CHAPTER 4

FUNCTIONALITIES OF DCCP PROTOCOL

4.1 INTRODUCTION

Congestion is not uniform over the given network and will be felt at different magnitudes at different sections of WSN due to its multi-hop nature (Abdur Rahman et al 2008). It is evident that high congestion is around the sink node as the traffic from other sections of the network converges towards the sink. In such high rate sensor network applications a fairly reliable solution is mandatory to avoid congestion and to maintain complete and efficient data transfer between many sources and one or more sinks (Alam and Hong 2009). One of the desirable characteristic of congestion control protocols is to provide a better QoS in terms of packet loss ratio, packet delay, throughput etc., (Chonggang Wang1 et al 2006).

The quality of the network is further deteriorated by poor and time varying channel quality, asymmetric communication channels, multi-hop environment etc., (Hull et al 2004). From the analysis given in the previous chapter it has been identified that DCCP is one of the best suited transport protocols for WSN in controlling congestion and hence, an introduction on the basic functionalities of DCCP and the presently available congestion control mechanisms of DCCP are presented in this Chapter.

DCCP is so designed that it provides tailored support for time critical applications such as voice chat and video conferencing, which prefer
timeliness over reliable delivery (Kohler et al 2006). Although it has already been some years since its IETF standardization back in 2006 (Kohler et al 2006), DCCP is still hardly used by potential applications. This can be partially attributed to the fact that most popular operating systems still do not natively support DCCP, and that Network Address Translation (NAT) traversal can still not be guaranteed. Recently, a short term solution to that misery was proposed, which encapsulates DCCP by UDP (Phelan et al 2012).

Session Description Protocol is used at connection startup to probe whether end points either natively support DCCP or whether to fall back on a user-level DCCP over UDP implementation. For Linux, there exists a pretty stable kernel implementation of DCCP that covers almost all mandatory functionality of the respective Request For Comment (RFC) and although there is still much work in progress with respect to experimental congestion control algorithms, the core module, paired with CCID 2 (TCP-like) or CCID 3 TCP-TFRC, is fully operational.

### 4.2 DCCP FUNDAMENTALS AND FRAMEWORK

From the above it may be arrived that based on the prevailing conditions the objective is to design a protocol that is lightweight and minimal, while providing a transport, on which other mechanisms can be layered with the primary purpose being congestion control. Given that different applications have different requirements of congestion control, the protocol must support negotiation of congestion control mechanisms. As congestion control involves keeping state for the flow in the end-points, well defined mechanisms are required to set up and cleanly tear down the state. And the pragmatics of deployment in today’s internet mean that NAT and firewall traversal must be taken into account (Kohler et al 2006).
4.2.1 Functions of DCCP

DCCP is having several interesting features which are discussed in this section. The main functions of DCCP are to establish, maintain or tear down unreliable connections and to utilize the congestion control mechanisms for unreliable connections. DCCP provides reliable connection establishment and feature negotiation, though data transfer is unreliable i.e. lost data packets are not retransmitted. During connection establishment, teardown and feature negotiation DCCP keeps transmitting and retransmitting control packets until the sender gets a response from the other end. To facilitate a better and cooperative transmission, DCCP is supported with several packet options. For instance, the option Ack vector records receiver’s receiving status and data-dropped option records the cause of a packet drop. Though DCCP offers only two congestion control mechanisms presently, viz. TCP-like and TFRC, designated as CCID 2 and CCID 3 respectively, provisions are plenty to include versatile congestion control mechanisms in future. Further, it permits the connection to select a proper congestion control mechanism dynamically through negotiation during connection establishment. In addition the rate at which the acknowledgements are to be received may also be adjusted dynamically according to the degree of congestion in return path. Synchronization (SYN) flooding attack, a major setback of TCP is prevented in DCCP as the latter uses cookies during connection establishment phase (Kohler et al 2006).

4.2.2 Packet Formats

The general format of a DCCP packet and a DCCP generic header are depicted in Figures 4.1 and 4.2 respectively. In a DCCP packet the first section is generic packet header and common for all packets. Currently there are 10 packets Types defined in DCCP and depending on the Type of a packet, additional fixed length fields and a variable length options list follow.
the general format. The supported packet Types are DCCP-request, DCCP-Response, DCCP-Data, DCCP-Ack, DCCP-DataAck, DCCP-
CloseReq, DCCP-Close, DCCP-Reset, DCCP-Sync, and DCCP-SyncAck (Yuan-Cheng Lai 2008). The “header size” is the location offset from the
beginning of the header to the beginning of data and given in the field Data
set offset. The next two fields “CCval” and “CsCov” are available only in
DCCP and the type of application specific dynamically adopted congestion
control mechanism, either CCID 2 or CCID 3, is indicated by the field
“CCval” where as the range that the checksum covers is determined by
“CsCov”. The checksum from all packets contents will be taken into account
when “CsCOV” is set to zero and the first (CsCOV – 1) × 4 bytes of data, in
addition to header and options, will be covered by checksum otherwise, so as
to enhance the performance on error-prone links for applications that can
endure corruption. DCCP has an option mechanism similar to that of TCP’s
mechanism, but in case of DCCP, the header allows much more option space,
a maximum of 1008 bytes, 25 times more than that of TCP. These options are
used for selecting certain parameters, for example, acknowledgement
reporting and parameter negotiation.

<table>
<thead>
<tr>
<th>Generic Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additional Field (Depending on)</td>
</tr>
<tr>
<td>Options</td>
</tr>
<tr>
<td>Data</td>
</tr>
</tbody>
</table>

**Figure 4.1 Format of DCCP packet**

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Set Offset</td>
<td>CCval</td>
</tr>
<tr>
<td>Res</td>
<td>Type</td>
</tr>
<tr>
<td>Sequence Number (Low Bits)</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.2 Format of DCCP’s generic header**
X = 1

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Acknowledgement Number (High Bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acknowledgement Number (Low Bits)</td>
</tr>
</tbody>
</table>

X = 0

| Reserved | Acknowledgement Number (Low Bits) |

**Figure 4.3 Format of DCCP’s acknowledgement sub-header**

The length of the packet header in DCCP is either 16 bytes or 12 bytes according to the bit value of X. If X is set to 1 the packet header length is 16 bytes and the length of the packet sequence number (seqno) is 48 bits in which 16 bits appear as higher order and remaining 32 bits as lower order. On the other hand if X is 0 then the packet header length is 12 bytes and the length of seqno is 24 bits with 8 reserved bits and 16 high order bits. It may be noticed that the packet header’s overhead in DCCP lies between that of UDP (8 bytes) and TCP (20 bytes). The unit of DCCP sequence number is packets and whenever a sender send packets, may be a data or non-data, it is increased by one. If a packet is neither DCCP-Request nor DCCP-Data, it appends a 4 byte or 8 byte acknowledgement sub-header, located in the additional field as given in Figure 4.3. The received packets trigger their acknowledgements and the sequence numbers of the received packets are treated as acknowledgement numbers (ackno) which are also included in the sub-header discussed above.

### 4.3 CONNECTION PHASES

DCCP is connection oriented protocol which supports bidirectional flow of both data and acknowledgements over a single bidirectional
connection, similar to that of TCP. In general, to facilitate asymmetric data transfer in applications such as video streaming with minimum overheads, DCCP supports bidirectional communication. A DCCP connection between points A and B is broken into two logical half connections, one from A and B while the other is in the reverse direction. In the first logical half data packets are from A to B and acknowledgements are from B to A whereas in the other half data are from B to A and acknowledgements are from A to B as shown in Figure 4.4.

**Figure 4.4 Two half-connections that make up a DCCP connection**

In either case acknowledgement need not be generated when the sender have no data to send and hence, the overhead is reduced. (Floyd et al 2006) The feature negotiation for both halves are independent and may occur simultaneously with different congestion control mechanisms being opted by the two half connections. Further, two half-connection in back-to-back may be regarded as single bidirectional connection.

In spite of significant benefits and flexibility, half-connection adds some complexity. The selection of two different congestion control mechanisms need to cooperate among themselves to produce a single piggybacked packet i.e. packets with both data and acknowledgement. In unidirectional communication also this piggybacked packet mechanism reduces the packet overheads. There are three phases in DCCP connection
establishment given as Connection Establishment, Data Transfer and Connection Termination and nine different states are involved in these three phases. The states of DCCP are CLOSED, LISTEN, REQUEST, RESPOND, PARTOPEN, OPEN, CLOSEREQ, CLOSING, and TIMEWAIT (Floyd et al 2006).

4.3.1 Connection Establishment

When an application is essentially unidirectional, DCCP offers a unique mechanism called as Quiescence for better management of available resources without unreasonable overhead. If one of the endpoints, say A, stops sending data for some predefined time, which is twice RTT presently, the other endpoint B, detects that A has gone into quiescent state and prefers a unidirectional pattern of communication instead of present bidirectional pattern and the endpoint A is expected to send only acknowledgements. While A might demand Ack Vector for feedback on its data packets, it likely will not require such precise feedback or, perhaps, any feedback at all, for its acknowledgements. Therefore, B, the non-quiescent endpoint, will limit the acknowledgements it sends to exactly those acks-of-acks required for its congestion control mechanism.

A three way handshaking technique is used in DCCP to establish a connection between a client and a server. Figure 4.5 depicts the complete scenario of DCCP communication phases. The server is initially in LISTEN state to accept requests from a client. Upon receiving a DCCP-Request packet from the client to establish a connection, the receiver responds with a DCCP-Response packet and the receiver enters in to RESPOND state.
Figure 4.5 Three-way handshake method of DCCP connection establishment

The client in turn acknowledges with a DCCP-Ack packet to the server. On receiving this DCCP-Ack packet, the server enters into OPEN state and at the same time client moves in to a partially open state known as PARTOPEN state from REQUEST state and finally to full OPEN state. With this connection establishment phase is complete and exchange of data between server and client may follow.

The client can’t move into OPEN state immediately after receiving DCCP-Response packet, if DCCP-Ack packet is lost due to which the server is prevented from entering into OPEN state. Assuming the sender is already in OPEN state, a deadlock will occur and so, the client enters into PARTOPEN state instead, after sending DCCP-Ack. In such case after remaining in this state for a fixed period of time, the client sends DCCP-Ack/DCCP-DataAck packets repeatedly until it receives an acknowledgement or four times the
Maximum Segment Lifetime (MSL) expires which means the failure in connection establishment. The maximum survival time of a packet is given by MSL. On the other hand, if the connection establishment between the client and server is successful feature negotiation takes between them. Though feature negotiation may take at any point of time between the client and server, it generally happens during connection establishment.

The feature values that the client prefer is embedded in a DCCP-Request packet and as a reply DCCP-Response packet carries a negotiation option with feature values proposed by the server. Four different options viz. Change L, Change R, Confirm L and Confirm R are used by the server and the client during a feature negotiation to have a consensus on the feature values. The beginning of a feature negotiation is denoted by Change option and the completion by Confirm option. Option “L” is sent by the feature location where as the option “R” by the feature remote either with Change or with Confirm followed by list of preferred or confirmed values respectively. The list sent along with Change or Confirm is ordered with the most preferred as the first one. For example if the client prefers DCCP-TCP-like congestion control algorithm it sends the Change R(CCID,2,3) which clearly indicates that the preference of the client is DCCP-TCP-like over DCCP-TFRC. The server in turn may respond with Confirm L(CCID, 3, 3, 2) option where the first number in the list denotes the choice of server which is followed by the preference list of the server. Hence, the selected congestion control mechanism is CCID 3, i.e. DCCP-TFRC as agreed by both sides.

The three-way handshaking process may be obstructed by SYN flooding attack to the server and DCCP uses an init-cookie option to avoid this using encryption-decryption process at server side. Upon the reception of a DCCP-Request by the server it embeds the init-cookie option into DCCP-Response packet and sends it to the client. The details such as initial sequence
number, acknowledgement number, port number, options etc. may be encrypted by a private key and sent as cookie content which cannot be decrypted by the client. On receiving the init-cookie option the client has to send a reply. The init-cookie is so designed by the server to validate the returned cookie by decrypting returned cookie with the same private key and by checking whether the decrypted version matches with the original content before the establishment of connection, thus successfully avoiding SYN flooding completely (Kohler et al 2006).

### 4.3.2 Data Transfer

In DCCP, once the Connection between the server and the client is established data packet exchange takes place between them and this transmission is unreliable though controlled by the preferred congestion control mechanism. A unique ID is associated with each DCCP packet, known as sequence number (seqno) and the client side packet sequence numbers are independent from those of server side. If initial sequence numbers are chosen sufficiently randomly, (Bellovin 1996) attackers must snoop data packets to achieve any reasonable probability of success. Unlike TCP, where the smallest seqno of unreceived packet is always reported, DCCP doesn’t track a cumulative acknowledge number as there is no retransmission in it and reports only the packet received very recently. Figure 4.6 describes a typical DCCP data exchange scenario. As shown in Figure 4.6, the client sends two data packets with seqno 1 and 2 and the server responds with data packet with seqno 5 and the acknowledgement ackno 2, for the recently received data packet with seqno 2 with respect to time scale. Client responds with data packet seqno 3 and ackno 5 and this process continues.

Suppose that DCCP end points use DCCP-Sync and DCCP-SyncAck packets for synchronization and assume that all the packet with
seqno 70 to seqno 151 sent by client are lost except the first and last packets i.e. only the packets with seqno 70 and seqno 151 arrive at the server. Obviously the packet with seqno 151 is totally out of the expected range at the server end and hence, a synchronizing process is invoked by sending DCCP-Sync packet from the server and acknowledging the reception of the latest packet received by sending ackno 151. Upon receiving the DCCP-Sync packet the client immediately responds with DCCP-Syncack packet and on the reception of this syncack, the server adjusts its expected range of seqno to synchronize both sides.

![DCCP data transfer and synchronization](image)

**Figure 4.6** DCCP data transfer and synchronization
The characteristics of DCCP acknowledgements differ from those of TCP in two ways. In DCCP the acknowledgements are restricted by congestion control and the sender has to report its acknowledgement reception to the receiver periodically. The receiver uses an 8 bit block, called as ackvector to convey its packet reception status to the sender. Out of the 8 bits in an ackvector two bits are used to represent three defined states and the rest for run length. The received state is signified by 0, received with an ECN state by 1 and not yet received by 3. Since retransmissions are not necessarily useful for these time sensitive applications, they have a great deal to gain from the use of ECN (Ramakrishnan et al 2001), which lets congested routers mark packets instead of dropping them. The number of consecutive packets in the given state is represented by run length. As the receiver has to report about more receiving conditions to the sender, the size of the ackvector increases gradually on receiving more packets by the receiver. It is the responsibility of the sender to notify the receiver about the acknowledgements received by it over period of time. Once acks-of-acks is received the receiver clears the receiving conditions of packets that the sender has recognized thereby reducing the size of ack vector.

Currently, the choice is between TCP-like, whose sawtooth rates quickly utilize available bandwidth, and TFRC, which achieves a steadier long-term rate (Floyd et al 2000). In future, DCCP will support experimentation with new congestion control mechanisms, from low-speed TFRC variants to more radical changes. (Katabi et al 2002). Each of these variants may require different acknowledgement mechanisms. For instance, TFRC’s acknowledgements are much more parsimonious than TCP’s. Thus, DCCP supports a range of acknowledgement types, depending on the selected congestion control method.
Data-drop is another important option in DCCP, which enables the sender to identify the cause of each packet loss and the length of this option is 8 bits. Most significant bit of data-drop option designated to represent the success or failure in transferring the received packets to the application layer by the receiver. The next three bits indicate the reasons of drop and last four bits for run length. The possible reasons for data drop and their representative values are protocol constraints 0, application not listening 1, receiver buffer 2, corrupt 3 and delivery corrupt 7.

4.3.3 Connection Termination

Figures 4.7 and 4.8 demonstrate the process by which DCCP terminates a connection. DCCP-Close packet and DCCP-Reset packet are used in DCCP to terminate a connection, irrespective of the side which initiates the process. Whenever the client intends to terminate a connection, it transmits DCCP-Close packet to the server and in response to this the sender sends a DCCP-Reset packet, signalling the end of connection-termination process.

![Figure 4.7 Two-way handshaking for DCCP connection termination](image-url)
On receiving a DCCP-Reset packet, the client waits for 2 MSL before terminating the connection. Same sequence with the packet flow reversed is followed whenever the server intends to terminate a connection. However, in case of server initiating the termination process, the client may be permitted to be in TIMEWAIT state after receiving DCCP-Reset packet and under such conditions the server first sends DCCP-CloseReq packet to the client.

![Three-way handshaking for DCCP connection termination](image)

**Figure 4.8 Three-way handshaking for DCCP connection termination**

### 4.4 CONGESTION CONTROL

With UDP as the base, DCCP was developed for effective and efficient handling of congestion over WSN resulting in more reliable transmission of datagrams or packets (Floyd et al 2006). The main objective of DCCP is to extend support for implementing different congestion control schemas out of which the most suitable one may be selected by the applications, so as to provide efficient congestion control. Hence, according to the type of data being transmitted, a schema is selected to assure a better flow of packets. A mechanism, known as CCID is implemented in DCCP,
enabling it to assign separate CCID for each direction of data flow. The nature of congestion is defined by CCID and the source as well as the destination selects appropriate mechanism to handle this congestion by feature negotiation, a method that selects the best suited algorithm for the present scenario (StanimirStatev and SeferinMirtchev 2008). A TCP-like congestion control mechanism is offered by CCID 2 and suitable for bursty packet flows where as TFRC congestion control mechanism is offered by CCID 3, which is suitable for smooth packet flows.

4.4.1 DCCP-TCP-like

CCID 2 is based on AIMD congestion control mechanism wherein the sender adopts a suitable congestion window to control transmission, according to the information embedded in the acknowledgements. The three parameters used by CCID 2 are congestion window size (cwnd), slow-start threshold (ssthresh) and pipe where cwnd represents the maximum allowable amount of outstanding packets and ssthresh represents a threshold of the congestion window that separates slow start from congestion avoidance. The third parameter represents the number of estimated outstanding packets. CCID 2 is almost similar to Selective Acknowledgement TCP except that the farmer uses packets as units and latter bytes as units. DCCP being an unreliable protocol doesn’t support fast retransmission and fast recovery as in the case of SACK TCP.

Acknowledgements are so coined by the congestion control mechanism over period of time that the congestion caused by them is prevented. Acknowledgements’ sending rate from the receiver is dynamically adjusted by the sender using ack-ratio feature which is initially set to its default value 2 i.e., the receiver replies to an acknowledgement after receiving two packets. On detecting congestion, the sender doubles ack-ratio to halve the acknowledgement reply rate. On the other hand, if the reception of
acknowledgements is from congestion free window the expected number of acknowledgements increases by one and the increase in the acknowledgements $k$ is given by

$$k = \frac{\text{cwnd}}{\text{ackratio}} - \frac{\text{cwnd}}{\text{ackratio}-1}$$ \hspace{1cm} (5.1)

Thus, if the sender receives $k$ acknowledgements from congestion-free window, the ack-ratio is decreased by one and hence the number of packets that the receiver has to wait before sending the reply for an acknowledgement is increased by one from its present value.

The half-connection communication in DCCP is established as discussed in section 4.3. The connection establishment phase exclusively with respect to DCCP-TCP-like is discussed in this section. The connection initiation process is similar to that of DCCP as discussed in section 4.3. The different steps involved in DCCP-TCP-like half connection are given below.

- As Ackvectors are mandatory in CCID 2 the feature Use Ack Vector is negotiated on the Request packet sent. Each data packet sent by the sender will have either ECT(0) or ECT(1) code point set.

- As the congestion control is provided by TCP-like on acknowledgements, the receiver sends its DCCP-Ack packets as ECN-capable. For every Ack Ratio of data packets received, the receiver sends a DCCP-Ack packet with Ack vector, and it is for the sender to ascertain that the receiver is not misbehaving. These Ackvectors include ECN Nonce Echo (ENE).
Upon the reception of acknowledgements, the sender examines the ECN Nonce Echo and if it is found correct, it checks for lost or marked packets and cwnd is updated accordingly. The level of congestion on the return path is assessed by the sender by detecting the missed or marked DCCP-Ack packets and Ack-Ratio is modified as needed.

The acknowledgements sent by the receiver are acknowledged by the sender at least once per congestion window and the sending depends on the activity of the other half-connection. In case, if the other half-connection is active, these acknowledgements are included in the acknowledgements of receiver’s data. On the other hand, if the other half-connection is in quiescent state, the sender has to send acknowledge occasionally.

The packets sent by the sender are considered as lost if no feedback is received from the receiver before the lapse of TimeOut value. In DCCP-TCP-like RTT is used to calculate TimeOut value, much like the calculation of RTO in TCP.

The congestion is identified in CCID 2 by checking the sequence Number of Duplicate Acknowledgement (NUMDUPACKS) packets. If the sequence number of NUMDUPACKS is higher than that of a sent packet, then that packet is considered as lost.

4.4.2 DCCP-TFRC

CCID 3, the second congestion control mechanism offered by DCCP, uses TFRC model congestion control and meant for applications require smooth transmission. During the initial slow-start phase, at the end of
each RTT the transmission is simply doubled until a packet loss is reported by the receiver or packets with ECN is received. Once such condition is reported, the sender immediately aborts the slow-start phase and uses the following formula to estimate the transmission rate \( T \) using the loss event ratio measured by the receiver to adjust the sender transmission rate accordingly.

\[
T = \frac{S}{RTT \sqrt{\frac{2p}{3} + t_{\text{RTO}}} \left( \frac{3p}{8} \right) \left( 1 + 32p^2 \right)}
\]  

(5.2)

where

- \( S \) - Packet size
- \( RTT \) - Round Trip Time
- \( t_{\text{RTO}} \) - Retransmission time
- \( p \) - Loss event ratio.

TFRC is a receiver based congestion control mechanism that uses throughput equation to estimate the allowable congestion rate and provides a smoother sending rate than TCP. The steps involved in a typical DCCP-TFRC half connection with ECN are as follows.

- The negotiation for the features, Loss event rate and Use Ack Features are done during connection establishment phase. Here also ackvectors are required to carry the ECN Nonce Echo (ENE).

- Each data packet sent by the sender will have either ECT(0) or ECT(1) codepoint set and the sending rate is dictated by TCP-throughput equation. The CCVAL field of the packet header is filled with a window counter value, which is
incremented by one for every quarter of RTT that has elapsed since the last packet was sent.

- The receiver in turn responds with a feedback packet with the calculated loss event rate and the receive rate, at least once per RTT. A loss event represents the number of packets marked or lost in a single RTT. DCCP-Ack packets are marked as ECN-incapable as no congestion control is provided on acknowledgements. Each packet contains the ENE in the form of an ack vector option.

- Upon the reception of an acknowledgement, the sender verifies the ENE and if found correct, the sending rate is updated according to the information received.

- If the other half-connection is quiescent, the acknowledgements of the receiver are acknowledged by the sender at least once per RTT.

- The estimated RTT is used to determine the TimeOut value and the sender behaves accordingly. If no feedback is received from the receiver after the lapse of TimeOut period, the sending rate is halved by the sender.

The major modifications in CCID 3 are the inclusion of ECN, alteration of mechanisms in packet-loss detection and loss interval calculation and support for oscillation prevention. A packet is considered lost than delayed in CCID 3, if NUMDUPACKS packets with higher sequence number have been received.
4.5 DCCP DIFFERENCES FROM TCP

There are certain differences between TCP and DCCP and major aspects in this regard are discussed in this section.

- **Option Space:** Comparing to TCP DCCP provides copious space for option. In TCP the size of the option is restricted to 0 to 320 bits, whereas in DCCP it can go up to 1008 bytes or PMTU.

- **Acknowledgement Formats:** DCCP supports different acknowledgement formats, based on the mechanism selected. In CCID 2, per 2 packets of data one ack is used and each ack must declare exactly which packets were received. In CCID 3, it is about one ack per RTT and acks must have information about the recent loss intervals.

- **Denial of Service (DoS) Protection:** The number of states to be maintained, which may be decided by the possibly-misbehaving clients, is limited in DCCP. For instance, the TIMEWAIT state in DCCP is limited to only one connection end-point as only the server sends DCCP-CloseReq packet and that state is passed to the client. Extensive computation or packet generation by the server are limited using various rate-limit methods in DCCP. An Init-cookie option in DCCP, which is similar to TCP’s SYN-cookies, avoids SYN-flood-like attacks.
• Distinguishing Losses: In DCCP, when a packet is lost, the endpoint declare the reason for the drop, such as corruption, receive buffer overflow etc., through Data-Dropped option which facilitates a more appropriate rate-control for non-networking congestion losses.

• Acknowledgebility: A DCCP packet may be acknowledged once its header is processed successfully, where as in TCP it happens only when the data is reliably queued for application delivery. This enables DCCP to accept the requests from the application, such as drop-from-front receive buffer. However, acknowledgebility in DCCP does not guarantee data delivery and the Data Dropped option may report later that the application data of the packet was discarded.

• No Half-Closed State: In TCP one of the half-connections is permitted to close explicitly while the other is still active, which is not allowed in DCCP. However, this may be achieved by setting the option part of Data-Dropped option to Drop-code 1 which implies Application Not Listening.

• As TCP is a flow control protocol, it is provided with Receive-Window and there is no such option in DCCP, since it is basically a congestion control protocol. Further, Unlike TCP, DCCP does not support simultaneous open and hence, in every connection in DCCP there is only one client and one server.
4.6 SUMMARY

The fundamentals of DCCP along with its framework are discussed at the beginning of this chapter. This is followed by a brief overview of functions of DCCP and the formats of different types of packets supported by DCCP. A detailed description on the three phases of a typical DCCP communication process viz. connection establishment, data transfer and connection termination is also presented with relevant diagrams. This is followed by a brief discussion on congestion control and the sequence of operations of the two congestion control mechanisms of DCCP known as CCID 2 and CCID 3. The differences between a typical TCP and DCCP processes are also discussed in this chapter.