Upgrading Transport Protocols using Untrusted Mobile Code

Parveen Patel
Jay Lepreau
Tim Stack
(University of Utah)
Andrew Whitaker
David Wetherall
(University of Washington)

Key Point

- Untrusted mobile code can allow anybody to build and use new transport protocols cleanly, safely and without delay.
- Self-spreading Transport Protocols (STP) is our prototype solution.

New transport protocols keep coming

- Karn/Tarjan algorithm (1986)
- Header Prediction (1989)
- RFC 1050 (1990)
- RFC 1134 (1990)
- Pack (1990)
- TCP Vegas (1995)
- RSVP (1995)
- TCP-SACK (1996)
- TCP congestion control (1996)
- Sign-exchange (1996)
- Backoff (1998)
- Conversion Manager (1998)
- TCP Connection Migration (1999)
- The RRe-aggregation (2000)
- D-SACK (2000)
- Inter-site Traffic (2000)
- ICN (2001)
- SSDP (2001)
- SID (2002)
- NICE (2002)
- TCP-ME (2002)
- TCP-Wide (2002)
- TCP-Relay (2002)
- TCP sender Hybrid (2002)
- TCP sender bi-directional (2002)

Problem scenario

- A content provider (e.g., Yahoo) develops a new transport protocol to deliver content to its customers
- A mobile client needs “TCP connection migration” at a telnet server to allow itself to move
- How do they deploy new protocols?

Upgrading transports takes years

- Research and simulation
- Prototype
- Standards committee
- Implementation in OS 1
- Implementation in OS 2
- ... Addition into standard build OS 1
- Addition into standard build OS 2
- ...
- Enable by default
- Enable by default on peer

Fallback: backwards-compatible change

- Often does not work
  - Can’t exchange new information
  - Example: TCP Migrate requires cooperation from both ends
- Does not work very well
  - Lose the benefit of cooperation between both ends
  - Example: one-way delay estimation using rtt includes reverse-path noise
Solution: STP

- Host can upgrade its connection peer with new transports by sending untrusted code

STM

Self-spreading Transport Protocols

STP Challenges

1. Network safety – should not hog bandwidth or attack other nodes

2. Host safety – must isolate and limit resource consumption

3. Performance – should not undermine improvement due to extensions

STP Design

Download/Policy mgr

APPLICATION 1

Compiler

Sockets Layer

STP

Network Layer

TP-A

TP-B

STP SANDBOX

Loss Detection in STP

Through the design of its API, STP enforces loss detection that is independent of transport protocol header formats.

sender

receiver

TP-A

TP-A

STP

packet with nonce

packet with nonce

TCP-friendliness is well-defined [SIGCOMM ’98]

Rate = \frac{1}{R^{-(2\cdot L/3)} + (t\cdot RTO/3) \cdot (2^{L/8}\cdot L^{-(1+32\cdot L)})}

R = Round-trip time, L = Loss-rate

TCP sending speed governed by inflow of acks from receiver. Prevent a TCP receiver from taking acks (hiding loss) by requiring it to echo a nonce. [ICNP’01]

TCP-friendliness is well-defined [SIGCOMM ’98]

Rate = \frac{1}{R^{-(2\cdot L/3)} + (t\cdot RTO/3) \cdot (2^{L/8}\cdot L^{-(1+32\cdot L)})}

R = Round-trip time, L = Loss-rate

TCP sending speed governed by inflow of acks from receiver. Prevent a TCP receiver from taking acks (hiding loss) by requiring it to echo a nonce. [ICNP’01]
Loss Detection in STP

2. Host safety
- Constrained domain: no shared state between transports
  - Makes resource accounting straightforward
  - Makes termination tractable
- Memory safety: type-safety of Cyclone [PLDI ’02]
- CPU timer-based CPU resource protection

3. Performance
- Connections proceed without delays
  - Code is downloaded out of the critical path
  - Benefits later connections
  - Exploits communication pattern of today’s Internet
- Efficient to interface C with Cyclone
  - Share data between the kernel and Cyclone code
  - Not necessary to use garbage collection

Implementation
- Prototype in FreeBSD 4.7
- Ported UDP-Flood, TCP NewReno and TCP SACK to the STP API

Evaluation
- Network Safety
- Overall Performance
- CPU Overhead
- Transport Experience

STP enforces TCP-friendliness
STP does not restrict TCP

STP is as fast as TCP for Internet-like paths

STP transports achieve gigabit speed

CPU utilization (gigabit link)

Transport experience

Future work

<table>
<thead>
<tr>
<th>Code</th>
<th>TCP</th>
<th>SACK</th>
<th>UDP Flooding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source (Gzip)</td>
<td>87K</td>
<td>95K</td>
<td>10K</td>
</tr>
<tr>
<td>Object</td>
<td>31K</td>
<td>33K</td>
<td>4K</td>
</tr>
</tbody>
</table>

- API supports all 27 studied extensions except 2 that are inherently not TCP-friendly.
- Shipping whole protocols is practical:

- So far:
  - STP is proof-of-concept of a system that synthesizes a set of ideas.
- Next up: Make the vision more real:
  - Stress-test system with adversarial transports
  - Prove that API is sufficient and OS-portable
  - Learn what policies work well in practice

- Overhead inherent in Cyclone’s type-safety (bounds/null checks) is low: 6%.
- Suspect most of overhead due to marshaling that will be straightforward to optimize in newer version of compiler.
Conclusions

- STP lets anybody build and use new transport protocols cleanly, safely and without delay.
  - Built on untrusted mobile code
  - Avoids hacks, standards and OS vendors

- This is a qualitative change!
  - Imagine real experience before standards
  - Fundamental change in incentive balance

END OF TALK

....

BACKUP/DETAIL SLIDES