Better transport
Michael Welzl
Context

• 15 years of almost only transport layer research
  – Because it’s so bad and I’d like to fix it

1. Gradual, deployable (reasonable)
  – Strong focus on IETF
    (TCP, AQM, WebRTC, TAPS)
  – RITE EC FP7 project

2. More, um, “forward looking” (crazy?)
  – RINA architecture
  – PRISTINE EC FP7 project
Part 1:
The Internet's transport API, done the right way

TRANSPORT SERVICES (TAPS)
Problem addressed

- Internet transport layer = TCP, UDP
  - Service = what these protocols provide
  - Does not match the diversity of today’s applications

- OSI had the right idea! :-) abstraction.
  - Layers merely provide a service
  - Lower layers + their internal operation hidden ➔ could be replaced

- Transport layer should be especially easy to change!
Why is this so bad?

- Without abstraction, innovation is very limited
- We’re stuck with the Internet services (protocols) of the 80’s!
The 80s

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

- 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 gain
  - Pain vs. gain just isn’t right

- QoS has a very similar problem
  - RFC 2990 describes chicken-egg situation
The 90s

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

- Internet (IP over everything) + **strict** QoS guarantees was never a good fit
  - Alternatives have always been proposed – e.g. Alternative Best Effort (ABE) Service (Paul Hurley, Jean-Yves Le Boudec, Patrick Thiran)
  - Could do that, or try QoS and fall back
  - **Former:** currently being proposed in IETF as `draft-lai-tsvwg-normalizer` (part of the AEON effort)
  - **Latter:** indirectly being proposed via `draft-ietf-rtcweb-qos` (related to new DART WG)
One hammer for two nails

1. Introduce abstraction: Applications specify a transport service (what they need) instead of “TCP” or “UDP” (how it is implemented)
   – Else:
     • How do you provide a low-latency-but-less-bandwidth service to a flow when you don’t know that it wants it?
     • How do you make a flow benefit from faster delivery of out-of-order packets when all flows expect TCP-like service?

2. A system underneath this API could automatically make the best of what is currently available, with a fall-back, typically to TCP (best effort)
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 😊
Zig-zagging for community support

TAPS IN THE IETF
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)

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
IETF-87 in Berlin, July 2013

• Presentation of Minion in TSVAREA
• ISOC panel discussion
  – Stuart Cheshire, Apple: “when we look at the last 30 years of computing, it's amazing how things have changed. But if you wrote an application 30 years ago, your choice was: TCP and UDP. If you write an application now, your choice is: TCP and UDP.”

• Some relatively recent related IETF work:
  – LEDBAT, RTMFP, MPTCP (RFC6897, appendix A: Requirements on a Future Advanced MPTCP API)

• People started talking about QUIC

• I (with Jon Crowcroft, Toby Moncaster) decided: Let’s do a BOF in London, March 2014!
August-October 2013

• Started mailing list and website
  – Copied Harald Alvestrand’s RMCAT approach
  – Still there, and still the one central “meeting point”: https://sites.google.com/site/transportprotocols/services/

• Started work on charter
  – Plan based on me thinking of my bottom-up work
  – Avoided: goal = API, rather: specify services, describe an example API
  – Major concern: how to decide which services are in/out? Where to draw the line? Turned out to be no problem
IETF-88 in Vancouver, November 2013

• Bar BOF very well attended and lively
  – Major concern raised: bottom-up not application-oriented enough; should specify more abstract services
    Must be relevant to what app programmers really want

• Other signs of growing interest; e.g. TSVAREA session: “We would like to give time to the Transport Area to discuss any potential need to evolve the IETF transport protocols.”
  – QUIC was presented in this session
  – People started mentioning “Transport Services”
December 2013 - February 2014

• Incorporated feedback in the charter (now: describe bottom-up; describe top-down; define mapping)
  – And specify an example implementation
  – Common view: no protocol defined → not IETF

• Several Internet-drafts: problem statement, use cases, … most importantly, survey of common APIs and how they could be supported with transport services
  – Involved Martin Sustrik, creator of ZeroMQ

• Requested BOF; was made “non-WG forming”
  – Decision by Transport Area Directors, to help
IETF-89 in London, March 2014

- Non-WG forming: no wordsmithing ➞ no charter discussion

**Agenda:**
- Introduce problem (Jon Crowcroft)
- Explain how it could support middlewares (Martin)
- Explain how it fits with ongoing IETF work (MIF) (Margaret)
- Describe what an implementation could look like (Gorry)

- 129 participants, more than 6650 words spoken in debate
  - Major concern: provide what application programmers want (again…)

Middleware M

<table>
<thead>
<tr>
<th>Non TAPS-enabled application</th>
<th>M's API</th>
<th>M's internals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy transport (TCP, UDP)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M + TAPS internals

Available transports (TCP, UDP, Minion, SCTP...)
And now?

• Something is happening
  – Interest is clear
  – I was asked to write an article about TAPS for the IETF journal
  – There might be a (WG-forming) BOF at the next IETF, but I probably won’t chair it
  – People in charge consider broader picture: other BOF proposals (e.g. AEON is highly complementary), there might be related things happening in IAB…

• If we get a WG, so what?
  – Indeed… this is an old story, and lots of standards are unused
  – Will need a lot of energy to really achieve something
  – To quote Jon Crowcroft: “success seems to depend on code quality”
Congestion control, done the right way

CONGESTION CONTROL IN RINA
What’s so wrong about Internet congestion control?

• Its end-to-end’ness
  – Justification with e2e-argument is wrong; e2e-argument is about application-specific functionality (also doesn’t forbid complex routing)

• IP over everything, but TCP CC. too…

• No wonder we have research on TCP-over-link-layer-X!
  – But is this really useful research, or a waste of time caused by wrong design?
Congestion control, better

• Link layers have their own overload control; could we just connect them somehow and provide backpressure?
  – Idea influenced by:

• John Day’s book “Patterns in Network Architecture”, describes Recursive InterNetwork Architecture (RINA)
  – Fits this vision well but no congestion control defined
  – This will be investigated in the PRISTINE project
RINA

• Recursive, meaning: all layers (Distributed InterProcess Facilities, DIFs) have the same basic functions
  – Layers provide scope
RINA: example, just one step away from IP…

- Alice can’t immediately reach Bob
- Alice tells NAT1: I’d like to talk to Bob
- NAT1 can find & ask NAT2…
- Addressing in scope 1 and scope 3 can be completely different
RINA: much about addressing / naming

- Doing this right facilitates mobility, multihoming
- More secure: enrollment authenticated, else can’t even reach a node
- Management should be easier
Of course, many usual functions needed...

- But: clean separation between “policy” and “mechanism”
RINA: DIF implementations differ (policies)
Congestion control in PRISTINE
Envisioned operation

- Permanently active control loops (aggregates) between all edges
- Each aggregate has a weight (number of flows)
- Each edge must know mapping between adjacent aggregates and internal aggregates
Aggregate-based congestion control

• Large number of potential benefits
  – Control where the problem is
  – Can use explicit signaling based CC in a DIF
  – Theoretically, infrastructure-specific CC possible
  – Aggregating flows together yields less competition for resource (less queuing => less delay/jitter)
  – Bandwidth given up by application-limited flows can be given to greedy flows
    – Priority support, even QoS should be possible
    – Enable load-based routing!

• But: how well can this scale?
  – E.g., if DIF changes at every hop, this becomes hop-by-hop CC.
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 attempt management
  - setup SCTP association
  - read/write from TCP connection
  - read/write on SCTP association
  - open/close new TCP connection
  - gateway signaling protocol

- **Host A**

- **Host B**

- **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, …)