How Should We Think About Transport Abstractions?

Bryan Ford
Yale University

w/ Janardhan Iyengar, Michael Nowlan,
Nabin Tiwari, Syed Obaid Amin

DIMACS Workshop on
Algorithmic Foundations for the Internet
May 22, 2012

http://dedis.cs.yale.edu/

Tng Project: Relevant Papers

- Structured Stream Transport (SIGCOMM '07)
  - http://bford.info/pub/net/sst-abs.html
- Breaking Up the Transport Logjam (HotNets '08)
  - http://bford.info/pub/net/logjam-abs.html
- Efficient Cross-Layer Negotiation (HotNets '09)
  - http://www.bford.info/pub/net/nego-abs
- Square Pegs in Round Pipes (NSDI '12)
  - http://dedis.cs.yale.edu/2009/tng/papers/nsdi12-abs

Evolutionary Pressures

- **Applications** need more flexible abstractions
  - semantic variations [RDP, DCCP, SCTP, SST, ...]
- **Networks** need better congestion control
  - high-speed [Floyd03], wireless links [Lochert07], ...
- **Users** need better use of available bandwidth
  - dispersion [Gustafsson97], multihoming [SCTP], logistics [Swany05], multipath [Iyengar06]…
- **Operators** need administrative control
  - Performance Enhancing Proxies [RFC3135], NATs and Firewalls [RFC3022], traffic shapers

The Transport Layer is (Still) Stuck in an Evolutionary Logjam!

[HotNets '08 – w/ Janardhan Iyengar]

Many Solutions, None Deployable

- New transports **undeployable**
  - NATs & firewalls
  - chicken & egg: app demand vs kernel support
- New congestion control schemes **undeployable**
  - impassable "TCP-friendliness" barrier
  - must work E2E, on *all* network types in path
- Multipath/multiflow enhancements **undeployable**
  - "You want *how many* flows? Not on *my* network!"
  - Fundamentally "TCP-unfriendly"?

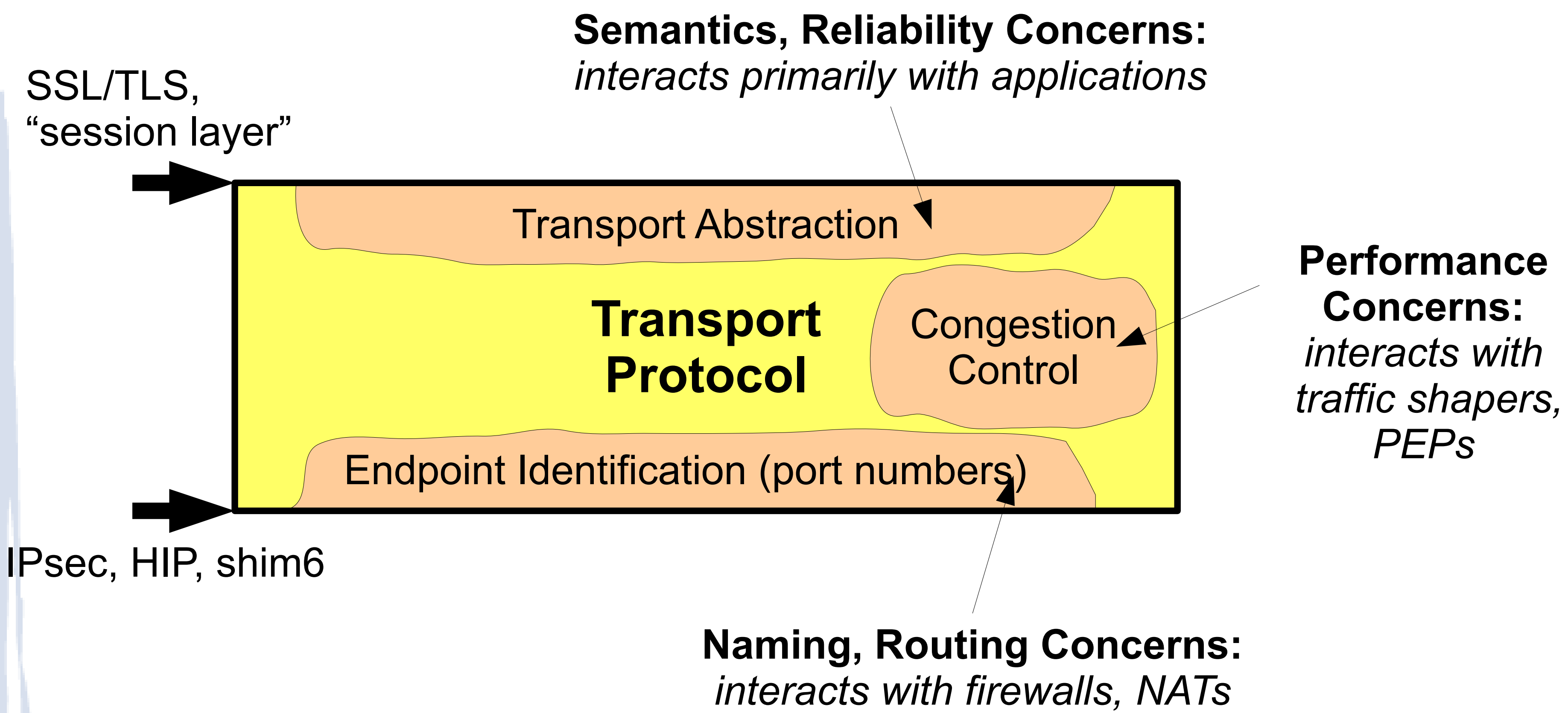
Transport Abstractions

What "abstractions" do transports provide?

- Units of Data Movement (packets, streams)
- Units of Reliable Transmission (e2e principle)
- Units of Rate Control (flow, congestion)
- Units of Resource Sharing (inter-flow fairness)
- Units of Logical Endpoint Naming (ports)
- Units of Pluggability (narrow waist principle)

Analysis of Transport Functions

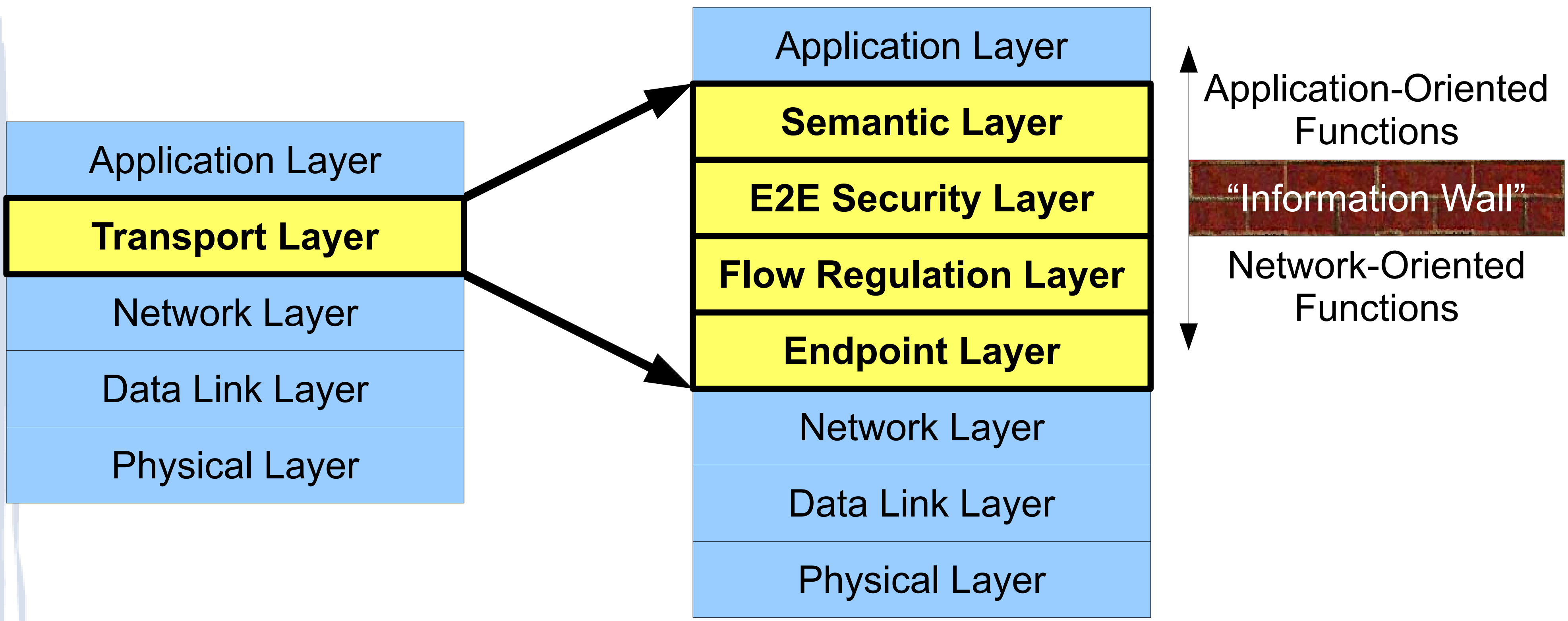
Current transports conflate **application-oriented** and **network-oriented** functions...



where do security and location-independence go?

"Transport Next Generation" (Tng)

Break up the Transport into further sub-layers according to these classes of functions:

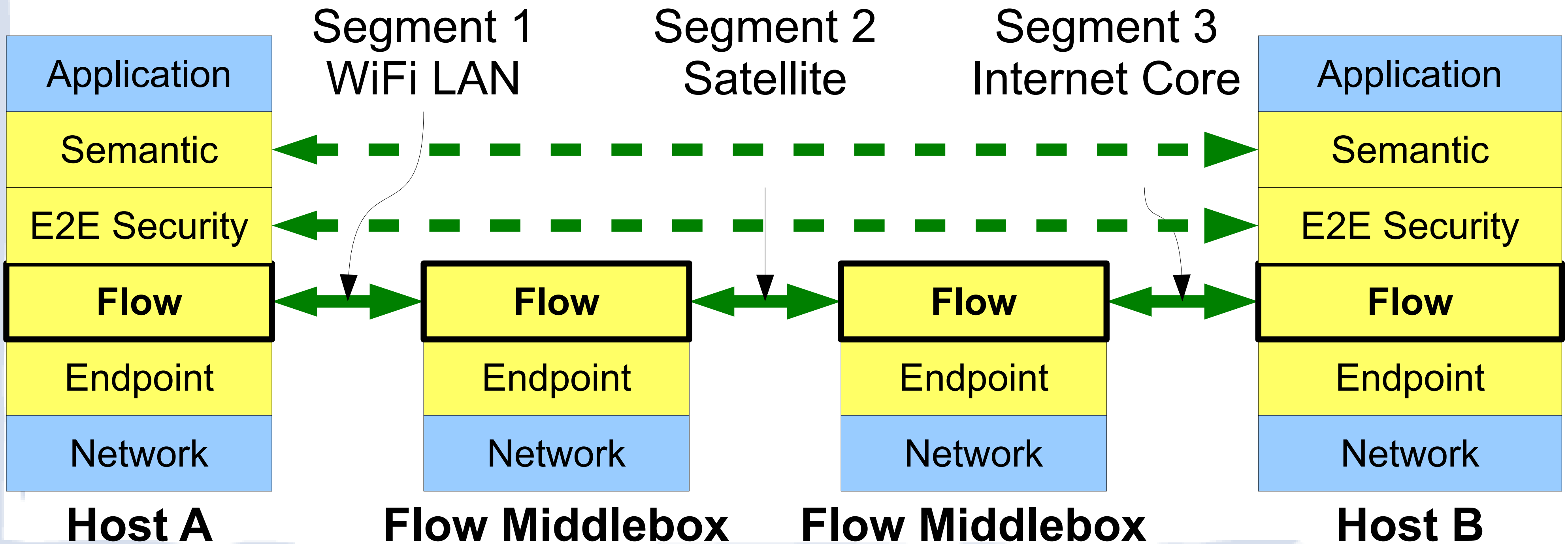


"Cool Stuff You Can Do" in Tng

Can split E2E flow into separate CC *segments*

- Specialize CC scheme to **network technology**
- Specialize CC scheme within **admin domain**

without interfering with E2E transport *semantics*



Random Annoying Questions About Transport Abstractions

- Do abstractions matter *fundamentally*, or only based on performance properties of their *currently available* implementations?
- Should we choose or design abstractions for the *network* or for the *application?*
- What is the right *granularity* for abstractions, or how do we handle *granularity mismatches?*

Data Movement Abstractions

Some data movement abstractions we've seen:

- Small Blobs (packets) [UDP, DCCP, SCTP]
- Byte-Stream [TCP]
- Packet-Stream [RDP, SCTP]
- Multi-Stream [SCTP, SST]
- Large Blobs [CDNs, DTN, DOT]
- ???

How Different Are They?

Application choices between TCP and UDP are mainly about the *performance characteristics* of their *available implementations*

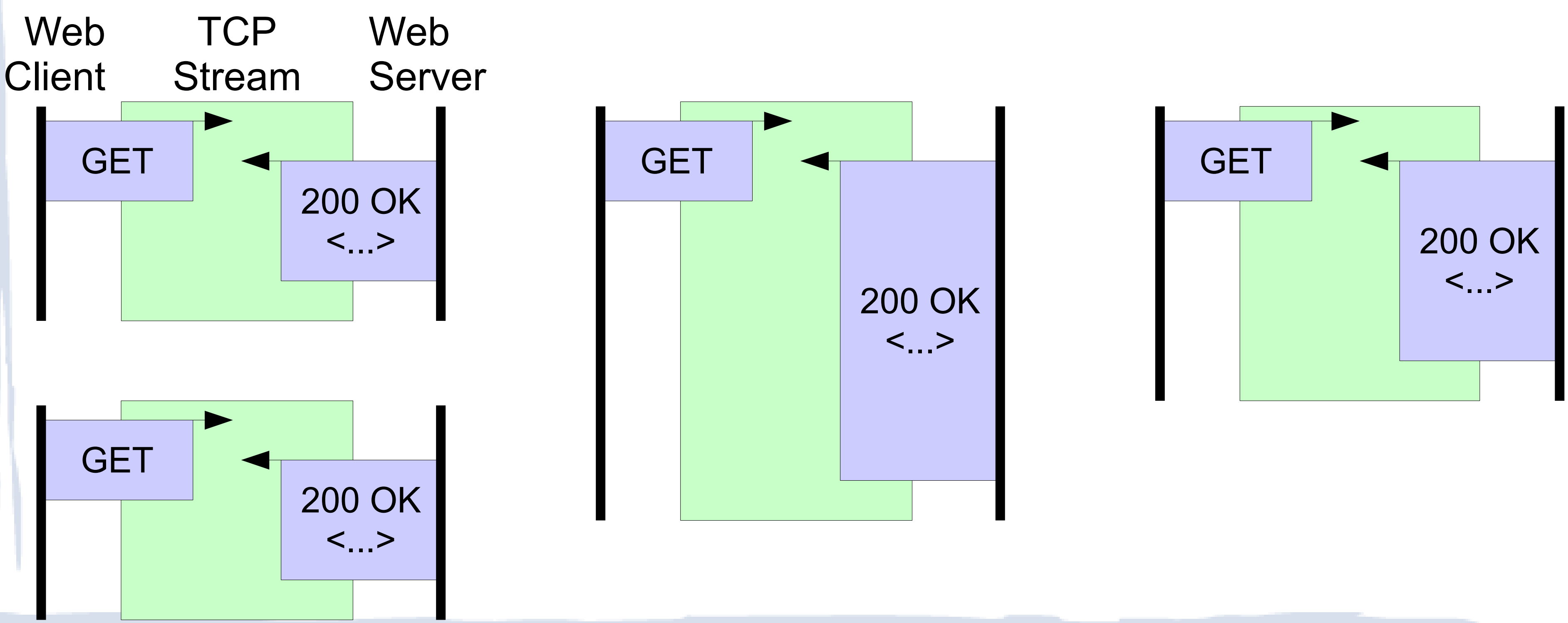
- UDP datagrams: low-overhead and atomic, but only work at all when “small” (~8K max)
- TCP streams: arbitrary-size and incremental, but higher setup/shutdown/state overheads

In Structured Stream Transport [SIGCOMM '07], *one abstraction* serves both roles efficiently...

Example Use of TCP Abstraction

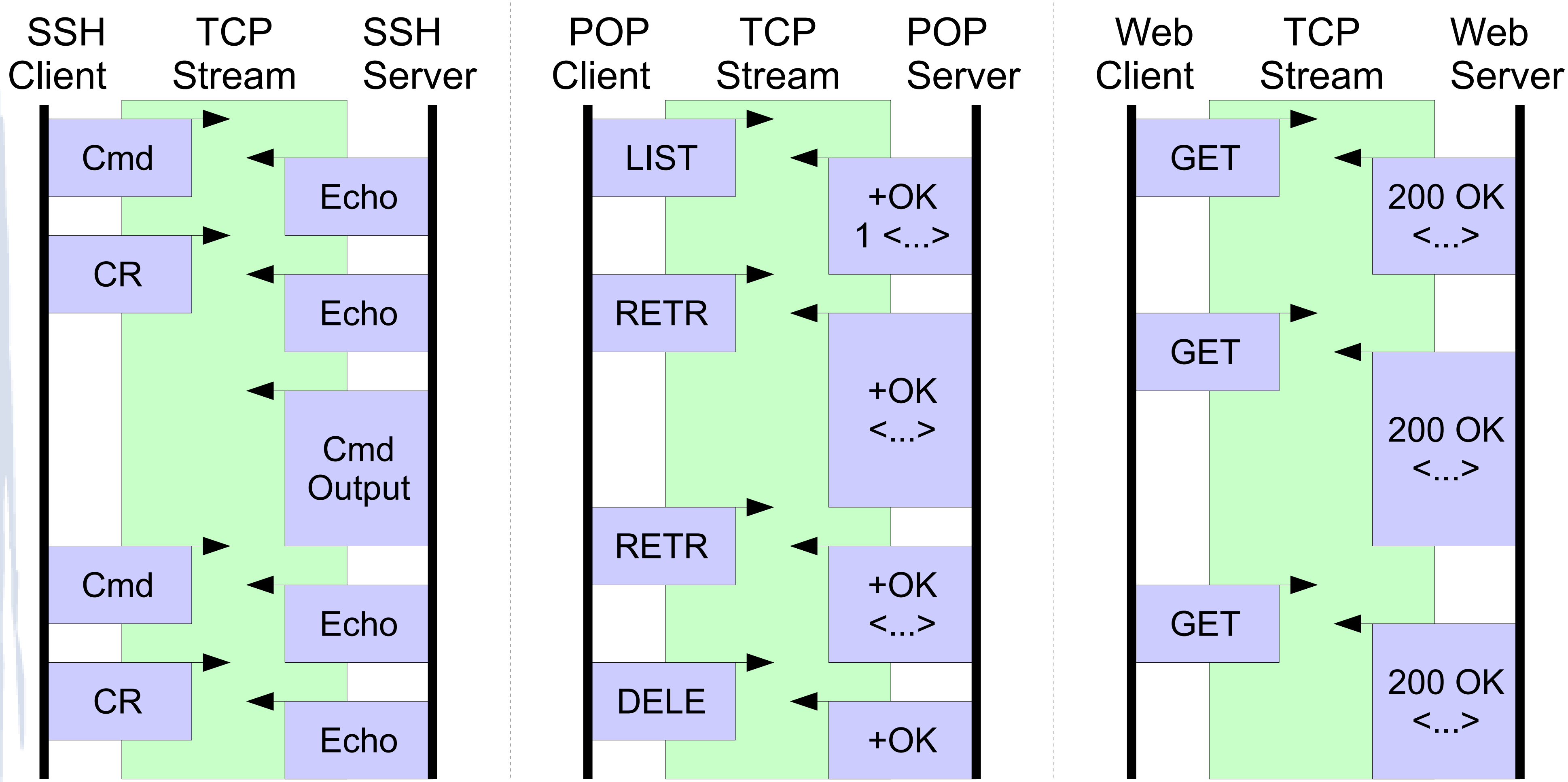
Natural approach: streams as **transactions** or **application data units (ADUs)** [Clark/Tennenhouse]

Example: HTTP/1.0

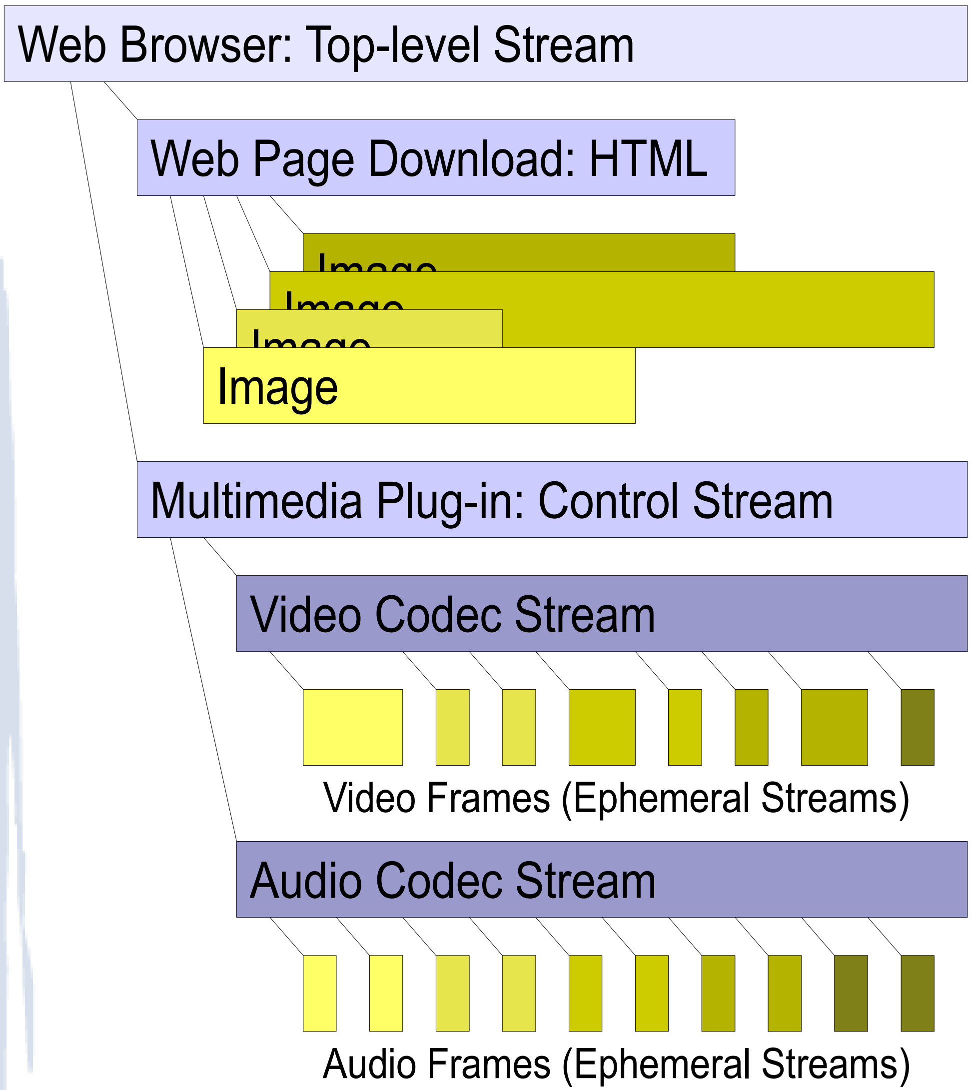


Example Use of TCP Abstraction

Practical approach: streams as ***sessions***



But If Streams Were Cheap...



The Structured Stream "abstraction":

- Like TCP, but *cheap*
- Stream per object
- Stream per datagram
- Stream per AV frame

Do we really need *new abstractions* or just *better implementation*?

Network vs Application Abstractions

What's important in a transport “abstraction”: what the *application* or the *network* sees?

- Apps can get abstractions from middleware built in user space atop TCP, UDP, whatever
- Network abstractions matter for interoperability and for long-term compatibility

So should abstractions be driven by applications or by the network?

The Minion Suite [NSDI '12]

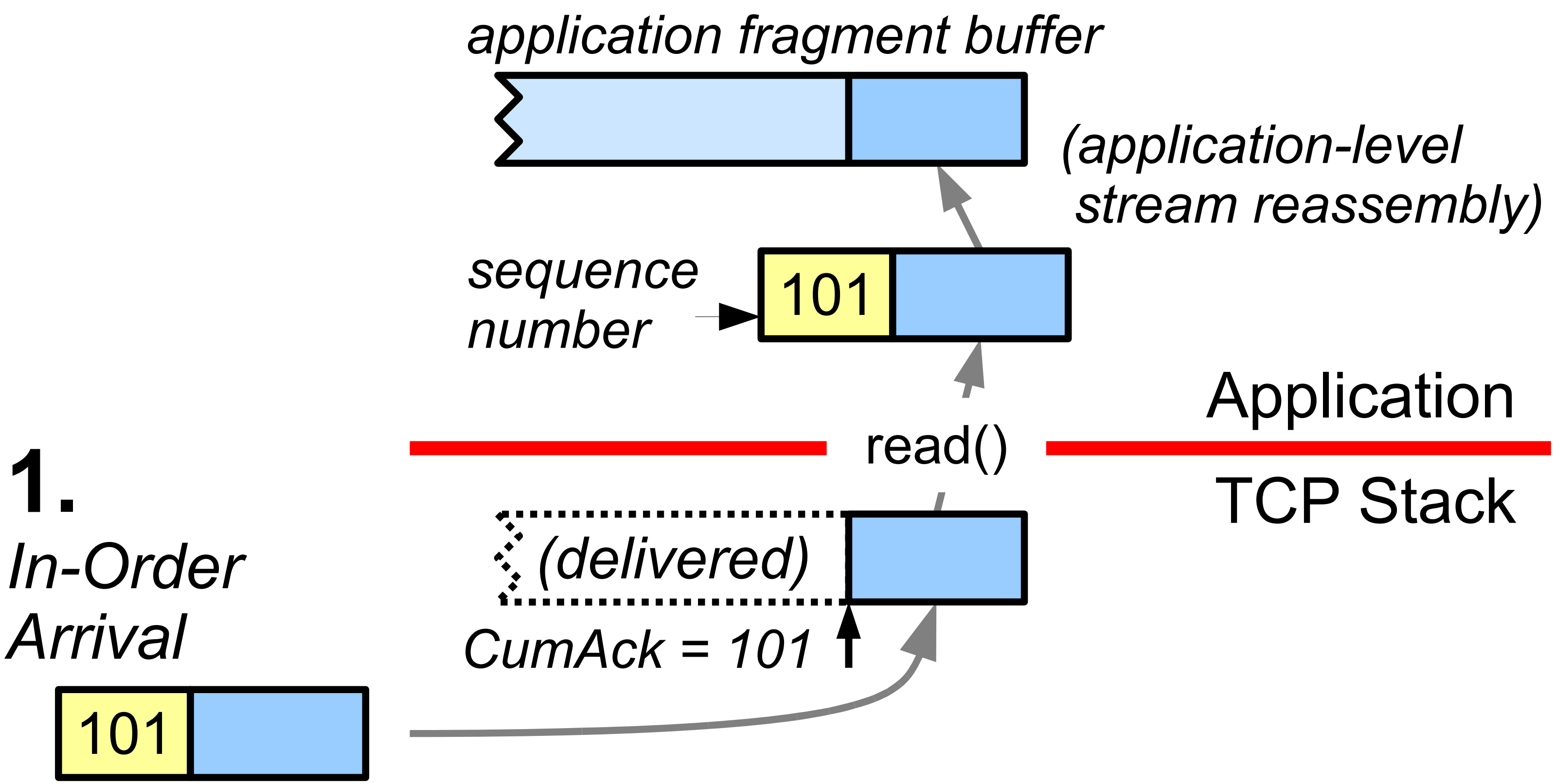
Recognizing that:

- Apps no longer need TCP for *convenience*, but as an efficient, compatible *substrate*
- But in-order delivery adds *unrecoverable delay*

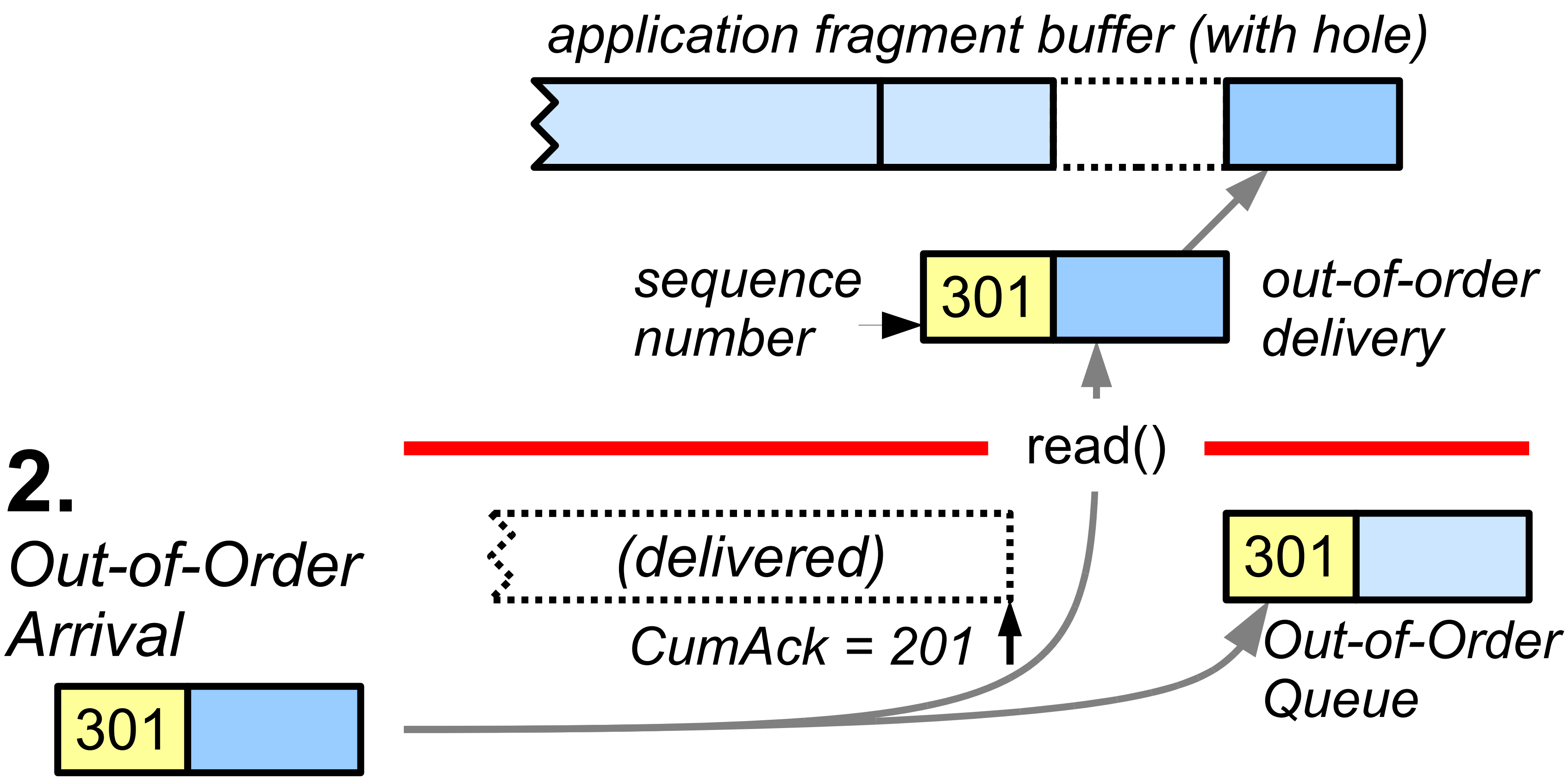
Minion offers:

- Out-of-order delivery in TCP and SSL/TLS
- *No change* in network-visible TCP behavior
  - Walks, squawks like a TCP stream!
- But *application* can receive data out-of-order

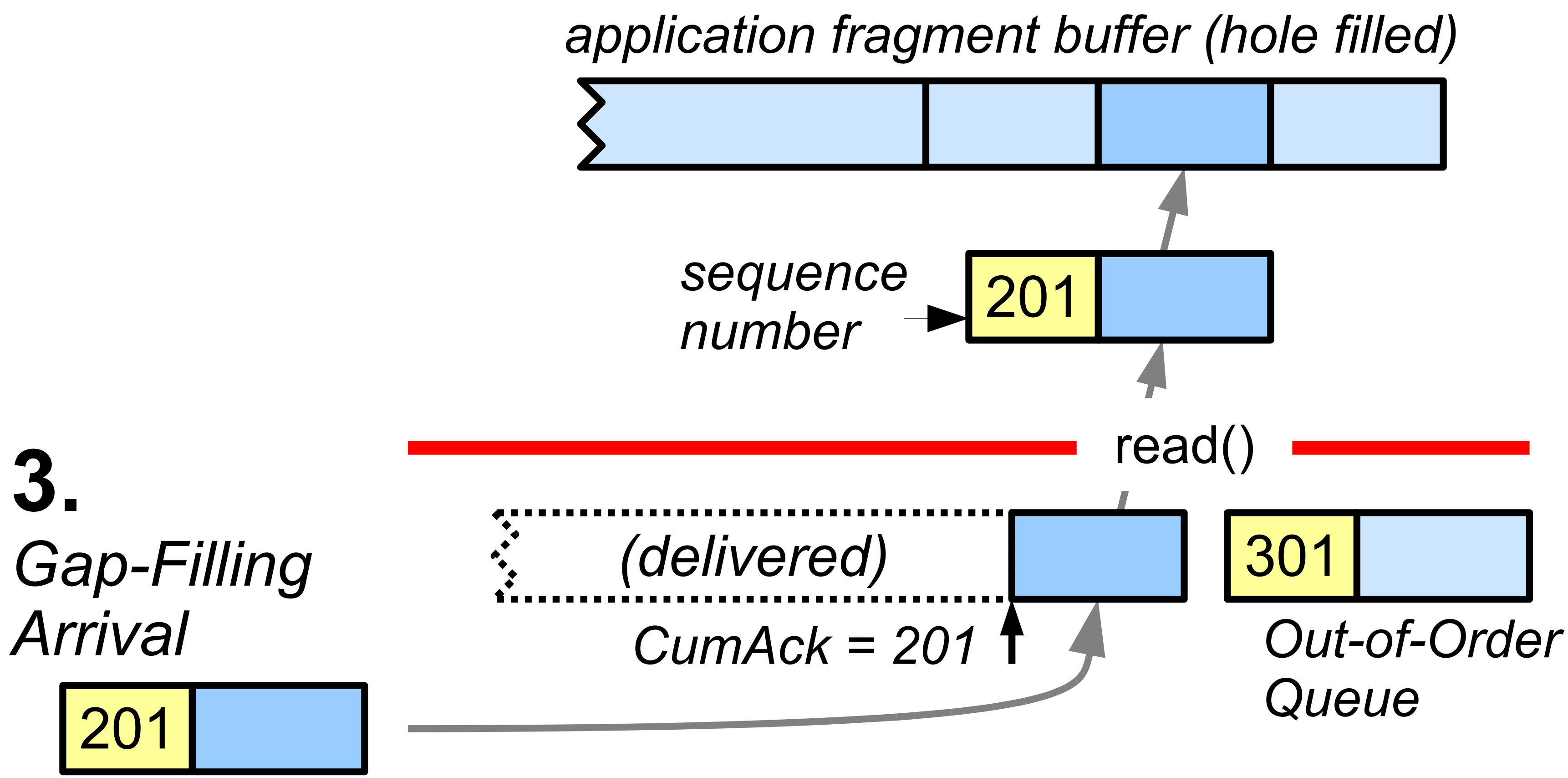
Delivery in Minion/uTCP



Delivery in Minion/uTCP



Delivery in Minion/uTCP



Is Minion a “New Abstraction”?

From “IETF philosophy” (wire format, not API)

- Same network behavior → same “abstraction”
  - Stream of bytes with seqnos, all get ACKed, …

But looks pretty different to application!

- Unordered datagrams, fancy COBS encoding
  - Or whatever application builds on top of it!

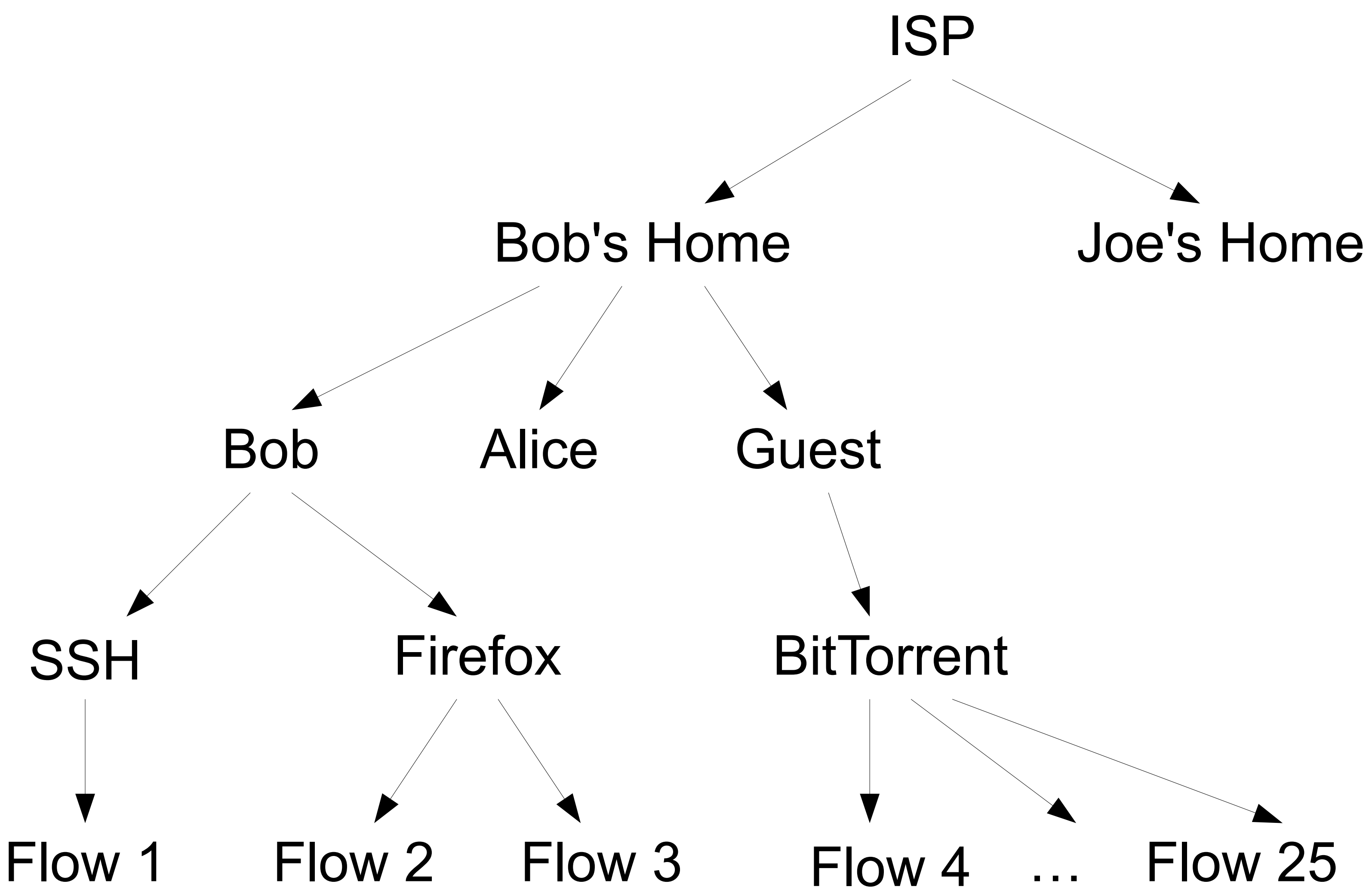
Consideration: do we need abstractions for *application convenience* or for *interoperability?*

Rate Control and Fairness

Transport connections are the traditional units of rate control and fair-sharing

- Flow, congestion control *supposed to* happen end-to-end between end hosts
  - Oops: Performance Enhancing Proxies (PEPs)
- Congestion control gives each competing TCP flow a “fair share” of bandwidth
  - Oops, wrong granularity for most purposes

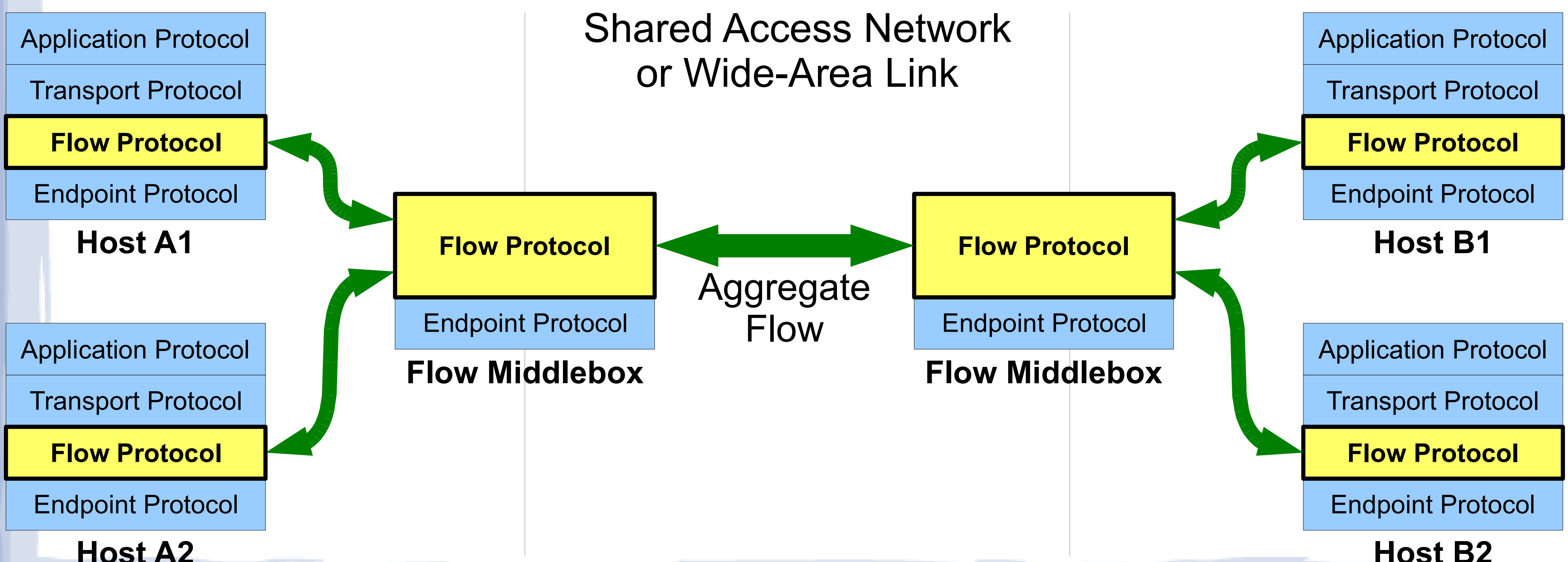
Stream as "Fairness Abstraction": *Wrong on So Many Levels*



What Might Work (but not sure...)

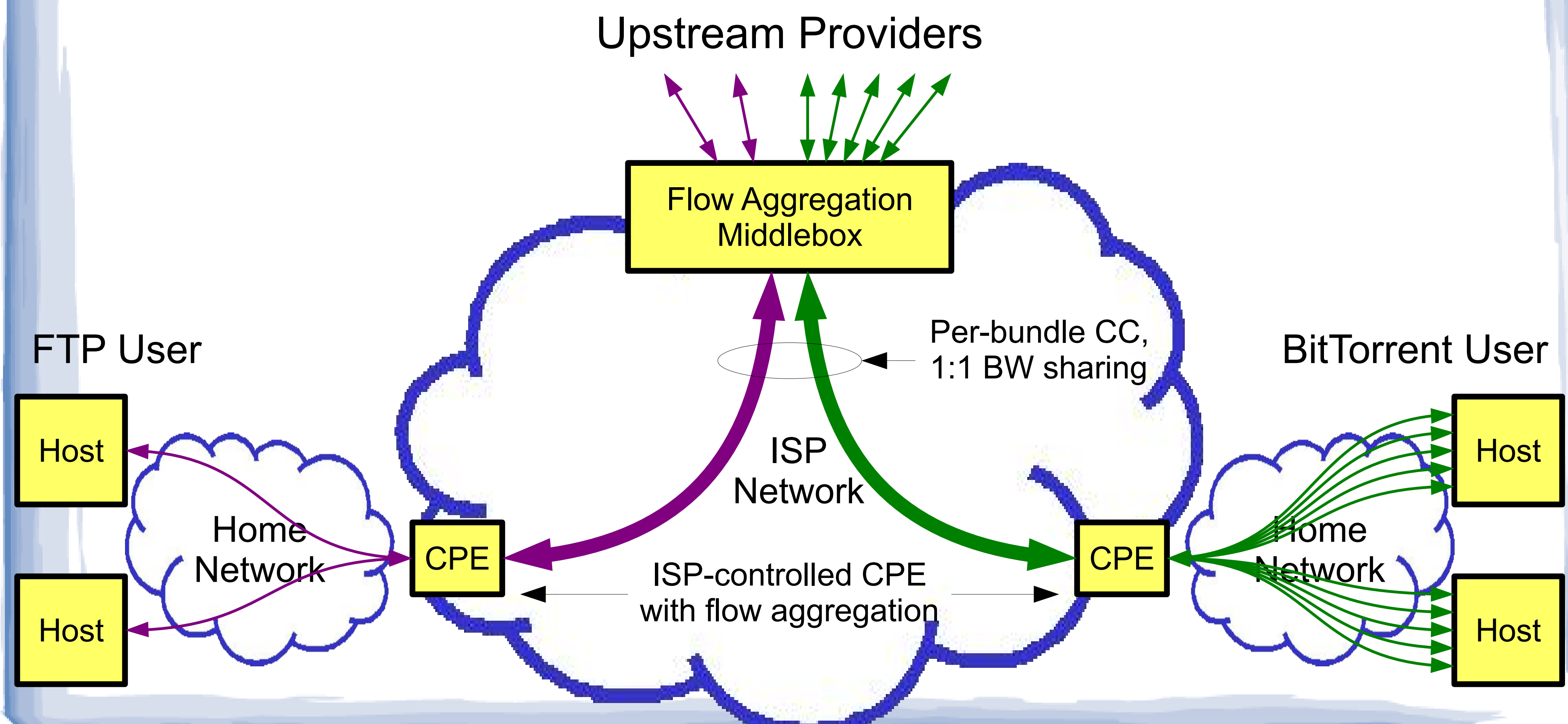
Tunnels within Tunnels, Layers upon Layers...

- Aggregation at “Flow Layer” [HotNets '08]
- Recursive Internet designs [Day, Zave]



“Fairness Enhancing Middleboxes”

Give customers **equal shares** of upstream BW
independent of # connections per customer



(Non-)Conclusion

Transports “roll many abstractions into one”

- Data Movement, Rate Control, Fair Sharing, Reliability, Endpoint Naming, Pluggability

How *should* we choose transport abstractions?

- Are abstraction choices *fundamental* or just about properties of *current* implementations?
- Are they about the *network* or the *application*?
- What are the implications of *granularity,* and how can we get the *right* granularity?