DCCP protocol

Introduction

Datagram Congestion Control Protocol (DCCP) is an unreliable, connection oriented protocol designed to solve issues present in UDP and TCP, particularly for real-time and multimedia (streaming) traffic. It divides into a base protocol (RFC 4340) and pluggable congestion control modules called CCIDs. Like pluggable TCP congestion control, at least one CCID needs to be enabled in order for the protocol to function properly. In the Linux implementation, this is the TCP-like CCID2 (RFC 4341). Additional CCIDs, such as the TCP-friendly CCID3 (RFC 4342), are optional. For a brief introduction to CCIDs and suggestions for choosing a CCID to match given applications, see section 10 of RFC 4340.

It has a base protocol and pluggable congestion control IDs (CCIDs).

DCCP is a Proposed Standard (RFC 2026), and the homepage for DCCP as a protocol is at http://www.ietf.org/html.charters/dccp-charter.html

Missing features

The Linux DCCP implementation does not currently support all the features that are specified in RFCs 4340...42.

The known bugs are at:

> http://www.linuxfoundation.org/collaborate/workgroups/networking/todo#DCCP

For more up-to-date versions of the DCCP implementation, please consider using the experimental DCCP test tree; instructions for checking this out are on: http://www.linuxfoundation.org/collaborate/workgroups/networking/dccp_testing#Experimental_DCCP_source_tree

Socket options

DCCP_SOCKOPT_QPOLICY_ID sets the dequeuing policy for outgoing packets. It takes a policy ID as argument and can only be set before the connection (i.e. changes during an established connection are not supported). Currently, two policies are defined: the "simple" policy (DCCPQ_POLICY_SIMPLE), which does nothing special, and a priority-based variant (DCCPQ_POLICY_PRIO). The latter allows to pass an u32 priority value as ancillary data to sendmsg(), where higher numbers indicate a higher packet priority (similar to SO_PRIORITY). This ancillary data needs to be formatted using a cmsg(3) message header filled in as follows:

```
cmsg->cmsg_level = SOL_DCCP;
cmsg->cmsg_type  = DCCP_SCM_PRIORITY;
cmsg->cmsg_len   = CMSG_LEN(sizeof(uint32_t));  /* or CMSG_LEN(4) */
```

DCCP_SOCKOPT_QPOLICY_TXQLEN sets the maximum length of the output queue. A zero value is always interpreted as unbounded queue length. If different from zero, the interpretation of this parameter depends on the current dequeuing policy (see above): the "simple" policy will enforce a fixed queue size by returning EAGAIN, whereas the "prio" policy enforces a fixed queue length by dropping the lowest-priority packet first. The default value for this parameter is initialised from /proc/sys/net/dccp/default/tx_qlen.

DCCP_SOCKOPT_SERVICE sets the service. The specification mandates use of service codes (RFC 4340, sec. 8.1.2); if this socket option is not set, the socket will fall back to 0 (which means that no meaningful service code is present). On active sockets this is set before connect(); specifying more than one code has no effect (all subsequent service codes are ignored). The case is different for passive sockets, where multiple service codes (up to 32) can be set before calling bind().

DCCP_SOCKOPT_GET_CUR_MPS is read-only and retrieves the current maximum packet size (application payload size) in bytes, see RFC 4340, section 14.

DCCP_SOCKOPT_AVAILABLE_CCIDS is also read-only and returns the list of CCIDs supported by the endpoint. The option value is an array of type uint8_t whose size is passed as option length. The minimum array size is 4 elements, the value returned in the optlen argument always reflects the true number of built-in CCIDs.

DCCP_SOCKOPT_CCID is write-only and sets both the TX and RX CCIDs at the same time, combining the operation of the next two socket options. This option is preferable over the latter two, since often applications will use the same type of CCID for both directions; and mixed use of CCIDs is not currently well understood. This socket option takes as argument at least one uint8_t value, or an array of uint8_t values, which must match available CCIDS (see above). CCIDs must be registered on the socket before calling connect() or listen().

DCCP_SOCKOPT_TX_CCID is read/write. It returns the current CCID (if set) or sets the preference list for the TX CCID, using the same format as DCCP_SOCKOPT_CCID. Please note that the getsockopt argument type here is `int`, not uint8_t.

DCCP_SOCKOPT_RX_CCID is analogous to DCCP_SOCKOPT_TX_CCID, but for the RX CCID.

DCCP_SOCKOPT_SERVER_TIMEWAIT enables the server (listening socket) to hold timewait state when closing the connection (RFC 4340, 8.3). The usual case is that the closing server sends a CloseReq, whereupon the client holds timewait state. When this boolean socket option is on, the server sends a Close instead and will enter TIMEWAIT. This option must be set after accept() returns.

DCCP_SOCKOPT_SEND_CSCOV and DCCP_SOCKOPT_RECV_CSCOV are used for setting the partial checksum coverage (RFC 4340, sec. 9.2). The default is that checksums always cover the entire packet and that only fully covered application data is accepted by the receiver. Hence, when using this feature on the sender, it must be enabled at the receiver, too with suitable choice of CsCov.

DCCP_SOCKOPT_SEND_CSCOV sets the sender checksum coverage. Values in the
: range 0..15 are acceptable. The default setting is 0 (full coverage), values between 1..15 indicate partial coverage.

DCCP_SOCKOPT_RECV_CSCOV is for the receiver and has a different meaning: it
: sets a threshold, where again values 0..15 are acceptable. The default of 0 means that all packets with a partial coverage will be discarded. Values in the range 1..15 indicate that packets with minimally such a coverage value are also acceptable. The higher the number, the more restrictive this setting (see [RFC 4340, sec. 9.2.1]). Partial coverage settings are inherited to the child socket after accept().

The following two options apply to CCID 3 exclusively and are getsockopt()-only. In either case, a TFRC info struct (defined in <linux/tfrc.h>) is returned.

DCCP_SOCKOPT_CCID_RX_INFO
: Returns a `struct tfrc_rx_info` in optval; the buffer for optval and optlen must be set to at least sizeof(struct tfrc_rx_info).

DCCP_SOCKOPT_CCID_TX_INFO
: Returns a `struct tfrc_tx_info` in optval; the buffer for optval and optlen must be set to at least sizeof(struct tfrc_tx_info).

On unidirectional connections it is useful to close the unused half-connection via shutdown (SHUT_WR or SHUT_RD): this will reduce per-packet processing costs.

Sysctl variables

Several DCCP default parameters can be managed by the following sysctls (sysctl net.dccp.default or /proc/sys/net/dccp/default):

request_retries
: The number of active connection initiation retries (the number of Requests minus one) before timing out. In addition, it also governs the behaviour of the other, passive side: this variable also sets the number of times DCCP repeats sending a Response when the initial handshake does not progress from RESPOND to OPEN (i.e. when no Ack is received after the initial Request). This value should be greater than 0, suggested is less than 10. Analogue of tcp_syn_retries.

retries1
: How often a DCCP Response is retransmitted until the listening DCCP side considers its connecting peer dead. Analogue of tcp_retries1.

retries2
: The number of times a general DCCP packet is retransmitted. This has importance for retransmitted acknowledgments and feature negotiation, data packets are never retransmitted. Analogue of tcp_retries2.

tx_ccid = 2
: Default CCID for the sender-receiver half-connection. Depending on the choice of CCID, the Send Ack Vector feature is enabled automatically.

rx_ccid = 2
: Default CCID for the receiver-sender half-connection; see tx_ccid.

seq_window = 100
: The initial sequence window (sec. 7.5.2) of the sender. This influences the local ackno validity and the remote seqno validity windows (7.5.1). Values in the range Wmin = 32 (RFC 4340, 7.5.2) up to 2^32-1 can be set.

tx_qlen = 5
: The size of the transmit buffer in packets. A value of 0 corresponds to an unbounded transmit buffer.

sync_ratelimit = 125 ms
: The timeout between subsequent DCCP-Sync packets sent in response to sequence-invalid packets on the same socket (RFC 4340, 7.5.4). The unit of this parameter is milliseconds; a value of 0 disables rate-limiting.

IOCTLS

FIONREAD
: Works as in udp(7): returns in the `int` argument pointer the size of the next pending datagram in bytes, or 0 when no datagram is pending.

SIOCOUTQ
: Returns the number of unsent data bytes in the socket send queue as `int` into the buffer specified by the argument pointer.

Other tunables

Other tunables

Per-route rto_min support

CCID-2 supports the RTAX_RTO_MIN per-route setting for the minimum value of the RTO timer. This setting can be modified via the 'rto_min' option of iproute2; for example:

```
> ip route change 10.0.0.0/24   rto_min 250j dev wlan0
> ip route add    10.0.0.254/32 rto_min 800j dev wlan0
> ip route show dev wlan0
```

CCID-3 also supports the rto_min setting: it is used to define the lower bound for the expiry of the nofeedback timer. This can be useful on LANs with very low RTTs (e.g., loopback, Gbit ethernet).

Notes

DCCP does not travel through NAT successfully at present on many boxes. This is because the checksum covers the pseudo-header as per TCP and UDP. Linux NAT support for DCCP has been added.