DCCP Overview

Cui Lin, Seok J. Koh

{cuilin, sjkoh}@cs.knu.ac.kr

Communications Protocols Laboratory

Department of Computer Science

Kyungpook National University

Abstract

DCCP, the Datagram Congestion Control Protocol, is a new unreliable transport protocol standardizing in progress with end-to-end congestion control on bidirectional, unicast connections of datagrams.

It aims to add the minimal designed congestion control support to a UDP-like foundation, such as possibly-reliable transmission of acknowledgement information, so as to make DCCP suitable as a building block for more higher-level application semantics, such as streaming media, online games and internet telephony, etc.

We try to sketch out a short overview of Datagram Congestion Control Protocol (DCCP), and motivate the protocol and discuss some of its design principles here.

Table of Contents

1. Introduction

Over recent few years, there has been a steady growth of the applications with characteristics of long-lived, delay sensitive, preferring timeliness to reliability, such as online games, streaming media and internet telephony. These applications historically used UDP and implemented their own congestion control mechanisms or no congestion control at all. This lack of congestion control posed a threat to internet. If these congestion-unware applications' usage continued to further grow, the Internet might take a risk of congestion collapse. Under this situation, it is imperative to develop a new transport protocol, Datagram Congestion Control Protocol (DCCP), with minimal-kernel built-in congestion control mechanism in addition to UDP-like foundation.

Historically, the preexistence of DCCP is Datagram Control Protocol (DCP), which is first published in Jul, 2001 and evolved to DCCP in May, 2002. The newest internet draft about DCCP was published on Mar. 10, 2005.

DCCP aims to add the necessary lightweight mechanisms in kernel to support congestion control by possibly-reliable transmission of Acks and Acks-of-Acks, have as little overhead as possible both in the packet header size and in the CPU running at end hosts, only possesses necessary minimal functionalities, leaving other functionalities, such as forward error correction (FEC), semi-reliability, and multiple streams, to be layered on top of DCCP as desired. This minimal design should make DCCP suitable as a building block for more upper-layer application for transferring large amounts of data fairly.

Specifically, DCCP possesses the following primary features:

i A bidirectional, unicast connections of congestion-controlled, unreliable datagrams, with acknowledgements.

ii Reliable handshakes for connection setup and teardown.

iii Reliable negotiation of options (e.g. negotiation of a suitable CCIDs).

iv Two half-connections of the same connection can be governed by different CCIDs.

v Servers can avoid holding state for unacknowledged connection attempts and already-finished connections.

vi Congestion control incorporating Explicit Congestion Notification (ECN) and the ECN Nonce, as per [RFC 3168] and [RFC 3540].

vii Robust acknowledgement mechanisms which can feed the accurate information back to the sending application for events such as ECN marking, corruption and dropping in buffer.

viii Path Maximum Transfer Unit (PMTU) discovery, as per [RFC 1191].

ix A choice of modular congestion control mechanisms. Currently, there are two congestion control algorithms for DCCP, referred to by Congestion Control Identifiers (CCIDs). CCID2 is a TCP-like additive increase multiplicative decrease (AIMD) algorithm [CCID2]. CCID3 is an implementation of TCP-Friendly Rate Control (TFRC) [RFC 3448], [CCID3].

The main differences between DCCP and TCP are concluded as follows:

i DCCP is a packet stream protocol, not a byte stream protocol.

ii DCCP will never retransmit a datagram. Options are retransmitted as required to make feature negotiation and ack information reliable.

iii Sequence numbers refer to packets, not bytes. Every packet sent by a DCCP endpoint gets a new sequence number, even including pure acknowledgements.

iv Copious space for options (up to 1008 bytes or the PMTU) while TCP has only 40 bytes option space.

v Two half-connections can choose different CCIDs for congestion control while it is impossible in TCP.

vi Most robust acknowledgement scheme with Data Dropped option for distinguishing different kinds of loss.

vii Different acknowledgement rationale. In TCP, a packet may be acknowledged only once the data is reliably queued for application delivery. But in DCCP, the acknowledgeability does not guarantee data delivery, where A DCCP packet may be acknowledged as soon as its header has been successfully processed. However: the Data Dropped option may later report that the packet's application data was discarded because of a certain reason.

viii No receive window in DCCP. Because DCCP is a congestion control protocol, not a flow control protocol.

ix No simultaneous open. Every connection has one client and one server.

x No half-closed states in DCCP while it is possible in TCP.

xi Feature negotiation. This is a generic mechanism by which endpoints can agree on the values of "features", or properties of the connection. For example, the Ack Ratio feature lets HC-Senders influence the rate at which HC-Receivers generate DCCP-Ack packets, thus controlling reverse-path congestion. This differs from TCP, which presently has no congestion control for pure acknowledgement traffic.

xii 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.

2. Related Works

2.1 A VoIP variant of TFRC

TFRC was intended for applications that use a fixed packet size, and was designed to be reasonably fair when competing for bandwidth with TCP connections using the same packet size. Therefore, TFRC should not be used by applications that change their sending rate by varying the packet size, rather than varying the rate at which packets are sent. For that, a new CCID, VoIP variant, is being scandalized by IETF.

The VoIP variant of TFRC is designed for applications that send small packets and can achieve the same bandwidth in bps as a TCP flow using 1500-byte data packets. This variant is referred to in RFC 3448 as TFRC-PS, and might vary their packet size in response to congestion. In addition, the VoIP variant of TFRC enforces a Min Interval of 10 ms between data packets, to prevent a single flow from sending small packets arbitrarily frequently. But so far, there are no IANA considerations in related specifications.

2.2 Integrated Rate Control Scheme for TCP-friendly

An integrated rate control scheme has been proposed by C. H. Shih et al for MPEG-4 video transmission over TCP-friendly rate control (TFRC) protocol.

For the oscillation of TFRC transmit rate, a rate smooth algorithm is used to smooth the TFRC rate, and then sends the smoothed rate to the MPEG-4 rate control mechanism as shown in Fig 2-1. In the application layer, the MPEG-4 encoder obtains the request data rate every GOP (Group Of Pictures), and adaptively allocates the data rate of the I-frame to reduce the continuous skipping of the subsequent P-frames. The goal is to improve the temporal quality based on the requirement of dynamic data rate.

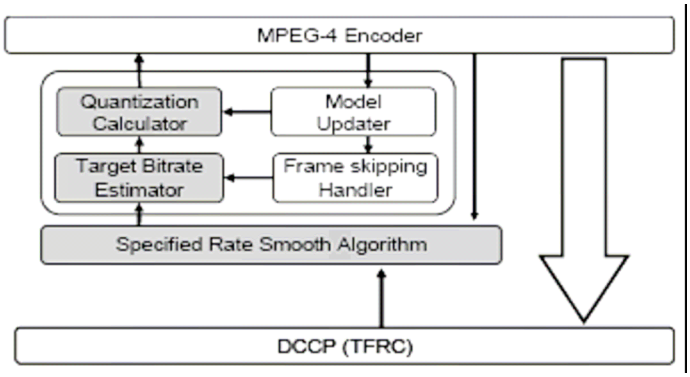

Fig. 2-1. Integrated rate control scheme.

2.3 Packet ring based DCCP API

Another efficient, flexible kernel DCCP API has also been developed by California University. The API uses a packet ring data structure, used for send control and synchronization, stored in memory shared between the application and the kernel. The application enqueues packets for transmission by putting them on the end of the packet ring,without bothering the kernel. When the kernel gets control, it can send as many packets as have been enqueued.

The shared-memory zero-copy design improves send performance by reducing data copies. Because it reduces control transfers, the packet ring also sends zero-length packets through the kernel faster than conventional UDP.

Simulation results show that applications using this API can achieve lower-latency transmissions than TCP, since DCCP doesn't retransmit or delay data to enforce ordering constraints, while remaining friendly to the network.

Simulation results also show that on a congested network, some packets sent using congestion-controlled DCCP can arrive at the receiver faster than packets sent using constant-bit-rate UDP, and an MPEG-like application using this DCCP API can deliver more than twice as many important “I-frames" than a UDP sender in the same network conditions.

3. DCCP Packets

Currently, ten packet types implement DCCP's protocol functions, referring to Table 3-1, the first eight packet types occur during the progress of a typical connection, and the two remaining packet types are used to resynchronize after bursts of loss.

Wherein, DCCP-Ack, DCCP-Close, DCCP-CloseReq, DCCP-Reset, DCCP-Sync, and DCCP-SyncAck are classified into Non-data packets, and DCCP-Request, DCCP-Response, DCCP-Data, and DCCP-DataAck are classified into data packets, although not all DCCP-Request and DCCP-Response packets will actually carry application data.

Table 3-1. The overview of packet types.

type	Name	Purpose
0	DCCP-Request	Sent by the client to initiate a connection (the first part of the three-way initiation handshake).
1	DCCP-Response	Sent by the server in response to a DCCP-Request (the second part of the three-way initiation handshake).
2	DCCP-Data	Used to transmit application data.
3	DCCP-Ack	Used to transmit pure acknowledgements.
4	DCCP-DataAck	Used to transmit application data with piggybacked acknowledgements.
5	DCCP-CloseReq	Sent by the server to request that the client close the connection.
6	DCCP-Close	Used by the client or the server to close the connection; elicits a DCCP-Reset in response.
7	DCCP-Reset	Used to terminate the connection, either normally or abnormally.
8	DCCP-Sync	Used to resynchronize sequence numbers after large bursts of loss.
9	DCCP-SyncAck	Used to resynchronize sequence numbers after large bursts of loss.
10-15	Reserved	

3.1 Packet Format

All DCCP packets start with a 12 or 16 bytes sized generic header unconditionally, following this comes any additional fixed-length fields and option field required by the particular packet type, and every particular packet types include their own fixed-size header data. Hence different type's packets can have different packet header size.

The Fig. 3-1 sketches the profile of DCCP packets:

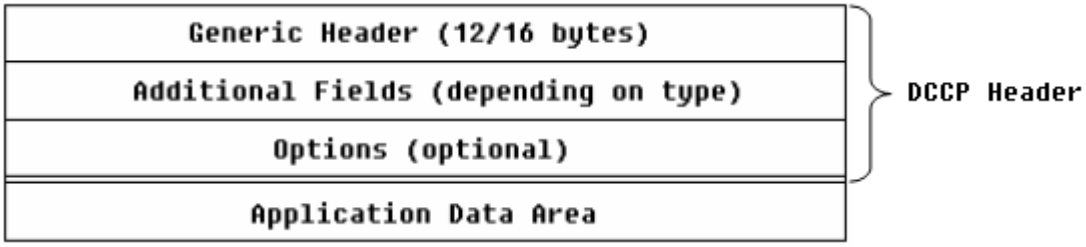

Fig. 3-1. The profile of DCCP packets.

Generally, additional fields carry the following information:

i Acknowledgement Number in 4/8 bytes (except DCCP-Request and DCCP-Data).

ii Service code (DCCP-Request, DCCP=Response).

iii Reset code & data 1/2/3 fields (DCCP-Reset).

Moreover, the application data field also can carry the Error Text in a human-readable text by the DCCP-Reset packets.

3.2 Generic Header

The DCCP generic header takes different forms depending on the value of X, the Extended Sequence Numbers bit. If X is one, the Sequence Number field is 48 bits long and the generic header takes 16 bytes (See Fig. 3-2). On the contrary, If X is zero, only the low 24 bits of the Sequence Number are transmitted, and the generic header is 12 bytes long (See fig. 3-3).

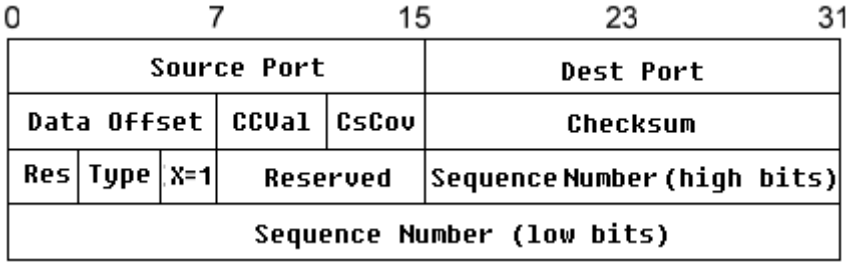

Fig. 3-2. The generic header with X=1

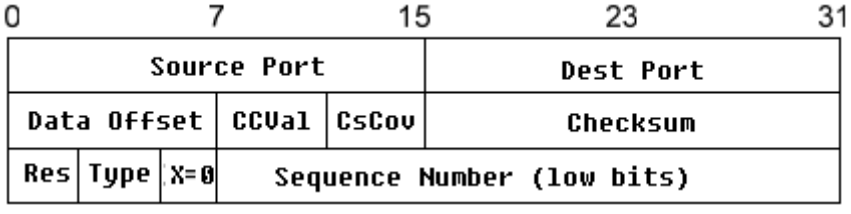

Fig. 3-3. The generic header with X=0

In detail, the generic header fields are defined as follows:

Table 3-2. Definition of every field in DCCP generic header.

Fields	Length	Comment
Source port & Destination Port	16 bits each	Similar with TCP & UDP
Data Offset	8 bits	The offset from the start of the packet's DCCP header to the start of its application data area, in 32-bit words.
CCVal	4 bits	Used by the HC-Sender CCID.
CsCov	4 bits	Indicates different Checksum Coverage.
CheckSum	16 bits	The Internet checksum, its coverage range depends on the value of CsCov.
Res	3 bits	Reserved, senders MUST set this field to all zeroes.
Type	4 bits	Referring to Table 2-1 in detail.
X	1 bit	1 indicates the use of an extended generic header with 48-bit Sequence and Acknowledgement Numbers, while 0 indicates the use of the only low 24-bit Sequence and Acknowledgement Numbers. DCCP-Data, DCCP-DataAck, and DCCP-Ack packets MAY set X to zero or one. The rest 7 type packets must set X to one (Endpoints MUST ignore any such packets with X set to zero). High-rate connections SHOULD set X to one on all packets to gain increased protection against attacks.
Sequence Number	48/24 bits	Depending on the value of X.

3.3 DCCP-Request Packet

A client initiates a DCCP connection by sending a DCCP-Request packet. These packets may contain application data, and MUST use 48-bit sequence numbers (X=1). The 32-bit Service Code field describes the application-level service to which the client application wants to connect.

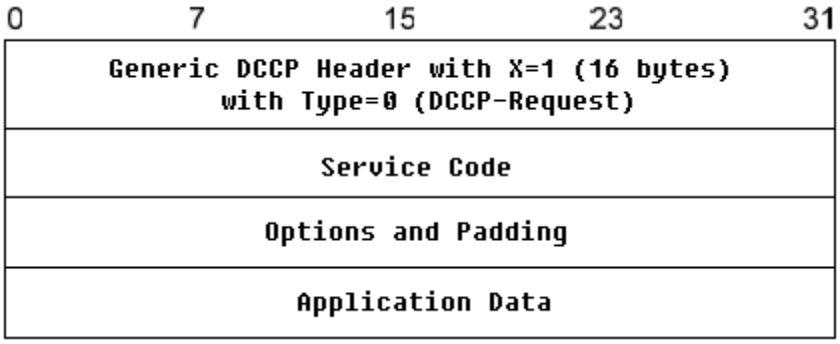

Fig. 3-4. DCCP-Request packet's format

3.4 DCCP-Response Packet

The server responds to valid DCCP-Request packets with DCCP-Response packets. This is the second phase of the three-way handshake. DCCP-Response packets also may contain application data, and must use 48-bit sequence numbers (X=1). 48-bit Acknowledgement Number must equal the Sequence Number on a received DCCP-Request, and 32-bit Service Code must equal the Service Code on the corresponding DCCP-Request.

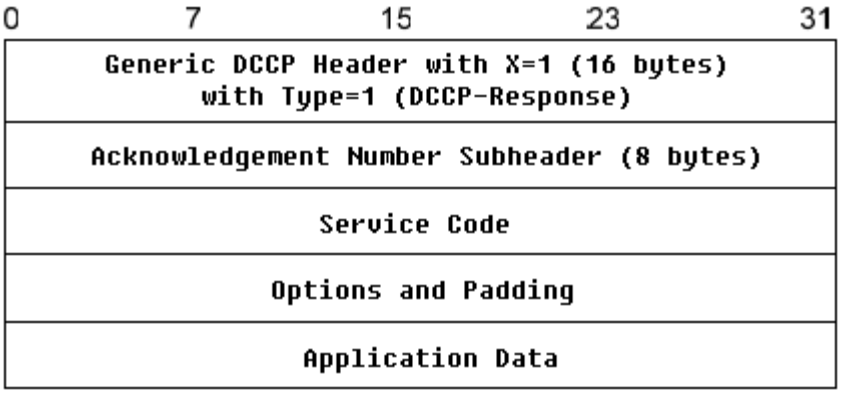

Fig. 3-5. DCCP-Response packet's format

3.5 DCCP-Data, DCCP-Ack, and DCCP-DataAck Packets

DCCP-Data, DCCP-Ack, and DCCP-DataAck packets are used to transfer application data and acknowledgement. They may use either 24-bit or 48-bit sequence / acknowledgement numbers depending on the value of the X field. Wherein, DCCP-Data packets carry only application data without acknowledgements, and DCCP-Ack is in contrary to DCCP-Data. Specifically, DCCP-DataAck packets carry both application data and an Acknowledgement Number in the manner of acknowledgement information being piggybacked on a data packet.

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

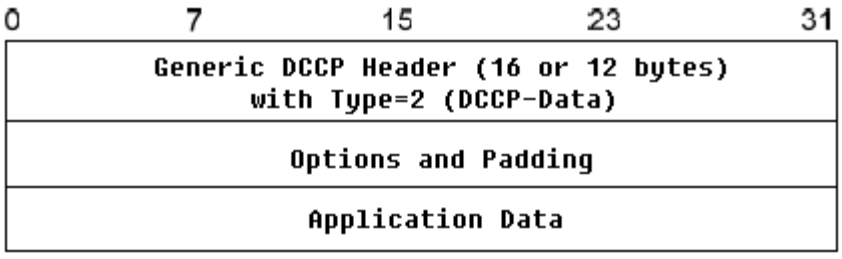

Fig. 3-6. DCCP-Data packet's format

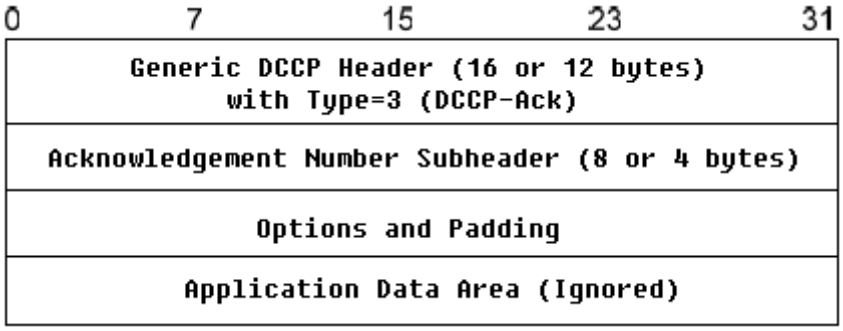

Fig. 3-7. DCCP-Ack packet's format

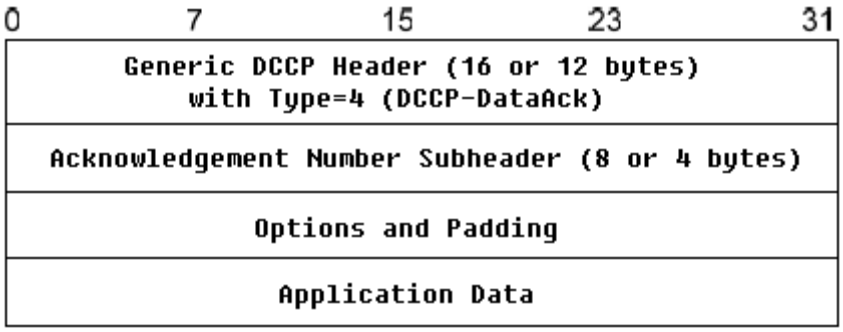

Fig. 3-8. DCCP-DataAck packet's format

3.6 DCCP-CloseReq and DCCP-Close Packets

DCCP-CloseReq and DCCP-Close packets begin the handshake that normally terminates a connection. Only the server can send a DCCP-CloseReq packet, which indicates that the server wants to close the connection, but does not want to hold its TIMEWAIT state. Either client or server may send a DCCP-Close packet, which will elicit a DCCP-Reset packet. Both packet types must use 48-bit sequence numbers (X=1).

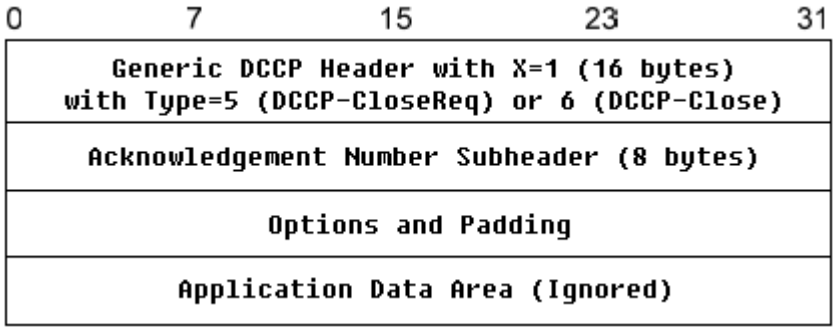

Fig. 3-9. DCCP-CloseReq & DCCP-Close packets' format

3.7 DCCP-Reset Packets

DCCP-Reset packets unconditionally shut down a connection and it must use 48-bit sequence numbers (X=1). In detail, Reset Code represents the reset reason in manner of number, and Data 1, 2, 3 fields provide additional information about why. Particularly, DCCP-Reset packets carry the reset reason in a human-readable text in application data area.

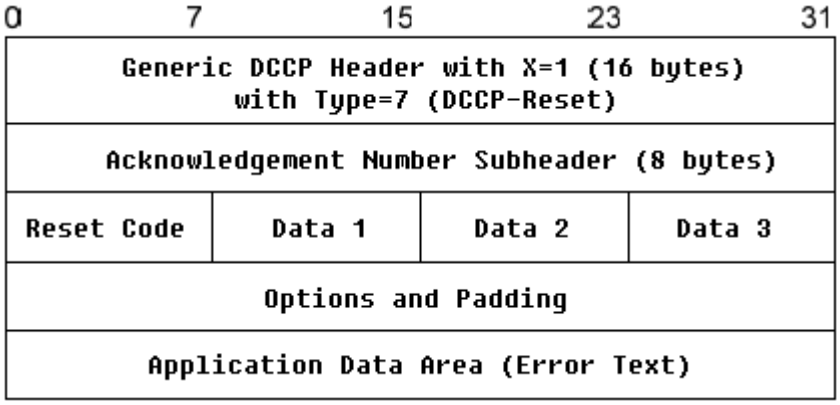

Fig. 3-10. DCCP-Reset packet's format

Table 3-3. Definition of Reset Codes and corresponding Data 1/2/3 fields.

Reset Code	Name	Data 1	Data 2 & 3
0	Unspecified	0	0
1	Closed	0	0
2	Aborted	0	0
3	No Connection	0	0
4	Packet Error	pkt type	0
5	Option Error	option type	option type
6	Mandatory Error	option type	option type

7	Connection Refused	0	0
8	Bad Service Code	0	0
9	Too Busy	0	0
10	Bad Init Cookie	0	0
11	Aggression Penalty	0	0
12-127	Reserved		
128-255	CCID-specific codes		

3.8 DCCP-Sync and DCCP-SyncAck Packets

DCCP-Sync packets help DCCP endpoints recover synchronization after bursts of loss, or recover from half-open connections. Each valid received DCCP-Sync immediately elicits a DCCP-SyncAck. Both packet types must use 48-bit sequence numbers (X=1).

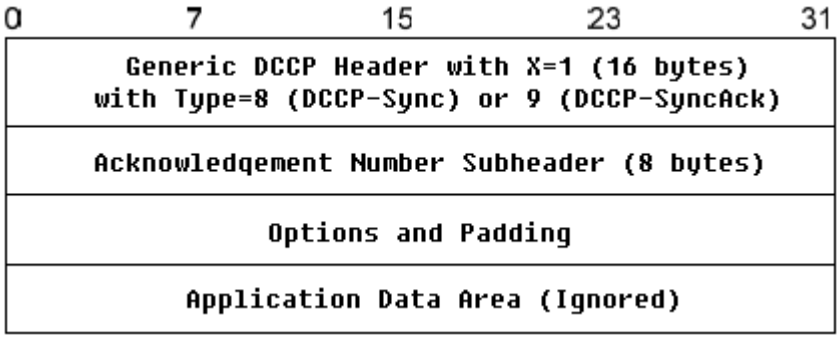

Fig. 3-11. DCCP-Sync and DCCP-SyncAck packets' format

4. Options and Features

4.1 Options

Any DCCP packet may contain options, which occupy space at the end of the DCCP header. Each option is a multiple of 8 bits in length and all options must add up to a multiple of 32 bits (If necessary, by padding options). Any options present are included in the header checksum.

Options are processed sequentially, starting at the first option in the packet header. Not all options are suitable for all packet types, for example, the Ack Vector option isn't suitable on DCCP-Request and DCCP-Data packets since they have no Acknowledgement Number.

Table 4-1. Current definition of options.

Type	Option length	Meaning	DCCP-Data?
0	1	Padding	Y
1	1	Mandatory	N
2	1	Slow Receiver	Y
3-31	1	Reserved	
32	Variable	Change L	N
33	Variable	Confirm L	N
34	Variable	Change R	N
35	Variable	Confirm R	N
36	Variable	Init Cookie	N
37	3-5	NDP Count	Y
38	Variable	Ack Vector [Nonce 0]	N
39	Variable	Ack Vector [Nonce 1]	N
40	Variable	Data Dropped	N
41	6	Timestamp	Y
42	6/8/10	Timestamp Echo	Y
43	4/6	Elapsed Time	N
44	6	Data Checksum	Y
45-127	Variable	Reserved	
128-255	variable	CCID-specific options	-

4.2 Features

From type 32 to 35 in table 4-1, four DCCP options are used to negotiate feature values, they are Change L, Confirm L, Change R, and Confirm R. Wherein, Change options initiate a negotiation; Confirm options complete that negotiation. The "L" indicates the option is sent by the feature location, and the "R" indicates the option is sent by the feature remote. In addition, Change options are retransmitted to ensure reliability.

All current DCCP features use one of two reconciliation rules, server-priority ("SP") and non-negotiable ("NN"). Server-priority feature option contains a list of values ordered by preference, with the most preferred value coming first, while non-negotiable feature option contains exactly one feature value, giving both format figures in the following:

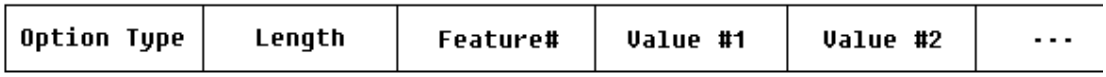

Fig. 4-1. The format of server-priority feature option

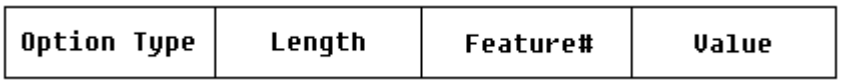

Fig. 4-2. The format of non-negotiable feature option

In the manner of server-priority, if there is no shared entry, the feature's value must not change, and the Confirm option will confirm the feature's previous value unless the Change option was Mandatory.

If an endpoint receives an invalid Change option with an invalid value, the other endpoint will respond with an empty Confirm option containing the problematic feature number, but no value. Such options have length 3.

Moreover, Change R and Confirm L options must not be sent for non-negotiable features.

Table 4-2. DCCP Feature Numbers.

Number	Meaning	Reconciliation rule	Initial Value	Well-known (Y) /Extension (N)
0	Reserved			
1	Congestion Control ID (CCID)	SP	2	Y
2	Allow Short Seqnos	SP	1	Y
3	Sequence Window	NN	100	Y
4	ECN Incapable	SP	0	N
5	Ack Ratio	NN	2	N
6	Send Ack Vector	SP	0	N
7	Send NDP Count	SP	0	N
8	Minimum Checksum Coverage	SP	0	N
9	Check Data Checksum	SP	0	N
10-127	Reserved			
128-255	CCID-specific features			

Feature state transitions at a feature location are implemented according to the following diagram:

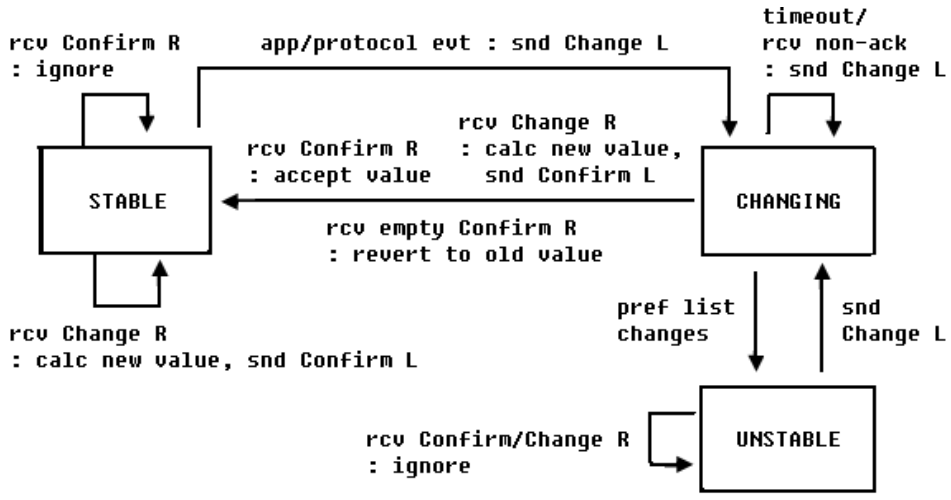

Fig. 4-3. The processing procedure after receiving options.

5. Sequence Number

DCCP sequence numbers are packet-based. That is, sequence numbers increase by one, modulo 2^48, for every packet, whether or not it contains application data. This lets DCCP implementations detect network duplication, retransmissions, and acknowledgement loss, and is a significant departure from TCP practice.

5.1 Variables

DCCP endpoints maintain a set of sequence number variables for each connection, they are:

ISS: The Initial Sequence Number sent by this endpoint.

ISR: The Initial Sequence Number received from the other endpoint.

GSS: The Greatest Sequence Number sent by this endpoint.

GSR: The Greatest Sequence Number received from the other endpoint on an acknowledgeable packet.

GAR: The Greatest Acknowledgement Number received from the other endpoint on an acknowledgeable packet that was not a DCCP-Sync.

SWL and SWH: Sequence Number Window low and high (The extremes of the validity window for received packets' Sequence Numbers).

AWL and AWH: Acknowledgement Number Window low and high (The extremes of the validity window for received packets' Acknowledgement Numbers).

5.2 Initial Sequence Numbers

The endpoints' initial sequence numbers are set by the first DCCP-Request and DCCP-Response packets sent. By using TCP's strategies described in RFC 793, Initial sequence numbers must be chosen to avoid two problems: Delivery of old packets and Sequence number attacks.

5.3 Validity and Synchronization

DCCP, like TCP, uses sequence number checks to detect those invalid packets whose Sequence and/or Acknowledgement Numbers are out of Sequence Number validity window or Acknowledgement Number validity window respectively. The profile of both windows are provided in the following, where W is the value of the Sequence Window/B feature, W' is the value of the Sequence Window/A feature.

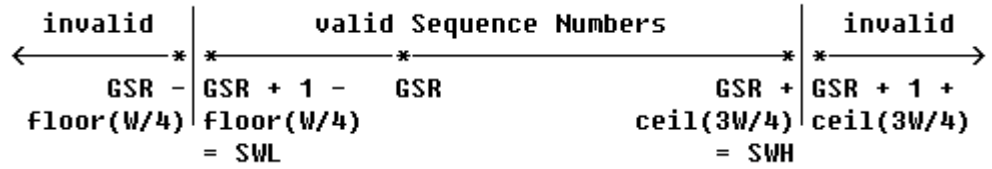

Fig. 5-1. The figure of Sequence Number validity window.

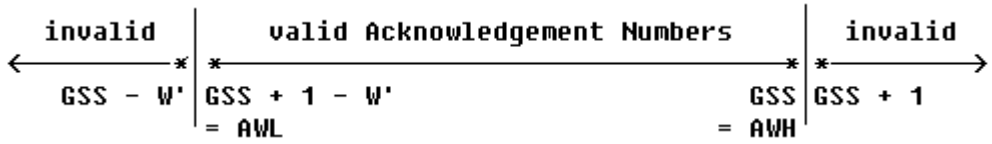

Fig. 5-2. The figure of Acknowledgement Number validity window.

SWL and AWL are initially adjusted so that they are not less than the initial Sequence Numbers received and sent, respectively:

SWL := max(GSR + 1 - floor(W/4), ISR),

AWL := max(GSS - W' + 1, ISS).

Unlike TCP, DCCP requires a synchronization mechanism to recover from large bursts of loss. One endpoint might send so many packets during a burst of loss that when one of its packets finally got through, the other endpoint would label its Sequence Number as invalid. A handshake of DCCP-Sync and DCCP-SyncAck packets recovers from this case.

5.4 Short Sequence Numbers

DCCP sequence numbers normally are 48 bits long. However, DCCP-Data, DCCP-Ack, and DCCP-DataAck packets, which make up the body of any DCCP connection, may reduce header space by transmitting only the lower 24 bits of the relevant Sequence and Acknowledgement Numbers. The receiving endpoint will extend these numbers to 48 bits. But Short sequence numbers increase the risks of certain kinds of attacks, including blind data injection.

5.5 NDP Count and Detecting Application Loss

DCCP sequence number scheme is suitable for detecting any network loss, but not for detecting the loss of application data. The NDP Count option reports the length of each burst of non-data packets according to the following format. This lets the receiving DCCP reliably determine whether a burst of loss included application data or not.

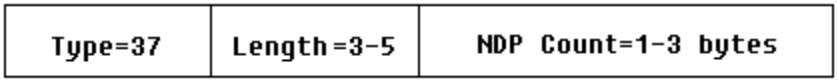

Fig. 5-3. The format of NDP Count option.

The value stored in NDP Count equals the number of consecutive non-data packets in the run immediately previous to the current packet just as the following example shown in Fig. 5-4:

N0	N1	D2	N3	D4	D5	N6	D7	D8	D9	D10	N11	N12	D13
-	1	2	-	1	-	-	1	-	-	-	-	1	2

Fig. 5-4. An example of NDP Count.

Where, Nx indicates non-data packets and Dx indicates data packets.

Say that K is the number of consecutive missing sequence numbers in a burst of loss, Then some application data was lost within those sequence numbers if the value of first NDP Count option after that hole is smaller than k.

6. Event Processing

6.1 Connection Establishment

DCCP connections' initiation phase consists of a three-way handshake, and the whole establishment phase is shown in Fig. 6-1.

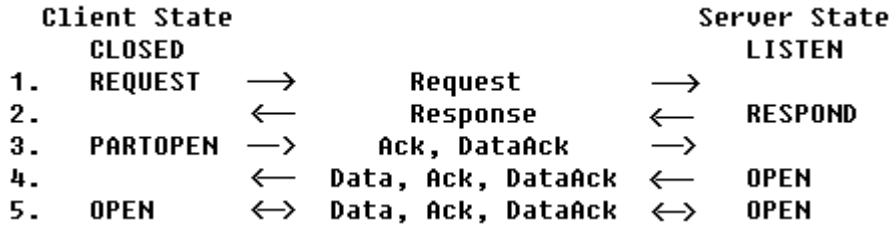

Fig. 6-1. The procedure of connection establishment.

In Request state, a client use an exponential-backoff timer to send new DCCP-Request packets if no response is received. The first interval between retransmissions is approximately one second, backing off to not less than one packet every 64 seconds. If no response after some time (3 minutes, for example), the client will give up.

In addition, each DCCP-Request contains a 32-bit Service Code, which corresponds to application services and protocols for choosing appropriate CCIDs by server.

In server side, Init Cookie option is used to avoid holding any state until the three-way handshake has completed. The Init Cookie option must not be sent on DCCP-Request or DCCP-Data packets, and its size are limited to at most 253 bytes in length.

6.2 Termination

DCCP connection termination uses a handshake consisting of an optional DCCP-CloseReq packet, a DCCP-Close packet, and a DCCP-Reset packet as shown in Fig. 6-2.

Fig. 6-2. The procedure of normal connection termination.

The client also can decide to close the connection like this:

Fig. 6-3. The procedure of connection being closed by client.

Once the server decides to hold TIMEWAIT state, then there is no necessary to send DCCP-CloseReq packet like this:

Fig. 6-4. The procedure of server holding TIMEWAIT state.

In TIMEWAIT state, an endpoint quietly preserves a socket for 2MSL (4 minutes) after its connection has closed for ensuring that no connection duplicating the current connection's source and destination addresses and ports can start up while old packets might remain in the network.

6.3 DCCP State Diagram

The common state transitions can be summarized in the following state diagram.

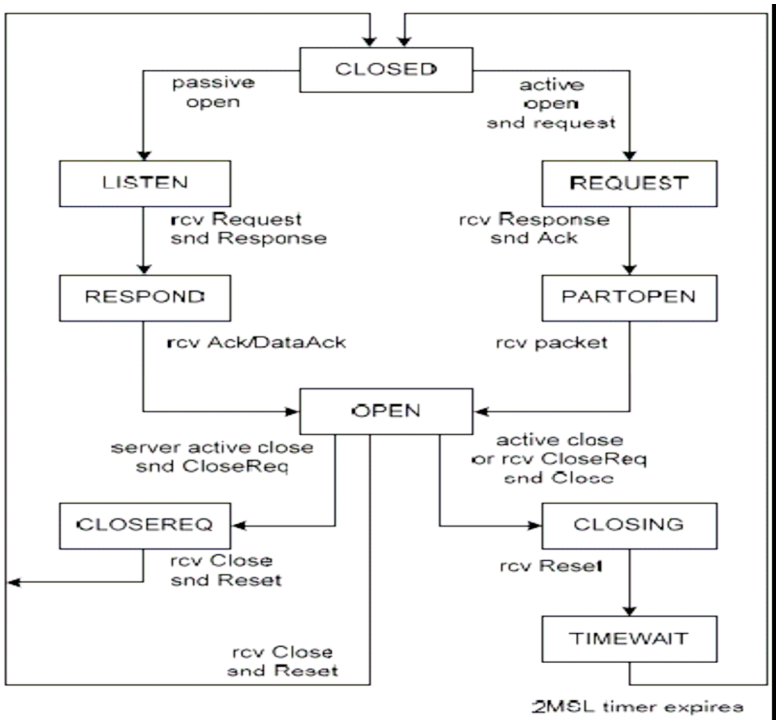

Fig. 6-5. DCCP State Diagram.

7. Checksums

7.1 Header Checksum

DCCP uses the TCP/IP checksum algorithm. The Checksum field in the DCCP generic header equals the 16 bit one's complement of the one's complement sum of all 16 bit words in the DCCP header, DCCP options, a network-layer pseudoheader, and some or all of the application data (depending on the value of the Checksum Coverage (CsCov), if CsCov = 0, all application data will be covered; if 1, then none of the application data is protected by the header checksum; otherwise, only the initial (CsCov-1)*4 bytes of the packet's application data will be included. The value of CsCov must be never bigger than 15).

Generally, this checksum will cover all application data by default. However, either endpoint can send The Minimum Checksum Coverage feature to ask its peer whether to accept packets with reduced Checksum Coverage. Minimum Checksum Coverage has feature number 8, and is server-priority.

As calculating the checksum, the Checksum field itself is treated as 0. If a packet contains an odd number of header and payload bytes to be checksummed, 8 zero bits are added on the right to form a

16 bit word for checksum purposes. The pad byte is not transmitted as part of the packet.

7.2 Data Checksum Option

There is an additional Data Checksum Option in addition to header checksum. The Data Checksum option holds a 32-bit CRC-32c cyclic redundancy-check code of a DCCP packet's application data.

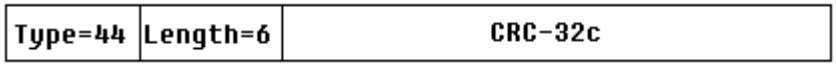

Fig. 7-1. The format of data checksum option.

The sending DCCP computes the CRC of the bytes comprising the application data area and stores it in the data checksum option field. The CRC-32c algorithm used for Data Checksum is the same as that used for SCTP.

8. Congestion Control

In DCCP, two half-connections can be governed by different congestion control mechanisms, and each mechanism is denoted by one-byte congestion control identifier, or CCID: a number from 0 to 255. Currently allocated CCIDs are as follows.

Table 8-1. DCCP Congestion Control Identifiers.

CCID	Meaning
0-1	Reserved
2	TCP-like Congestion Control
3	TFRC Congestion Control
4-255	Reserved

CCID is a server-priority feature. New connections start with CCID 2 for both endpoints. If unacceptable, must send mandatory Change(CCID) options on its first packets.

8.1 TCP-like Congestion Control

CCID 2, TCP-like Congestion Control, denotes Additive Increase, Multiplicative Decrease (AIMD) congestion control mechanism, including congestion window, slow start, timeout, etc. Its behavior modeled directly on SACK-based TCP. A CCID2 data sender maintains three integer parameters

measured in packets: cwnd, the congestion window; ssthresh, the slow-start threshold; and pipe. These three parameters have the same meaning as that in SACK-based TCP except that they are measured in packets, not bytes. Sender halves its cwnd in response to each congestion event which leads to the abrupt rate changes. On the other hand, DCCP never retransmits data, this differs from TCP. Moreover, ACK Ratio feature is implied to adjust ACK sending rate so as to achieve the congestion control to acknowledgement.

CCJD 2 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 should use CCID 2 if they prefer maximum bandwidth utilization to steadiness of rate, for example, On-line game.

In CCID 2, the sender determines the current allowed sending rate and congestion state according to the size of congestion window and the measurement of the number of packets outstanding in the network in addition to an estimate of the round-trip time.

The CCID2 has the following features:

(1) Sender maintains a congestion window and sends packets until that window is full.

(2) One ack per 2 packets by default.

(3) ACK declares exactly which packets were received.

(4) Dropped packets and ECN [RFC 3168] indicate congestion.

(5) Response to congestion is to halve the congestion window (AIMD).

(6) Acknowledgements in CCID 2 contain the sequence numbers of all received packets within some window, similar to a selective acknowledgement (SACK) [RFC 2018].

8.2 TFRC Congestion Control

CCID 3 denotes TCP-Friendly Rate Control (TFRC), an equation-based rate-controlled congestion control mechanism which is designed to be reasonably fair when competing for bandwidth with TCP-like flows. TFRC measures loss rates by estimating the loss event ratio as described in [RFC 3448], and uses this loss event rate to determine the sending rate in packets per round-trip time. The sender starts in a slow-start phase, roughly doubling its allowed sending rate each round-trip time. After the slow-start phase is ended by the receiver's report of a packet drop or mark, the sender calculates the allowed sending rate based on the round-trip time and on the loss event rate or equivalent information reported by the receiver. The feedback packets from the receiver contain a Receive Rate option specifying the rate at which data packets were received by the receiver since the last feedback packet. The allowed sending rate can be at most twice the rate that the receiver received in the last round-trip

time.

TFRC has a much smooth throughput over time compared with TCP, it leads to somewhat difference from TCP in the short term. Therefore, CCID 3 is appropriate for flows that would prefer to minimize abrupt changes in the sending rate, such as streaming media applications where a relatively smooth sending rate is of importance.

9. Acknowledgements

All currently defined packet types except DCCP-Request and DCCP-Data carry an Acknowledgement Number Subheader in the four or eight bytes immediately following the generic header. When X=1, Acknowledgement Number takes 48 bits and 16-bit Reserved field must be set all zeros likes Fig. 9-1:

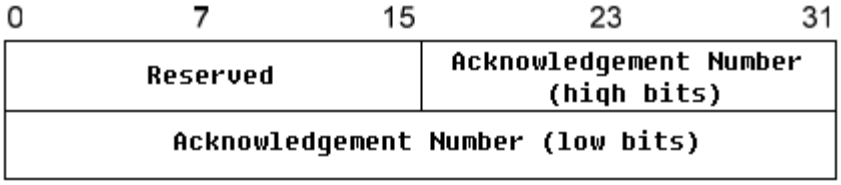

Fig. 9-1. The format of Acknowledgement field with X=1

When X=0, only the low 24 bits of the Acknowledgement Number with all zeros' 8-bit Reserved field are transmitted, giving the Acknowledgement Number Subheader format:

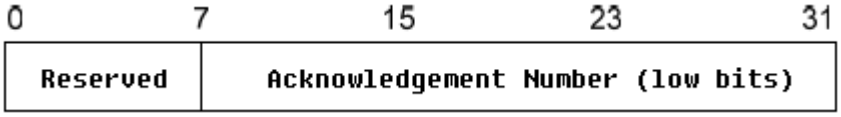

Fig. 9-2. The format of Acknowledgement field with X=0

Differing from TCP, the Acknowledgement Number must equal GSR, the Greatest Sequence Number, and most acknowledgements use additional most robust Ack Vector options in the mean time.

9.1 Acks of Acks and Unidirectional Connections

CCID2 uses a reliable Ack but not reliable Ack-of-Ack scheme. Hence the HC-Sender should occasionally inform the HC-Receiver that it has received an ack. If it did not, the HC-Receiver might resend complete Ack Vector information, going back to the start of the connection, with every DCCP-Ack packet! However, acks-of-acks need not be reliable themselves: when an ack-of-acks is lost,

the HC-Receiver will simply maintain, and periodically retransmit, old acknowledgement-related state for a little longer. Therefore, there is no need for acks-of-acks-of-acks.

In contrast, in CCID 3, acknowledgements generally need not be reliable, since they contain cumulative loss rates; TFRC works even if every DCCP-Ack is lost. Therefore, a DCCP sender need never acknowledge an acknowledgement.

9.2 Ack Piggybacking

Acknowledgements can be piggybacked on data to become a DCCP-DataAck packet, as long as that does not delay the acknowledgement longer than the A-to-B CCID would find acceptable.

However, data acknowledgements often require more than 4 bytes to express. A large set of acknowledgements appended to a large data packet might exceed the allowed maximum packet size. In this case, DCCP B can send separate DCCP-Data and DCCP-Ack packets, or wait, but not too long, for a smaller datagram.

9.3 Ack Ratio Feature

In DCCP, HC-Senders can control the rate of DCCP-Ack sent by HC-Receiver through The Ack Ratio feature, thus control reverse-path congestion. Higher Ack Ratios correspond to lower DCCP-Ack rates; the sender raises Ack Ratio when the reverse path is congested and lowers Ack Ratio when it is not.

CCID 2 usually uses Ack Ratio for acknowledgement congestion control, but CCID 3 does not. However, each Ack Ratio feature has a value whether or not that value is used by the relevant CCID.

9.4 Ack Vector

An HC-Receiver using CCID 2, TCP-like Congestion Control, sends Ack Vectors containing completely reliable acknowledgement information. In Ack vector option, The vector itself consists of a series of bytes, and each byte of the vector gives the number of consecutive packets with the same state looks like this:

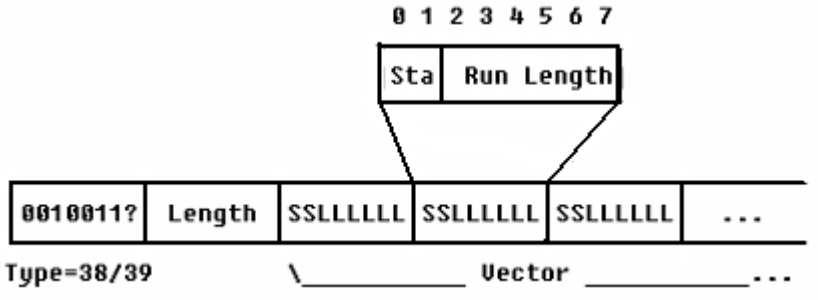

Fig. 9-3. The format of Ack Vector option.

Wherein, Sta[te] occupies the most significant two bits of each byte, and can have one of four values, as follows.

Table 9-1. DCCP Ack Vector States.

Sta[te]	Meaning
0	Received
1	Received ECN Marked
2	Reserved
3	Not Yet Received

Run Length, the least significant six bits of each byte, specifies how many consecutive packets have the given State. Run Length zero says the corresponding State applies to one packet only; Run Length 63 says it applies to 64 consecutive packets. Run lengths of 65 or more must be encoded in multiple bytes.

The first byte in the first Ack Vector option refers to the packet indicated in the Acknowledgement Number; subsequent bytes refer to older packets. (Ack Vector must not be sent on DCCP-Data and DCCP-Request packets, which lack an Acknowledgement Number.)

A single Ack Vector option can acknowledge up to 16192 data packets. If need, multiple Ack Vector options can be sent.

The Ack Vector will cover all packets which have been received by receiver but not yet received corresponding Acks-of-Acks.

9.5 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. Slow Receiver is a one-byte option, and option type number is 2.

9.6 Data Dropped Option

The Data Dropped option indicates that some received application data did not actually reach the application and why. Using Data Dropped, DCCP endpoints can discriminate between different kinds of loss. The format of data dropped option is similar with Ack Vector's, the Vector also consists of a series of bytes, called Blocks.

Drop Blocks, which have high bit 1, indicate that received packets in the Run Len[gth] were not delivered as usual. The 3-bit Drop Code [DrpCd] field says what happened.

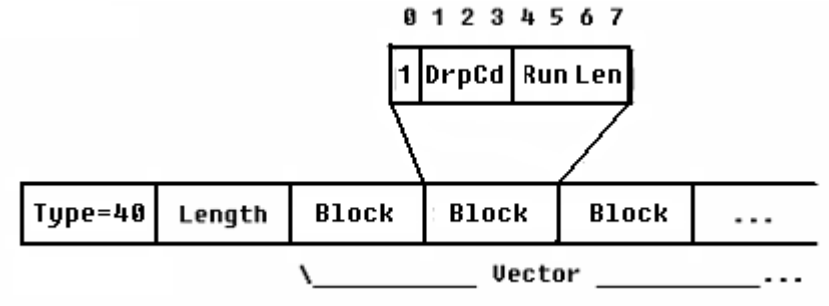

Fig. 9-4. The format of Data Dropped Option.

Wherein, DrpCd (Drop Code) has one of eight values, as follows.

Table 9-1. DCCP Drop Codes.

Drop Code	Meaning
0	Protocol Constraints
1	Application Not Listening
2	Receive Buffer
3	Corrupt
4-6	Reserved
7	Delivered Corrupt

In contrast, Normal Blocks, which have high bit 0, indicate that any received packets in the Run Length had their data delivered to the application normally, and left 7 bits indicates the Run Length.

10. Explicit Congestion Notification

The DCCP protocol is fully ECN-aware [RFC 3168] and DCCP endpoints are ECN-aware by default. But the ECN Incapable feature lets an endpoint reject the use of Explicit Congestion Notification. The use of this feature is not recommended. ECN Incapable has feature number 4, and is server-priority. It takes one-byte Boolean values.

10.1 ECN Nonces and Aggression Penalties

For preventing the receiver to falsify acknowledgement information in order to avoid halving congestion window, ECN Nonce mechanism are used to prevent ECN cheating (or loss cheating). Two values for the two-bit ECN header field indicate ECN-Capable transport, 01 and 10. The second code point, 10, is the ECN Nonce. In DCCP, the ECN Nonce Echo field is encoded in acknowledgement options.

In general, a protocol sender chooses between these code points randomly on its output packets, remembering the sequence it chose. The protocol receiver reports, on every acknowledgement, the number of ECN Nonces it has received thus far. This is called the ECN Nonce Echo.

Since there are only two code points, a receiver that lies about an ECN mark or packet drop has a 50% chance of guessing right and avoiding discipline. But once misbehavior is verified by sender, receiver will Incur "Aggression Penalty".

11. Timing Options

The Timestamp, Timestamp Echo, and Elapsed Time options help DCCP endpoints explicitly measure round-trip times. These options are permitted in any DCCP packet. Timestamp Value is a 32-bit integer that increases monotonically with time, at a rate of 1 unit per 10 microseconds. At this rate, Timestamp Value will wrap approximately every 11.9 hours.

Elapsed Time option indicates how much time has elapsed, in hundredths of milliseconds, since the packet being acknowledged was received. The option may take 4 or 6 bytes, depending on the size of the Elapsed Time value.

Generally, a DCCP endpoint should send one Timestamp Echo option for each Timestamp option it receives as soon as convenient. The first four bytes of option data, Timestamp Echo, carry a Timestamp Value taken from a preceding received Timestamp option. The Elapsed Time value, similar to that in the Elapsed Time option, indicates the amount of time elapsed since receiving the packet whose

timestamp is being echoed. A missing Elapsed Time field is equivalent to an Elapsed Time of zero.

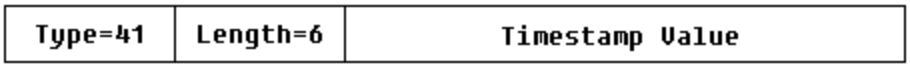

Fig. 11-1. The format of Timestamp Option.

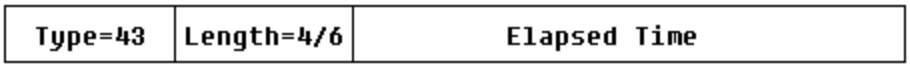

Fig. 11-2. The format of Elapsed Time Option.

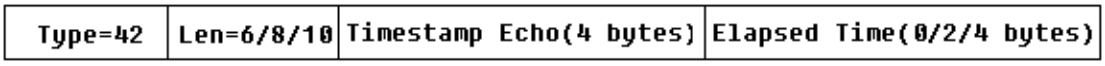

Fig. 11-3. The format of Timestamp Echo Option.

12. Maximum Packet Size

A DCCP implementation maintains the maximum packet size (MPS) allowed for each active DCCP session, those packets whose sizes are larger than CCMPS must be rejected in any case. However, if application insists, fragmentation can be used in this case.

Application can occasionally request PMTU discovery process again. This will reset the PMTU to the outgoing interface's MTU. Such requests should be rate limited, to one per two seconds, for example.

DCCP-Sync packets are used to upward probe of the PMTU [PMTUD], where the DCCP endpoint begins by sending small packets with DF set, then gradually increases the packet size until a packet is lost, since DCCP-Sync probes do not risk application data loss. The DCCP implementation inserts arbitrary data into the DCCP-Sync application area, padding the packet to the right length; and since every valid DCCP-Sync generates an immediate DCCP-SyncAck in response, the endpoint will have a pretty good idea of when a probe is lost.

It is undesirable for PMTU discovery to occur on the initial connection setup handshake in which packet sizes may not be representative and it will unnecessarily delay connection establishment. Typically, 1500-byte is usually used as DCCP packet size.

13. Considerations

13.1 Middlebox Considerations

The Service Code field in DCCP-Request packets provides useful information to middleboxes. With Service Code, a middlebox can tell what protocol a connection will use without relying on port

numbers. Middleboxes can disallow connections that attempt to access unexpected services by sending a DCCP-Reset with Reset Code 8, "Bad Service Code".

Middleboxes should not modify packet streams by adding or removing packets, it equal to modify DCCP Sequence Numbers and Acknowledgement Numbers as well as acknowledgement options, such as Ack Vector, CCID-specific options, etc. Moreover, On ECN-capable connections, the middlebox would have to keep track of ECN Nonce information for packets it introduced or removed, so that the relevant acknowledgement options continued to have correct ECN Nonce Echoes, or risk the connection being reset for "Aggression Penalty".

13.2 RTP (RFC3550) Consideration

The Real-Time Transport Protocol, RTP [RFC 3550], is currently used over UDP by many of DCCP's target applications.

There are two potential sources of overhead in the RTP-over-DCCP combination, duplicated acknowledgement information and duplicated sequence numbers. Together, these sources of overhead add slightly more than 4 bytes per packet relative to RTP-over-UDP, and that eliminating the redundancy would not reduce the overhead.

Acknowledgement overhead in RTP-over-DCCP is not significantly higher than for RTP-over-UDP, at least for CCID 3. RTP-over-DCCP applications might request either DCCP Ack Vectors or RTCP Extended Report Loss RLE Blocks, but not both.

The sequence number in DCCP-Data header, another overhead source, is 12 bytes long without options, including a 24-bit sequence number. This is 4 bytes more than a UDP header. Any options required on data packets would add further overhead, although many CCIDs (for instance, CCID 3, TFRC) don't require options on most data packets. However, The 4 byte header cost is a reasonable tradeoff for DCCP's congestion control features and access to ECN.

13.3 Congestion Manager and Multiplexing

DCCP does not provide built-in, SCTP-style support for multiple sub-flows, since DCCP doesn't provide reliable, ordered delivery, multiple application sub-flows may be multiplexed over a single DCCP connection with no inherent performance penalty.

Some applications might want to share congestion control state among multiple DCCP flows that share the same source and destination addresses. This functionality could be provided by the Congestion Manager [RFC 3124], a generic multiplexing facility. However, the CM would not fully support DCCP without change; it does not gracefully handle multiple congestion control mechanisms.

13.4 Security Considerations

DCCP does not provide cryptographic security guarantees. Applications desiring cryptographic security services (integrity, authentication, confidentiality, access control, and anti-replay protection) should use IPsec or end-to-end security of some kind.

However, DCCP can provide simple protect against some classes of attackers likes hijack by choosing initial sequence numbers well. DCCP also provides mechanisms to limit the potential impact of some denial-of-service attacks using Init Cookie, the DCCP-CloseReq packet, etc.

The partial checksum facility has a separate security impact, particularly in its interaction with authentication and encryption mechanisms. When IPsec is used with ESP payload encryption, DCCP partial checksums provide no benefit in this case.

13.5 IANA Considerations

DCCP introduces eight sets of numbers whose values should be allocated by IANA. In addition, DCCP requires a Protocol Number to be added to the registry of Assigned Internet Protocol Numbers. IANA is requested to assign IP Protocol Number 33 to DCCP; this number has already been informally made available for experimental DCCP use.

14. Possible Future Evolution

14.1 More CCIDs

DCCP connections are congestion controlled, but unlike in TCP, DCCP applications have a choice of congestion control mechanism and two half-connections can be governed by different mechanisms. These mechanisms are denoted by one-byte congestion control identifiers, or CCIDs. CCIDs 2 and 3 are currently defined; CCIDs 0, 1, and 4-255 are reserved. Other CCIDs may be defined for different applications in the future.

For example, as described above, CCID 3 should not be used by applications that change their sending rate by varying the packet size, rather than varying the rate at which packets are sent. A new CCID will be required for these applications.

14.2 The evolution of TFRC

There are a number of cases where the behavior of TFRC as specified in RFC 3448 does not match the desires of possible users of DCCP. These include the following:

1. The initial sending rate of at most four packets per RTT, as specified in RFC 3390.

2. The receiver's sending of an acknowledgement for every data packet received, when the receiver receives less than one packet per round-trip time.

3. Limiting the sending rate to at most twice the reported receive rate over the previous round-trip time.

4. The limitation of halving the allowed sending rate after an idle period of four round-trip times (possibly down to the initial sending rate of two to four packets per round-trip time).

5. Another change that is needed is to modify the response function used in RFC 3448 to match more closely the behavior of TCP in environments with high packet drop rates [RFC 3714].

One suggestion for higher initial sending rates is to use an initial sending rate of up to eight small packets per RTT, when the total packet size, including headers, is at most 4380 bytes. Because the packets would be rate-paced out over a round-trip time, instead of sent back-to-back as they would be in TCP, an initial sending rate of eight small packets per RTT with TFRC-based congestion control would be considerably milder than the impact of an initial window of eight small packets sent back-to-back in TCP.

In addition, with CCID 3, the sender is in slow-start in the beginning, and responds promptly to the report of a packet loss or mark. However, in the absence of feedback from the receiver, the sender can maintain its old sending rate for up to four round-trip times. One possibility would be that for an initial window of eight small packets, the initial nofeedback timer would be set to two round-trip times instead of four, so that the sending rate would be reduced after two round-trips without feedback.

Research and engineering will be needed to investigate the pros and cons of modifying these limitations in order to allow larger initial sending rates, to send fewer acknowledgements when the data sending rate is low, to allow more abrupt changes in the sending rate, or to allow a higher

sending rate after an idle period.

14.3 Mobility and Multihoming

Because so far no suitable mechanism was found for mobility and multihoming at transport layer, even seemingly trivial multihoming mechanisms like SCTP's can cause security problems in practice, hence mobility and multihoming have removed from the main DCCP specification. But once a convictive breakthrough is made in other protocol, e.g. in SCTP, the mobility and multihoming support may be reincorporated in main DCCP specification.

DCCP mobility and multihoming support should fulfill the following requirements and non-requirements:

- An endpoint does not need to announce a new address before moving to that address.
- Move requests must be safe against hijacking.
- Mobility must not create new, large opportunities for denial-of-service attacks.
- Endpoints must be able to move freely between different NAT domains using the mobility mechanism.
- Simultaneous moves need not be supported.
- Cryptography is allowed.

The mobility design should has the following features.

0. Support for mobility is optional and defaults to off.
1. Each endpoint of a mobility-capable connection has a public 128-bit Mobility ID.
2. The endpoints share a Mobility Secret, a key communicated over a secure channel. The Secret is either transmitted out-of-band, or via public-key cryptography or Diffie-Hellman exchange. It is changed on every successful move.
3. A Mobility Sequence number increases monotonically with moves, and identifies which Mobility Secret a packet is using.

Moving a connection has the following four steps, which will take approximately two round-trip times.

1. The mobile host moves, then sends a DCCP-Move-Request packet from its new address with the stationary host's Mobility ID and encrypted mobility token.
2. The stationary host maps the DCCP-Move-Request packet to the old connection using the Mobility ID, then checks the token against the packet data. If this check succeeds, it sends a DCCP-Move-Response packet with a similar token. The stationary host remembers both the new Mobility Secret and the old Mobility Secret.
3. The mobile host sends a DCCP-Move-Confirm packet with new token which is encrypted with the new Mobility Secret.
4. The stationary host receives the DCCP-Move-Confirm packet, removes its old Mobility Secret(s), and sends a DCCP-Move-Complete packet back to the sender, both to end this episode of mobility and to inform NATs and middleboxes that the connection's endpoints have definitively changed.

In addition, It is noted that the original proposed type number of DCCP-Move packet, 8, has been assigned to DCCP-Sync packet in latest DCCP draft.

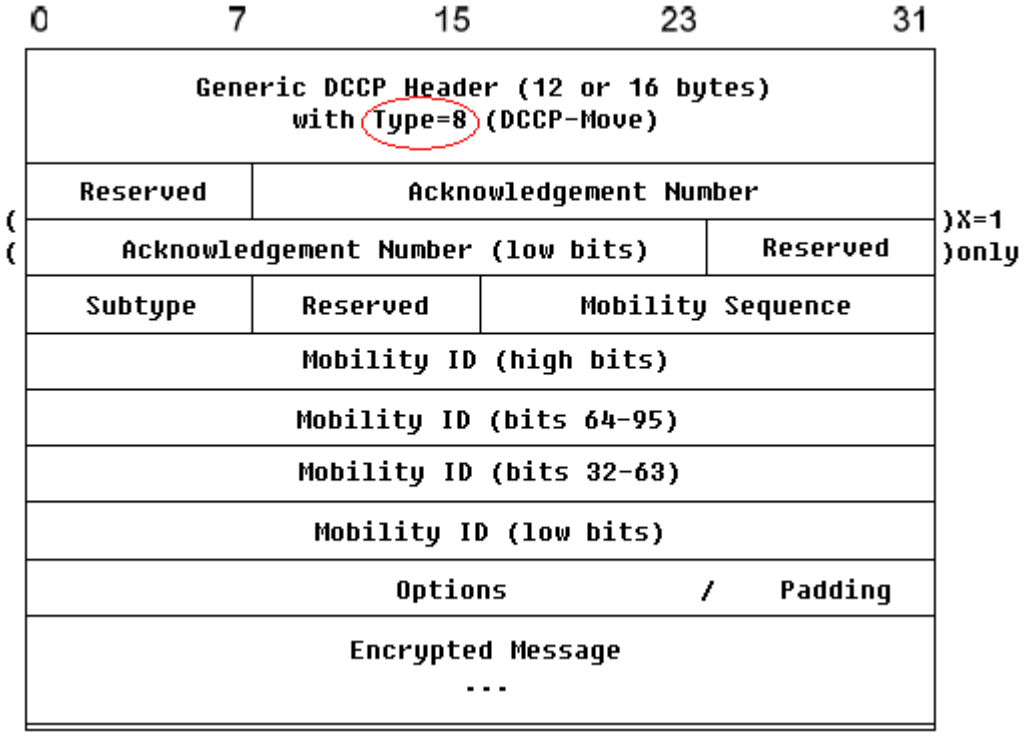

Fig. 13-1. The original proposed format of DCCP-Move header.

Subtype: 8 bits. Used to distinguish between different message types in the mobility handshake.

Mobility Sequence: 16 bits. Defines the Mobility Secret used to encrypt the token. Its initial value for the connection is 0.

Mobility ID: 128 bits. The value of the receiver's Mobility ID feature.

Encrypted Message: Consists of the following fields:

1. The third, fourth, and subsequent words of the DCCP header, up to, but not including, any options. This includes the Sequence Number, Acknowledgement Number, Type, Subtype, Mobility Sequence, and Mobility ID fields, among others.

2. New Mobility Sequence.

3. New half-Mobility Secret (format not specified yet).

4-6. Same as 1-3.

An endpoint will ignore any DCCP-Move packet whose application data area doesn't have the proper format.

15. Conclusion

As a possible substitute for UDP, DCCP integrates the advantages of TCP and UDP. It aims to add the minimal designed congestion control support to a UDP-like foundation by making use of the achievements on congestion control in recent 2 decades, such as possibly-reliable transmission of acknowledgement information, so as to make DCCP suitable as a building block for more higher-level application semantics, So it is highly promising that DCCP will become the de facto standard for some emerging applications, such as streaming media, online games and internet telephony, etc.

In addition, We have to point out that DCCP is still a work-in-progress, it may change further in the future.

References

[1] Datagram Congestion Control Protocol (DCCP), Internet Engineering Task Force INTERNET-DRAFT, 10 March 2005.

[2] Profile for DCCP Congestion Control ID 2: TCP-like Congestion Control, Internet Engineering Task Force INTERNET-DRAFT, 10 March 2005.

[3] Profile for DCCP Congestion Control ID 3: TFRC Congestion Control, Internet Engineering Task Force INTERNET-DRAFT, 10 March 2005.

[4] TCP Friendly Rate Control (TFRC) for Voice VoIP Variant, Internet Engineering Task Force INTERNET-DRAFT, 19 July 2005.

[5] Datagram Congestion Control Protocol Mobility and Multihoming, Internet Engineering Task Force INTERNET-DRAFT, 12 July 2004.

[6] Datagram Congestion Control Protocol - Lite (DCCP-Lite), Internet Engineering Task Force INTERNET-DRAFT, August 2003.

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

[8] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, RFC 3550, January 2003.

[9] An Integrated Rate Control Scheme for TCP-friendly MPEG-4 Video Transmission Shih, C.H.; Wang, J.Y.; Shieh, C.K.; Hwang, W.S.; Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on 23-26 May 2005 Page(s):2124 – 2127.

[10] A Congestion-Controlled Unreliable Datagram API, Junwen Lai and Eddie Kohler, Princeton University & University of California.

[11] Datagram Congestion Control Protocol (DCCP) Overview, Kohler /Handley /Floyd /Padhye, 9 July 2003.

[12] Designing and Issues of DCCP, Joungsik Kim, Uhjin Joung, Dept. of Computer Engineering, Kyungpook National University, 14 December, 2004.