Breaking Up the Transport Logjam

Bryan Ford
Max Planck Institute for Software Systems
baford@mpi-sws.org

Janardhan Iyengar
Franklin & Marshall College
jiyengar@fandm.edu

HotNets-VII, October 6-7, 2008
Evolutionary Pressures on Transports

- **Applications** need more flexible abstractions
  - better datagrams [DCCP], streams [SCTP, Ford07]
- **Networks** need new congestion control schemes
  - high-speed [Floyd03], wireless links [Lochert07], ...
- **Users** need better use of available bandwidth
  - dispersion [Gustafsson97], multihoming [SCTP], logistics [Swany05], concurrent multipath [Iyengar06]…
- **Operators** need administrative control
  - Performance Enhancing Proxies [RFC3135], NATs and Firewalls [RFC3022], traffic shapers
- We have partial solutions, **but no deployment**
Many solutions, None deployable

- New transports **undeployable**
  - NATs & firewalls
  - which comes first: App-demand or OS kernel support?

- New congestion control schemes **undeployable**
  - impassable “TCP-friendliness” barrier
  - must work end-to-end, on all network types in path

- Multipath/multiflow enhancements **undeployable**
  - “You want how many flows? Not on my network!”
  - TCP-unfriendly?
The Transport Layer is Stuck in an Evolutionary Logjam!
The Problem

Traditional transports conflate 3 function areas...

- Semantics, Reliability Concerns (applications care)
- Performance Concerns (users, opers care)
- Naming, Routing Concerns (NATs, firewalls care)

To break transport logjam, must separate concerns
Our Proposal

Break up the Transport according to these functions:
Endpoint Layer
Current transports have separate port spaces.
But What Are Ports?

• Ports are really **routing info**!
  – IP address ⇒ Inter-Host Routing
  – port numbers ⇒ *Intra*-Host Routing

• … *should* have been part of **Network Layer**?

• NATs/Firewalls treat ports as **routing info**!
  – Care about *application* endpoints, not just hosts
  – Therefore, must understand transport headers

• **Result:** Only TCP, UDP can get through
Proposed Solution

Factor endpoint info into uniform **Endpoint Layer**
Surprise!

Workable starting point exists — UDP!
Practical Benefits

Can now evolve separately:

- **Transport functions:**
  - New transports get through NATs, firewalls
  - Easily deploy new user-space transports, interoperable with kernel transports
  - Application controls negotiation among transports

- **Endpoint functions:**
  - Better cooperation with NATs [UPnP, NAT-PMP, ...]
  - identity/locator split, port/service names [Touch06], security and authentication info ...?
Flow Layer
Traditional “Flow Regulation”

Transport includes end-to-end **congestion control**
- to regulate flow transmission rate

But one E2E path may cross *many*...
- … different **network technologies**
  - Wired LAN, WAN, WiFi, Cellular, AdHoc, Satellite, …
  - Each needs different, specialized CC algorithms!
- … different **administrative domains**
  - Each cares about CC algorithm in use!
Proposed Solution

Factor flow regulation into underlying **Flow Layer**

- Transport **Semantics**, Reliability
- Flow **Performance** Regulation
- Endpoint Naming
Practical Benefits (1/3)

Can split E2E flow into separate CC segments

- Specialize CC algorithm to network technology
- Specialize CC algorithm within admin domain

... without interfering with E2E transport semantics!
Practical Benefits (2/3)

Incrementally deploy performance enhancements
- multihoming, multipath, dispersion, aggregation...
... without affecting E2E transport semantics!
Practical Benefits (3/3)

- Can aggregate flows cleanly within domains for
  - Efficient traffic measurement, management
  - Fairness at “macro-flow” granularity
Developing the Flow Layer

• Two likely “starting points” already exist:
  – Congestion Manager [Balakrishnan99]
  – DCCP [Kohler06] (just stop thinking of it as a “transport”)

• Major work areas:
  – Support for flow middleboxes, path segmenting
  – Interfaces between (new) higher & lower layers
Transport Layer
Transport Layer

Contains “what's left”:

- Semantic abstractions that apps care about
  - Datagrams, streams, multi-streams, …
- Reliability mechanisms
  - “Hard” acknowledgment, retransmission
- App-driven buffer/performance control
  - Receiver-directed flow control
  - Stream prioritization
  - …
Breaking Up the Transport Logjam

- New transports **deployable**
  - Can traverse NATs & firewalls
  - Can deploy in kernels or applications

- New congestion control schemes **deployable**
  - Can specialize to different network types
  - Can deploy/manage within administrative domains

- Multipath/multiflow enhancements **deployable**
  - Can deploy/manage within administrative domains
Only the Beginning...

Promising architecture (we think), but lots of details to work out

- Functionality within each layer
- Interfaces between each layer
- Application-visible API changes

Big, open-ended design space

- We are starting to explore, but would love to collaborate with others!
- If you know of spaces where you could use this framework, we'd love to know!
Conclusion

- Transport evolution is **stuck**

- To unstick, need to separate:
  - Endpoint naming/routing into separate **Endpoint Layer**
  - Flow regulation into separate **Flow Layer**

- Leave semantic abstractions in **Transport Layer**
Complexity

• More layers
  => **increase**

• Puts necessary hacks into framework
  => **decrease**

• What's the balance?
What about the e2e principle?

- Flow layer implements in-network mechanisms that focus on communication performance
  - Precisely the role for which the e2e principle justifies in-network mechanisms
- All state in the flow middleboxes is performance-related soft state
- Transport layer retains state related to reliability
  - End-to-end fate-sharing is thus preserved
- Transport layer is still the first end-to-end layer
Kernel/User Transport Interoperation

Host A

User

Application

User-space Transport

Kernel

UDP

Network Protocol

Host B

User

Application

Kernel-space Transport

Kernel

Network Protocol
Kernel/User Transport Interoperation

Host A
- Application
  - User-space Transport
    - Endpoint Protocol
      - Network Protocol

Host B
- Application
  - Kernel-space Transport
    - Endpoint Protocol
      - Network Protocol
“Zero-RTT” Transport Negotiation

Host A

Transport Negotiation “Meta-SYN”

Transport 1 SYN | Transport 2 SYN | Transport 3 SYN

Transport 2 SYN/ACK

Host B

B chooses Transport 2