Designing DCCP: Congestion Control Without Reliability

By Eddie Kohler, Mark Handley and Sally Floyd
SIGCOMM’06, September 11-15, 2006, Pisa, Italy.

Presented By: Travis Grant & Harshal Pandya
10/03/06
CS577: Professor Robert Kinicki – Fall ’06

Acknowledgements: Adaptation from Presentation by Greg Kemp
The Need for Congestion Control

- UDP used instead of TCP by applications that prefer timeliness over reliability
- UDP lacks TCP’s congestion control
- Especially a problem with long-lived flows and lots of traffic (streaming video, audio, internet telephony)
- Greater use increases risk of congestion collapse
Related Work - What can be done?

• Below UDP: too low
• Above UDP: implement CC at application level
  – Lots of work, reinventing the wheel each time
  – CC is complex, might not be done correctly
  – New protocol more interoperable than a user-level library
  – (Alternatives: Congestion Manager)
• Modify TCP, UDP, RTP, SCTP
  – Makes these protocols complex (feature bloat)
  – Not general enough
  – Forces a reasonably fundamental change
• Alongside UDP and TCP Makes most sense
  – Primary goal is to allow dynamic CC Selection to accommodate varying application requirements
  – Hence -> Dynamic Congestion Control Protocol
History and state of the art

• Initially Datagram Control Protocol (DCP)
• July 2001: First Internet Draft
• February 2002: DCCP Problem Statement
• May 2002: Changed name to DCCP
• October 2003: Latest Internet Draft
• Implementations circa 2002 and late 2003
  – FreeBSD (kernel-level)
  – Linux (kernel-level and user-level)
Application Requirements

• Internet Telephony – VOIP
  – CBR like with extreme sensitivity to delay and quality fluctuation
  – Pressures transport layer to reduce overhead

• Video Conferencing
  – idle periods followed by need for immediate rate return

• Streaming Media
  – buffering can mask rate variation, but timeliness is priority of reliability
  – some CODECs drive drastic datagram size variance (and is expected)

• Interactive Gaming
  – timeliness is key for position information
  – outstanding data related to old positioning information may be entirely worthless and lead to wasted resources end-to-end

• Key takeaways
  – Application Requirements vary (can be extremely different)
  – UDP sometimes used by default to avoid both TCP constraints and Application Development efforts but lacks key features
Goals

1. Minimalism
   - simplicity in protocol design, implementation, and ability to leverage for applications
   - Protocol Overhead reduction

2. Robustness
   - Difficult to abuse
   - tread lightly on the core infrastructure

3. Framework for CC
   - Plug & Play without missing TCP key features

4. Self-sufficiency (i.e. API)
   - receivers capable of congestion detection
   - senders capable of rate calculation and fairness
   - CC parameters negotiated in-band

5. Support timing-reliability tradeoffs
   - coarse or fine grained control made available to the application
   - i.e. determining priority packets by type or age
DCCP Derived Requirements

• Some features are “good ideas” and can be borrowed from TCP, UDP:
  – Port numbers, checksums, sequence numbers (with difficulty), acks (congestion and ECN info), piggybacked acks
  – Three-way handshake to set up, two-way with wait to tear down

• New features/concepts:
  – Negotiate CC mechanism and parameters on setup
  – Two half-connections (A → B, B → A)
Deliberate Omissions

1. **Flow Control**
   - implicitly achieved through CC
   - Possibly implemented above DCCP if desired

2. **Selective Reliability**
   - no clear benefit for strict differentiation between prioritization of packets and re-transmitting

3. **Streams**
   - half-connection abstraction and inherently unreliable nature make blocking non-issue
   - Trivial to layer above DCCP if required

4. **Multicast**
   - Complexities associated with all aspects of DCCP and multicast deemed out of scope
DCCP Overview

- unreliable, unicast, connection oriented, w/ bi-directional data flow
- Header >= 16Bytes (vs. TCP>=20Byte & UDP>=8Byte)
- 10 Packet Types
  - Different Packet Types = diff. options
  - Allows flexibility
  - Avoids unnecessary clutter
- Data Offset (to start of data)
  - 8 bits long
  - allows 1000 bytes of option
- Even Ack# is optional
  - Potential header overhead reduction

Fig. 1: DCCP packet flow
DCCP Overview (Cont.)

- DCCP has no equivalent to TCP
  → rec. window, urgent ptr field, PUSH or URG flags
- TCP has no equivalent to DCCP
  → CCVal, CsCov/Checksum Coverage, or ACK Vector
- Seq. and Ack #s are 48 bits long
  - vs. TCP@32Bits
  - some packets permit a compact form
  - 48 bits always for connection initiation, synchronization & teardown
  - 24 bits possible for data and ACK packets (negotiated by endpoints)
Issues dealing with Sequence Numbers

- Problems with TCP sequence numbers
- DCCP sequence numbers
- Synchronization
- Acknowledgements
- Sequence number length
- Robustness against attack
Problems with TCP Sequence Numbers

• The main problem with TCP is that pure acknowledgements that don’t contain data, SYN & FIN cannot be acknowledged.

• This is because SYN, FIN occupy sequence space whereas others do not.

• So TCP cannot evaluate the loss rate for pure acknowledgements. Nor can it detect or react to reverse path congestion.
DCCP Sequence Numbers

• Expectations from DCCP:
  – DCCP must be able to detect loss without application support
  – DCCP headers must include sequence numbers that measure datagrams rather than bytes because unreliable applications generally send and receive datagrams rather than portions of a byte stream

• Solutions:
  – Most DCCP packets carry an acknowledgement number as well as a sequence number
  – In DCCP, every packet, including pure acknowledgements, occupies sequence space and uses a new sequence number
  – Cumulative acknowledgements don’t make sense as there are no retransmissions
  – DCCP’s ackno reports the latest packet received, rather than the earliest not received.
Synchronization

• DCCP supports explicit synchronization

• An endpoint receiving an unexpected sequence or acknowledgement number sends a Sync packet asking its partner to validate that sequence number

• The other endpoint processes the Sync and replies with a SyncAck packet

• When the original endpoint receives a SyncAck with a valid ackno, it updates its expected sequence number windows based on that SyncAck’s seqno
Consider the ackno on a Sync packet. In the normal case, this ackno should equal the seqno of the out-of-range packet, allowing the other endpoint to recognize the ackno as in its expected range.

However, the situation is different when the out-of-range packet is a Reset, since after a Reset the other endpoint is closed.

If a Reset had a bogus sequence number (due maybe to an old segment), and the resulting Sync echoed that bogus sequence number, then the endpoints would trade Syncs and Resets until the Reset’s sequence number rose into the expected sequence number window (First Figure).

Instead, a Sync sent in response to a Reset must set its ackno to the seqno of the latest valid packet received; this allows the closed endpoint to jump directly into the expected sequence number window (Second Figure).
Acknowledgements

• There is a lot of state that the receiver has to maintain about the packets that are received & the acknowledgements that are sent
• To help the receiver prune this state, occasionally, pure acknowledgements must also be acknowledged by the sender
• Acknowledgements don’t necessarily guarantee that data has been delivered to the application. So older packets will be dropped if there are many newer packets in queue
• There are many options in DCCP acks that precisely tell the sender about the fate of the packet
Sequence Number Length

- Initially DCCP used only 24-bit sequence numbers as shown. This had a problem of wrapping too quickly.
- But 24 bits are too less. For ex. a 10 Gb/s flow of 1500-byte packets will send $2^{24}$ packets in just 20 seconds.
- Hence the best solution was to lengthen sequence numbers to 48 bits.
- However forcing the overhead on all the packets was considered unacceptable.
Sequence Number Length (contd..)

• Hence endpoints would now choose between short & long sequence numbers
• The following procedure takes a 24-bit value \( s \) and an expected sequence number \( r \) and return’s \( s’ \) 48-bit extension
• It includes two types of comparisons, absolute (written \(<\) ) and circular mod \( 2^{24} \) (written \( \ominus \))

\[
\begin{align*}
  r_{\text{low}} & := r \mod 2^{24}; \quad r_{\text{high}} := \lfloor r/2^{24} \rfloor; \\
  \text{if} \ (r_{\text{low}} \ominus s < r_{\text{low}}) & \quad \text{// \( s \) incremented past} \ 2^{24} - 1 \\
  & \quad \text{return} \ ((r_{\text{high}} + 1) \mod 2^{24}) \times 2^{24} + s; \\
  \text{else if} \ (s \ominus r_{\text{low}} < s) & \quad \text{// \( s \) decremented past} \ 0 \ (reordering) \\
  & \quad \text{return} \ ((r_{\text{high}} + 2^{24} - 1) \mod 2^{24}) \times 2^{24} + s; \\
  \text{else} & \quad \text{return} \ r_{\text{high}} \times 2^{24} + s;
\end{align*}
\]
Robustness against attack

• TCP:
  – SYN flooding is a popular attack on TCP. Another less popular form of attack is data injection into hosts, by guessing sequence & acknowledgement numbers.

• DCCP:
  – But the 48-bit sequence numbers of DCCP make the attacks much more difficult to execute.
  – DCCP is also immune to SYN attack. If a Request packet hits the sequence window of an active connection, the receiving endpoint simply responds with a Sync.

• So the goal of reducing overhead by introducing short sequence numbers & removing acknowledgement numbers, actually conflicts with security.
Issues dealing with Connection Management

• Asymmetric communication
• Feature negotiation
• Mobility & multihoming
• Denial-of-service attacks
• Formal Modelling
Asymmetric Communication

- DCCP provides a single bidirectional connection: data and acknowledgements flow in both directions.

- If B is sending only acknowledgements to A, then A should acknowledge B’s packets only as necessary to clear B’s acknowledgement state; these acks-of-acks are minimal and need not contain detailed loss reports.

- To solve these issues cleanly, DCCP logically divides each connection into two half-connections.

- A half-connection consists of data packets from one endpoint plus the corresponding acknowledgements from the other.

- When communication is bidirectional, both half-connections are active, and acknowledgements can often be piggybacked on data packets.

- Each half-connection has an independent set of variables and features, including a congestion control method.
Feature Negotiation

• Per endpoint property on whose value both endpoints must agree. They are essentially a set of parameters.

• Some of the examples of features are: Congestion control mechanism, whether or not short sequence numbers are allowed, mechanisms to be implemented etc.

• It involves two option types:
  – Change Options: They are retransmitted as necessary for reliability.
  – Confirm Options: It’s a single exchange options.

• Both the option types contain preference lists which the endpoints analyze to find the best match.
Mobility and Multihoming

• It essentially talks about mobility of hosts when DCCP is implemented

• Mobility could be implemented entirely at the network layer, as with Mobile IP, but choosing the transport layer has advantages

• The transport layer is naturally aware of address shifting, so its congestion control mechanism can respond appropriately, and transport-layer mobility avoids triangle routing issues

• DCCP’s mobility & multihoming mechanism joins a set of component connections each of which may have different endpoint addresses, ports, sequence numbers & connection features into a single session
Denial-of-service Attacks

• Attack:
  – In a transport-level denial-of-service attack, an attacker tries to break a victim’s network stack by overwhelming it with data or calculations
  – For example, the attacker might send thousands of TCP SYN packets from fake (or real) addresses, filling up the victim’s memory with useless half-open connections

• Defense Strategy:
  – The basic strategy is to push state to the client whenever possible
  – In DCCP, a server responding to a Request packet can encapsulate all of its connection state into an Init Cookie option, which the client must echo when it completes the three-way handshake
  – This lets the server avoid keeping any information about half-open connections
  – DCCP servers can also shift Time-Wait state onto willing clients
  – All DCCP connections end with a single Reset packet, and only the receiver of that Reset packet holds Time-Wait state.
  – Normal connections end with a Close–Reset handshake, but only the server can initiate shutdown with a CloseReq packet, which effectively asks the client to accept Time-Wait state
Formal Modeling

• The initial DCCP design was completed without benefit of formal modeling
• Later an independently developed colored Petri net (CPN) model from the University of South Australia was used. This tool was extremely useful in revealing several subtle problems in the protocol
• The resulting precision revealed several places where the design could lead to deadlock, livelock, or other confusion. Ex. The CPN model found the half-open connection recovery problem
Congestion Control

• DCCP Provides a Framework
  – Choice of CC
• CCID – neg. at connection startup
  – CCID 2: TCP-Like
  – CCID 3: TFRC
CCID 2: TCP-Like

• Ack Vector Option (vs. TCP SACK)
  – similar variables – cwnd, slow-start threshold & estimated data packets outstanding

• Reverse-path Congestion – Ack Ratio = \( R \)
  – Rough Ratio of data packets per acknowl.
  – \( R \) default is 2 (akin to TCP-like delayed-ack)
    – <2 (min. 4 packet cwnd)
    – max. \( R \) is cwnd/2 (rounded up)

• Algorithm
  – For each cwnd of data where \( \geq 1 \) ack is lost or marked \( \rightarrow \) then \( R \) is doubled
  – For each cwnd/(\( R^2 - R \)) consecutive cwnd where 0 acks were lost (or not marked) \( \rightarrow \) then \( R \) is decreased by 1
  – since \( R \) is an integer we find \( k \)
    • after \( k \) congestion free windows \( \rightarrow \) cwnd/R+k=cwnd/(R-1)
CCID 3: TFRC

- sending rate used (instead of cwnd)
- receiver sends feedback (~per RTT)
- sender used feedback to determine sending rate
- if no feedback is received for several RTT then sending rate is cut in half
- To avoid security concerns (hijacker sending erroneous loss information) an *loss intervals* are used
CCID 3: TFRC (Cont.)

Loss Intervals

- # of acks is limited → does not require cc for acks
- Sender attaches a coarse-grained timestamp (4bytes)
- Sender calculates loss event rate
- Each Loss interval contains a maximal tail of non-dropped, non-marked packets
  - DCCP header option – Loss Intervals report each tail’s ECN nonce echo
  - receiver reports <= 9 most recent Loss Intervals

- Key takeaway: Unlike TCP SACK, CCID 3 allows several distinct losses to be represented in a single range representing a single congestion event inside each RTT
CC Challenges

• Problematic Application Demands
  – Fast small packet send rate
    • vs. large packets with slower send rate
  – Rapid startup after idle periods
    • i.e. VOIP conversations
  – Abrupt changes in data rate
    • i.e. MPEG I-frame vs. B/P-frames

• AMR CODEC Example
  – requires < 12kbps (>5KBps) but given end-user experience
    playout must be constant (forcing constant packet rate -> ability
    of application to playout)
  – 20ms audio frame >= 14bytes (very small)
  – Packet Rate vs. throughput becomes key focus area
  – Requires min. 50 packets/second
Send rate vs. drop rate & CC Choices

Start @ 50 Packets/s

Fairness Impacted?
File Transfer vs. VOIP

Effective Throughput?
@ Start = for all 3

Right Choice?
Bytes vs. Packet Drop
Flat line is good (end-user)
similar curve is fair
Partial checksums

- From UDP-Lite
- Checksum covers DCCP header and (optionally) any number of bytes into payload
  - CsCov Field
- Allows delivery of slightly damaged data
  - Preferred for some target Applications
- May be useful on error-prone links (eg. wireless)
  - non-congestion associated corruption and packet loss
- NOTE: Still needs to be proven useful
Conclusions

• Supports Modular CC
  – Adaptable to Ongoing CC Improvements
• Flexibly handles varying Application requirements
• Control Loop of CC Mechanisms forced acknowledge format
• Robustness and Security proved difficult but achievable
  – Formal Modeling helped design team considerably
  – Simplicity due to Unreliable nature was an incorrect assumption
• Adoption is yet to be determined
  – faces common challenge of competing with TCP
  – Linux and FreeBSD implementations available
  – RFC and IETF ongoing work