DCCP Resolved Issues

Eddie Kohler
UCLA/ICIR
IETF 58 DCCP Meeting
November 10, 2003
Overview

- “New Jersey” feature negotiation
- Extended sequence numbers
- Synchronization/sequence number validity
- Mobility
- Other
California feature negotiation

- Change, Prefer, Confirm
  
  Or Change, Prefer, Change, Prefer, Change, Prefer, ... 

  “Do this” / “Tubular, dude, but I’d rather do this” / “But, like, I’d rather do this” / ... 

- Complex state machine, endless negotiation? 

- Worse: Simultaneous negotiation results in disagreement
  
  Lost packets plus delayed packets could lead to the two ends assigning different values
New Jersey feature negotiation

• Problem: reconciliation
  How can the two ends reconcile their preferences?
  Formerly: Punt; endpoints implement arbitrary algorithms by sending Changes and Prefers
  Now: Define fixed reconciliation algorithms

• Advantages
  All negotiations take one exchange
  Result of negotiation determined only by option contents
  No simultaneous negotiation problems

• Thanks to Junwen Lai
Reconciliation algorithms

- **Non-negotiable**
  
  Feature location accepts any valid value

- **Server priority**
  
  Both endpoints have a preference list
  
  Select the first value in the server's list that also occurs in the client's list
  
  If no item in common, reset connection
Options

- Change R
  
  Sent by feature Remote; “please change your value”

  Payload: new value (NN) or preference list (SP)

- Confirm L

  Sent by feature Location; reports new value

  Payload: new value (NN) or new value and preference list (SP)

- Also Change L, Confirm R
Examples

DCCP A                     DCCP B
1. Change R(CCID, 2 3 1) --->
   ("2 3 1" is DCCP A's value preference list)
2.                          <-- Confirm L(CCID, 3, 3 2 1)
   (3 is the negotiated value;
   "3 2 1" is B's pref list)
   * agreement that (CCID,B) = 3 *

1. XXX <-- Change L(CCID, 3 2 1)
2.                          Retransmission:
   <-- Change L(CCID, 3 2 1)
3. Confirm R(CCID, 3, 2 3 1) --->
   * agreement that (CCID,B) = 3 *

1. Change R(Ack Ratio, 3) --->
2.                          <-- Confirm L(Ack Ratio, 3)
   * agreement that (Ack Ratio,B) = 3 *
Extended sequence numbers

- Problem: LFNs
  Sequence numbers could wrap on fast connections
  10 Gb/s, 1500-byte packets, wrap after 20 s

- 32-bit sequence numbers don't work
  Not enough space in header
  Still not enough bits for packet sizes < 1500 bytes

- Solution: Optional extended sequence numbers
  48 bits
  Different header format
Extended header

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 & 1 \\
\hline
\text{Source Port} & \text{Dest Port} \\
\text{Data Offset} & \text{CCVal} & \text{CsCov} & \text{Checksum} \\
\text{Type} & \#\text{ NDP} \\
\text{Sequence Number (high bits)} & \text{Sequence Number (low bits)} & \text{Reserved} & T \\
\text{Reserved} & \text{Acknowledgement Number (high bits)} & \text{Acknowledgement Number (low bits)} & \text{Reserved} \\
\end{array}
\]
Transition difficulties

- Hard to transition from non-extended to extended

- Extensive set of rules in spec
  - Use Transition bit in extended header

- Recommend avoiding transition: do it at the start or not at all
  - Endpoint can reject transition
  - Currently no feature to govern this (?)
### Synchronization/sequence number validity

<table>
<thead>
<tr>
<th>DCCP A</th>
<th>DCCP B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(GSS=1, GSR=10)</td>
<td>(GSS=10, GSR=1)</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>--&gt; DCCP-Data(seq 2)</td>
<td>XXX</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>--&gt; DCCP-Data(seq 100)</td>
<td>XXX</td>
</tr>
<tr>
<td>--&gt; DCCP-Data(seq 101)</td>
<td>???</td>
</tr>
</tbody>
</table>

- seqno out of range

- Large bursts of loss can get endpoints out of sync
- **Need mechanism to recover synchronization**
  - TCP uses flow control, retransmission
  - Not appropriate here (unreliable protocol)
Old mechanism

- When you get a packet with odd sequence numbers
  Send a DCCP-Ack with a Challenge

- When you get an Ack with a valid Challenge
  Respond with Identification

- When you get an Ack with a valid Identification
  Ignore its sequence numbers, it’s definitely valid
  Update your synchronization

- Challenge and Identification are MD5 hashes involving a shared secret

```
-------> DCCP-Data(seq 100)  XXX
-------> DCCP-Data(seq 101)  ----->  ???
!!!  <------  DCCP-Ack(seq 11, ack 101,  Challenge)
must respond to Challenge  ----->  DCCP-Ack(seq 102, ack 11,  Identification)  OK (GSS=11,GSR=102)
```
Problem

- Random attacker initiates long conversation

```
ATTACKER ----> DCCP-Data(seq 10^6) ----> ???
  seqno out of range; send Ack with Challenge
???
  <------- DCCP-Ack(seq 11, ack 10^6, <----
    ackno out of range; Challenge)
send Ack with Challenge
  <-------- DCCP-Ack(seq 2, ack 11, ----> must
    Challenge)
OK
  DCCP-Ack(seq 12, ack 2, <-------- Identification)
```

Generate 3 MD5 hashes, check 3 MD5 hashes...

Can we do better?
DCCP-Sync

- Problem: Need to accept some packets with bad seq/ack numbers

- Before: A packet is valid if it has good seq+ack numbers or good Identification

- Insight: During resynchronization, sequence numbers are bad, but acknowledgement numbers are good

- New: Add a packet type, DCCP-Sync, that is valid when its ack number is good
  
  Don’t check sequence number
  
  Update sequence number on receiving valid Sync

- Simplifies rules, avoids MD5 hashes, reduces conversation caused by attack
Examples

Recovery from loss

---

DCCP-Data(seq 100) XXX

DCCP-Data(seq 101) --->

seqno out of range; send Sync

OK

DCCP-Sync(seq 11, ack 101) <-

DCCP-Sync(seq 102, ack 11) --->

OK (GSS=11, GSR=102)

• Sync’s ackno equals out-of-range seqno

But endpoint doesn’t update its GSR until it receives new Sync

Response to attack

ATTACKER ---> DCCP-Data(seq 10^6) ---> ???

seqno out of range; send Sync

??? <-

DCCP-Sync(seq 11, ack 10^6) <----

ackno out of range; ignore
More examples

Recovery from a Half-Open Connection

(Crash)
CLOSED
REQUEST ———> DCCP-Request(seq 400) ———> ???
!! <——— DCCP-Sync(seq 11, ack 400) <——— OPEN
REQUEST ———> DCCP-Reset(seq 401, ack 11) ———> Abort!!
REQUEST
REQUEST ———> DCCP-Request(seq 402) ———> ...
REQUEST
Mobility and NATs

- Before: DCCP-Move packet mentioned old address and port

  So stationary endpoint can determine the relevant connection

- Problem: NATs

- Solution: DCCP-Move packet mentions a 64-bit Mobility ID assigned earlier by the stationary endpoint

```
<table>
<thead>
<tr>
<th>Reserved</th>
<th>Acknowledgement Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobility ID (high bits)</td>
<td></td>
</tr>
<tr>
<td>Mobility ID (low bits)</td>
<td></td>
</tr>
</tbody>
</table>
```
Other updates

- Mandatory option
  “Either process the next option in the list, or reset the connection”
  Sounds funny

- Some SHOULD → MUST

- More information in DCCP-Reset and Ignored

- CCID-specific Reset Reasons

- CCID 3: Changes to accommodate extended sequence numbers

- CCIDs: Disallow automatic tracking of updates in TCP/TFRC

- More