Datagram Congestion Control Protocol (DCCP) Spec Walkthrough

Eddie Kohler
International Computer Science Institute

IETF 57 DCCP Meeting
July 16, 2003

Outline

- Problem & alternatives
- Design choices & philosophy
- Connection overview
- Generic header & sequence numbers
- Packet types
- Reliable feature negotiation
- Acknowledgement options
- CCIDs

Much

will

be

skipped

. . .

Problem

- Increasing use of UDP for long-lived flows
  - Streaming media, telephony, on-line games
  - Prefer timeliness over reliability
  - TCP can introduce arbitrary retransmission delay
- Growth of long-lived, non-congestion-controlled traffic poses a threat to the health of the Internet

Application requirements & preferences

- Different congestion control mechanisms
  - TCP-like: higher throughput, abrupt rate changes (games)
  - TFRC: steadier rate, lower throughput in changing env (telephony)
- Middlebox traversal
- Low per-packet byte overhead
- Buffering control: don’t deliver old data
- Access to ECN
- DoS avoidance

Alternatives

- Congestion control above UDP?

  Burdensome to app designer

  Access to ECN problematic

- Congestion control below UDP? (CM)

  Application feedback for acknowledgements burdensome

  Multiple CC mechanisms?

- Unreliable SCTP?

  Verbose header (multiple stream support)

  Single CC mechanism

- A new transport protocol?

  Best option

Fundamental design choices

- In-band signalling

  Alternative: assume a separate signalling channel

- Bidirectional communication

  Alternative: one-way data flow

- Per-packet sequence numbers

  Even pure acknowledgements occupy sequence number space

  Alternatives: per-byte or per-data-packet

Design philosophy

- Focus: modern congestion control
  - Provide access to all features required or helpful
  - Multiple CC mechanisms, ECN, ECN Nonce, partial checksums, . . .
- Ancillary features: consider inclusion if they cannot be layered on top
  - No support for multiple streams, partial reliability, FEC, . . .
  - Mobility cannot be layered on top
- “General principle of robustness”
  - Be conservative in what you do, liberal in what you accept from others (modulo security)
  - Reserve MUST for absolute interoperability requirements
  - Example: Reserved fields MUST be ignored, SHOULD be set to zero

Note

- This spec walkthrough refers to DCCP as it currently is defined, with changes suggested by reviewers. Most, but not all, of those changes are in the most recently available drafts.

Packet types

DCCP-Request	client → server: open connection
DCCP-Response	server → client: response
DCCP-Data	transmit data (no ack info)
DCCP-Ack	transmit ack info (no data)
DCCP-DataAck	DCCP-Data + DCCP-DataAck
DCCP-CloseReq	server → client: close connection
DCCP-Close	close connection
DCCP-Reset	destroy connection
DCCP-Move	move IP address/port

- No simultaneous open

States

CLOSED	nonexistent connection
LISTEN	server in passive listening state
REQUEST	client beginning handshake
RESPOND	server responding to request
OPEN	data transfer (TCP’s ESTABLISHED)
CLOSEREQ	server asking client to close
CLOSING	waiting for final Reset
TIME-WAIT	2MSL wait (at receiver of Reset)

- No half-closed states

Two half-connections

- A half-connection is data flowing in one direction, plus the corresponding acknowledgements
- A DCCP connection contains two half-connections

  A $\longrightarrow$ B data plus B $\longrightarrow$ A acks

  B $\longrightarrow$ A data plus A $\longrightarrow$ B acks

  Can piggyback acks on data (DCCP-DataAck)
- Conceptually separate

  May use different congestion control mechanisms
- Terminology

  Given a half-connection, the HC-Sender is the endpoint sending data, the HC-Receiver the endpoint sending acks

CCIDs & feature negotiation

- Congestion control mechanism represented by a CC Identifier (CCID)
  - CCID 2 = TCP-like, CCID 3 = TFRC
  - Defines how the HC-Sender limits data rates and how the HC-Receiver sends congestion feedback
- Feature negotiation
  - A generic mechanism to reliably negotiate the values of shared parameters
  - Example feature type: CCID
  - Each feature type corresponds to two independent features, one per half-connection

Choosing a CCID

- CCID 2 (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
- CCID 3 (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
- Neither appropriate for apps that vary packet size in response to congestion
  - Wait for standardization (TFRC-PS, . . .)

Sample connection: client close

	DCCP A				DCCP B
0.	CLOSED				LISTEN
1.	App opens				
	REQUEST	→	DCCP-Request	→	RESPOND
2.	OPEN	←	DCCP-Response	←	RESPOND
3.	OPEN	→	DCCP-Ack	→	OPEN
4.	Initial feature negotiation (CC mechanism, . . .)				
	OPEN	↔	DCCP-Ack	↔	OPEN
5.	Data transfer				
	OPEN	↔	DCCP-Data, -Ack, -DataAck	↔	OPEN
6.	App closes				
	CLOSING	→	DCCP-Close	→	CLOSED
7.	TIME-WAIT	←	DCCP-Reset	←	CLOSED

Sample connection: server close

	DCCP A				DCCP B
0.	CLOSED				LISTEN
1.	App opens				
	REQUEST	⟶	DCCP-Request	⟶	RESPOND
2.	OPEN	⟵	DCCP-Response	⟵	RESPOND
3.	OPEN	⟶	DCCP-Ack	⟶	OPEN
4.	Initial feature negotiation (CC mechanism, . . .)				
	OPEN	⟷	DCCP-Ack	⟷	OPEN
5.	Data transfer				
	OPEN	⟷	DCCP-Data, -Ack, -DataAck	⟷	OPEN
6.					App closes
	CLOSING	⟵	DCCP-CloseReq	⟵	CLOSEREQ
7.	CLOSING	⟶	DCCP-Close	⟶	CLOSED
8.	TIME-WAIT	⟵	DCCP-Reset	⟵	CLOSED

Packet 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           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type  | CCval |              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | # NDP | Cslen |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |          Acknowledgement Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Different packets have different headers
    - In all cases, header followed by options
    - Sometimes, options followed by payload

Ports, type, data offset

```
 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            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type  | CCval |              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | # NDP | Cslen |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |          Acknowledgement Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- `Source Port` and `Dest Port` as in TCP, UDP
- `Type` identifies packet type

  Not flags word; 7 types left for expansion

- `Data Offset`: header length, including options, in 32-bit words

  Up to 1008 bytes of options

CCval

```
 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           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type  | CCval |              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | # NDP | Cslen |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |          Acknowledgement Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- 4-bit space reserved for use by HC-Sender CCID

  Can remove the need for options, reducing byte overhead

  Example: TFRC’s Window Counter option

Checksum and Cslen

```
 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           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type  | CCval |              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | # NDP | Cslen |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |          Acknowledgement Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Checksum: Internet checksum of the DCCP header, options, a pseudoheader, plus some amount of the payload
- Cslen determines how much payload is covered by Checksum

  0: no payload covered

  15: all payload covered

  1–14: that many initial 32-bit words of payload covered

Partial checksums

- Inspired by UDP-Lite
- Motivation: Some links frequently deliver corrupt data
  - Link-layer retransmissions can greatly delay delivery
  - Our target applications can deal with loss, many can also deal with corruption
  - Delivering corrupt data may improve user’s perception of service quality
- Corruption is not always an indication of congestion
  - Congestion response to corruption too harsh on links with constant nonminimal corruption rate
  - Want to differentiate corruption loss and congestion loss, whether or not app can handle corrupt data
- Partial checksums not useful with IPsec AH

Payload Checksum option

- Useful particularly with partial checksums

    Partial header checksum cannot detect corruption in payload
    Payload checksum option detects payload corruption only

```
+--------+--------+--------+--------+
|00101101|00000100|    Checksum     |
+--------+--------+--------+--------+
 Type=45  Length=4
```

- `Checksum` is Internet checksum of payload

    If checksum broken, discard payload (or give to application with explicit corruption notification)

    Packet still “received”! (Data Dropped, later)

NDP

```
 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           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type  | CCval |              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | # NDP | Cslen |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |          Acknowledgement Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Number of non-data packets sent on the connection mod 16
- Intended mostly for HC-Receiver’s application

  Was any of my payload lost?

  Derive application sequence number from DCCP Sequence Number and # NDP

- Ambiguous after $\geq$ 16 consecutive lost packets

Sequence numbers

```
 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           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type  | CCval |              Sequence Number                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data Offset  | # NDP | Cslen |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |          Acknowledgement Number               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- `Sequence Number`

  Increases by one on every packet sent, including pure acks

  Wrapping an issue

- `Acknowledgement Number`

  Acknowledges GSR, greatest (mod 2^{24}) valid seqno received

  Not present on DCCP-Request and DCCP-Data packets

  Not a cumulative ack, not a promise of data delivery

Sequence number validity

- DCCP checks packet's Sequence and Acknowledgement Numbers for validity
    - Defense against delivery of old segments
    - Defense against half-open connections
    - Defense against attack
- General approach: Loss Window
    - Sequence numbers within Loss Window are valid
- Compare TCP's receive window
    - No cumulative ack, so packets older than GSR may be OK (reordering)
    - Not a flow control mechanism

Staying in sync

- Problem: sequence numbers advance on every packet
- A long enough burst of loss could cause the endpoints’ sequence numbers to get out of sync relative to any window

  Even if only acks are sent

- Need a mechanism to get back into sync

  Identification option

Hold on to your hats

Loss window width

- HC-Sender decides on a loss window width W_S for sequence numbers

  Should reflect how many packets the sender expects to be in flight

  Suggestion: 3–4x the maximum number of packets sent per RTT

  HC-Sender informs HC-Receiver of W_S through feature negotiation

  Too small $\longrightarrow$ often out of sync; too large $\longrightarrow$ attackable

  Defaults to 1000

- HC-Receiver decides on a loss window width W_A for ack numbers

  Equals the loss window width it chose in its role as HC-Sender on the other half-connection

Loss window definitions

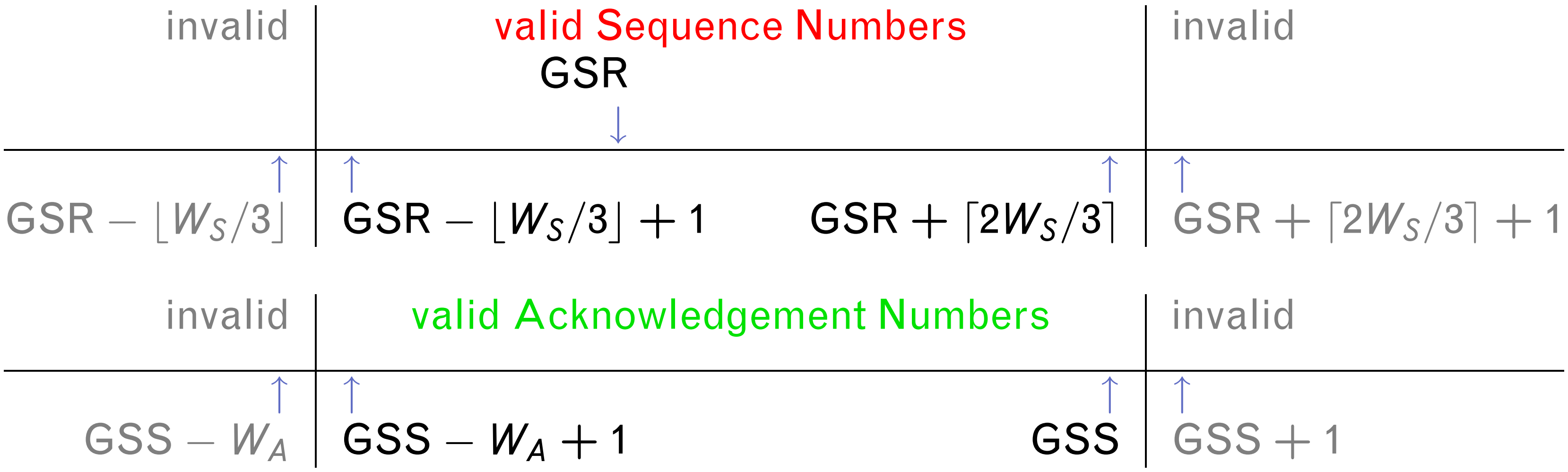
- CLOSED and LISTEN states

  All packets are sequence-valid

- Other states

  Sequence number must lie in $[\mathrm{GSR} - \lfloor W_S/3 + 1 \rfloor, \mathrm{GSR} + \lceil 2W_S/3 \rceil]$

  Acknowledgement number must lie in $[\mathrm{GSS} - W_A + 1, \mathrm{GSS}]$



invalid

valid Sequence Numbers

invalid

GSR

$\mathrm{GSR} - \lfloor W_S/3 \rfloor$

$\mathrm{GSR} - \lfloor W_S/3 \rfloor + 1$

$\mathrm{GSR} + \lceil 2W_S/3 \rceil$

$\mathrm{GSR} + \lceil 2W_S/3 \rceil + 1$

invalid

valid Acknowledgement Numbers

invalid

$\mathrm{GSS} - W_A$

$\mathrm{GSS} - W_A + 1$

GSS

$\mathrm{GSS} + 1$

Requirements for sequence validity

- (1) The Acknowledgement Number is in the relevant window, AND EITHER:
- (2a) The Sequence Number is in the relevant window, OR
- (2b) The packet has a correct Identification or Challenge option, OR
- (2c) The packet is a DCCP-Reset and its Sequence Number is zero.
- Explanation

  (1) prevents replay attacks

  (2b) is necessary for getting back in sync

  (2c) is necessary for cleaning up half-open connections

If a packet is sequence-invalid

- Send a DCCP-Ack

  Acknowledge the packet's Sequence Number (not GSR!)

  Include a Challenge option

  Exception: send nothing if the packet was a Reset

- DoS protection

  SHOULD ignore packets with bad Sequence Numbers if connection active (valid packet received within $\sim$ 1s or 1 RTT)

  MAY ignore packets with bad Sequence Numbers for some time after receiving an incorrect Identification option (checking Identification may be CPU intensive)

  MAY rate limit DCCP-Ack generation

Identification and Challenge

- Endpoints exchange random Connection Nonces at startup

  Or exchange them over a secure channel

- Identification option:

```
+--------+--------+--------+-- ... --+--------+
|00101010|00010010|    Identification Data    |
+--------+--------+--------+-- ... --+--------+
 Type=42   Len=18 \_________16 bytes__________/
```

  MD5 sum of packet's Sequence and Acknowledgement Numbers, this endpoint's Nonce, and the other endpoint's Nonce

  Sequence and Acknowledgement Numbers prevent replay

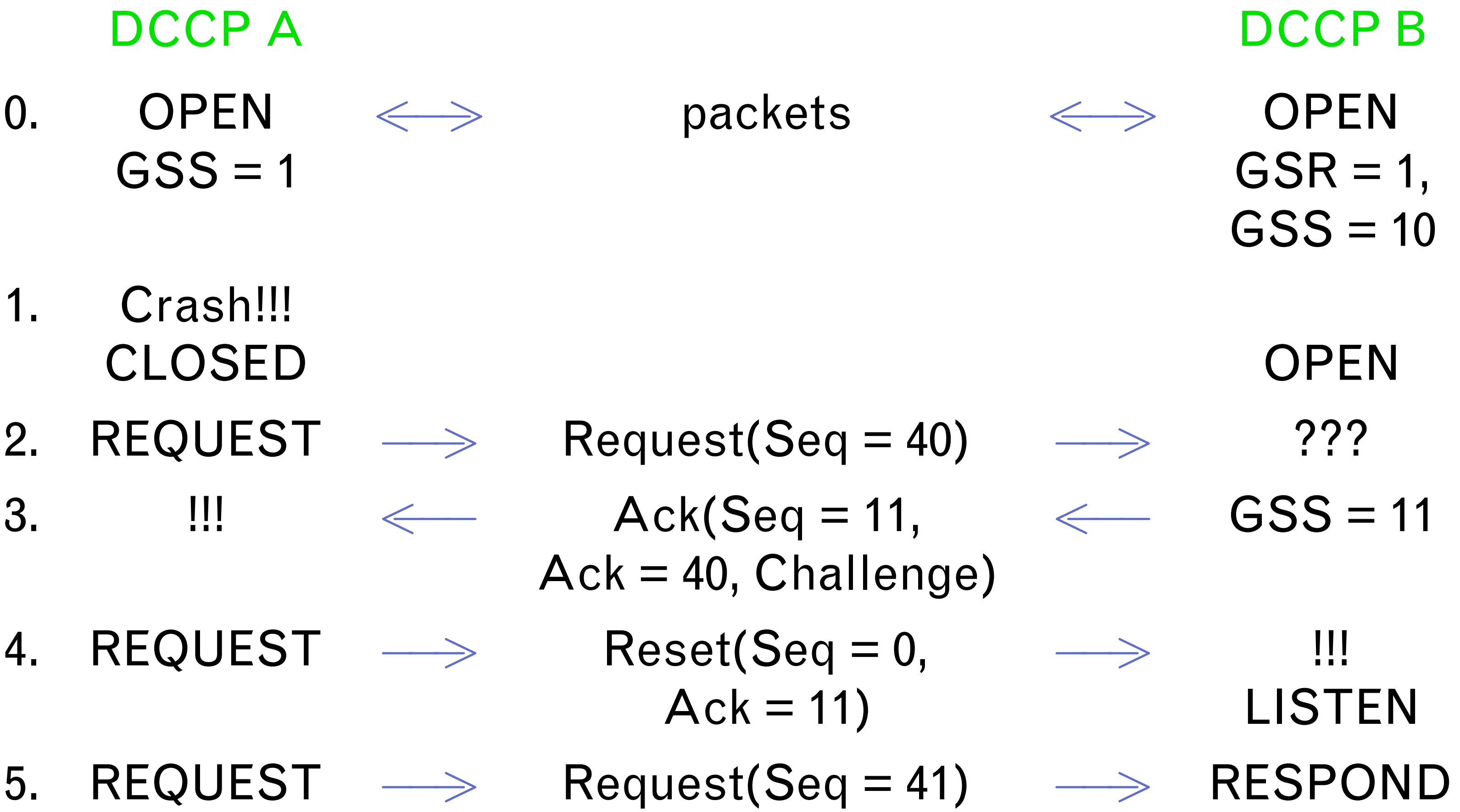
  Nonces prevent spoofing

- Challenge option is like Identification, but receiver should respond with Identification

Resync after burst of loss

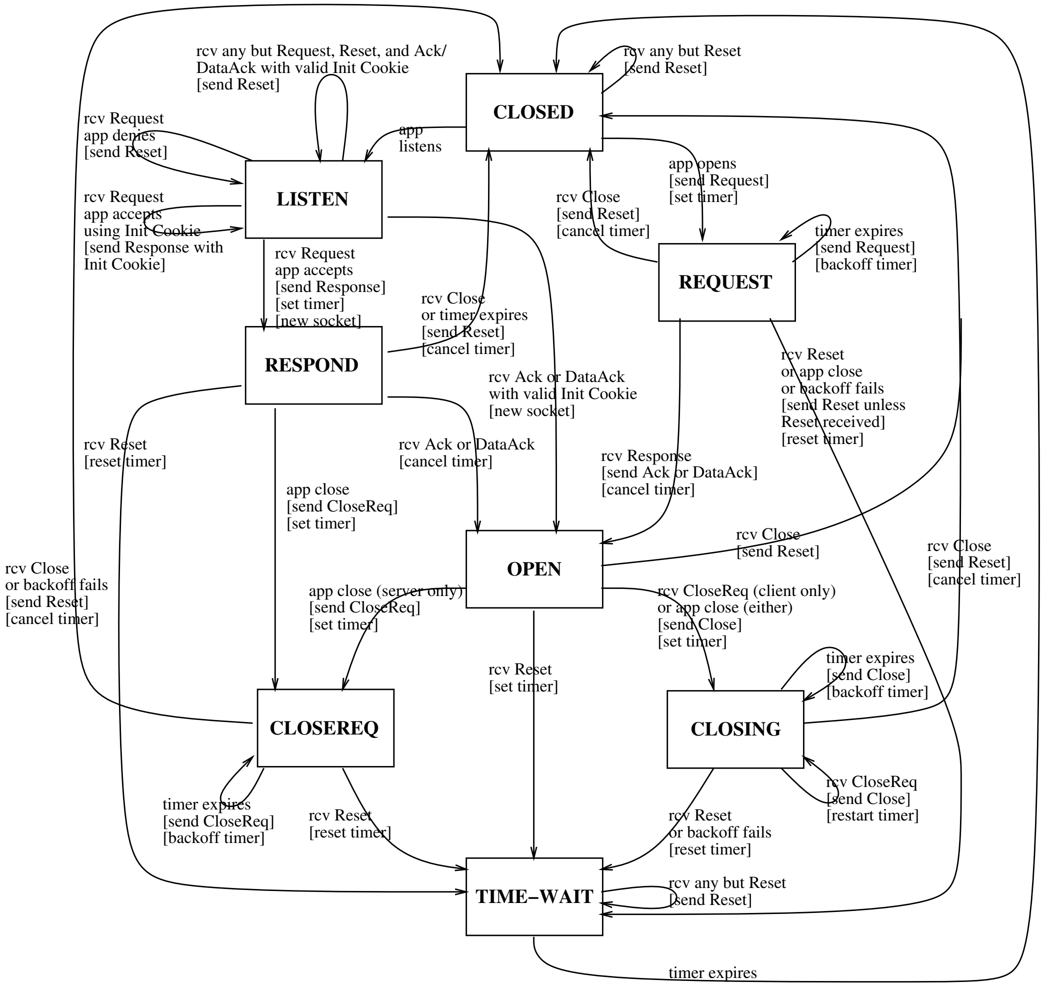
	DCCP A				DCCP B
0.	GSS = 10, GSR = 5				GSS = 5, GSR = 10
1.	GSS = 11	→	Data(Seq = 11)	XXX	
			. . .		
	GSS = 30	→	Data(Seq = 30)	XXX	
2.	GSS = 31	→	Data(Seq = 31)	→	???
3.	GSR = 6	←	Ack(Seq = 6, Ack = 31, Challenge)	←	GSS = 6, GSR = 10
4.	GSS = 32	→	Ack(Seq = 32, Ack = 6, Identification)	→	!!! GSR = 32

Half-open connection cleanup



	DCCP A				DCCP B
0.	OPEN GSS = 1	↔	packets	↔	OPEN GSR = 1, GSS = 10
1.	Crash!!! CLOSED				OPEN
2.	REQUEST	→	Request(Seq = 40)	→	???
3.	!!!	←	Ack(Seq = 11, Ack = 40, Challenge)	←	GSS = 11
4.	REQUEST	→	Reset(Seq = 0, Ack = 11)	→	!!! LISTEN
5.	REQUEST	→	Request(Seq = 41)	→	RESPOND

State diagram



In a table

- Respond to sequence-valid packets and timeouts as follows:

			Data/Ack/				
State	Request	Resp.	Move	C-Req	Close	Reset	[Timeout]
CLOSED	Rst	Rst	Rst	Rst	Rst	-	
LISTEN	RESP.	Rst	Rst(1)	Rst	Rst	-	
REQUEST	Rst	OPEN	Rst	Rst	Rst	TW	REQ
RESPOND	-/RESP.	Rst	Rst/OPEN	Rst	C-ED	TW	C-ED
Server OPEN	-/Rst	Rst	OPEN	Rst	C-ED	TW	
Client OPEN	Rst	-/Rst	OPEN	C-ING	C-ED	TW	
CLOSEREQ	-/Rst	Rst	C-REQ	Rst	C-ED	TW	C-REQ
CLOSING	Rst	-/Rst	C-ING	C-ING	C-ED	TW	C-ING
TIME-WAIT	Rst	Rst	Rst	Rst	Rst	-	C-ED

- Packets sent on state transitions:

REQUEST	Request
RESPOND	Response
OPEN	Ack/DataAck
CLOSEREQ	CloseReq
CLOSING	Close
CLOSED	Reset
TIME-WAIT	-

- App events:

Passive open	LISTEN
Active open	REQUEST
Close	C-REQ/CLOSING

DCCP-Request

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/                   with Type=0 (DCCP-Request)                  /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Service Name                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Options                   /   [padding]   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Service Name
  - Specifies an app-level service (’HTTP’ = 1213486160)
  - Ports also come with Service Names; mismatch causes Reset
  - IANA registry: first-come, first-serve
- Contains data
  - Server may ignore

DCCP-Response

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/                  with Type=1 (DCCP-Response)                  /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |           Acknowledgement Number              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Options                   /   [padding]   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Send in response to Requests
  - Including retransmitted Requests
  - Ignore data on retransmitted Requests
- For DoS protection, use Init Cookie

DCCP-[Data][Ack]

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/   with Type=2 (DCCP-Data), 3 (DCCP-Ack), or 4 (DCCP-DataAck)  /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |           Acknowledgement Number              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Options                   /   [padding]   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- DCCP-Data = black + green
- DCCP-Ack = black + blue
- DCCP-DataAck = black + green + blue

DCCP-Data: minimal overhead in common (unidirectional) case

DCCP-Ack: separate type enables 0-length datagrams

DCCP-Close[Req]

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/         with Type=5 (DCCP-Close) or 6 (DCCP-CloseReq)         /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |           Acknowledgement Number              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Options                   /   [padding]   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

DCCP-Reset

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/                   with Type=7 (DCCP-Reset)                    /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |           Acknowledgement Number              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Reason     |    Data 1     |    Data 2     |    Data 3     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Options                   /   [padding]   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Reason specifies why the connection was reset

  The Data bytes give more detail

- Example Reasons: Closed (normal close), Aborted, Fruitless Negotiation (a feature negotiation took too long), etc.

Mobility

- Cannot be layered on top

  Part of our charter

- Basic mechanism: endpoint moves, sends DCCP-Move from new address

  DCCP-Move contains old address so flow can be identified

  Also security mechanisms (Identification)

  Stationary endpoint acknowledges with DCCP-[Data]Ack

- Authors vote “+0.1” on mobility

  But cleanly separable from rest of protocol

  Doesn’t add complexity outside of Move itself

DCCP-Move

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                 Generic DCCP Header (12 bytes)                /
/                    with Type=8 (DCCP-Move)                    /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Reserved    |           Acknowledgement Number              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Old Address Family       |           Old Port            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                          Old Address                          /
/                                               /   [padding]   /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Options, including Identification      /   [padding]   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
|                              ...                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- `Old Address Family, Address, Port` = old address

  What about NATs?!

DCCP-Move security

- Mandatory `Identification` prevents hijacking

  Unless attacker snooped on Nonce exchange

- Ignore invalid Moves

  Invalid sequence numbers, Identification, or connection not mobility capable

  Do not send Reset or Ack—would leak information!

- DoS resistance

  MAY ignore all Moves for some time after receiving an invalid Move

Options

- One-byte options: Type = 0 . . . 31

```
+--------+
|Opt Type|
+--------+
```

- Multibyte options: Type = 32 . . . 255

```
+--------+--------+--------+--------+--------
|Opt Type| Length | Data ...
+--------+--------+--------+--------+--------
```

Length ≥ 2

Ignored option

```
+--------+--------+--------+--------+--------
|00100000| Length |Opt Type|  Opt Data ...
+--------+--------+--------+--------+--------
 Type=32
```

- Informs receiver that one of its options was not understood

CCID-specific options

- CCIDs will need to allocate options

  New acknowledgement formats, . . .

  A shame to deal with IANA for this

- Options 128 . . . 255 are CCID-specific

  128 . . . 191: option sender is HC-Sender

  192 . . . 255: option sender is HC-Receiver

Feature negotiation

- The endpoints must agree on several of the connection's parameters

  The CCIDs, CCID-specific settings, Loss Window, Connection Nonces, . . .

- This agreement must be reliable

  A shame to reinvent reliability for each feature

- Invent a general framework for features

  = Things that will be reliably negotiated

  Identified by one-byte feature numbers

  Use three options to negotiate feature values

Change, Prefer, Confirm

- Change: “Please use this value for a feature”

```
+--------+--------+--------+--------+--------+--------
|00100001| Length |Feature#| Value or Values ...
+--------+--------+--------+--------+--------+--------
 Type=33
```

- Prefer: “I would rather use one of these values”

```
+--------+--------+--------+--------+--------+--------
|00100010| Length |Feature#| Value or Values ...
+--------+--------+--------+--------+--------+--------
 Type=34
```

- Confirm: “OK, I am using this value”

```
+--------+--------+--------+--------+--------+--------
|00100011| Length |Feature#| Value ...
+--------+--------+--------+--------+--------+--------
 Type=35
```

Reliability through retransmission

- Retransmit Change until you get a response
  - Response = Prefer, Confirm, or Ignored
- Retransmit Prefer until you get a response
  - Response = Change, or Ignored
- Piggyback feature negotiation on existing traffic, or use additional Acks as allowed by CCID
- State diagrams in draft
  - Need more specification of retransmission algorithm
  - Only for non-reordered packets

Feature negotiation examples

DCCP A & Assumed Value					DCCP B & Actual Value	
KNOWN	4		(Initial State)		KNOWN	4
CHANGING	4	→	Change(CC, 2)	→	KNOWN	2
KNOWN	2	←	Confirm(CC, 2)	←	KNOWN	2
CHANGING	4	→	Change(CC, 2)	→	CONFIRMING	4
CHANGING	4	←	Prefer(CC, 1, 3)	←	CONFIRMING	4
CHANGING	4	→	Change(CC, 3)	→	KNOWN	3
KNOWN	3	←	Confirm(CC, 3)	←	KNOWN	3
CHANGING	4	→	Change(CC, 2)	→	KNOWN	2
		XXX	Confirm(CC, 2)	←	KNOWN	2
CHANGING	4	→	Change(CC, 2)	→	KNOWN	2
KNOWN	2	←	Confirm(CC, 2)	←	KNOWN	2

CCID-specific features

- Features 128 . . . 255 are reserved for CCIDs

  128 . . . 191: feature located at HC-Sender

  192 . . . 255: feature located at HC-Receiver

- Examples

  Say A $\longrightarrow$ B using CCID 2, B $\longrightarrow$ A using CCID 3

  A $\longrightarrow$ Change(128, Foo) $\longrightarrow$ B refers to CCID 3's feature 128 @ B

  A $\longrightarrow$ Change(192, Foo) $\longrightarrow$ B refers to CCID 2's feature 192 @ B

  A $\longrightarrow$ Prefer(128, Foo) $\longrightarrow$ B refers to CCID 2's feature 128 @ A

  A $\longrightarrow$ Prefer(192, Foo) $\longrightarrow$ B refers to CCID 3's feature 192 @ A

  A $\longrightarrow$ Confirm(128, Foo) $\longrightarrow$ B refers to CCID 2's feature 128 @ A

  A $\longrightarrow$ Confirm(192, Foo) $\longrightarrow$ B refers to CCID 3's feature 192 @ A

Other options

- Init Cookie
  - Like a large SYN Cookie: DoS protection
  - Server sends on Response
  - Packages up connection state
  - Client must echo on its Ack
  - Server can forget connection until the Ack arrives

```
+--------+--------+--------+--------+--------+--------
|00100100| Length |         Init Cookie Value    ...
+--------+--------+--------+--------+--------+--------
 Type=36
```

- Timestamp, Elapsed Time, Timestamp Echo
  - See draft

CCIDs

- Each congestion control mechanism corresponds to a CCID
- Each half-connection has a CC feature: that half-connection’s CCID

  Feature number 1

- Assigned CCIDs:

0	Reserved
1	Unspecified Sender-Based Congestion Control
2	TCP-like Congestion Control
3	TFRC Congestion Control

CCID negotiation

- Negotiated at connection startup

  Renegotiation may not work

- Change(CC), Prefer(CC) options take a prioritized list

  “Change(CC 2, 3)”: I would rather you use CCID 2, but CCID 3 is also acceptable.

- 2 is default

  If not appropriate, don’t send data, negotiate first thing

- 1, 2 suggested for interoperability

Data congestion control

- CCID says when its HC-Sender can send data

  Like a function ccid-allows-data: Packet $\longrightarrow$ bool

- CCIDs will refer to IETF-approved congestion control mechanisms

  Currently all TCP-friendly

CCID 1

- Forward compatibility for sender-based mechanisms
  - Server can implement new CC mechanism without waiting for ubiquitous deployment
  - Not intended for production deployment of proprietary or experimental protocols; production uses MUST have been approved by the IETF
  - Proposing CCID 1 only (with no backup) is outlawed
- Depends on receiver being abe to provide the relevant feedback
  - Probably Ack Vector

Acknowledgements

- Acknowledgement Number is GSR
  - Cumulative ack meaningless in an unreliable protocol
  - Additional ack information required to detect losses
- Different CCIDs require different acknowledgement formats
- Generic ack option: Ack Vector
  - Run-length-encoded vector: exactly which packets have been received
- TFRC ack options: Receive Rate, Loss Event Rate, Loss Intervals
- Acknowledgements must be reliable
  - Retransmit until received

Acks of acks

- Acknowledgement information represents state

  Consider Ack Vector

- Must occasionally free the state

  . . . once the sender has received the information

- Thus, sender must ack the receiver's acks

  Data flowing on both HCs: no problem

  Data flowing on only one HC: . . . ?

Unidirectional connections and quiescence

- Must free ack information even if data flowing on only one HC
  - Complex ack data, such as Ack Vector, probably not required
  - Just send an Acknowledgement Number every now and then
- Must detect quiescence
  - When an HC falls silent
  - CCID-specific, but usually no data sent within max(0.2s, 2 RTT)
- When one CCID is quiescent, the other CCID says how to handle acks-of-acks
  - CCID 2: send at least one Acknowledgement Number per cwnd
  - CCID 3: if Ack Vector, same as CCID 2; if not, do nothing

Acknowledgement congestion control

- Acks take up sequence number space

    So we can detect their loss

    And perform congestion control

- CCID says when its HC-Receiver can send acks

    Another function ccid-allows-ack: Packet $\longrightarrow$ bool

- TCP-friendliness not necessary

    Intended to be "better than TCP's acks"

- Ack Ratio feature

    Send one ack per *R* data packets

    R defaults to 2

    Delayed Acks OK

    Some CCIDs may do ack CC in another way

Ack Vector option

- Run-length encoded history of data packets received

  Steroidal SACK

```
+--------+--------+--------+--------+--------+--------  Codes (SS)
|001001??| Length |SSLLLLLL|SSLLLLLL|SSLLLLLL|  ...     0 received non-marked
+--------+--------+--------+--------+--------+--------  1 received ECN marked
Type=37/38         \___________ Vector ___________...   3 not yet received
```

  Start at Acknowledgement Number, move backwards

  Up to 16192 data packets acknowledged per option

- Includes ECN Nonce Echo (Type 37 = Nonce 0, 38 = Nonce 1)

  Nonce Echo = XOR of all Nonces on Code 0 packets in Vector

  Probabilistic verification that receiver is reporting ECN CE correctly

- Want API to provide Ack Vector information to app

Ack Vector code meaning

- Codes 0 and 1

  MUST have been processed by the receiving DCCP

  MUST have been header-checksum-valid and sequence-valid

  MUST have had their options processed

  Data might not have been processed; it may even have been dropped

- Code 3

  MUST NOT have been processed by the receiving DCCP

  MUST NOT have had their options processed

  Acknowledgement Number MUST NOT correspond to a Code 3 packet

- Summary: “Acknowledgement means header acknowledgement”

Ack Vector consistency

- Two Ack Vectors might acknowledge a packet differently

  Packet arrives between Acks, Acks reordered, only one copy of a duplicated segment gets ECN marked, . . .

- Combine codes according to these tables:

HC-Receiver (Ack generation)

		Received Pkt		
		0	**1**	**3**
Old	**0**	0	0/1	0
Ack	**1**	0/1	1	1
Code	**3**	0	1	3

HC-Sender (Ack processing)

		Received Code		
		0	**1**	**3**
Old	**0**	0	0/1	0
Ack	**1**	1	1	1
Code	**3**	0	1	3

Flow control

- What if the receiver is slower than the available bandwidth?

  Explicit receive window, like TCP's flow control, inappropriate for unreliable traffic

  Besides, the "correct" application response to CPU overload might be to send more traffic! (Less compression = less CPU)

- DCCP has three flow control mechanisms

  Slow Receiver: Don't increase your rate

  Receive buffer drops with Data Dropped: Reduce your rate

  False drop/ECN mark reports: Reduce your rate a lot

Slow Receiver option

- The HC-Receiver is having trouble keeping up

  HC-Sender CC semantics: Do not increase your rate (cwnd, whatever) for about 1 RTT after seeing Slow Receiver

```
+--------+
|00000010|
+--------+
 Type=2
```

Data Dropped option

- Ack Vector says whether packets' *headers* were processed
- Data Dropped option does the equivalent for *payloads*
    - Precise feedback on which packets were dropped *and why*
    - Useful for application
- Report receive buffer drops with Data Dropped
- Use the same mechanism to report other payload drops
    - Protocol constraints (for instance, no payloads accepted on Requests)
    - Application no longer listening (half-closed socket)
    - Corruption drop (Payload Checksum failed)
- Enables richer responses to non-congestion losses

Data Dropped format

```
+--------+--------+--------+--------+--------+--------
|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 Codes
+-+-+-+-+-+-+-+-+      +-+-+-+-+-+-+-+-+   0 dropped, protocol constraints
|0| Run Length  |  or  |1|Dr St|Run Len|   1 dropped, app not listening
+-+-+-+-+-+-+-+-+      +-+-+-+-+-+-+-+-+   2 dropped, receive buffer
  Normal Block            Drop Block       3 dropped, corrupt
                                           7 delivered, corrupt
```

Data Dropped semantics

- CC mechanisms may respond Data Dropped

  Each Data Dropped packet SHOULD be treated as ECN marked unless otherwise specified

- Particular codes

  0 (protocol constraints): Don’t send data until protocol constraint lifted

  1 (app not listening): Send no more data ever

  2 (receive buffer drop): Reduce cwnd by 1 (TFRC TBD)

ECN

- Protocol ECN capable

  All acknowledgement formats support ECN Nonce Echo

  Requirement for verifiable Nonce Echo changed several option designs (for instance, Data Dropped)

- ECN capability not required

  Negotiate ECN Capable feature to 0

  For instance, HC-Sender doesn't want to verify Nonce Echo ⟶ turn off ECN

- Responding to nonce errors

  One kind of aggression: misbehaving receiver wants more than it deserves

  Consistent nonce errors can lead to connection reset (Aggression Penalty)

MTU

- Protocol supports PMTU discovery
  - Need to track PMTUD
- CCID may also set its own MTU
- Connection MTU = min(PMTU, CCID MTU)
- User allowed to turn off PMTU discovery (leave DF off)
  - User cannot avoid CCID MTU

Middlebox considerations

- Service Name particularly useful
- Modifying Sequence and Acknowledgement Numbers painful

  Must modify Ack Vector—can’t just bump a cumulative ack

  CCID-specific options like TFRC’s Loss Intervals

  Identification includes sequence numbers in cryptographic hash

  Must respond to congestion on introduced packets or risk Aggression Penalty

  But it’s a datagram protocol, so many data manipulations easier than in TCP

End DCCP Spec

CCID 2

- TCP-like Congestion Control
  - Good TCP-friendly available B/W utilization
  - Abrupt AIMD rate changes
- Congestion control algorithms based on SACK TCP
  - cwnd, ssthresh, pipe
  - Round-trip time estimation
  - Acknowledgements use Ack Vector
- Ack congestion control
  - Very roughly TCP-friendly manipulations of Ack Ratio

CCID 2 congestion control overview: Variables

- cwnd = congestion window

  Maximum number of data packets allowed in the network

- ssthresh = slow-start threshold

  Controls adjustments to cwnd

- pipe

  Sender's estimate of number of outstanding data packets

  MAY send data packets iff pipe $<$ cwnd

  Increate pipe by 1 on every newly sent data packet

pipe reduction

- HC-Sender reduces pipe as it infers data packets have left the network
- Reduce pipe by 1 for each data packet newly acked as A-V Code 0 or 1
- Reduce pipe by 1 for each data packet inferred as lost due to "dupacks"
    - *P* inferred lost when at least NUMDUPACK packets after *P* have been acked as A-V Code 0 or 1
    - The NUMDUPACK packets need not be data packets specifically
- "Retransmit" timeouts, in case a whole window lost
    - Estimate RTT à la TCP
    - Set RTO à la TCP (but minimum RTO not necessary)
    - When RTO occurs, set pipe to 0

cwnd manipulation

- Congestion events halve cwnd, set ssthresh = new cwnd

  One or more packets lost or marked from a window of data

  Marked = A-V Code 1; lost = inferred through NUMDUPACK

- RTOs set ssthresh = cwnd/2, then set cwnd = 1

- Congestion window increases

  When cwnd $<$ ssthresh, increase cwnd by 1 for every newly acknowledged data packet, up to some max)

  Otherwise, increase by 1 for every window of data acknowledged without lost or marked packets

Sending acknowledgements

- Send about one ack per *R* data packets received
  - *R* is the Ack Ratio
- Reasons to send more acks
  - Delayed ack timer à la TCP
  - Ack piggybacking doesn't count towards *R*
- Acks can be sent with ECN Capable Transport since they are congestion controlled

Congestion control on acknowledgements

- Rough guidelines

  Just try to be somewhat better than TCP

- For each cwnd of data with at least one lost or marked ack, double R (Ack Ratio)

- For each cwnd$/(R^2 - R)$ cwnds of data with no lost or marked acks, decrease R by one

  Derivation in draft

Quiescence and acks of acks

- HC-Receiver detects that HC-Sender is quiescent when max(0.2 sec, 2 RTT) have passed without receiving data
- When other CCID is quiescent, HC-Sender sends about one ack per cwnd

Differences from TCP

- Congestion control in terms of packets, not bytes
  - No consideration of different packet lengths
  - CCID 2 will specify an MTU of 1500
- Congestion window increases in slow start
  - In line with ABC
- Ack Ratio

End

CCID 2

CCID 3

- TFRC Congestion Control

  TCP-friendly, but avoids abrupt rate changes

  Problems utilizing available bandwidth in rapidly changing environments

- TFRC congestion control algorithms

  Equation-based congestion control

  Receiver calculates loss event rate, sender adjusts accordingly

  Acknowledgements need not use Ack Vector

CCID 3 congestion control overview

≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈ ≈

- HC-Sender sends data packets at most at the rate specified by the TCP throughput equation [PFTK98]
  - Rate-based congestion control
  - Data packets include Window Counter (helps receiver distinguish packets sent in different RTTs); sent in CCval
- HC-Sender updates rate based on the loss event rate specified on acknowledgement packets
  - Or it can calculate that rate itself from Ack Vector or Loss Intervals
- Draft refers to TFRC
  - Perhaps too much

Loss events

- A Loss Interval:

  Begins with a lost or marked packet

  Continues for one round-trip time's worth of packets (lost, marked, or not)

  Concludes with an arbitrary-length tail of non-lost, non-marked packets

- The Loss Event Rate:

  The inverse of a weighted average of the last 8 Loss Interval lengths

Acknowledgements

- Elapsed Time and/or Timestamp Echo options

  Aid RTT estimation

  Particularly important since feedback packets sent once per RTT, so Elapsed Time may be large

- Receive Rate option

  How fast has the receiver been receiving data?

- One or more options describing the loss event rate

  Loss Event Rate: lists rate explicitly

  Loss Intervals: the beginning and end of each loss interval

  Ack Vector: sender can calculate loss intervals

Loss Event Rate option

```
+--------+--------+--------+--------+--------+--------+
|11000000|00000110|          Loss Event Rate          |
+--------+--------+--------+--------+--------+--------+
 Type=192    Len=6
```

- Receiver’s calculation of loss event rate
- But not verifiable

Loss Intervals option

```
                    ___ Loss Interval ___
                   /                     \
+--------+--------+----...----+----...----+--------+--------+--------
|11000011| Length | Left Edge |E|  Offset | Up to 7 Loss Intervals ...
+--------+--------+----...----+----...----+--------+--------+--------
 Type=195             3 bytes     3 bytes
```

- Lists the last 8 loss interval lengths

  Left Edge = first sequence number in the loss interval's loss- and mark-free tail

  Offset = length of loss- and mark-free tail

  E = ECN Nonce Echo of loss- and mark-free tail

- Sender can calculate Loss Rate
- Sender can verify ECN Nonce Echo

End CCID 3