Internet Engineering Task Force
INTERNET-DRAFT
draft-lanphear-dccp-user-guide-00.ps

Damon Lanphear
RealNetworks

25 October 2002
Expires: April 2003

Datagram Congestion Control Protocol (DCCP) User Guide

Status of this Document

Abstract

This document is an informative reference for the implementation of the Datagram Congestion Control Protocol (DCCP) and its interface with the application layer.

Table of Contents

1. Introduction

This document presents issues that pertain to the semantics and logistics of interaction between the Datagram Congestion Control Protocol (DCCP) and an application that uses DCCP. The choices that are to be made in implementing such semantics are also discussed. This discussion may serve as a reference for the use and implementation of the Datagram Congestion Control Protocol (DCCP). The text of this document should be regarded as informative, rather than normative.

A discussion of the interaction between an application and a lower level service begs the question of an API specification. This text does not assume a specific API paradigm. Rather, API issues are discussed in the abstract. The purpose of which is to allow the reader to focus on fundamental implementation issues without limiting the discussion to a particular problem domain. An example implementation using a specific API is presented, however, as a means by which API design choices can be demonstrated in more certain terms.

DCCP presents a set of unique challenges to the provision of an application interface. These challenges are compounded by the fact that many DCCP implementations will be done in kernel space. API paradigms which are presented by existing kernels may not be sufficient for the most complete exposure of DCCP's services. Mitigating the trade off between DCCP functionality muted by an API and providing a sufficiently flexible API considered throughout this document.

The reader's knowledge of DCCP and its design rationale are assumed. Reviewing the DCCP specification [DCCP] and the DCCP problem statement [DCCP-PROBLEM] before reading this document is therefore suggested.

2. Considerations for Target Applications

DCCP specifically targets a subset of network applications. These applications have elected to use UDP over TCP for data transfer. The reason for this choice is typically that the characteristics of TCP data transfers are not congruent with the specific needs of the application. Examples of such needs are:

(1) Startup Delay: Applications may wish to avoid the delay of a three-way handshake.

(2) Statelessness: Applications may wish to avoid holding connection state, and the potential state-holding attacks that come with this.

(3) Trading of Reliability against Timing: For some applications, data being sent is timely in the sense that if it is not delivered by some deadline (typically a small number of RTTs) then the data will not be useful at the receiver. [DCCP-PROBLEM]

(4) Smoothness of Delivery: Applications such as media delivery and IP Telephony are susceptible to the oscillations of delivery rate induced by TCP. Periodic fluctuations of throughput can cause media codecs to run dry, inducing a perceptible gap in the presentation, if a continuous stream of data is not received at the decode rate.

DCCP has been designed to resolve these issues in a way that will be acceptable to applications that would otherwise use UDP. The extent to which the design of a protocol can address these issues is limited by the extent to which an implementation is sensitive to these issues. It is therefore relevant that a DCCP implementation takes into account the needs of the overlying application through the way in which it exposes the features and semantics of DCCP.

This section discusses the needs of applications targeted by DCCP. Emphasis is placed on the impact of these needs on an implementation of DCCP.

2.1. Real Time Media Applications

2.1.1. The Post Network Buffer and Rate Control

Real time media applications typically employ a post network buffer, or a play out buffer, in which media packets are queued prior to decoding. Fluctuations in the rate and jitter of the media transmission are absorbed by the play out buffer. A client in a media application will attempt to maintain a certain degree of occupation of the post network buffer throughout the session in order to insulate the codec from such fluctuations. In the event that this buffer is drained by the codec prior to the termination of the session, the media application will typically pause the presentation to refill the buffer. This is an undesirable state since it implies a gap in the continuity of the media experienced by the user.

The dynamics of the post network buffer introduce a set of requirements to rate control that exist in parallel with those of congestion control. Real time media applications may attempt to manage the occupation of the post network buffer by asking the server to either send more or less than the transmission rate required by the codec to decode the media in real time. Although this dynamic aspect of real time media applications' rate control falls outside of the purview of DCCP, it does have implications on DCCPs congestion control processing.

In DCCP congestion control processing is particular to the CCID selected by the user of a given DCCP. In any case an implementation may have to consider how an application's rate control may interact with the specified congestion control algorithms. In certain cases it may be appropriate for the congestion control scheme to regard application cues in the same way as it would congestion cues from the network. This may be achieved through the use of DCCP's slow receiver notification.

[DCCP] articulates a specific interpretation of Slow Receiver options, that DCCP will not increase the sending rate for one RTT. An application, however, may need to alter its behavior to respond to Slow Receiver reports in a way that is independent from DCCP's response. As such, it may be useful for an API to specify to the application that a Slow Receiver signal has been received. In this way the application can more tightly integrate its flow control requirements with the congestion control processing of DCCP, avoiding redundant work and unintended side effects of interactions between application and transport layer flow control.

2.1.2. Decode Costs and Media Adaptation

TBD

2.1.3. Selective Transmission and Retransmission

Real time media applications have specific delivery latency requirements. A media packet must arrive at the the receiver within a certain period of time in order for the the packet to have relevance in the decoded media stream. This requirement has an impact both on transmission policy and on retransmission policy.

DCCP's unreliable data flows, which defer implementation of ARQ to the application, provide a clear advantage over TCP's requirement that all data be presented to the application in order and without loss. By deferring retransmission to the application, the application can enforce retransmission policies that suit its presentation requirements. A media application may not want to pause the time line while awaiting the fulfillment of a retransmission request. Likewise, the utility of retransmitting a given packet as a function of the coding scheme may not be equal to that of retransmitting any other packet in the stream. This problem is compounded by the fact that the frequency of retransmitted packets may impact the available throughput for more current or relevant data.

DCCP has information about the loss and transmission latency that may affect an application's ability to make a decision about selective retransmission requests. Providing this information to an application in an elegant and sufficiently flexible way is a challenge for the API design. For certain API paradigms it may not be efficient to represent to the application layer the identity of packets that are lost. In such cases informing an application of why packets are lost and the opportunity for successful retransmission over a given window of time may be a helpful alternative to per packet loss data.

Some media applications may wish to set transmission policies based on the likelihood that a packet will be received on time. Although an application may be aware that the current transmission rate is sufficient for the codec to decode the media stream continuously, it may not be aware that the dynamics of the send and receive buffers in the DCCP are inducing a latency that prevents the timely delivery of data. This issue becomes a particular problem during the synchronization of two distinct streams at the client. It will therefore be useful for an application to have information about the delay induced by send and receive buffers.

Presentation of delay information from DCCP to the application should be achieved through a representation of a compound of two metrics, the estimated RTT and the estimated buffer depth. Although the estimated RTT is not necessarily representative of the forward propagation delay, it does provide an indication of the extent of delay induced by the network. The number of buffers in the send queue may also be used by the application to determine the amount of delay the sending DCCP may add to the propagation of the transmission of a given datagram. Providing such metrics to the application is a matter of maintaining these metrics, and providing them upon application request.

2.1.4. Latency Requirements

Within the domain of real time media applications exists a subset of broadcast applications which have a varying latency requirements. In instances, it may be permissible to have arbitrary latency between the real time encoding source and the recipients of that source. Such cases instances include broadcast media scenarios in which there is no reference point against which to ascertain the extent to which the received media is synchronized with wall clock time. Likewise, there are instances in which such synchronization is absolutely essential to the success of the application. For example, broadcast sessions that are used for security purposes where timely human response to events communicated in the broadcast is essential.

DCCP congestion control processing and protocol processing should be done in such as way as to always minimize propagation delay. The application should be provided with a mechanism to inform any trade offs between propagation delay and DCCP functionality. DCCP feature negotiation, for example, may cost a delay in the stream while a given feature is agreed upon. An implementation of DCCP may want to allow the application to configure default responses that can foreshorten feature negotiation requests. This will allow mitigation of events that may otherwise stall the session until feature negotiation completes.

An API to DCCP may, for example, allow the application to specify preferences for negotiable features prior to the initiation of the session. This will allow DCCP to perform feature negotiation as per application preferences without explicit interaction with the application. The same API may also accommodate the case where the application needs a more dynamic presence in the DCCP feature negotiation process.

2.1.5. RTP/RTCP

TBD

2.1.6. IP Telephony

TBD

2.1.7. Video Teleconferencing

TBD

2.1.8. Multi-Player Game Applications

TBD

2.2. Cross Application Issues

2.2.1. Loss Signaling

Although DCCP does provide unreliable delivery of data, it does have a notion of which datagrams where dropped, and a limited notion of why they were dropped. Such information is useful to applications to perform ARQ, or take other adaptive measures. The provision of such information to the application may blur the separation of functionality maintained by the respective communication layers. As such, an implementation that provides explicit loss information to the application for each datagram may be both inefficient and incongruent with certain API paradigms.

2.2.2. Drop Preference

One of DCCP's strong advantages over TCP is its unreliable delivery of packets. Unreliable delivery allows an application to enforce retransmission policies, for example, since not all packets need to be regarded with equal relevance as they in are TCP. This paradigm can be extended to scenarios where DCCP must drop packets when resources become scarce.

In the event that DCCP must drop a packet due to buffer overflow, a decision must be made as to which packet to drop. Some applications, such as real time media applications, may place preference on new data versus old data within a given queue. As such, a DCCP implementation may want to provide the application with the ability to specify a drop semantic to follow.

2.2.3. Mobility

Mobile capability in DCCP presents a set of problems to applications that transmit a continuous stream of data. Such applications include a certain subset of real time media applications and gaming applications. In the event that a mobile enabled DCCP initiates a move, the transmission of data must stop until the move completes successfully. The API semantics of handling this event may be as simple as responding to write requests with a failure code indicating that a move event is occurring. It is critical, however, that the application be afforded an opportunity to make an adaptive decision in the presence of gaps in service induced by a move event.

DCCP specifies that a receiving DCCP HC may drop all data from a the old IP address/port pair in response to a successful move event, or follow an alternate route of accepting a loss window's worth of data. The semantics chosen for this event should be configurable by the application.

Since session attributes may have changed as a result of a Move event, it may be useful to inform the application of the successful completion of a move event so it can reset state contingent upon such attributes. Path MTU is an example of such an attribute that affects application behavior, and is susceptible to change as a result of a Move event.

2.2.4. Send Option on Option Change With or Without Data

[DCCP] indicates that options may be sent either with or without data. Sending options with data may require that the DCCP wait to send the option until the application has more data to send. This may not be a desired behavior for an application. Likewise and application may want to piggyback options onto data for purposes of efficiency. DCCP applications may be afforded the opportunity to inform DCCP that an option change request should be sent immediately rather than after some amount of data has been written to the DCCP. This can be achieved through a flag sent with the options change request.

2.2.5. Protocol Information Shared with Overlying Layers

Application layer framing protocols, such a RTP, may represent session information that is redundant with that used by DCCP. Examples of such data include sequence numbering and latency calculations. A DCCP implementation may provide a way to share information with the overlying protocol in an effort to minimize redundancy.

Such a model, however, may neither be feasible nor appropriate for certain existing kernel API paradigms such as BSD sockets. In a clean design, DCCP need not have knowledge of the header information added by the overlying protocol. A trade off must be made between inducing an impact on existing APIs, or creating a new API with added complexity, and coping with the overhead of redundant data.

3. API and Configuration Considerations

DCCP semantics and functionality must be rendered meaningfully to the application layer through an API. This section collects the semantical issues of DCCP that have an impact on an API to DCCP.

3.1. Configuration

DCCP's flexibility has a profound influence on the variability of DCCP's semantics. Those areas in which DCCP's semantics may vary as per the needs of the application are represented here as a collection of configuration variables which an API designer may take into account when exposing DCCP's functionality.

Configurable variables for DCCP fall into one of two categories. Either configurable variable may be for an option provided in [DCCP] , or an option may be exposed provided by the semantics of a particular implementation.

3.1.1. Configurable Variables Specific to DCCP

The specific context and precise meaning of the configurable options is found in [DCCP]. In some cases [DCCP] provides for a default configuration, in others no default is specified. Those options with a specified default are indicated with a *. Otherwise the default value indicated is a suggested value.

Feature	Default Value
CCID	2 *
Connection Nonce	1 *
ECN Capable	1 *
Mobility Capable	0 *
PMTU	0
Data Discard	0
Init Cookie	0
Loss Window	1000 *
Ack Ratio	2 *
Use Ack Vectors	0 *
Service Name	n/a

3.1.2. Configurable Variables Specific to an API Implementation

The configurable options collected here represent levels of flexibility that are provided for in [DCCP] but are not explicitly articulated. An API to DCCP may expose these options in cases where an implementation wants to allow default semantics to be modified to fit the needs of an application.

3.1.2.1. Feature Negotiation Limit

Section 5.3 of [DCCP] discusses Feature Negotiation. It is possible for DCCP's Feature Negotiation to enter an ambiguous state if a given end point continues to negotiate for a specific option which the remote end point does not prefer. In this case, it may be useful for the application to specify the number of failed change requests before the connection is aborted. The default value should be 3.

3.1.2.2. Preferred Feature Priorities, Exclusions and Inclusions

An application may want to specify a set of preferred options with which to negotiate against a remote end point. The utility of doing so is to allow a DCCP to try a number of application specified options before failing a connection, or feature change attempt. This semantics mitigates the cost of forcing DCCP to negotiate through the application layer for each failed option change.

An implementation of DCCP may also provide the application with the ability to indicate certain features that it prefers to use or nor to use. The inclusion or exclusion of negotiable features for a DCCP allow that DCCP to indicate application preferences during feature negotiation

3.1.2.3. Loss Window Ratio

The Loss Window feature, described in section 5.9 of [DCCP] , determines the window over which a DCCP endpoint regards received datagrams as valid. Since the receiver may position the loss window around the most recent sequence number seen, this variable allows the application to set the ratio of the loss window space before the current sequence number to the loss window space after the last sequence number. The utility of this to an application may be based on the needs of the application for timely data. That is application that is too far beyond the current sequence number may not be of any use to the application and can safely be dropped. The default is 0, which implies that DCCP should select the ratio and/or adjust it dynamically.

3.1.2.4. Slow Receiver Notification

Slow Receiver is described in section 7.6 of [DCCP] A sending application may want to know if a DCCP HC is blocked because of congestion or because the receiver is slow. This is because the sending application may wish to take adaptive measures that are more appropriate to a slow receiver than decreasing the sending rate. The default value for this option is 0.

3.1.2.5. Slow Receiver Option

By setting the slow receiver option, the application allows the underlying DCCP to send Slow Receiver options to the sending DCCP. The default value for this option is 0.

3.1.2.6. Buffer Close Drop Notification

This option indicates that a sending application wants to be notified in the event that packets are being dropped at the remote end point, and therefore stop sending data to the underlying DCCP for transmission. The default value for this option 1.

3.1.2.7. Drop Preference

An application may wish to specify the way in which packets are dropped from a queue. For example, certain real time applications have a preference for the most recent data over the oldest data in the queue. This option may be articulated as a boolean which indicates that the DCCP should drop-head (1) or drop-tail (0). The default should be 1.

3.1.2.8. Mobile Receiver Drop Behavior Preference

Section 9.4 of [DCCP] discusses indicates that a DCCP receiver may drop all data received from the old endpoint once a DCCP-Move has completed successfully, or it may accept one Loss Window’s worth of data. This option lets the application determine the behavior in this case. A value of 0 indicates that the receiver should drop all data from the old endpoint after a move, and a value of 1 indicates that the receiver should accept one loss window’s worth of data from the remote endpoint after the move.

3.1.2.9. Use Service Name

The Service Name is discussed in section 4.5 of [DCCP]. If this option is set to 1 then DCCP must force every socket to specify a Service Name. The default value should be 1.

3.1.2.10. Minimal Buffering Option

An implementation may wish to allow applications to specify the amount of buffering that the DCCP will do on the sender side. So, for example, a sending DCCP may support a semantic such that the DCCP does not make itself available for write requests unless the current CCID indicates that a write request would result in an immediate transmission.

3.1.2.11. PMTU Change Behavior

Section 10 of [DCCP] suggests that if the MTU has changed to exceed the sizes of the pending datagrams, then those datagrams may either be dropped or transmitted with the DF bit cleared. This behavior should be configurable by the application.

3.2. Connection Establishment and Termination

3.2.1. Connection Establishment

Due to Feature Negotiation, it is typical that a DCCP connection attempt will not be completed immediately. An API to DCCP that supports non-blocking method invocations should provide a mechanism for polling the connection state so an application can determine if a connection attempt is in progress, has failed, or has completed.

3.2.2. Data Piggybacking

[DCCP] permits data piggybacking onto connection requests. An API need not support this. An API implementation may, however, provide queuing of transmitted data in the DCCP prior to connection establishment to mitigate application latency between connection realization and the initiation of data flow. Implementation of data piggybacking on a connection request may have an impact on the non-blocking interface semantics of a DCCP when it a connection is in progress. Write requests on a connecting DCCP socket may not, for example, block until the sending DCCPs buffer has filled.

3.2.3. Connection Termination

A DCCP must close both of its half connections as a pair. In lieu of closing a single DCCP half connection, an application may inform the DCCP that it no longer wishes to receive data. At which point the application should be allowed to retrieve all data in the DCCP's receive buffer. The DCCP should then drop all incoming data for that half connection, informing the sending end point of the buffer closed drop.

If a sending end point receives notification of a receive buffer drop, it should notify the application via some asynchronous notification or upon a subsequent write request. This way the sending application can be made aware that the receiving half connection can no longer accept data.

A closed receive buffer should be regarded as a terminal state for that half connection prior to closing the full DCCP connection. That is, the half connection may not return from a buffer closed state to a state in which it can receive more data without opening a new DCCP session.

3.3. Feature Negotiation

3.3.1. Negotiable Features

The following features require negotiation with the remote DCCP:

- Congestion Control (CC)
- ECN Capable
- Use Ack Vector
- Mobility Capable

- Certain CCID-Specific Features

3.3.2. Impact on Configuration Interfaces

A non-blocking API may need to implement a mechanism to determine the state of an options change request on a DCCP in the case that the options change request can not be handled immediately. The cases in which an options change request may be handled immediately are:

(1) The DCCP is not in connected, and therefore will not negotiate a feature change until a connection is initiated.

(2) The requested feature is non negotiable.

An implementation may chose to provide non-blocking semantics that parallel the semantics for a connect attempt. A request to perform an operation on a DCCP that is in an unknown state due to a pending feature negotiation may result in the return of an error code that indicates a feature change is in progress. The API may then provide a mechanism for asynchronous notification or polling of the status of the pending or completed feature change.

3.3.3. Feature options with data

DCCP does allow option change requests to contain application data that is scheduled to be transmitted. An implementation of DCCP may present a feature change mode where the feature change does not begin until there is user data to transmit along with the feature change request. The caveat for such a feature is that feature changes may suffer from undue latency if the time between application write requests is long.

3.3.4. Impact of feature negotiation on I/O semantics

The report of the success of a feature negotiation may depend on more than just the acceptable completion of the negotiation exchange. Outstanding data in the DCCP may need to be processed under the previous feature parameters before new data may be accepted from the application. This semantic is particularly relevant for CCID changes, in which the application may reasonably assume that all data sent prior to a CCID change will be sent according to the prior CCID. This issue will ultimately the determine the timing of reporting the completion of a feature negotiation to the application.

3.4. Error Codes

An API to DCCP may wish to expose error codes that represent the following conditions:

Connection In Progress

- Feature Change In Progress
- Feature Change Timed Out
- Feature Change failed due to lack of agreement
- Remote Buffer Closed
- Receive Buffer Empty
- Mobile Location Change Failed
- Connection Proof Failed
- Datagram Too Large for MTU

- Slow Receiver

3.5. Statistics

A DCCP implementation may maintain the following statistics for exposure for use by the application:

- PMTU
- Estimated RTT
- Estimated Buffer Depth
- Estimated inter arrival time of datagrams
- Estimated inter retrieval time of datagrams
- Estimated loss metric
- Estimated congestion controlled data rate
- Outstanding un-Acked data

4. Implementation Considerations

DCCP implementations will confront a number of common design choices. This section identifies and outlines those choices along with possible solution paths.

4.1. Connection State

DCCP provides a mechanism to support common connection semantics, while relieving the server of the need to keep state. This is achieved through an Init Cookie, which the server uses to encapsulate the original connection request, along with its response. The echo of this data upon completion of the three way hand shake allows the server to establish the connection with the correct parameters without holding state throughout the processing of the connection.

If the server wishes to avoid holding state during connection processing, then it also must be able to reject data payloads piggybacked with a DCCP-Request packet. Otherwise the server would have to allocate the requisite structures to manage the pending user data until the connection is established. These semantics leave open the issues of the way in which the Init Cookie ought to be implemented, and the way in which a client copes with a data discard event.

DCCPs which allow the application to piggyback data on to connection requests may have to handle cases where the server discards the piggybacked data because it does not want to hold state. This scenario may be quite common, since the stateless connection feature of DCCP may be widely implemented and used by applications. DCCPs may support the retransmission of data piggybacked on a DCCP-Request packet in a subsequent DCCP-Data packet once the connection is established. The concept of such retransmission, however, is contrary to DCCPs design as an unreliable protocol. A more appropriate implementation may provide a way to notify the application that a retransmit is required.

4.2. Provision of sequence number, and loss data to application

DCCP's maintenance of a sequence number space and awareness of lost data may leave room for optimizations with protocols that operate at layers above DCCP. Certain application layer protocols, for example, may require an awareness of lost data that is congruent with that maintained by DCCP and may desire to share DCCP's representation of this information.

There are, however, concerns with this approach. Failure to observe the layering of protocols may introduce semantical complexities to the API that could otherwise be avoided. Given that most DCCP implementations will be in kernel space, the frequency with which loss data or sequence number representation must be communicated across the kernel/user boundary becomes a reasonable consideration for scalability. The trade off for the elimination of redundant maintenance of sequence numbers is therefore not necessarily useful. Application layer implementations of DCCP may, however, find an approach to the sharing of sequence information to be viable.

4.3. Probabilistic Verification of Loss Rate using ECN for TFRC

TBD

4.4. Supporting Application Feature Preference

A full featured DCCP implementation may provide a set of negotiable features to the remote DCCP that the application may not wish to present. In such cases it will be useful for an DCCP implementation to maintain a table of features permitted by the application as per the application's configuration input. This table should then be consulted in response to DCCP feature change requests when a feature preference must be indicated.

4.5. Slow Receiver Interpretation and Notification

A DCCP receiver may notify its sender that the overlying application keeping up with the inbound data rate. In order to relieve the application of having to explicitly specify Slow Receiver to DCCP, an implementation may provide a heuristic to determine when to send Slow Receiver itself.

A simple heuristic would involve sending Slow Receiver once the DCCP's receive buffer is within some specified number of bytes of being full. The problem with this approach is that such a condition could represent a steady state behavior for the receive queue that is acceptable to the application.

How a sender reacts to a slow receiver notification will depend largely on behavior required by the application. For this reason, an implementation should employ a mechanism to inform the application that the sender is slow and for the application to indicate a course of action for the sending DCCP to follow. There should be no default behavior for the sender's response to a Slow Receiver notification, since an appropriate response is not dependent on information that is maintained by the transport layer.

5. Security Considerations

TBD

6. IANA Considerations

This document does not introduce any numbers for consideration by IANA.

7. References

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

[DCCP-PROBLEM] Sally Floyd, Mark Handley, Eddie Kohler. Problem Statment for DCCP. Work in progress.

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

8. Author's Address

Damon Lanphear <damonlan@real.com>

Media Systems Development
RealNetworks, Inc.
PO Box 91123
Seattle, WA 98111-91123
USA