The Transport Layer is Dead – Long Live the Transport Layer!

Michael Welzl
Net Group, University of Rome Tor Vergata
30.09.2017
Outline

1. The problem

2. The solution
   1. IETF Transport Services (TAPS) WG
   2. NEAT EC project (and software!)
The Transport Layer is Dead ....

(the problem)
The beauty ;-) of OSI...

• A N-layer provides a service to the N+1-layer
  – what's below the service interface is hidden

• Protocols inside layers are interchangeable

• Transport layer (and above) should be especially easy to change!
Internet transport is ossified

- Applications expect **precisely** the behavior of TCP and UDP
- Routers expect to see **precisely** TCP or UDP (sometimes with even more limitations)
Transport layer development

• Computer, 30 years ago

• A program that sends data over the Internet, 30 years ago

(using Berkeley sockets)
Transport layer development

- Computer, today

- A program that sends data over the Internet, today
  (using Berkeley sockets)
More than just sockets...

• Previous example: Berkeley sockets
  – Today, applications use many different APIs

• These higher-layer systems *use* sockets
  ➔ can only offer the services of TCP and UDP
  1. A reliable byte stream
  2. Unreliable packet ("datagram") transmission

It’s a way of *thinking* about communication!
... and it’s from the 80s!

1) Reliable byte stream, TCP
2) Datagram, UDP
What about...

• Priorities?
  – Among your own flows, and between yours and others?

• Careful “do not disturb anybody” background (“scavenger”) communication?

• Intelligent use of multiple network interfaces?

• Maybe using data even when a checksum fails?
  – Codecs can handle that, but UDP will throw away all your data

• Using protocol features that yield lower latency?
  – Maybe trade latency against bandwidth?
  – Control the send buffer?
Example: unordered message delivery

- Some applications can accept data chunks out of order, even when requiring reliability
  - TCP causes Head-Of-Line (HOL) blocking delay

  TCP receiver buffer

  | Chunk 2 | Chunk 3 | Chunk 4 | Chunk 1 |

  App waits in vain!

- Some protocols can deliver data out of order (e.g. SCTP)
- If such a protocol is not available: fine to use TCP!
  - in-order delivery is correct! Just slower

... BUT: unless an application asks for it, we can NEVER give it data in the wrong order.

⇒ This service needs to be in ALL APIs! (not just socket level)
By the way: the 90s were no better!

IntServ? DiffServ?
IntServ over DiffServ!
("U can't touch this" ISP core router)

No, It Isn't Hammertime.

From http://blog.sendmemobile.com/music-humor/ten-1990s-artists-who-need-a-comeback
Internet (end-to-end!) 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
... Long Live The Transport Layer!

(the solution)
Solution part 1: A Standard.

IETF Transport Services (TAPS)
mid-2013, the time had come: LEDBAT, RTMFP, MPTCP, SPDY, Minion
  • Can we get some order into this chaos?

1 year of community-convincing; 1 bar BoF; 2 BoFs => TAPS chartered 24.9.2014
  • History available at: https://sites.google.com/site/transportprotocolservices/
How TAPS wants to solve the problem

• Unordered reliable message delivery is only one example

• What is the minimum set of transport services...
  – from the services that IETF transport protocols offer
  ... that we “must” make available everywhere for them to become usable?

• Once we know that, we can specify how to implement a TAPS system
TAPS charter: 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.”
Bottom-up method to derive a minimum set of transport services

- IETF Transport Services (TAPS) Working Group
  1. Survey of services provided by transport protocols – RFC 8095 (54 pages)
  2. In-depth analysis of 30+ RFCs: TCP, MPTCP, UDP, UDP-Lite, SCTP, LEDBAT (2 Internet-drafts, 79 pages)
  3. Derivation of a minimum set: from 2), keep only services that require application-specific knowledge and do not prohibit falling back to TCP / UDP; resolve some resulting peculiarities (1 Internet-draft, 53 pages)
Fall-backs enable one-sided deployment
Result (latest version): high-level view

1. **Flow creation and flow grouping**
   - Most configuration features only configure a group (e.g., can’t configure timeout for an SCTP stream only)
   - Should use configuration early, ideally before connecting

2. **Send-before-connect**
   - Could also be implemented as a parameter to “connect”
   - Specify “idempotent data” for extra-fast transmission (TFO)

3. **Limited connect / listen / close / abort semantics**
   - Support UDP, or transparently use streams
   - Connect may not invoke accept!
   - No half-closed connections

**Note:** assume TCP fall-back only (assuming UDP fall-back limits this list)
Result (latest version): high-level view  /2

- Be informed about sender buffer running dry
  - Late decision about choice of data to send
- **Configure**: per-flow priorities, network-wide priorities (DSCP), timeout, authentication, checksums
- Some queries, e.g. related to packet sizes
- Send messages, receive bytestream
  - Messages stay intact, but may be reordered or dropped (configured by sender)
  - Receiver must be aware (detect boundaries)
Sending messages, receiving a bytestream

• Can we make this combination work?
  – Be compatible to TCP but still benefit from messages?

• Alternative not very attractive: always telling an application “sorry, you only get a stream here” is not much different than saying “sorry, use TCP instead”
  – Let’s minimize # hoops an app developer has to jump through

• Message-oriented TCP apps already frame their data
  – Unnecessary to repeat this in transport layer
  – Requirement to tell receiver app “here is your complete message” creates a major limitation and is often unnecessary
**Application-Framed (AFra-)Bytestream**

- Normal TCP-like bytestream API
  - Optional: some additional information provided by sender app

- **Sender app**: hands over a stream of bytes, informs transport about frame boundaries and requirements (order, reliability, ..)
  - Delimited **frames** stay intact, in order
  - More relaxed rules possible between frames
  - Delimiters assumed to be known by application

- **Receiver app**: receives stream of bytes
  - App-level delimiters turn it into messages

- **TCP = special case**: no delimiters used
  - Can talk to “normal” TCP applications on both sides
Unordered message delivery: **SCTP**

**Sender app**

- Msg 1
- Msg 2
- Msg 3

**Receiver app**

- Msg 1
- Msg 3
- Msg 2

---

*App-defined header. Could also be e.g. implicit knowledge about size*

- Inform where frame ends
- Inform where frame begins
- Configure: “unordered”

---

**Just a byte stream!**

- App knows how to identify messages
Unordered message delivery: TCP

Sender app

- Inform where frame begins
- Inform where frame ends
- Configure: “unordered”
  ...TCP just ignores this!

Receiver app

App knows how to identify messages

Just a byte stream!

Block 1
Block 2
Block 3

API

Msg 3
Msg 2
Msg 1

API

Msg 1
Msg 2
Msg 3

Block 1
Block 2
Block 3
Unreliable unordered msg delivery: SCTP

- Inform where frame begins
- Inform where frame ends
- Configure: “unreliable, unordered”

App knows how to identify messages

Just a byte stream!
Unreliable unordered msg delivery: TCP

**Sender app**
- Msg 3
- Msg 2
- Msg 1

**Receiver app**
- App knows how to identify messages
- Just a byte stream!

API

**Sender app**
- Block 3
- Block 2
- Block 1

API

**Receiver app**
- Block 1

**App**
- Inform where frame begins
- Inform where frame ends
- Configure: “unreliable, unordered” … TCP just ignores this!
Unreliable message delivery: SCTP, large messages

- Inform where frame ends
- Inform where block begins
- Configure: “Unreliable”
Unreliable message delivery: SCTP, large messages

Sender app

Receiver app

App knows how to identify messages

Just a byte stream!

Packets

SCTP
- Will you need some form of reliability?
  Yes: SCTP or TCP can be used.
  - Is any of the following useful to the application?
    * Choosing a scheduler, choosing priorities
    * Configurable message reliability
    * Unordered message delivery
    * Request not to delay message SACKs
  Yes: SCTP is preferred. No:
  - Is any of the following useful to the application?
    * Hand over a message to reliably transfer (possibly multiple times) before connection establishment
    * Suggest timeout to the peer
    * Notification of Excessive Retransmissions (early warning below abortion threshold)
    * Notification of ICMP error message arrival
  Yes: TCP is preferred.
  No: SCTP and TCP are equally preferable.

No: all protocols can be used.
- Is any of the following useful to the application?
  * Specify checksum coverage used by the sender
  * Specify min. checksum coverage required by receiver
  Yes: UDP-Lite is preferred; No: UDP is preferred.
Moving to a system specification...

• TAPS charter item 3: currently various documents / proposals
  – post-sockets: fall-back to TLS, system on both sides, extra features (e.g. connection migration)
  – guidelines for TAPS systems, security survey
  – NEAT project outputs (happy eyeballs, API, ..)
  – socket intents (mostly focused on interface choice)
Solution part 2: **Code.**

The neat project (and software)
What does NEAT offer?

• A user-space library for efficient e2e network Internet comm., designed to be easy to use
• Offers access to the features of TCP, MPTCP, SCTP, UDP, UDP-LITE and more researchy things via a callback-based API
• Policy system to control everything via json files
• One-sided deployment possible, can do fall-backs to TCP or UDP, can talk to native peers (e.g. TCP, SCTP, even WebRTC browser!)
  – Not precisely the minimum set, but much overlap (e.g.: send message, receive without guarantee of message boundaries for TCP; transparent use of SCTP multi-streaming), and much more
Internals: sequence of events in NEAT

1. Request to open flow and pass application requirements
2. Query PM about feasible transport candidates based on destination domain name
3. PM determines available transport candidates that fulfill policy (PIB) and cached information (CIB)
4. Return ranked list of feasible transport candidates as pre-filter for address resolution
5. Resolve addresses
6. Query PM about feasible transport candidates for resolved destination address
7. PM builds candidates, assigning priorities based on PIB/CIB matches
8. Return ranked list of feasible transport candidates for flow establishment
9. Do Happy Eyeballs with candidates, according to specified priorities
10. Return handle to selected transport solution
11. Cache results from Happy Eyeballs in the CIB
More deployment considerations

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

• Related NEAT development: NEAT socket API
  – Run legacy applications over NEAT using "withneat commandname params"
  – Benefit from: policy settings; transparent SCTP-stream-mapping
Stuff!

- **TAPS:**
  - [https://datatracker.ietf.org/wg/taps/](https://datatracker.ietf.org/wg/taps/)

- **NEAT:**
  - [http://www.neat-project.org](http://www.neat-project.org)
  - [https://github.com/NEAT-project/neat](https://github.com/NEAT-project/neat)
Questions, comments?