Datagram Congestion Control Protocol (DCCP) Overview

Eddie Kohler
UCLA/ICIR
IETF 58 AVT Meeting
November 12, 2003
DCCP is

- A congestion-controlled, unreliable flow of datagrams
- “UDP plus congestion control”
- Also a modern transport protocol
  Partial checksums, mobility, DoS resistance, fast connections, . . .
Target applications

- Long-lived flows that prefer timeliness to reliability
  - Streaming media, Internet telephony, on-line games, ...
- UDP not congestion controlled, apps must implement CC
- Apps want
  - Buffering control: don’t deliver old data
  - Different congestion control mechanisms (TCP vs. TFRC)
  - Low per-packet byte overhead
DCCP’s attractions for applications

• Congestion control implementation
  Experience shows CC is difficult to get right

• Integrated acknowledgements, reliable feature negotiation

• Access to ECN
  ECN capable flows must be congestion controlled
  UDP APIs would find this difficult to enforce

• Different congestion control mechanisms
TCP-like vs. TFRC congestion control

- TCP-like: quickly get available B/W
  
  Cost: sawtooth rate—halve rate on single congestion event
  
  May be more appropriate for on-line games
  
  More bandwidth means more precise location information; UI cost of whipsawing rates not so bad

- TFRC [RFC 3448]: respond gradually to congestion
  
  Single congestion event does not halve rate
  
  Cost: respond gradually to available B/W as well
  
  May be more appropriate for telephony, streaming media
  
  UI cost of whipsawing rates catastrophic
### Sample connection

<table>
<thead>
<tr>
<th>DCCP A</th>
<th>DCCP B</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0. CLOSED</strong></td>
<td>LISTEN</td>
</tr>
<tr>
<td><strong>1. App opens</strong></td>
<td><strong>REQUEST</strong></td>
</tr>
<tr>
<td><strong>2. OPEN</strong></td>
<td><strong>OPEN</strong></td>
</tr>
<tr>
<td><strong>3. OPEN</strong></td>
<td><strong>OPEN</strong></td>
</tr>
<tr>
<td><strong>4. Initial feature negotiation (CC mechanism, ...)</strong></td>
<td><strong>OPEN</strong></td>
</tr>
<tr>
<td><strong>5. Data transfer</strong></td>
<td><strong>OPEN</strong></td>
</tr>
<tr>
<td><strong>6. App closes</strong></td>
<td><strong>CLOSING</strong></td>
</tr>
<tr>
<td><strong>7. TIME-WAIT</strong></td>
<td><strong>TIME-WAIT</strong></td>
</tr>
</tbody>
</table>
Packet header

- Gray portion not on all packet types

- Different headers for different packet types (unlike TCP)

- Reduce byte overhead for unidirectional connections
### Packet header

![Packet header diagram](image_url)

- **CsCov** supports partial checksums
  - Errors in header result in packet drop
  - Errors outside Checksum Coverage ignored
  - 0–56 bytes of payload can be covered, or all payload
Ack Vector option

- Run-length encoded history of data packets received
  
  Cumulative ack not appropriate for an unreliable protocol
  
  Steroidal SACK

  +---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------+
  |001001??| Length |SSLLLLLL|SSLLLLLL|SSLLLLLL| ... |States (SS) |
  +---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------+---------------------------------+
  0 received non-marked
  1 received ECN marked
  3 not yet received

  Type=37/38 \___________ Vector \___________...

  Up to 16192 data packets acknowledged per option
  
  Includes ECN nonce
Data Dropped option

- Ack Vector says whether a packet’s header was processed
  Not whether packet’s data will be delivered to application
  Supports drop-from-head receive buffers, …

- Data Dropped says whether a packet’s data was delivered
  And if not, why not
  Enables richer [non-]congestion response functions

```
+--------+--------+--------+--------+--------+--------+--------+--------+
|00100111| Length | Block  | Block  | Block  | ... 
+--------+--------+--------+--------+--------+--------+--------+
```

Type=39 \___________ Vector __________ ...

```
  0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7   Drop States
  +----------+----------+   0 protocol constraints
  |0| Run Length | or |1|Dr St|Run Len| 1 receive buffer
  +----------+----------+   2 corrupted
  Normal Block | Drop Block 3 delivered corrupt
                | 4 app not listening
```
“VoIP issues” with CCID 3 (TFRC) and DCCP

- Protocol complexity
  
  New draft, CCID 3-Thin, enables a low-complexity subset

- Slow start?
  
  Now allow 4 packets/RTT (4380 payload bytes/RTT)
  
  40ms initial packetization interval for $\text{RTT} \leq 160\text{ms}$

- Rate slows down during idle periods
  
  Example: two-way phone
  
  TFRC limits sending rate to twice your actual sending rate in the last RTT
  
  Send idle packets?
“VoIP issues” 2

• Rate does not increase during app-limited period
  
  Again, can send up to twice your app-limited rate
  
  Don’t get to reserve bandwidth

• Variable rate considered harmful
  
  Apps might have discrete rates
  
  Seems fine to allow sending at slightly above the reference rate (up to $2\times$?)
  
  New draft needed

• Rate changes considered harmful
  
  Apps work at fixed rates, hard to switch
  
  App-specific
Conclusion

- [http://www.icir.org/kohler/dccp/](http://www.icir.org/kohler/dccp/)
  - *draft-ietf-dccp-spec-05.txt*: main specification
  - *draft-ietf-dccp-ccid\{2,3\}-04.txt*: CCID specs
- New drafts coming by the end of the month
- WGLC in December
  - Comments welcome!