UiO : Faculty of Mathematics and Natural Sciences
University of Oslo

ifi

Our pre-TAPS work on transport services

Michael Welzl

TAPS, 92nd IETF meeting
23. March 2015

Outline / disclaimer

- Overview of results documented in MSc. thesis + paper
  - [Stefan Jörer: A Protocol-Independent Internet Transport API, MSc. Thesis, University of Innsbruck, December **2010**]
  - [Michael Welzl, Stefan Jörer, Stein Gjessing: "Towards a Protocol-Independent Internet Transport API", FutureNet IV workshop, ICC 2011, June **2011**, Kyoto Japan]

- **Not** a proposal for how things should be: TAPS work should be more extensive, more up to date, make better, more informed decisions
  - But we learned some lessons back then, perhaps useful

Design method

- Bottom-up: TCP, UDP, SCTP, DCCP, UDP-Lite
  - start with lists from key references + RFCs
- 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. TCP, SCTP: ECN, selective ACK

Result of step 1

transport protocol	connection oriented	flow control	congestion control	app. PDU bundling	error detection	reliability	delivery type	delivery order	multi streaming	multi homing
TCP	x	x	x	0/1	x	t	s	o		
UDP					x		m	u		
UDP-Lite					x/p1		m	u		
DCCP	x	x	2/3/4		x/p1		m	u		
SCTP	x	x	x	0/1	x	t/p2	m	o/u	0/1	0/1

- x = always on, empty = never on; 0/1 = can be turned on or off
- 2/3/4 = choice between CCIDs 2, 3, 4
- P1 = partial error detection; t = total reliability, p2 = partial reliability
- s = stream, m = message; o = ordered, u = unordered

Expansion

- A line for every possible combination of features
  - 43 lines: 32 SCTP, 3 TCP/ UDP
- List shows reduction possibilities (step 2)
  - e.g. flow control coupled with congestion control
  - duplicates, subsets

service no.	transport protocol	connection-oriented	flow control	congestion control	app. PDU bundling	error detection	reliability	delivery type	delivery order	multi streaming	multi homing
1	TCP	x	x	x		x	t	s	o		
2	TCP	x	x	x	x	x	t	s	o		
3	UDP					x		m	u		
4	UDP-Lite					x		m	u		
5	UDP-Lite					p1		m	u		
6	DCCP	x	x	CC 2		x		m	u		
7	DCCP	x	x	CC 2		p1		m	u		
8	DCCP	x	x	CC 3		x		m	u		
9	DCCP	x	x	CC 3		p1		m	u		
10	DCCP	x	x	CC 4		x		m	u		
11	DCCP	x	x	CC 4		p1		m	u		
12	SCTP	x	x	x		x	t	m	o		
13	SCTP	x	x	x		x	t	m	o		x
14	SCTP	x	x	x		x	t	m	o	x	
15	SCTP	x	x	x		x	t	m	o	x	x
16	SCTP	x	x	x		x	t	m	u		
17	SCTP	x	x	x		x	t	m	u		x
18	SCTP	x	x	x		x	t	m	u	x	
19	SCTP	x	x	x		x	t	m	u	x	x
20	SCTP	x	x	x		x	p2	m	o		
21	SCTP	x	x	x		x	p2	m	o		x
22	SCTP	x	x	x		x	p2	m	o	x	
23	SCTP	x	x	x		x	p2	m	o	x	x
24	SCTP	x	x	x		x	p2	m	u		
25	SCTP	x	x	x		x	p2	m	u		x
26	SCTP	x	x	x		x	p2	m	u	x	
27	SCTP	x	x	x		x	p2	m	u	x	x
28	SCTP	x	x	x	x	x	t	m	o		
29	SCTP	x	x	x	x	x	t	m	o		x
30	SCTP	x	x	x	x	x	t	m	o	x	
31	SCTP	x	x	x	x	x	t	m	o	x	x
32	SCTP	x	x	x	x	x	t	m	u		
33	SCTP	x	x	x	x	x	t	m	u		x
34	SCTP	x	x	x	x	x	t	m	u	x	
35	SCTP	x	x	x	x	x	t	m	u	x	x
36	SCTP	x	x	x	x	x	p2	m	o		
37	SCTP	x	x	x	x	x	p2	m	o		x
38	SCTP	x	x	x	x	x	p2	m	o	x	
39	SCTP	x	x	x	x	x	p2	m	o	x	x
40	SCTP	x	x	x	x	x	p2	m	u		
41	SCTP	x	x	x	x	x	p2	m	u		x
42	SCTP	x	x	x	x	x	p2	m	u	x	
43	SCTP	x	x	x	x	x	p2	m	u	x	x

Reduction method for step 2

- Remove services that seem unnecessary as a result of step 1 expansion
- 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?*** *(but not purely because of environment conditions)*
  - Features that are just performance improvements if they are used correctly (i.e. depending on environment, not app) are not services

Step 2

- Connection orientation
  - Removing it does not affect service diversity
  - User view: API is always connection oriented
  - on the wire, non-congestion-controlled service will always use UDP or UDP-Lite
  - static distinction, clear by documentation

- Delivery type
  - easy for API to provide streams on top of message transport
  - no need to expose this as a service

Step 2, contd.

- Multi-streaming
  - Performance improvement, depending on environment conditions / congestion control behavior, not an application service
- Congestion control renamed ➔ “flow characteristic”
- Multi-homing kept although not an app. service
  - We felt this is a more complex discussion / decision
  - could still be removed above our API

Result of Step 2

service no.		flow charac-teristic	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

API Design

- Goal: make usage attractive = easy
  - stick with what programmers already know: deviate as little as possible 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. `PI_TCPLIKE_NODELAY`, `PI_TCPLIKE`, `PI_NO_CC_UNRELIABLE` for lines 1-3
- Sending / receiving: provide sendmsg, recvmsg; for services 1,2,11,17: send, recv

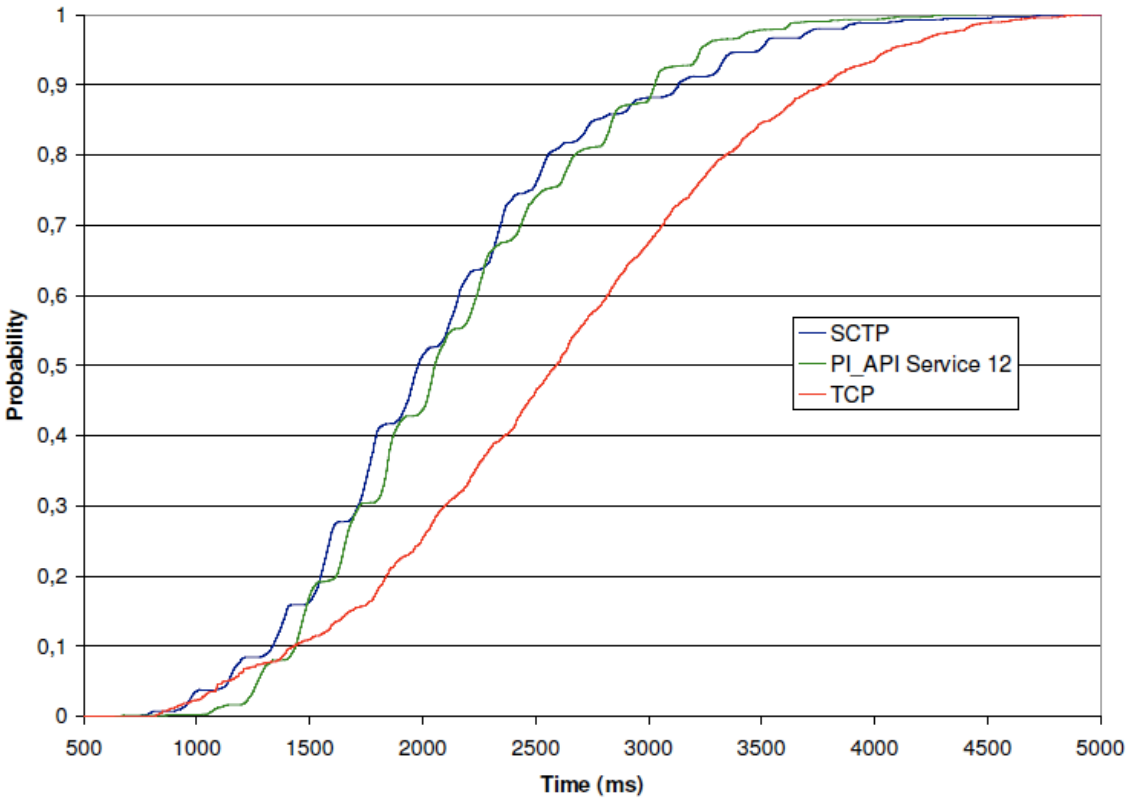
API Design /2

- We classified features as
  1. **static:** only chosen upon socket creation
     - flow characteristic
  2. **configurable:** chosen upon socket creation + adjusted later with setsockopt
     - error detection, reliability, multi-homing
  3. **dynamic:** no need to specify in advance
     - application PDU bundling (Nagle in TCP)
     - delivery order: socket option or flags field

Backup slides

Implementation example

- Unordered reliable message delivery with SCTP
  - removes head-of-line (HOL) blocking delay
- Local testbed, 2 Linux PCs



How is this achieved?

- Based on draft- ietf- tsvwg- sctpsocket-23
- Could not make this work in our testbed (suspect: bug in SCTP socket API)

```
struct sctp_sndrcvinfo *si;
struct cmsghdr *cmsg;
char cbuf[sizeof (*cmsg) + sizeof (*si)];
size_t cmsglen = sizeof (*cmsg) + sizeof (*si);

cmsg = (struct cmsghdr *)cbuf;
cmsg->cmsg_level=IPPROTO_SCTP;
cmsg->cmsg_type= SCTP_SNDRCV;
si = (struct sctp_sndrcvinfo *)(cmsg + 1);
si->sinfo_stream = 1;
si->sinfo_flags = SCTP_UNORDERED;

msg.msg_control = cbuf;
msg.msg_controllen = cmsglen;

sendmsg(sockfd, &msg, 0);
```

How is this achieved? /2

- SCTP, version 2 (this worked)
  - `socket(PF_INET, SOCK_STREAM, IPPROTO_SCTP)`
  - set `SCTP_NODELAY` with `setsockopt`
  - followed by (10 parameters!):
    ```
    sctp_sendmsg(sockfd, textMsg, msgLength,
    NULL, 0, 0, SCTP_UNORDERED, 1, 0, 0);
    ```
- PI_API version
  - `pi_socket(PF_INET, 12);`
  - `pi_sendmsg(sockfd, &msg, 0);`

Thank you!

Questions?