Scope and Status

- **Comparison of MPTCP and TCP**
  - Tutorial-style description of performance impact and potential problems
  - No significant change compared to -01

- **Operation of MPTCP with legacy applications**
  - Issues with existing sockets API: Address issues, socket options, default enabling, etc.
  - Some clarifications compared to -01

- **Basic API for MPTCP-aware applications**
  - Specification of a minimal MPTCP API
  - Completely new text in -02

- **Other compatibility issues**
  - Incompatibilities with other multihoming solutions, interactions with DNS
  - Extended text in -02

- **Advanced API: Out-of-scope of this draft**
Operation of MPTCP with Legacy Applications
Changes Compared to -01

- Different path management MAY be used if TCP_NODELAY is set
- A new note on stack-internal heuristics potentially used by MPTCP
  - E. g., to classify an application and adapt heuristics implicitly
  - Addresses a comment from Anaheim
  - Summary: “Use the TCP API in a reasonable way” - not that specific to MPTCP
Basic MPTCP API for MPTCP-Aware Applications

Scope

- **Focus of the basic API:** *Minimum set of functions*
  - API provides an **equivalent level of control and information** as exists for TCP
  - Only deals with **enabling** and **address management** of MPTCP
  - Should be simple and rather straightforward

- **Advanced API** could offer more control to applications
  - **Out-of-scope** of this draft, which only specifies the basic API
  - Currently, an **appendix** lists some initial ideas as a potential starting point
  - Suggestion: Describe advanced API in **another draft**, once there is more experience

- **Any comments on this split between basic and advanced API?**
Basic MPTCP API for MPTCP-Aware Applications
Functions getpeername() and getsockname()

- **Legacy apps**
  - MPTCP stack MUST always return the **addresses of the first subflow**

- **MPTCP-aware apps (which, for instance, explicitly enable MPTCP)**
  - Choice 1: **Return address of first subflow, too**
  - Choice 2: **Failure with EMULTIPATH**, since the basic API provides an alternative
  - Choice 3: **Leave behavior to implementation**
  - **No recommendation in current draft**, i. e., behavior is left to implementation

- **Any comments?**
Basic MPTCP API for MPTCP-Aware Applications

Suggested API

- Only new socket options
- **No new functions** (such as bindx), to be as backward compatible as possible
- Four new socket options:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Name</th>
<th>Get</th>
<th>Set</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable/disable</td>
<td>..._ENABLE</td>
<td>x</td>
<td>x</td>
<td>int</td>
</tr>
<tr>
<td>Bind MPTCP to a set of given local addresses</td>
<td>..._BIND</td>
<td>x</td>
<td></td>
<td>list of &quot;struct sockaddr&quot;</td>
</tr>
<tr>
<td>Get the addresses used by the MPTCP subflows</td>
<td>..._SUBFLOWS</td>
<td>x</td>
<td></td>
<td>list of pairs of &quot;struct sockaddr&quot;</td>
</tr>
<tr>
<td>Get the local connection identifier (e.g., local token)</td>
<td>..._CONNID</td>
<td>x</td>
<td></td>
<td>uint32</td>
</tr>
</tbody>
</table>
Basic MPTCP API for MPTCP-Aware Applications

Open Issues

- **TCP_MULTIPATH_BIND**
  - Allows to update the full list of “allowed” local addresses
  - Question: Is such an explicit update during connection lifetime reasonable?
  - Question: What if an interface is not present any more in the list?
  - Current text: MPTCP MAY close the corresponding subflows
  - Is this reasonable? Should it be stronger than a MAY for address removal? Or is this feature unnecessary once a connection has been set up?

- **TCP_MULTIPATH_CONNID**
  - Returns a local connection identifier for the MPTCP connection, which SHOULD be the same as the local connection identifier sent in the MPTCP handshake.
  - Provides a safe way for an application to uniquely identify a MPTCP connection (analogous to 5-tuple in single-path TCP).
  - Is there agreement that this is useful feature?
Next Steps

- **Main change compared to version -01: Focus on a basic API**
  - Document only specifies a **minimum API** for address management
  - An **advanced API is out-of-scope** and may be addressed in a separate draft

- **Application considerations** part of the draft seem to be **rather stable**
  - Basic API will be aligned with the ongoing implementation efforts and experiments
  - Feedback and reviews are still very welcome

- **Ready for WG adoption?**
  - Either with the basic API
  - Or, alternatively, **without the basic API**