Eddie Kohler
Sally Floyd
ICIR
9 July 2003

Datagram Congestion Control Protocol (DCCP) Overview

Abstract

We provide a short overview of Datagram Congestion Control Protocol (DCCP), which implements a congestion-controlled, unreliable flow of datagrams suitable for use by applications such as streaming media.

[This document consists mostly of excerpts from [DCCP]. Section 1 serves as a brief introduction to DCCP, while the remaining sections serve as reference material. Particularly note the typical connection in Section 4. For each section, the corresponding section in [DCCP] can be consulted for more information.]

1. Introduction

This document is an overview of the Datagram Congestion Control Protocol (DCCP).

DCCP provides the following features, among others:

- An unreliable flow of datagrams, with acknowledgements.
- A reliable handshake for connection setup and teardown.
- Reliable negotiation of features.
- A choice of TCP-friendly congestion control mechanisms, including, initially, TCP-like congestion control (CCID 2) and TCP-Friendly Rate Control [RFC 3448] (CCID 3). CCID 2 uses a version of TCP's congestion control mechanisms, and is appropriate for flows that want to quickly take advantage of available bandwidth, and can cope with quickly changing send rates; CCID 3 is appropriate for flows that require a steadier send rate.
- Options that tell the sender, with high reliability, which packets reached the receiver, and whether those packets were ECN marked, corrupted, or dropped in the receive buffer.
- Congestion control incorporating Explicit Congestion Notification (ECN) and the ECN Nonce.
- Mechanisms allowing a server to avoid holding any state for unacknowledged connection attempts or already-finished connections.
- Path MTU discovery.

DCCP is intended for applications that require the flow-based semantics of TCP, but have a preference for delivery of timely data over in-order delivery or reliability, or which would like different congestion control dynamics than TCP. To date most such applications have used either TCP, whose reliability and in-order semanitcs can introduce arbitrary delay, or used UDP and implemented their own congestion control mechanisms (or no congestion control at all). DCCP will provide a standard way to implement congestion control and congestion control negotiation for such applications, and enable the use of ECN, along with conformant end-to-end congestion control, for applications that would otherwise be using UDP.

Similarly, DCCP is intended for applications that do not require features of SCTP [RFC 2960] such as sequenced delivery within multiple streams.

1.1. Important Differences from TCP

This section lists some of the more important differences between DCCP and TCP.

- **Packet stream.** DCCP is a packet stream protocol, not a byte stream protocol. The application is responsible for framing.
- **Unreliability.** DCCP will never retransmit a datagram. Options are retransmitted as required to make feature negotiation and ack information reliable.
- **Packet sequence numbers.** Sequence numbers refer to packets, not bytes. Every packet sent by a DCCP endpoint gets a new sequence number, even including pure acknowledgements. This lets a DCCP receiver detect lost acks, but introduces some complications with endpoitns getting out of sync; see Sequence Number Validity in [DCCP].
- **Copious space for options** (up to 1020 bytes).
- **Feature negotiation.** This is a generic mechanism by which endpoints can agree on the values of "features", or properties of the connection.
- **Choice of congestion control.** One such feature is the congestion control mechanism to use for the connection. In fact, the two endpoints can use different congestion control mechanisms for their data packets: In an A<->B connection, data packets sent from A->B can use CCID 2, and data packets sent from B->A can use CCID 3.
- **Different acknowledgement formats.** The CCID for a connection determines how much ack information needs to be transmitted. In CCID 2 (TCP-like), this is about one ack per 2 packets, and each ack must declare exactly which packets were received (Ack Vector option); in CCID 3 (TFRC), it's about one ack per RTT, and acks must declare at minimum just the lengths of recent loss intervals.
- **No receive window.** DCCP is a congestion control protocol, not a flow control protocol.
- **Distinguishing different kinds of loss.** A Data Dropped option lets one endpoint declare that a packet was dropped because of corruption, because of receive buffer overflow, and so on. This facilitates research into more appropriate rate-control responses for these non-network-congestion losses (although currently all losses will cause a congestion response).
- **Definition of acknowledgement.** In TCP, a packet is acknowledged only when the data is queued for delivery to the application. This does not make sense in DCCP, where an application might request a drop-from-front receive buffer, for example. We acknowledge a packet when its options have been processed. The Data Dropped option may later say that the packet's payload was discarded.
- **Integrated support for mobility.**
- **No simultaneous open.**

2. Design Rationale

To minimize overhead, we included only minimal functionality in DCCP. Anything that could reasonably be layered on top, such as FEC and semi-reliability, we left out of the core protocol.

3. Concepts and Terminology

Each DCCP connection runs between two endpoints, which we often name DCCP A and DCCP B. Data may pass over the connection in either or both directions. We often consider a subset of the connection, namely a *half-connection*, which consists of the data packets sent in one direction, plus the corresponding acknowledgements sent in the other direction. In the context of a single half-connection, the HC-Sender is the endpoint sending data, while the HC-Receiver is the endpoint sending acknowledgements.

Each half-connection is managed by a congestion control mechanism, specified by single-byte congestion control identifiers, or CCIDs. The endpoints negotiate these mechanisms at connection setup. The CCID for a half-connection describes how the HC-Sender limits data packet rates; how it maintains necessary parameters, such as congestion windows; how the HC-Receiver sends congestion feedback via acknowledgements; and how it manages the acknowledgement rate.

3.1. Connection Initiation and Termination

Every DCCP connection is actively initiated by one DCCP, which connects to a DCCP socket in the passive listening state. We refer to the active endpoint as "the client" and the passive endpoint as "the server".

3.2. Features

DCCP uses a generic mechanism to negotiate connection properties, such as the CCIDs active on the two half-connections. These properties are called features.

The Change, Prefer, and Confirm options negotiate feature values. Change is sent to a feature location, asking it to change its value for the feature. The feature location may respond with Prefer, which asks the other endpoint to Change again with different values, or it may change the feature value and acknowledge the request with Confirm. Retransmissions make feature negotiation reliable.

4. DCCP Packets

DCCP has nine different packet types: DCCP-Request, DCCP-Response, DCCP-Data, DCCP-Ack, DCCP-DataAck, DCCP-CloseReq, DCCP-Close, DCCP-Reset, and DCCP-Move

The progress of a typical DCCP connection is as follows. (This description is informative, not normative.)

(1) The client sends the server a DCCP-Request packet specifying the client and server ports, the service being requested, and any features being negotiated, including the CCID that the client would like the server to use. The client may optionally piggyback some data on the DCCP-Request packet---an application-level request, say---which the server may ignore.

(2) The server sends the client a DCCP-Response packet indicating that it is willing to communicate with the client. The response indicates any features and options that the server agrees to, begins or continues other feature negotiations if desired, and optionally includes an Init Cookie that wraps up all this information and which must be returned by the client for the connection to complete.

(3) The client sends the server a DCCP-Ack packet that acknowledges the DCCP-Response packet. This acknowledges the server's initial sequence number and returns the Init Cookie if there was one in the DCCP-Response. It may also continue

feature negotiation.

(4) Next comes zero or more DCCP-Ack exchanges as required to finalize feature negotiation. The client may piggyback an application-level request on its final ack, producing a DCCP-DataAck packet.

(5) The server and client then exchange DCCP-Data packets, DCCP-Ack packets acknowledging that data, and, optionally, DCCP-DataAck packets containing piggybacked data and acknowledgements. If the client has no data to send, then the server will send DCCP-Data and DCCP-DataAck packets, while the client will send DCCP-Acks exclusively.

(6) The server sends a DCCP-CloseReq packet requesting a close.

(7) The client sends a DCCP-Close packet acknowledging the close.

(8) The server sends a DCCP-Reset packet whose Reason field is set to "Closed", and clears its connection state.

(9) The client receives the DCCP-Reset packet and holds state for a reasonable interval of time to allow any remaining packets to clear the network.

4.1. Examples of DCCP Congestion Control

The main draft gives two examples showing DCCP congestion control in operation.

5. Packet Formats

5.1. Generic Packet Header

All DCCP packets begin with a generic DCCP packet header:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |           Dest Port           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type  | CCval |                Sequence Number                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | # NDP | Cslen |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.2. DCCP-Data, DCCP-Ack, and DCCP-DataAck Packet Formats

The payload of a DCCP connection is sent in DCCP-Data and DCCP-DataAck packets, while DCCP-Ack packets are used for acknowledgements when there is no payload to be sent. DCCP-Data packets look like this:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/                    with Type=2 (DCCP-Data)                    /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Options                  /   [padding]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            data                               |
|                             ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

DCCP-Ack packets dispense with the data, but contain an acknowledgement number:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/                    with Type=3 (DCCP-Ack)                     /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |           Acknowledgement Number              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Options                  /   [padding]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

DCCP-Ack and DCCP-DataAck packets often include additional acknowledgement options, such as Ack Vector, as required by the congestion control mechanism in use.

6. Options and Features

All DCCP packets may contain options, which occupy space at the end of the DCCP header and are a multiple of 8 bits in length. All options are always included in the checksum. An option may begin on any byte boundary.

The following options are currently defined:

Type	Option Length	Meaning
0	1	Padding
2	1	Slow Receiver
32	3-4	Ignored
33	variable	Change
34	variable	Prefer
35	variable	Confirm
36	variable	Init Cookie
37	variable	Ack Vector [Nonce 0]
38	variable	Ack Vector [Nonce 1]
39	variable	Data Dropped
40	6	Timestamp
41	6-10	Timestamp Echo
42	variable	Identification
44	variable	Challenge
45	4	Payload Checksum
46	4-6	Elapsed Time
128-255	variable	CCID-specific options

6.1. Feature Numbers

The first data byte of every Change, Prefer, or Confirm option is a feature number, defining the type of feature being negotiated. The remainder of the data gives one or more values for the feature, and is interpreted according to the feature. The current set of feature numbers is as follows:

Number	Meaning	Neg.?
1	Congestion Control (CC)	Y
2	ECN Capable	Y
3	Ack Ratio	N
4	Use Ack Vector	Y
5	Mobility Capable	Y
6	Loss Window	N
7	Connection Nonce	N
8	Identification Regime	Y
128-255	CCID-Specific Features	?

7. Congestion Control IDs

Each congestion control mechanism supported by DCCP is assigned a congestion control identifier, or CCID: a number from 0 to 255.

The CCIDs defined by this document are:

```
CCID   Meaning
----   -------
  0    Reserved
  1    Unspecified Sender-Based Congestion Control
  2    TCP-like Congestion Control
  3    TFRC Congestion Control
```

8. Acknowledgements

Congestion control requires receivers to transmit information about packet losses and ECN marks to senders. DCCP receivers MUST report all congestion they see, as defined by the relevant CCID profile.

8.1. Ack Ratio Feature

Ack Ratio provides a common mechanism by which CCIDs that clock acknowledgements off of data packets can perform rudimentary congestion control on the acknowledgement stream. CCID 2, TCP-like Congestion Control, uses Ack Ratio to limit the rate of its acknowledgement stream, for example. Some CCIDs ignore Ack Ratio, performing congestion control on acknowledgements in some other way.

8.2. Slow Receiver Option

An HC-Receiver sends the Slow Receiver option to its sender to indicate that it is having trouble keeping up with the sender's data.

8.3. Data Dropped Option

The Data Dropped option indicates that some packets reported as received actually had their data dropped before it reached the application. The sender's congestion control mechanism MAY react to data-dropped packets; such responses MAY be less severe than responses triggered by a lost or marked packet.

8.4. Payload Checksum Option

The Payload Checksum option holds the 16 bit one's complement of the one's complement sum of all 16 bit words in the DCCP payload (the data contained in a DCCP-Request, DCCP-Response, DCCP-Data, DCCP-DataAck, or DCCP-Move packet). When combined with a Checksum Length of less than 15, this lets DCCP distinguish between corruption in a packet's payload and corruption in its header. Corrupted-header packets MUST be treated as dropped by the network, while corrupted-payload packets MAY be treated differently; for example, the sender's response to corruption might be less stringent than its response to congestion.

9. Explicit Congestion Notification

The DCCP protocol is fully ECN-aware.

10. Multihoming and Mobility

DCCP provides primitive support for multihoming and mobility via a mechanism for transferring a connection endpoint from one address to another. The moving endpoint must negotiate mobility support beforehand, and both endpoints must share their Connection Nonces. When the moving endpoint gets a new address, it sends a DCCP-Move packet from that address to the stationary endpoint. The stationary endpoint then changes its connection state to use the new address.

DCCP's support for mobility is intended to solve only the simplest multihoming and mobility problems. For instance, DCCP has no support for simultaneous moves. Applications requiring more complex mobility semantics, or more stringent security guarantees, should use an existing solution like Mobile IP or [SB00].

11. Path MTU Discovery

A DCCP implementation SHOULD be capable of performing Path MTU (PMTU) discovery.

12. Middlebox Considerations

This section describes properties of DCCP that firewalls, network address translators, and other middleboxes must consider, including parts of the packet that middleboxes must not change.

13. Abstract API

API issues for DCCP are discussed in another Internet-Draft, in progress.

14. Multiplexing Issues

In contrast to TCP, DCCP does not offer reliable ordered delivery. As a consequence, with DCCP there are no inherent performance penalties in layering functionality above DCCP to multiplex several sub-flows into a single DCCP connection.

15. DCCP and RTP

The real-time transport protocol, RTP [RFC 1889], is currently used (over UDP) by many of DCCP's target applications (for instance, streaming media). There are two potential sources of overhead in the RTP-over-DCCP combination, duplicated acknowledgement information and duplicated sequence numbers. We argue that together, these sources of overhead add just 4 bytes per packet relative to RTP-over-UDP, and that eliminating the redundancy would not reduce the overhead. However, particular CCIDs might make productive use of the space occupied by RTP's sequence number.

16. Security Considerations

DCCP does not provide cryptographic security guarantees. Applications desiring hard security should use IPsec or end-to-end security of some kind.

Nevertheless, DCCP is intended to protect against some classes of attackers. Attackers cannot hijack a DCCP connection (close the connection unexpectedly, or cause attacker data to be accepted by an endpoint as if it came from the sender) unless they can guess valid sequence numbers. Thus, as long as endpoints choose initial sequence numbers well, a DCCP attacker must snoop on data packets to get any reasonable probability of success. The sequence number validity and mobility mechanisms provide this guarantee.

17. IANA Considerations

DCCP introduces six sets of numbers whose values should be allocated by IANA.

18. Design Motivation

This section of [DCCP] attempts to capture some of the rationale behind specific details of DCCP design.

19. Informative References

[DCCP] E. Kohler, M. Handley, S. Floyd, and J. Padhye. Datagram Congestion Control Protocol, draft-ietf-dccp-spec-04.txt, work in progress, June 2003.

[RFC 1889] Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 1889.

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

[RFC 2960] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. RFC 2960.

[RFC 3448] M. Handley, S. Floyd, J. Padhye, and J. Widmer, TCP Friendly Rate Control (TFRC): Protocol Specification, RFC 3448, Proposed Standard, January 2003.

[SB00] Alex C. Snoeren and Hari Balakrishnan. An End-to-End Approach to Host Mobility. Proc. 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '00), August 2000.

20. Authors' Addresses

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

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