Transport Services – Internet Transport's Last Chance?
Michael Welzl
Outline

• Problem

• Solution
Problem
What’s the problem?

- The Internet works!
- Could be faster
- Could be safer
- Could have more features
- Could be more reliable
Internet transport is ossified

- **Service = what TCP and UDP provide**
  - Does not match the diversity of today’s applications

- If the Internet had just followed OSI...
  - Layers would merely provide a service
  - Lower layers + their internal operation hidden → could be replaced

- Transport layer should be especially easy to change!
The 80s

1) Reliable byte stream, TCP
2) Datagram, UDP
Why is this hard to fix?

- New protocols don’t usually work “out of the box”
- Checking for availability on the other side, compatibility with the network path, fall-back to TCP/UDP: all left up to the application programmer
  - Significant effort, for often no benefit
  - Pain vs. gain just isn’t right

- Today’s common “solution”: workaround
  - Make traffic look like as close as possible to normal TCP
  - MPTCP, Minion, ... Skype...
  - “Inner Space” (new Internet-draft by Bob Briscoe that places TCP Options in payload)
Problems with work-arounds

• Overhead, functional limitations

• Evolution in the wrong direction: spiral towards minimum everywhere-deployable element
  – Common middlebox maintainer view (and software default): security based - strange packets are a danger, we drop the unknown
  – Doesn’t fit “be conservative in what you accept” and has caused problems in the past (e.g. RFC 1191→4821)
  – Won’t get better if middleboxes never see other packets and if users don’t demand their support

• May be fine as fall-back method, but should not be default
The 90s

From http://blog.sendmemobile.com/music-humor/ten-1990s-artists-who-need-a-comeback
Internet QoS

- Circular dependency that’s very similar to the Transport Layer problem [RFC 2990]

- Internet (IP over everything) + **strict** QoS guarantees was never a good fit
  - Alternatives exist, but perhaps not as sexy for ISPs – e.g. Alternative Best Effort (ABE) Service (Paul Hurley, Jean-Yves Le Boudec, Patrick Thiran) and some papers by Sergey Gorinsky et al
  - Could do that, or just “try QoS” but not rely on it
  - **Former:** proposed in IETF as draft-lai-tsvwg-normalizer
  - **Latter:** will be done by WebRTC (see draft-ietf-rtcweb-qos / DART WG)
Solution
How would you change TCP to...

• ...decide whether to provide a low-latency-but-less-bandwidth service to a flow?
  • You can’t: you don’t know what the app wants

• ...make a flow benefit from faster delivery of out-of-order packets?
  • You can’t: all flows expect a data stream from TCP, no matter what they truly want
One hammer ;-) for two nails

Introduce abstraction!

1. Applications specify a transport service (what they need) instead of “TCP” or “UDP” (how it is implemented)

2. A system underneath this API automatically makes the best of what is currently available, with a fall-back, typically to TCP (best effort)
But how to achieve this?

• Long history of people saying these things
  – Also writing them down & investigating

• Typical approach: “top-down” (start with what applications want)
  – services more abstract, system more flexible

• My approach: “bottom-up” (begin with existing transport protocols to have a realistic common starting point)
  – services more transport protocol oriented, less flexible
Transport Service examples

• Faster out-of-order delivery (e.g. SCTP)
  – Fallback: slow in-order delivery (TCP)
• Partially unreliable delivery (e.g. SCTP)
  – Fallback: reliable, but throw away if it arrives too late (TCP)
• More capacity via multiple paths (e.g. MPTCP)
  – Fallback: less capacity via one path (TCP)
• Lower latency at the potential cost of throughput (e.g. more FEC in some NC-TCP-variant, or some queuing behavior via a DSCP)
  – Fallback: a lot of latency via TCP

• ...Yes, TCP fits for a lot of things 😊
Bottom-up as a compromise (top-down never had an effect)

Example to the right shows: possible to systematically arrive at a result (table shows services provided by TCP, SCTP, DCCP, UDP-Lite (RFCs, Dec. 2010)

### Table

<table>
<thead>
<tr>
<th>service no.</th>
<th>flow characteristic</th>
<th>app. PDU bundling</th>
<th>error detection</th>
<th>reliability</th>
<th>delivery order</th>
<th>multi-homing</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>o</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>TCP-like</td>
<td></td>
<td>pl</td>
<td>[p2]</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>[p2]</td>
<td>t</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>TCP-like</td>
<td></td>
<td>pl</td>
<td>u</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Smooth</td>
<td></td>
<td>x</td>
<td>o</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Smooth</td>
<td></td>
<td>pl</td>
<td>u</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Smooth-SP</td>
<td>x</td>
<td>u</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>o</td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>u</td>
<td>x</td>
</tr>
<tr>
<td>12</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>u</td>
<td>x</td>
</tr>
<tr>
<td>13</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>o</td>
<td>u</td>
<td>x</td>
</tr>
<tr>
<td>14</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>o</td>
<td>x</td>
</tr>
<tr>
<td>15</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>o</td>
<td>x</td>
</tr>
<tr>
<td>16</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>t</td>
<td>o</td>
<td>x</td>
</tr>
<tr>
<td>17</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>p2</td>
<td>o</td>
<td>x</td>
</tr>
<tr>
<td>18</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>p2</td>
<td>o</td>
<td>x</td>
</tr>
<tr>
<td>19</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>p2</td>
<td>u</td>
<td>x</td>
</tr>
<tr>
<td>20</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>p2</td>
<td>u</td>
<td>x</td>
</tr>
<tr>
<td>21</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>p2</td>
<td>u</td>
<td>x</td>
</tr>
<tr>
<td>22</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>p2</td>
<td>u</td>
<td>x</td>
</tr>
<tr>
<td>23</td>
<td>TCP-like</td>
<td></td>
<td>x</td>
<td>p2</td>
<td>u</td>
<td>x</td>
</tr>
</tbody>
</table>

x = always on
empty = never on
P1 = partial error detection
t = total reliability
p2 = partial reliability
o = ordered
u = unordered

Resulting API in that paper

• Goal: make usage attractive = easy; stick with what programmers know: minimize deviations from socket interface
• Most services chosen upon socket creation
  – int socket(int domain, int service)
  – service number identifies line number in table; understandable aliases: e.g. TCPLIKE_NODELAY, TCPLIKE, NO_CC_UNRELIABLE for lines 1-3
• Sending / receiving: provide sendmsg, recvmsg
• We classified features as:
  – static: only chosen upon socket creation
    • flow characteristic
  – configurable: chosen upon socket creation, adjusted later with setsockopt
    • error detection, reliability, multi-homing
  – dynamic: no need to specify in advance
    • application PDU bundling (Nagle in TCP)
    • delivery order: socket option or flags field
The IETF TAPS WG

without TAPS

<table>
<thead>
<tr>
<th>Application</th>
<th>Protocol X</th>
<th>ISP A (supports X only)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Protocol X</td>
<td>ISP B (supports X and Y)</td>
</tr>
</tbody>
</table>

with TAPS

<table>
<thead>
<tr>
<th>Application</th>
<th>TAPS system</th>
<th>Protocol X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISP A (supports X only)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>TAPS system</th>
<th>Protocol Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISP B (supports X and Y)</td>
<td></td>
</tr>
</tbody>
</table>

LEDBAT, RTMFP, MPTCP, QUIC, Minion

- 1 year of community-convincing; 1 bar BoF through “this doesn’t make sense without looking at what applications need” – “oh, we can’t seem to agree how this interface would look, and wait, it’s not supposed to be an ‘API’ right? Maybe start with protocol capabilities?”
A slide I showed at the second BoF...

We’ve been zig-zagging through charter land:
83 revisions since 27 August 2013
before inserted in IETF datatracker!

- Please look at our history!
  - All available via:
    https://sites.google.com/site/transportprotocolservices/
  - List archives before March 2014:
    https://sympa.uio.no/ifi.uio.no/arc/transport-services
  - Since: https://www.ietf.org/mailman/listinfo/taps
TAPS: planned outcomes

1. List of services provided by today’s transports

2. List: subset of services that systems supporting TAPS will provide + guidance on choosing among available mechanisms and protocols

3. Experimental spec: mechanisms to provide the services identified in item 2

("This document will explain how to select and engage an appropriate protocol and how to discover which protocols are available for the selected service between a given pair of end points. Further, it will provide a basis for incremental deployment.")
Deployment considerations

• TAPS system could be in user space, at least temporarily
  – Timing has improved in user space; think QUIC++

• “Application” could be a library or middleware
  – e.g., pub-sub doesn’t need 100% reliability
  – will only exploit a subset of TAPS capabilities, but re-compiling an app with the new middleware version makes the app use TAPS
Thank you!

Questions?
Backup slides
Example benefits


Transparent usage of SCTP’s multi-streaming underneath TCP

- **SCTP association with multi-streaming**
  - map each connection on different stream
  - message based data transmission
  - shared flow control
  - shared congestion control

- **benefits**
  - subsequent data transfers have new cwnd - value
  - faster startup if association already exists
  - multihoming only active if demanded or beneficial

- **TCP connection**
  - connect to GW
  - bytestream transfer
  - flow control
  - congestion control

- **Gateway**

- **Connection manager gateway**
  - connection attempt management
  - setup SCTP association
  - read/write from TCP connection
  - read/write on SCTP association
  - open/close new TCP connection
  - gateway signaling protocol

original TCP connection (possible to bypass the gateway)
1. Shows what can be achieved by using SCTP underneath the app without even changing the transport API

2. Shows that you don’t have to put it in the OS (user space, middle-box, …)