DCCP : Datagram Congestion Control Protocol (rev. 11)

Eugen Dedu

1st April 2005

Brief presentation of DCCP

- DCCP = “UDP with congestion control”
- Formerly called DCP (C from control)
  - easily misheard as TCP :o(
- DCCP is :
  - a new transport protocol
  - provides a congestion control
  - that's all...
- So DCCP is not :
  - reliable and/or ordered and/or flow control: no retrans

DCCP draft organisation

- DCCP gathers several drafts:
  - DCCP protocol (1 draft)
  - congestion control protocols (4 drafts)
  - mobility (1 draft)
  - user guide (1 draft)

Why presenting DCCP ?

- NetMoVie: video transport
  - DCCP might be the answer
- New Internet protocol (current trends)
  - currently at Internet Draft stage (work in progress)
- Many new ideas about transport protocol
- (We focus on reasons instead of presentation)

Biblionetographie

- http://www.icir.org/kohler/dccp/ -- a lot of info
  - "Designing DCCP: Congestion Control Without Reliability", Kohler, Handley & Floyd, 2003 – obsolete, but explains the reasons
  - "DCCP Overview", Kohler&Floyd, 2003

Plan

- Introduction
  - congestion control
- A new protocol requirements
  - Congestion Manager
- DCCP header
- DCCP functioning
  - TFRC protocol
- DCCP prototype implementations
- Conclusions and perspectives

1. Reasons of DCCP apparition (1/2)

- Nowadays, more and more applications with long-lived flows transport huge data over the Internet:
  - Internet telephony
  - video streaming
  - on-line game streaming
- They cannot use TCP, since it causes unacceptable delays (fiability, ordering, ...)
- They use UDP, which does not provide congestion control

Reasons of DCCP apparition (2/2)

- If congestion control is to be written:
  - it takes time to code it
  - it is difficult to achieve a “good” implementation (fairness towards existing TCP flows, use all the available bw)
- => Congestion control is seldom taken into account

Importance of congestion control on the Internet

- Normal users are defavorised over malicious users
  - e.g. a malicious user solely may take 90% of the bw
- Congestion collapse
  - packets are dropped at the last moment (router), after having already used network resources
  - network is used at e.g. 1% of its capacity

Forms of congestion collapse on the Internet

- Classical congestion collapse
  - more packets sent => more packets dropped & retr-ed
  - solved by TCP's congestion control (Jacobson 1988)
- From undelivered packets
  - packets are dropped before reaching destination, but after consuming network resources
- Other forms
  - fragmentation-based, increased control traffic, stale pkts
- (Floyd&Fall, “Promoting the use of End-to-End Congestion Control in the Internet”, 1999)

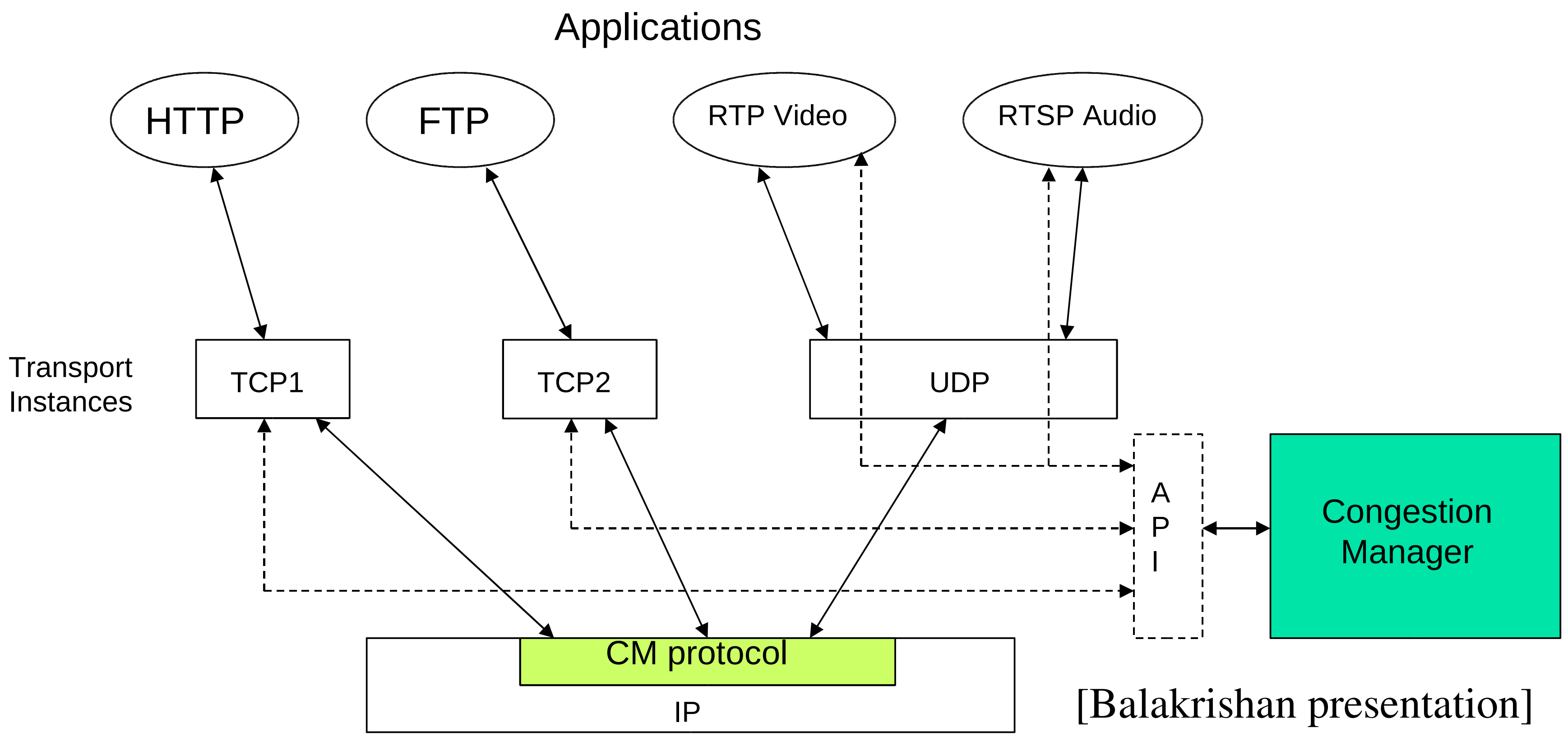
2. New protocol requirements

- Choice of congestion control mechanism
  - e.g. reactive/abrupt, monotonic/smooth
- Low per-packet overhead
  - Internet telephony, games send frequently small packets
- ECN support
  - especially for applis with tight timing constraints
- Allow middlebox traversal (firewalls, NATs)
  - connection-oriented

Possible solutions (1/2)

- Above UDP: user-level library
  - lower speeds (e.g. application-generated ACKs)
  - low interoperatibility between hosts
- Below UDP:
  - doesn't have multiple congestion control mechanisms
  - does not consider the middlebox traversal
  - still relies on application-level feedback
  - example: CM (Congestion Manager) [RFC 3124, Balakrishnan 2001]: a low-level module which takes care of all the network communications of a machine

Congestion Manager



[Balakrishan presentation]

- Cope naturally with multiple connections within the same application!

Possible solutions (2/2)

- Same level as UDP:
  - modify TCP: byte-oriented streams, cumulative acks
  - modify SCTP: have unnecessary functionality
  - RTP: different functions => different protocols
  - => DCCP, a transport protocol used separately by each application
- All major applications should be TCP-friendly
- DCCP is the basic stone and only that
- => will DCCP be widely deployed?
  - future: TCP over DCCP??

What DCCP provides

- An unreliable flow of datagrams, with acks
- A reliable handshake for connection init and end
- A choice of TCP-friendly congestion control mechanisms
- Reliable negotiation of features (eg. CC protocol)
- ECN-capable
- Options to tell the sender what packets have reached the receiver, ECN info
- Path MTU discovery [RFC 1191]...

3. DCCP header

- Contains:
  - generic header
  - type-dependent header
    - ack, request, response, ...
  - options (optional)

Generic header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |           Dest Port           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | CCVal | CsCov |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Res | Type  |X|    Reserved   |  Sequence Number (high bits)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Sequence Number (low bits)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Data offset:
  - inside the packet, in 32-bit words => header size
- CCVal (Congestion Control Value):
  - 4 bits of information useful to the CC mechanism of the sender

CsCov and Checksum fields

- CsCov: Checksum coverage (à la UDP-Lite)
- Reason: some appli (e.g. audio) prefer to receive partially damaged data rather than not receive them
  - unuseful if MAC-level performs the check too
- Parts covered by the checksum:
  - CsCov = 0: all headers + data (the whole packet)
  - CsCov=1-15: all headers + (CsCov-1)*4 bytes of data
    - if CsCov=1, only the headers are covered

Generic header

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |           Dest Port           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | CCVal | CsCov |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Res | Type  |X|    Reserved   |  Sequence Number (high bits)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Sequence Number (low bits)                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Sequence Number: increases by 1 for each packet
  - 48-bit, at least at connection initialisation
  - anti-hijacking, for high-rate connections
- X (Extended Sequence Numbers):
  - if 0, 24-bit sequence numbers instead of 48-bit
    - generic header and ack header are 32 bit smaller
- Res & Reserved: must be set to 0

Type field

- Unlike TCP, each DCCP packet is one of 16 types
  - 4 bits: 16 types in DCCP, compared to 4 in TCP
- Reduces the header size
- Types (detailed later):
  - request, response
  - data, ack, dataack
  - closereq, close, reset
  - sync, syncack
  - reserved (currently unused) types (6 types)

Type-dependent header

- All packet types have specific headers
- All headers except DCCP-Request and DCCP-Data carry this subheader:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Reserved            |     Ack Number (high bits)    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Acknowledgement Number (low bits)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Options

- All packets may contain options
- All the options occupy a multiple of 4 bytes
- 1st byte: option type
- For 0 <= type <= 31: single-byte options
  - padding, slow receiver, ...
- For type >= 32, the 2nd byte is the option length
  - change/confirm (see later), init cookie, ack vector, data dropped (at receiver), timestamp, elapsed time
    - elapsed time: between data reception and ack sending

Options: ack vector

- SACK [RFC 2018]: acks contains the list of non-contiguous seq number received (reliability-based)
- DCCP: receiver keeps telling the sender that packet k has been received until the sender acks some receiver message that included an ackvector covering k
- Ackvector: which packets have been received and their ECN status
- A second vector may be used for packets dropped by the receiver (e.g. full receiver buffer)

Options: data dropped

- Allows data drop differentiation
- On application packet drop the receiver must respond with packet received together with data drop option
- This option contains blocks of contiguous data
- Such block contains a drop code, e.g.:
  - 1: application no longer listening
  - 2: dropped in the receive buffer, probably overflow
  - 3: packet corrupted and removed
  - 7: packet corrupted, but delivered to the application

4. DCCP functioning

- Plan:
  - classical connection
  - feature negotiation
  - packet size
  - reliable acks
  - bidirectionality
  - quiescence
  - CC mechanisms
  - mobility

Classical connection: communication types

```
Client                                              Server

                 (1) Initiation

DCCP-Request -->

                                          <-- DCCP-Response

DCCP-Ack -->

                 (2) Data transfer

             <-- DCCP-Data, DCCP-Ack, DCCP-DataAck

DCCP-Data, DCCP-Ack, DCCP-DataAck -->

                 (3) Termination

                                          <-- DCCP-CloseReq

DCCP-Close -->

                                             <-- DCCP-Reset
```

DCCP-Sync, DCCP-SyncAck: synchronise sequence numbers after large bursts of loss

Feature negotiation

- During request/response packets of connection initiation
  - TCP: ECN, SACK, window scaling, timestamps
  - DCCP generalises: CC protocol used, ECN-capable, ...
- Negotiation options:
  - change
  - prefer
  - confirm
- Change/prefer are exchanged until consensus

Packet size

- The appli find out the MTU by PMTU discovery
- It sends packets with max size of MTU
  - headers included!

Reliable ACKs (1/2)

- Acks are used for congestion control, not for retransmission
  - TCP uses both (feature aggregation)
  - TCP-like cumulative acks make no sense
- Both ECN mechanism and sender application might need to know precisely what packets were lost
  - example: sender appli would like to adjust the number of video layers sent based on lost packet rate
  - => ack state at the receiver may grow indefinitely

Reliable ACKs (2/2)

- Possible solution: sender occasionally sends "I received feedback for everything up to seq s"
  - limited: works only for acks
- DCCP solution: acks take up sequence numbers
  - when sender acks an ack A, it means it received all A's ack options
- Advantages: sender:
  - knows what acks have been lost
  - detects reverse-path congestion (useful on bandwidth-asymmetric networks)

Bidirectionality

- Needed even for strong unidirectional appli, like video streaming, because of pause/rewind/...
- DCCP: a bidirectional connection divided in two logical half-connections
  - data A->B, acks B->A
  - data B->A, acks A->B
- They can use different features (e.g. CC mechanism)

Quiescence

- Quiescence = “repos”, i.e. unidirectional transmission:
  - A has data to send
  - at same point, B becomes quiescent => B sends only acks
- A considers B to become quiescent when B does not send data during a specified time, e.g. 2*RTT
- Example: TFRC uses cumulative acks, hence receiver acks do not need to be acked

Congestion control mechanisms

- TCP's bw is a zig-zag curve (may be halved), hence reactive, aggressive bw probing
- TFRC's bw curve is much more monotonic, better for video streaming
- Choice of CC, based on a CCID of standardised mechanisms:
  - CCID = 2: TCP-like
  - CCID = 3: TFRC

TCP-like CC

- A close variant of TCP
- SACK-based

TFRC CC

- [Floyd&al 2000, “Equation-based congestion...”]
- TCP: AIMD, uses congestion window
- TFRC: equation-based, uses sending rate
- Receiver sends feedback on losses once per RTT
- Sender adjusts correspondingly its sending rate
  - if no feedback for several RTTs, halve sending rate
- Packets are sent regularly
- => smooth bw changes

Mobility

- Removed from DCCP draft rev. 07 and put into a separate draft
- Incipient mobility
- Adds a DCCP-Move packet type
- When an endpoint changes IP address, it sends its new address in a DCCP-Move header
  - attack: if seq number is guessed, the receiver IP address may be changed to attacker address

Differences from TCP

- Packet stream vs. byte stream
- Unreliability
- No receive window
- Packet sequence numbers, acks have seq numbers
- Generic feature negotiation
- Choice of congestion control
- ...

5. Prototype implementations

- Implementations [http://www.icir.org/kohler/dccp]
  - kernel-space: linux, freebsd
  - user-space
  - some features not implemented (e.g. mobility)
- Application support [http://www.dccp.org]
  - ethereal: a patch exists (2003 Apr)
    - Internal DCCP represents another protocol!
  - ns2: a patch exists (2004 Mar)

6. Conclusions and perspectives

- DCCP provides only CC
- DCCP's major new ideas:
  - choice of CC mechanisms
  - partial checksums
  - feature negotiation
- Future: will TCP be based on DCCP??
  - would mix TCP features and DCCP design
- NetMoVie: create a video transport protocol above packet-based DCCP