Why care about transport evolution?
The Internet has 2 main transport protocols:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Throughput</th>
<th>Reliability</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>UDP</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

But application needs differ. What if you want low latency but high reliability? Or variable reliability?

So why hasn’t it happened?
Well, it has! Lots of proposals for alternatives DCCP, RCP, XCP. But none have succeeded...

What makes it so hard?
In a word: middleboxes.

- NATs block UDP, change protocol headers
- IDS and firewalls block unusual traffic
- Load Balancers re-segment TCP, change IP addr
- Traffic Management boxes block or shape traffic

What can we do about it?
Two clever approaches:

- Run on top of UDP/TCP - safe but inflexible
e.g. Minion, application layer transport which trades reliability for latency using TCP as a substrate.

- Masquerade as UDP/TCP - risky but flexible
e.g. µTP which allows reliable BitTorrent to run over UDP while using a less-than-best-effort congestion controller MPTCP which uses multiple subflows to increase throughput, reliability or connection resilience.

4 simple rules for evolution
1. Use a TCP handshake to establish each flow and preserve standard TCP control bits.
2. Fail gracefully in the presence of aggressive middleboxes. Have a safe fallback strategy.
3. Offer real deployment benefits with minimal effort by developers and operators.
4. Ensure the protocol is stable and resilient - in particular be mindful of self congestion.

Polyversal TCP (PVTCP)
- A simple extension to MPTCP.
- Subflows can have different wire formats and congestion control across a single session.
- Simple migration strategy which always falls back to TCP for robustness
- Offers optional new semantics for better connection management
- Includes novel transports like shmem

Open questions include:
- How to optimise connection negotiation
- How to use hardware offload (disabled in MPTCP)
- How to establish end-host capabilities

New application/Socket API
- Tailor behaviour to application
- Library integration OpenSSL, libcurl, etc.
- congestion-aware fast security
- New transports shmem, multipath
- PVTCP negotiation
- MPTCP extensions
- Transport selection
- Security extns.
- Needs

PVTCP example: Datacentre VM migration
- Currently applications use TCP because they don't know where the VMs physically reside.
- If they know they are on same physical die they could use a libvchan shmem channel instead - faster and more stable. But it breaks if the VM is migrated.
- With PVTCP the application can use shmem with a TCP fallback channel. If migration happens the fallback channel maintains connectivity until a new shmem channel is set up.