Proposal for Fast Subflow Creation

Quentin De Coninck
quentin.deconinck@uclouvain.be

Universite Catholique de Louvain
IETF 99, Prague

From "Every Millisecond Counts: Tuning Multipath TCP for Interactive Applications on Smartphones", tech. report.
http://hdl.handle.net/2078.1/185717
Why do we propose this?

Current implementations (e.g., Linux one) opens subflows on all available interfaces upon connection establishment

→ Great for bandwidth aggregation

But *Make-Before-Break* is sometimes useless

• Connection finished before establishment of additional subflows
• Application not requiring lot of capacity
  • But rather low latency...
  • ...and seamless network handover

Useless subflows waste network/energy resources
The Smartphone Usecase

Typically wants to remain on the WiFi if good

- Only create cellular subflows if bad/no WiFi
- Or if traffic actually requires large bandwidth
The Smartphone Usecase

Typically wants to remain on the WiFi if good

- Only create cellular subflows if bad/no WiFi
  - Or if traffic actually requires large bandwidth
Typically wants to remain on the WiFi if good

- Only create cellular subflows if bad/no WiFi
- Or if traffic actually requires large bandwidth
Proposing a *Break-Before-Make* Approach

We propose three mechanisms for the smartphone usecase

- "Global Scheduling": server following client’s choices
- "Multipath TCP Oracle": detecting bad performing subflows
- "Immediate Reinjections": fast subflow creation

Only discussing the last one here, as it affects interoperability
Limiting Handover Delay

The backup (cellular) path creation is delayed

- Nice from an energy consumption point of view...
- ...but incurs larger app perceived latency in mobility cases
  - Reactive approach: need to detect first bad network

Furthermore, additional Multipath TCP path creation takes time...
Towards Fast Establishment of Additional Subflows

**Figure 1:** Normal JOIN.
Towards Fast Establishment of Additional Subflows

**Figure 1:** Normal JOIN.

**Figure 2:** Fast JOIN with data.
Figure 3: Latency gain between Fast and Normal Joins depending on the request size.

- Saving at least 1 RTT
- Saving 2 RTTs if request size < MSS
Defining New Multipath TCP Options

Two new proposed options

- **FAST_JOIN_OUT**
  - When the client has data to send
- **FAST_JOIN_IN**
  - When the client has no data to send
    - But knows that main path(s) failed
    - And the server might still have data to send (e.g., middle of bulk download)
### FAST\_JOIN\_OUT Option Format – Initial SYN

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Subtype</th>
<th>rsv</th>
<th>E</th>
<th>B</th>
<th>Address ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver’s Token (4 bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data sequence number (4 or 6 bytes, depending on flags)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender’s Truncated HMAC (4 bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data-Level Length (2 bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With $HMAC_{CS}(DSN, token)$
## FAST_JOIN_OUT Option Format – SYN/ACK

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Subtype</th>
<th>rsv</th>
<th>E</th>
<th>B</th>
<th>Address ID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender’s Truncated HMAC (8 bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data ACK (4 or 8 bytes, depending on flags)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With $HMAC_{SC}(DataACK, DSN)$
### FAST_JOIN_IN Option Format – Initial SYN

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Subtype</th>
<th>rsv</th>
<th>E</th>
<th>B</th>
<th>Address ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With $HMAC_{CS}(DataACK, token)$
FAST JOIN_IN Option Format – SYN/ACK

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Subtype</th>
<th>rsv</th>
<th>E</th>
<th>B</th>
<th>Address ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver’s Token (4 bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sender’s Truncated HMAC (8 bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data sequence number (4 or 8 bytes, depending on flags)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With \( \text{HMAC}_{SC}(DSN, DataACK) \)
Possible Security Considerations

- Shortened HMAC size
  - Try to save as much TCP option space as possible
- SYN replay attacks
  - Prevent more than 1 subflow creation with a given DSN/Data ACK
  - Limit number of fast created subflows per connection
To Summarize

Tuning Multipath TCP for smartphone

- Cellular subflow consumes radio resources
- Only use cellular when needed

Applying changes at MPTCP design to suit this usecase

- Implemented in Linux MPTCP v0.91
- And in Nexus 5 (with Android 6.0.1)

→ Interest in writing a draft about this?
Thanks for your attention!
Feel free to ask questions or provide feedback 😊