and H. Alvestrand. Guidelines for Writing an IANA Considerations Section in RFCs. RFC 2434.

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

[RFC 2988] V. Paxson and M. Allman, Computing TCP's Retransmission Timer, RFC 2988, November 2000.

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

[RFC 3390] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's Initial Window. RFC 3390.

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

[RFC 3692] T. Narten. Assigning Experimental and Testing Numbers Considered Useful. RFC 3692.

Informative References

[CCID 3 PROFILE] S. Floyd, E. Kohler, and J. Padhye. Profile for DCCP Congestion Control ID 3: TFRC Congestion Control. draft-ietf-dccp-ccid3-08.txt, work in progress, November 2004.

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

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

[RFC 3540] N. Spring, D. Wetherall, and D. Ely. Robust Explicit Congestion Notification (ECN) Signaling with Nonces. RFC 3540.

[V03] Arun Venkataramani, August 2003. Citation for acknowledgement purposes only.

Authors' Addresses

Sally Floyd <floyd@icir.org>
ICSI Center for Internet Research
1947 Center Street, Suite 600
Berkeley, CA 94704
USA

Eddie Kohler <kohler@cs.ucla.edu>
4531C Boelter Hall
UCLA Computer Science Department
Los Angeles, CA 90095
USA

Full Copyright Statement

Intellectual Property