Transport Services (TAPS) BOF plan

T. Moncaster, M. Welzl, D. Ros:
draft-moncaster-tsvwg-transport-services-00

https://sites.google.com/site/transportprotocolservices

Michael Welzl, with help from (alphabetical):
Anna Brunström, Toby Moncaster, Gorry Fairhurst,
Reinaldo Penno, Bernd Reuther
+ all the folks who contributed to the draft charter

ICCRG @ 88th IETF Meeting
Vancouver, BC, Canada
5 November 2013

FP7 RITE
Reducing Internet Transport Latency

Context

- The plan is to request having a BOF on "**Transport Services (TAPS)**" at IETF-89 in London

- There is a website:

It's all there!

https://sites.google.com/site/transportprotocolservices/

with a draft charter:
https://sites.google.com/site/transportprotocolservices/home/charter-proposal-before-bof

- And a mailing list:
transport-services@ifi.uio.no
To subscribe:
https://sympa.uio.no/ifi.uio.no/info/transport-services

Problem

- Internet transport layer = TCP (1981), UDP (1980)
  - Does not match the diversity of today's applications

- More and more transport protocols and congestion control mechanisms available, with various features, even overlaps

  But: not "generally" used.
  - SCTP in browser for rtcweb data channel, and in special environments (telephony signalling)
  - QUIC, RTMFP, LEDBAT in app (over UDP)
  - Maybe Minion in OSX?
  - MPTCP now used in OSX in one special way, for one special application
    Consider:
    RFC6897, appendix A: Requirements on a Future Advanced MPTCP API

Transport layer ossification

- "Wasn't the transport layer supposed to be relatively easy to change, as layering ensures that IP routers don't care about the contents of packets?"
  [Mark Handley, "Why the Internet only just works", BT Technology Journal 24(3), 2006]

- Why can't we change it?
  1. Checking for availability on the other side, compatibility with the network path, fall-back to TCP/UDP: all left up to the application programmer
  2. Lack of abstraction: transport protocols are hard coded in the applications

- Today successful deployment of a new transport protocol requires
  - Many applications must explicitly make use of the protocol (because of 2.)
  - Applications must deal with problem 1. on their own

How to solve this

1. Introduce abstraction:
   Applications specify a **transport service** (i.e. what they need) instead of "TCP" or "UDP" (i.e. how it is implemented)

2. A system underneath this API could automatically make the best of what is currently available, with fall-back to TCP (best effort)

What to do here:

- Need to identify these services first
- IETF is in the best position to do so

What real problems does this solve?

Yuchung Cheng: *[transport-services mailing list, transport-services@ifi.uio.no]*
"What real problems do this new transport service solve?"

Michael Welzl: *[transport-services mailing list, transport-services@ifi.uio.no]*
"None, for your application, if (as in case of Google) it pays off for you to put your own new transport protocol in there

(..)

application programmers should be able to benefit from more than what TCP and UDP now give them

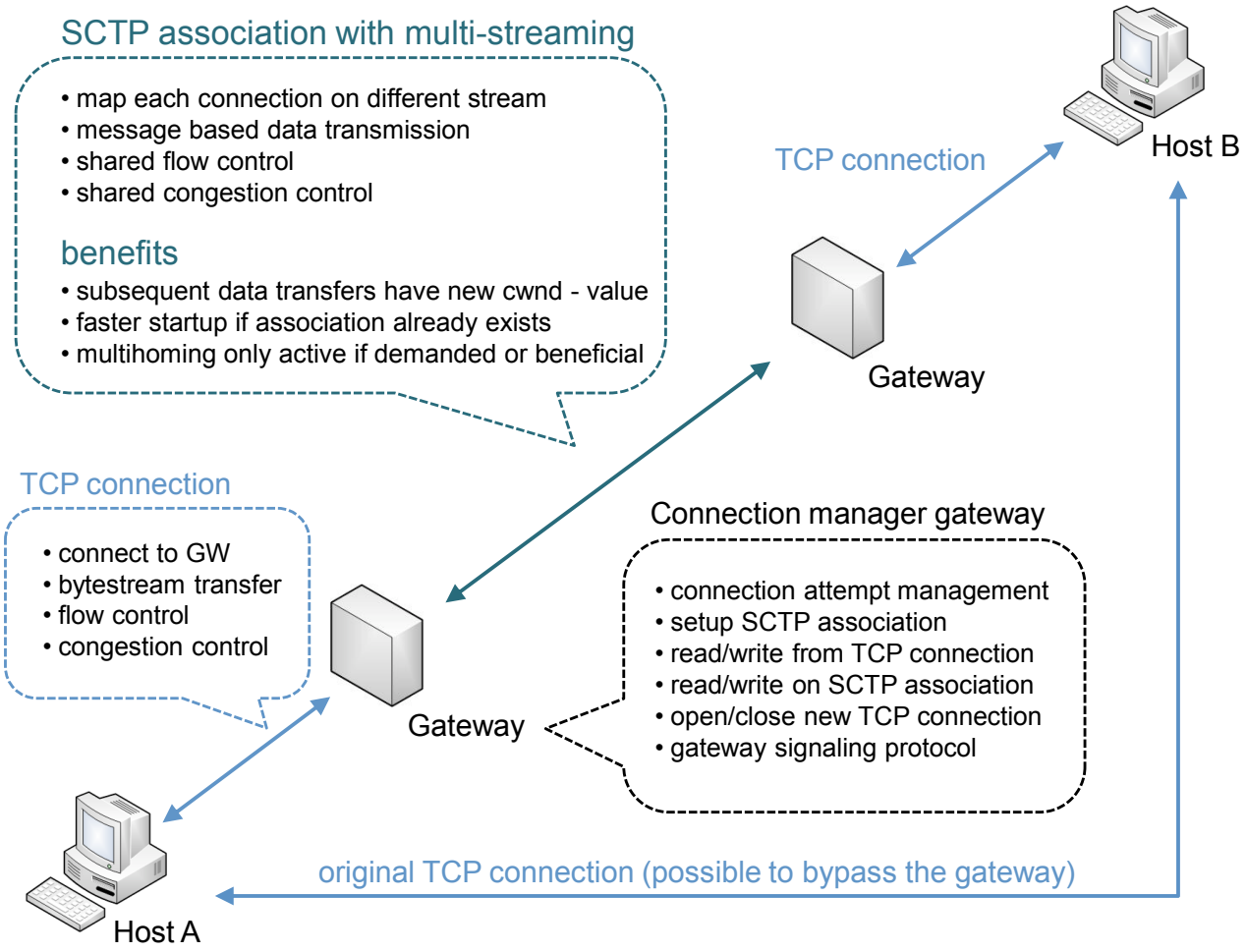
(..)

…take multistreaming for example - this could be done by the transport layer without bothering the application programmer with "do you want this or not?"

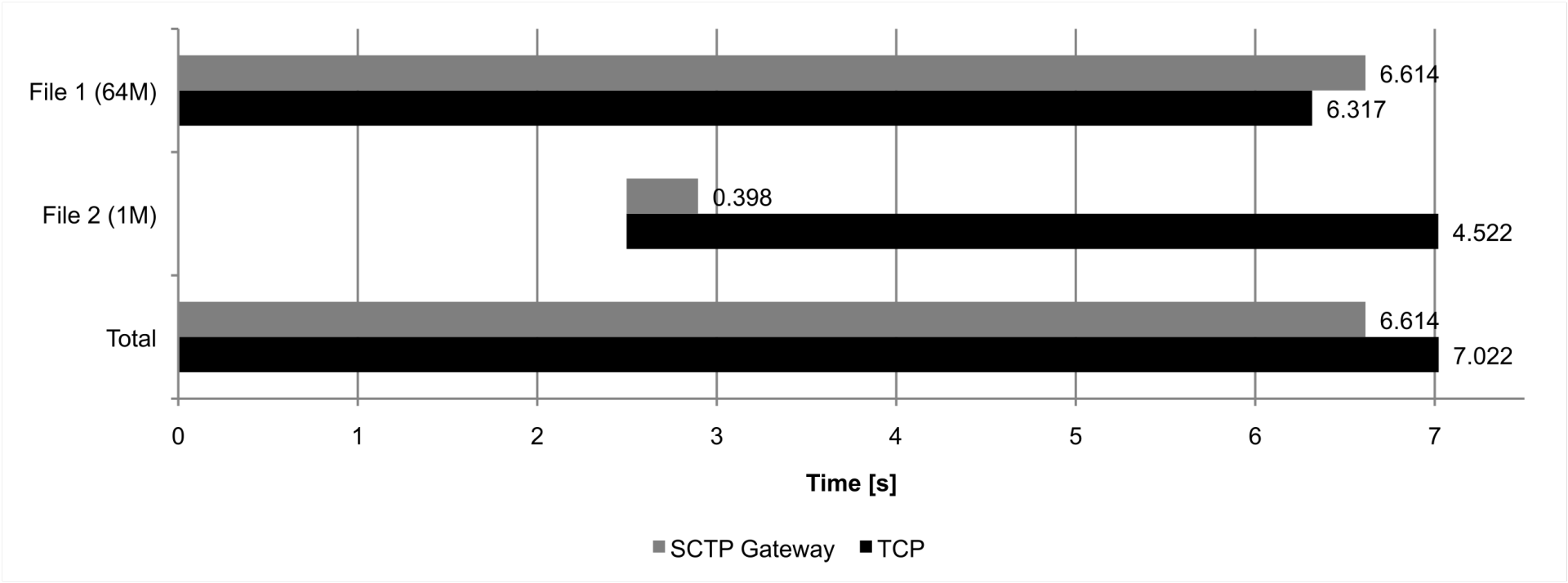
Example benefits

[M. Welzl, F. Niederbacher, S. Gjessing, "Beneficial Transparent Deployment of SCTP: the Missing Pieces", GlobeCom 2011]

Transparent usage of SCTP's multi-streaming underneath TCP



Test result



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, …)

Plans for a TAPS WG-to-be

Updated: includes discussion from yesterday's side meeting, not reflected by draft charter yet

- Specify the services
- Specify one transport system supporting them
  - SCTP (/UDP) with fall-back to TCP, using Happy Eyeballs
- Excluded: QoS and tunneling
- Transport service idea is at least a decade old, without any effect. Why will this succeed?
  - Previous work: "top-down" (what do applications need? ➔ potentially endless debate)
  - Our aproach: "bottom-up" (what do IETF transports provide? ➔ Every service has been discussed before)

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)

service no.		flow characteristic	app. PDU bundling	error detection	reliability	delivery order	multi-homing
1		TCP-like		x	t	o	
2		TCP-like	x	x	t	o	
3				x		u	
4				p1		u	
5		TCP-like		x	[p2]	u	
6		TCP-like		p1		u	
7		Smooth		x		u	
8		Smooth		p1		u	
9		Smooth-SP		x		u	
10		Smooth-SP		p1		u	
11		TCP-like		x	t	o	x
12		TCP-like		x	t	u	
13		TCP-like		x	t	u	x
14		TCP-like		x	p2	o	
15		TCP-like		x	p2	o	x
16		TCP-like		x	p2	u	x
17		TCP-like	x	x	t	o	x
18		TCP-like	x	x	t	u	
19		TCP-like	x	x	t	u	x
20		TCP-like	x	x	p2	o	
21		TCP-like	x	x	p2	o	x
22		TCP-like	x	x	p2	u	
23		TCP-like	x	x	p2	u	x

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

[M. Welzl, S. Jörer, S. Gjessing, "Towards a Protocol-Independent Internet Transport API", FutureNet workshop, ICC 2011]

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*

Ask, discuss, tear to shreds!

Backup slides

About [Michael Welzl, Stefan Jörer, Stein Gjessing: "Towards a Protocol-Independent Internet Transport API", FutureNet IV workshop, ICC 2011]

- Bottom-up: TCP, UDP, SCTP, DCCP, UDP-Lite
  - start with lists from key references
- Step 1: from list of protocol features, carefully identify application-relevant services
  - features that would not be exposed in APIs of the individual protocols are protocol internals (e.g. ECN)
- Result: table with a line for every possible combination of features
  - 43 lines: 32 SCTP, 3 TCP/UDP

About [Michael Welzl, Stefan Jörer, Stein Gjessing: "Towards a Protocol-Independent Internet Transport API", FutureNet IV workshop, ICC 2011] /2

- Step 2: carry out obvious further reductions
  - e.g. flow control coupled with congestion control
  - duplicates, subsets
- Apply common sense to go beyond purely "mechanical" result of step 1
  - Question: would an application have a reason to say "no" to this service under certain circumstances?
  - Features that are just performance improvements if they are used correctly (i.e. depending on environment, not app) are not services

Complete draft charter
(as of 31 October 2013, not including the discussion at the side meeting)

Conjointly, transport protocols such as SCTP, DCCP, MPTCP, UDP-Lite and the LEDBAT congestion control mechanism offer a large number of services to applications in addition to the long-standing two services provided by TCP and UDP. For an application programmer, using protocols other than TCP or UDP is hard: not all protocols are available everywhere, hence a fall-back solution to e.g. TCP or UDP must be implemented. Some protocols provide the same services in different ways. Layering decisions must be made (e.g. should a protocol be used native or over UDP?).

Because of these complications, programmers often resort to either using TCP or implementing their own customized solution over UDP, and chances of benefiting from other transport protocols are lost. If the socket interface provided a way for applications to request transport services without specifying the protocol, a transport system underneath the socket API could automatically try to make the best of its available resources. It could use available transport protocols in a way that is most beneficial for applications, and this approach could give more freedom for diversification to designers of Operating Systems.

Complete draft charter /2

To make implementing such systems possible, the Working Group will:

- develop a survey of existing IETF transport protocols and congestion control mechanisms, based on Standards-track and Experimental RFCs that were developed in the IETF's Transport Area
- shorten the resulting list of transport services, by allowing only those features that an application depends on to work correctly as well as hints about acceptable services that an application can give to the transport layer. A service should be removed if there is no clear way for an application to decide whether it should use this service or not.
- extend the shortened list with services that have seen a significant level of deployment and usage in applications, based on implementations that the WG agrees to perform well. Here, "usage in applications" excludes applications whose primary purpose is monitoring or manipulation of the network.

Security is necessary at a variety of network layers. The Working Group will work with the IETF Security area to better understand how security should be addressed in the specified list of transport services. It will publish a document on security implications and guidance. The Working Group is expected to work closely with the APP and RAI areas to continuously check whether the defined transport services match the requirements of application developers. It will also coordinate closely with other Transport area groups.

Complete draft charter /3

The following topics are out of scope of this Working Group:

- Specifying how a transport system operates; an example will be described
- Signaling that could improve the operation of the transport layer
- Quality-of-Service (QoS) and tunneling mechanisms and services

Deliverables:

- Informational RFC summarizing the services provided by IETF transport protocols and congestion control mechanisms, based on Standards-track and Experimental RFCs that were developed in the IETF's Transport Area (list #1)
- Proposed Standard RFC describing which services of IETF transport protocols and congestion control mechanisms should be offered to applications (list #2, which is a shorter version of list #1)
- Proposed Standard RFC specifying an extended set of services that a transport API should ideally provide (list #3, which is a longer version of list #2)
- Proposed Standard RFC specifying on which basis transport services are identified for inclusion in lists #1, #2 and #3.
- Proposed Standard RFC defining IANA procedures for including new services in lists #2 and #3.
- Informational RFC on Transport Service Security Implications and Guidance
- Informational RFC describing an example API
- Informational RFC describing example usage (i.e. how a transport system providing these services could interoperate with applications)