Breaking Up the Transport Logjam

Bryan Ford

Max Planck Institute
for Software Systems
and Yale University

baford@mpi-sws.org

Janardhan Iyengar

Franklin & Marshall
College

jiyengar@fandm.edu

TSVAREA Open Meeting, IETF 73, 19 November 2008

Evolutionary Pressures on Transports

- **Applications** need more flexible abstractions
  - many semantic variations [RDP, DCCP, SCTP, SST, ...]
- **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

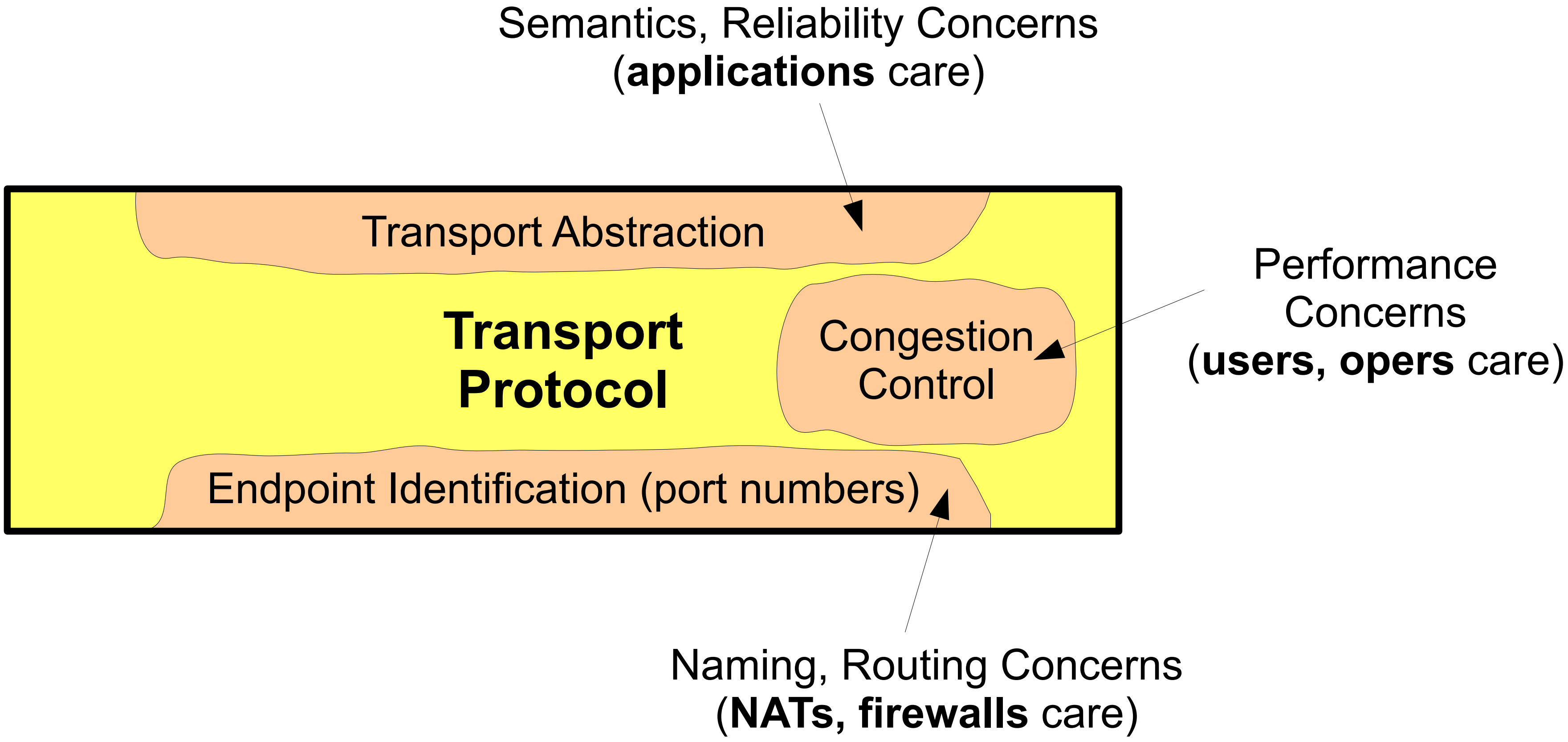
The Transport Layer is Stuck in an Evolutionary Logjam!

Many Solutions, None Cleanly Deployable

- New transports **undeployable**
  - NATs & firewalls
  - chicken & egg: application demand vs 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!"
  - Fundamentally "TCP-unfriendly"?

The Problem

Traditional transports conflate **3 function areas**...



Semantics, Reliability Concerns
(**applications** care)

Transport Abstraction

Transport Protocol

Congestion Control

Performance Concerns
(**users, opers** care)

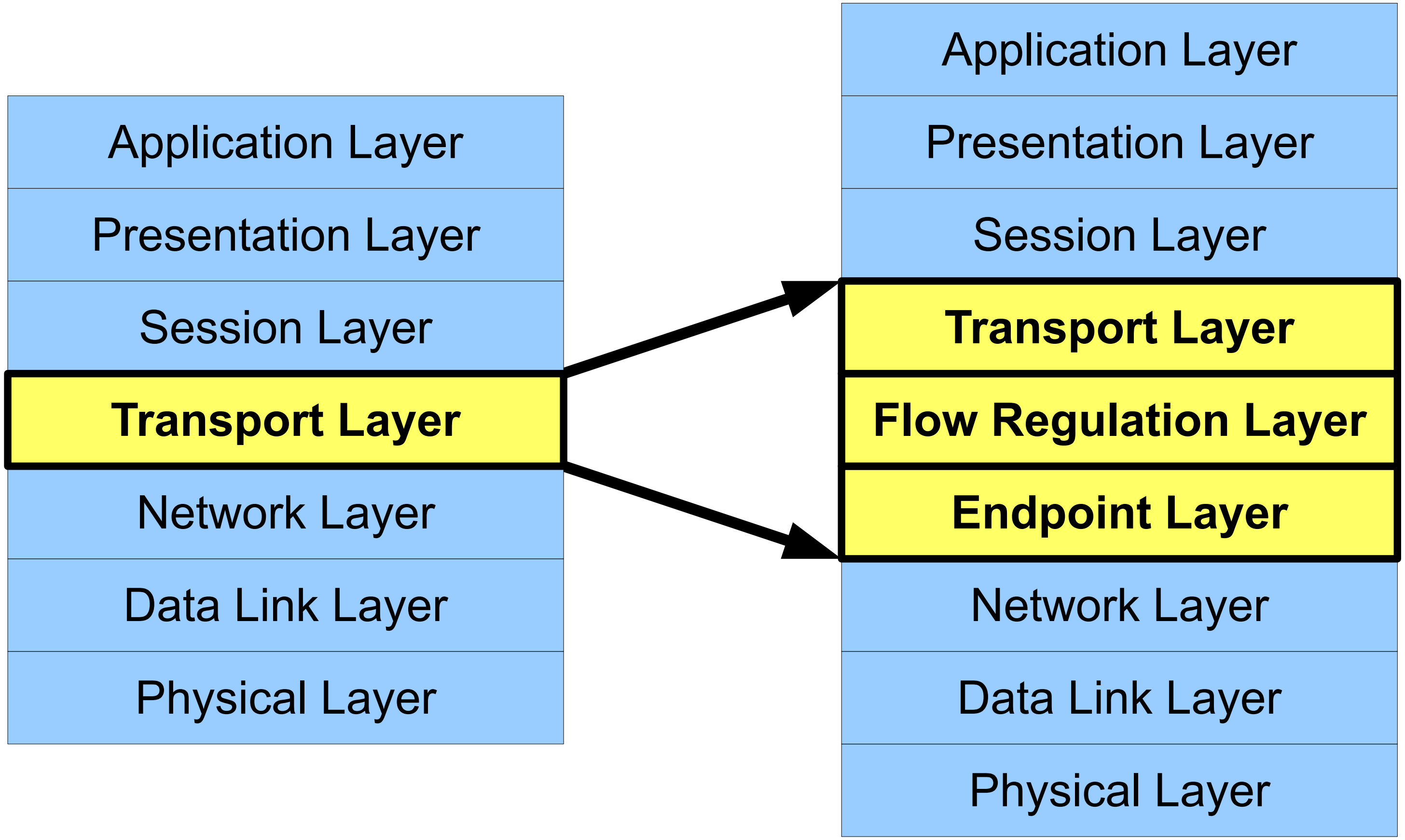
Endpoint Identification (port numbers)

Naming, Routing Concerns
(**NATs, firewalls** care)

To break transport logjam, must **separate concerns**

Our Proposal

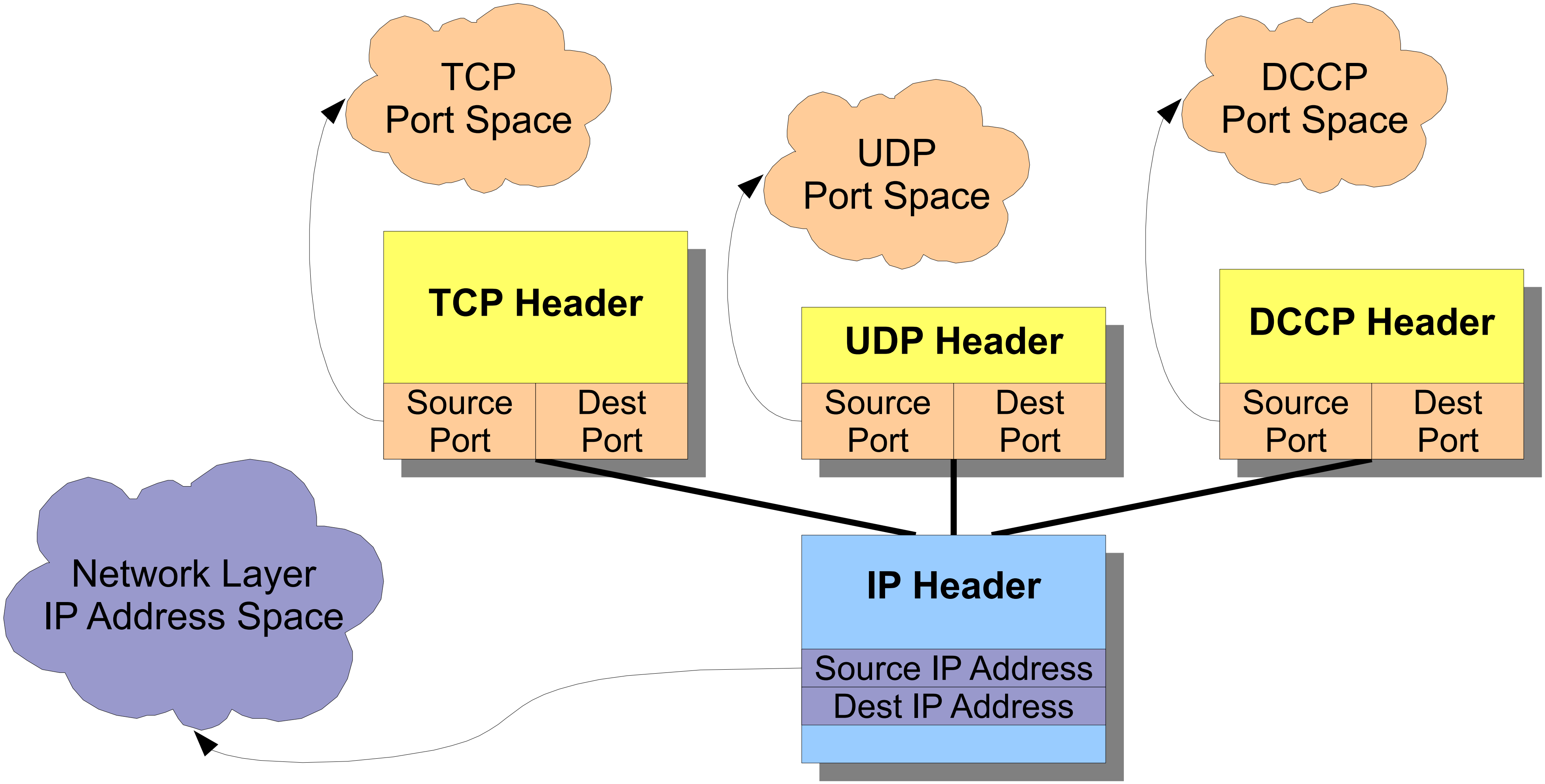
Break up the Transport according to these functions:



Endpoint Layer

Endpoint Identification via Ports

Current transports have **separate** port spaces



But What Are Ports?

Ports are **routing info**!

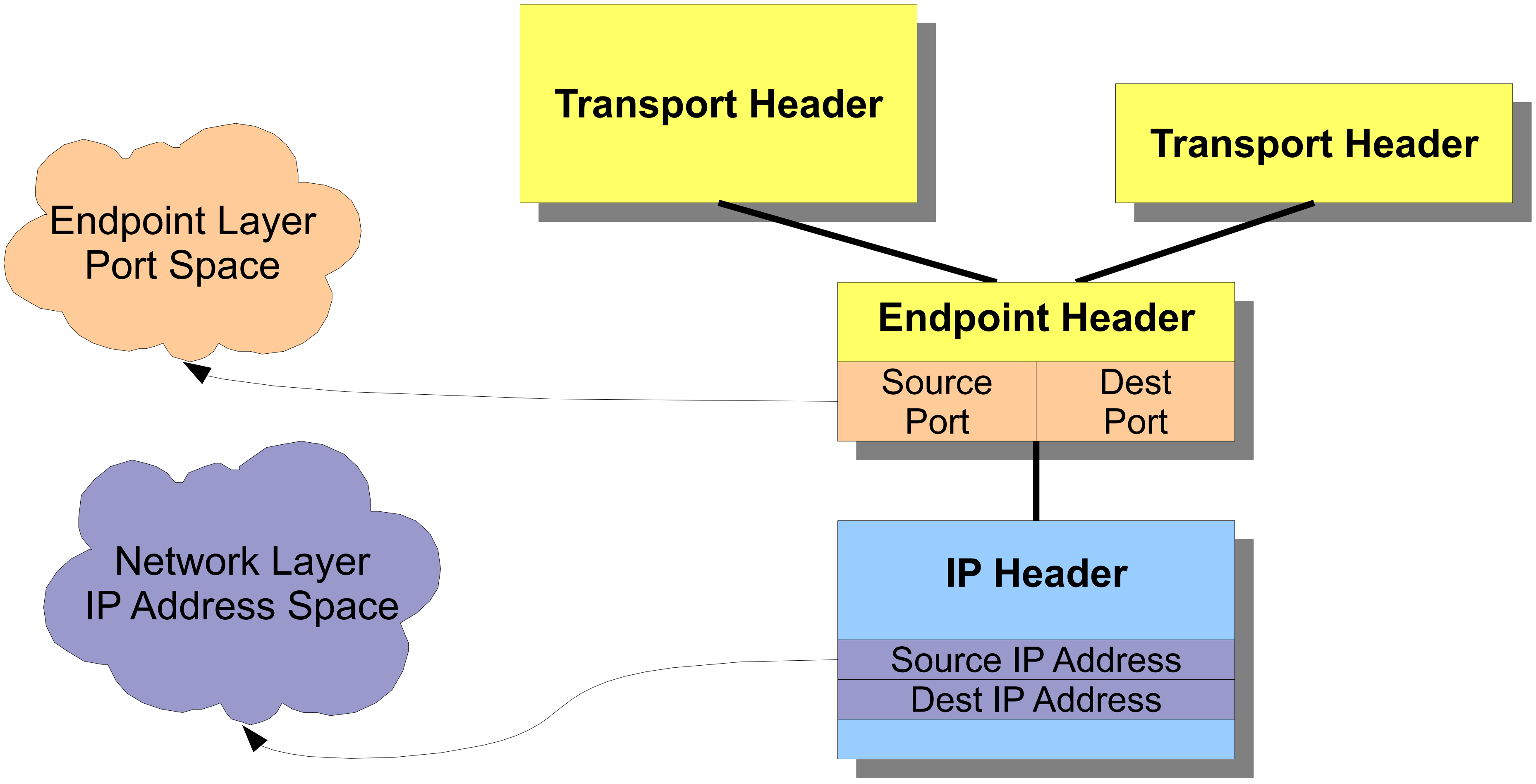
- IP address ⇒ Inter-Host Routing
- port numbers ⇒ *Intra*-Host Routing

Do ports *really* belong in the **Network Layer?**

- Firewalls, NATs, traffic shapers need to know ports
  - Parse transport headers ⇒ only TCP, UDP get through
- IPv4: ports increasingly just "16 more IP address bits"
  - DHCP port borrowing/sharing [Despres, Bajko, Boucadair]
- IPv6: *could* dispense with ports entirely
  - Assign each host a CIDR subnet, low bits = "port #"

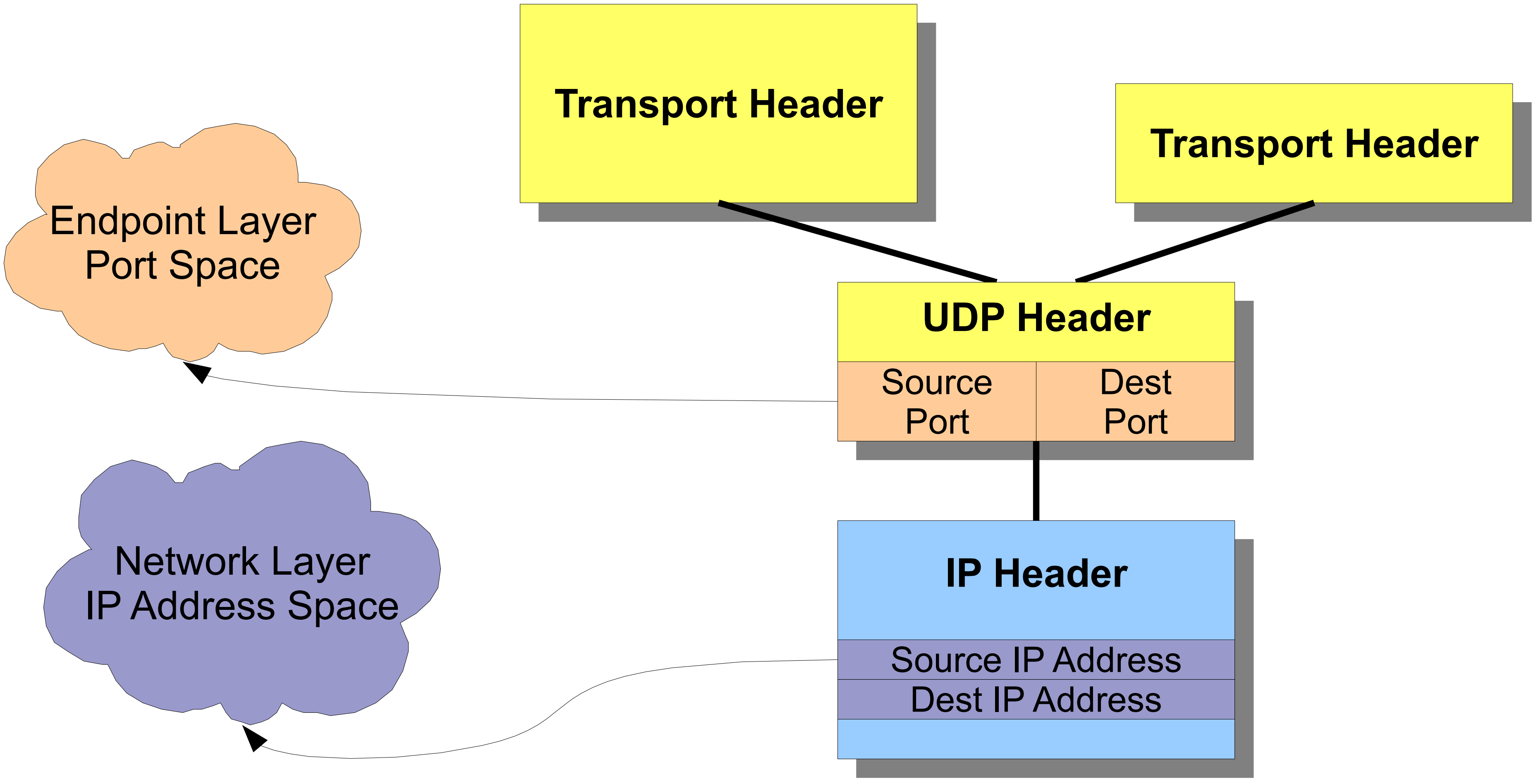
A Pragmatic Approach

Factor endpoints into shared **Endpoint Layer**



Surprise!

Workable starting point exists — **UDP**!



Embrace the Inevitable

It's happening in any case!

- TCP/UDP is "New Waist of the Internet Hourglass" [Rosenberg 08]
- Every new transport requires UDP encapsulations
  - SCTP [Ong 00, Tuexen 07, Denis-Courmont 08]
  - DCCP [Phelan 08]
- And a lot of non-transports do too
  - IPSEC [RFC 3947/3948], Mobile IP [RFC 3519], Teredo [RFC 4380], …

...but the new model also has **technical benefits**...

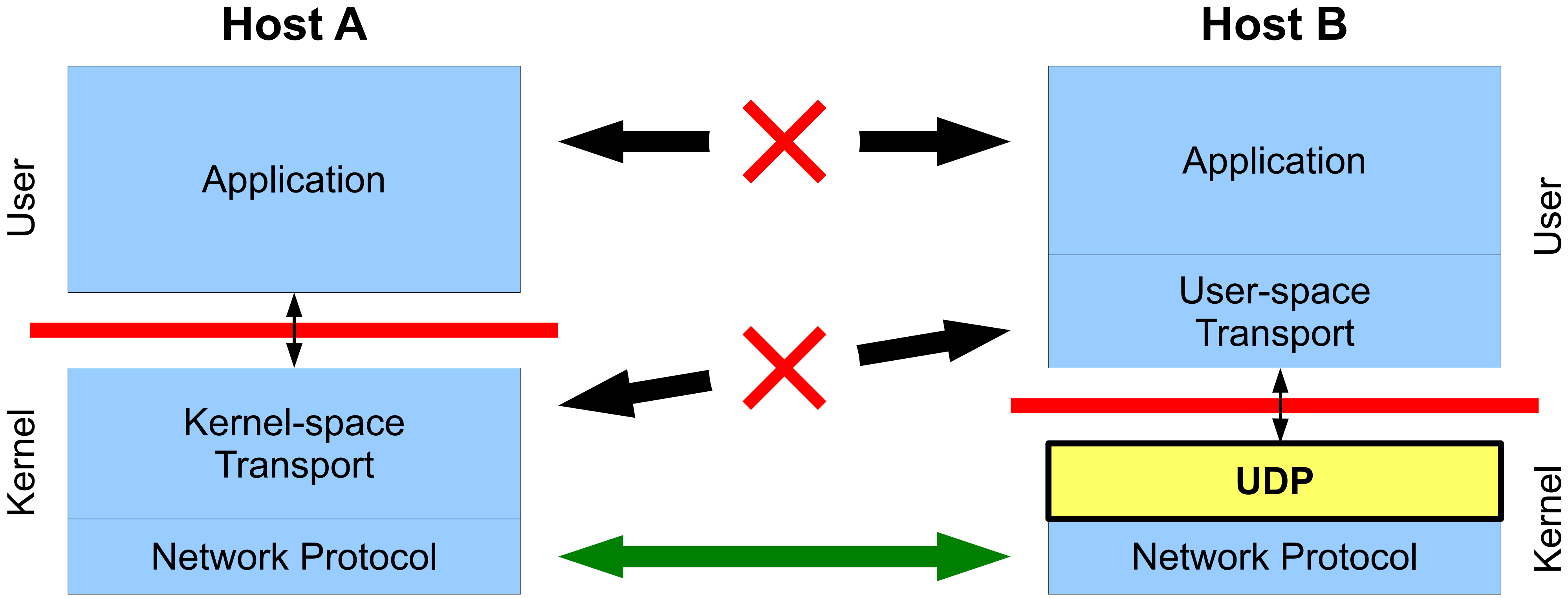
Practical Benefits

Can now **evolve separately:**

- **Transport functions:**
  - New transports get through firewalls, NATs, etc.
  - 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 ...?

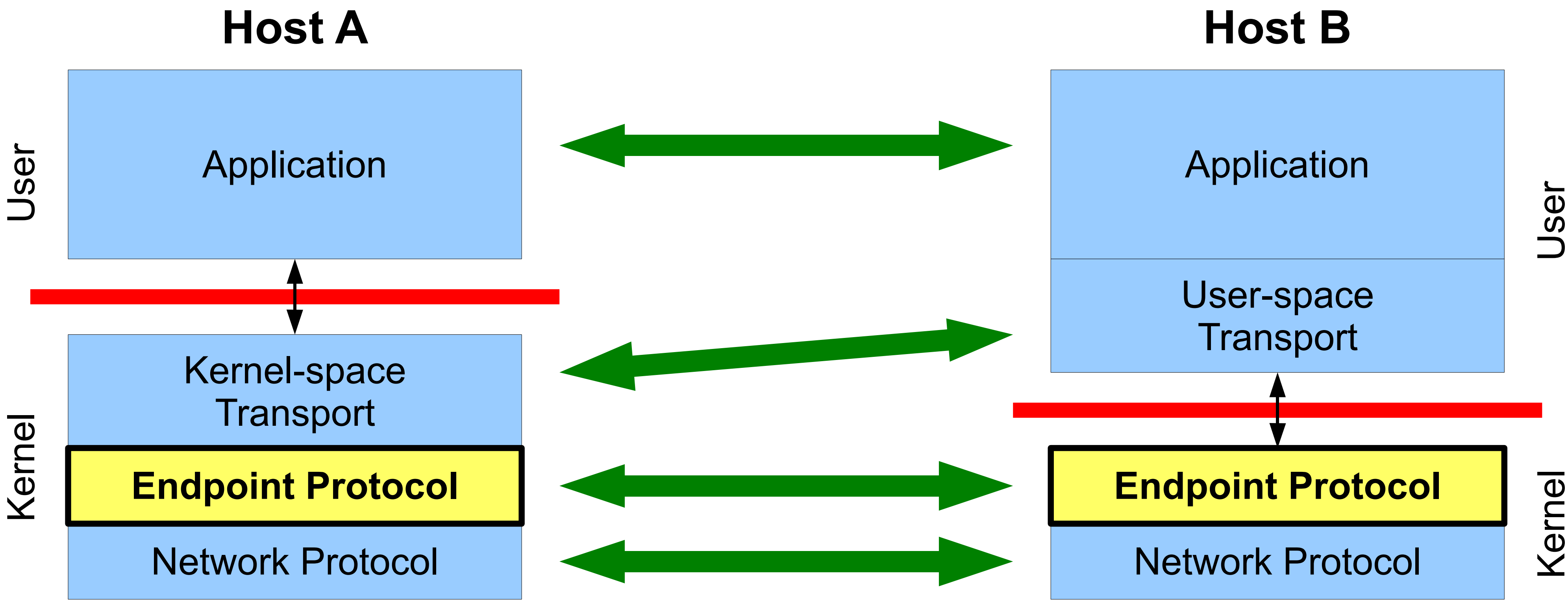
Kernel/User Transport **Non-**Interoperability

User-space transports are easy to deploy, but can't talk to kernel implementations of same transport!
(without special privileges, raw sockets, etc.)



Kernel/User Transport Interoperability

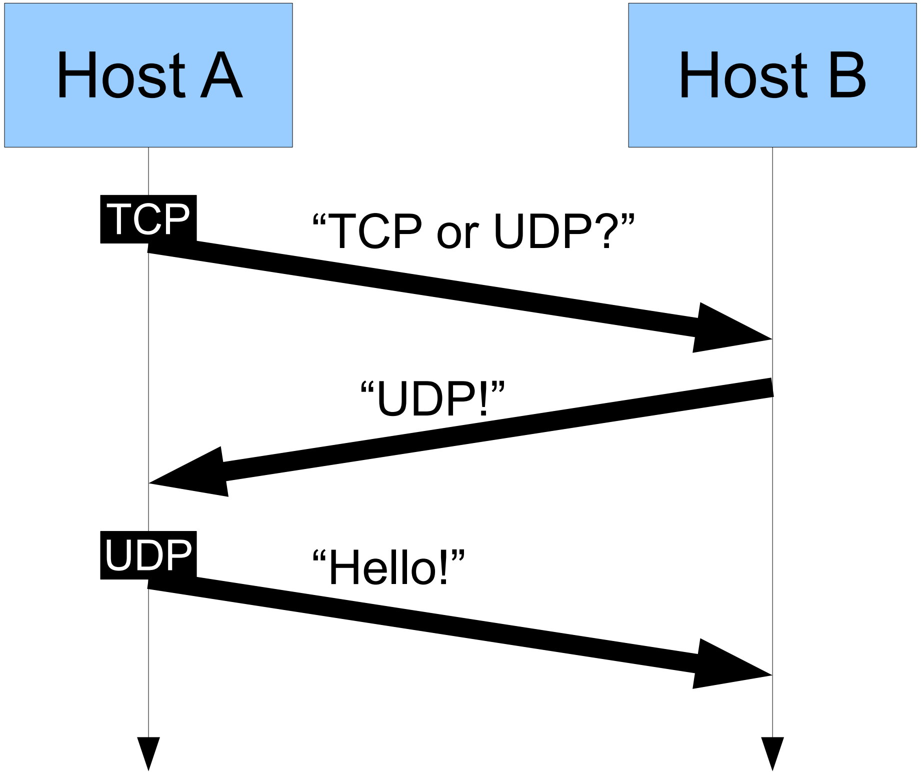
Endpoint layer provides **full interoperability**, user-space transports require **no special privileges**



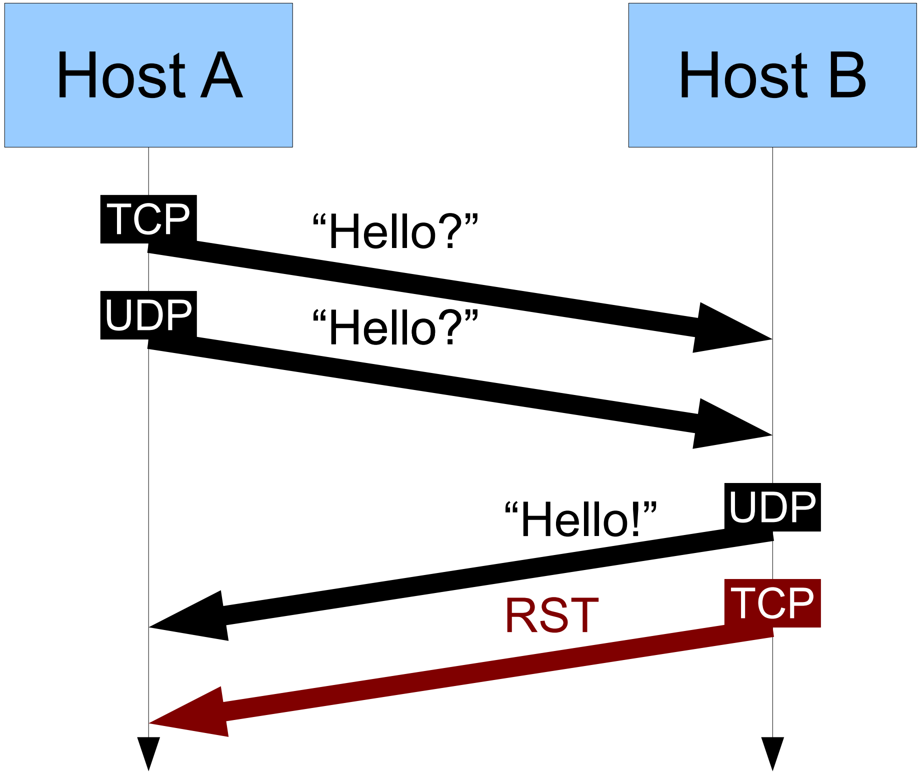
Transport Negotiation

Many applications support **multiple transports**, but can't **negotiate** them efficiently

“Cautious Negotiation”

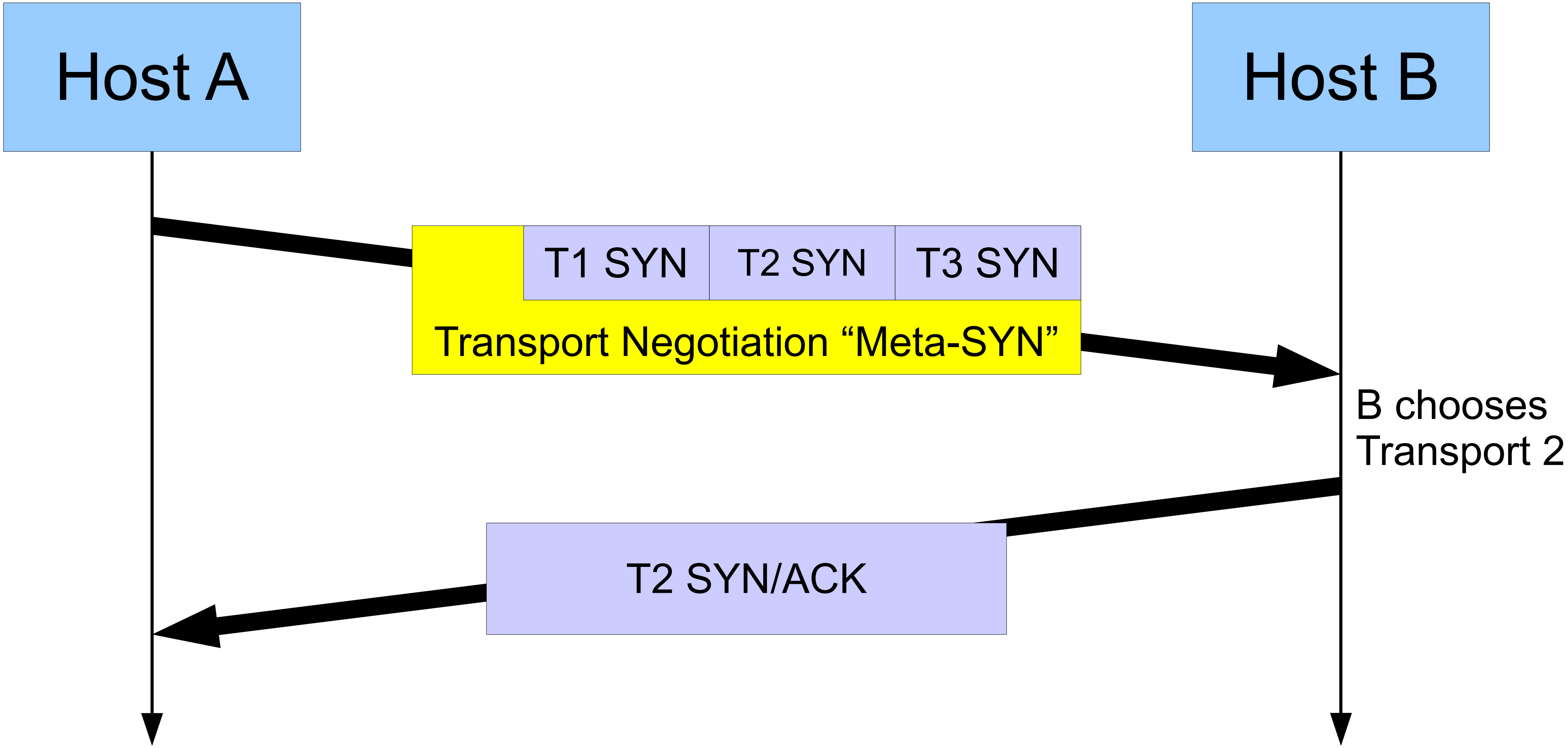


“Shotgun Negotiation”



“Zero-RTT” Transport Negotiation

When **application** controls its Endpoint Layer ports, it can combine transport **negotiation** with **setup**



Future Endpoint Layer Evolution

“Next-Generation Endpoint Layer” could:

- Remain backward-compatible with UDP
  - Use same port space, fall back on UDP transparently
- Annotate endpoints with richer information
  - Port names [Touch 06], user/service names, auth info, ...?
- Proactively advertise listen sockets [Cheshire?]
  - NATs could propagate listener advertisements upstream, translate inbound connections *as policy permits*
  - Enable cleaner solutions to “NAT signaling” mess? [UPnP, NAT-PMP, MIDCOM, NSIS, ...]

Flow Layer

Traditional "Flow Regulation"

Transport includes end-to-end **congestion control**

- regulates flow transmission rate to network capacity

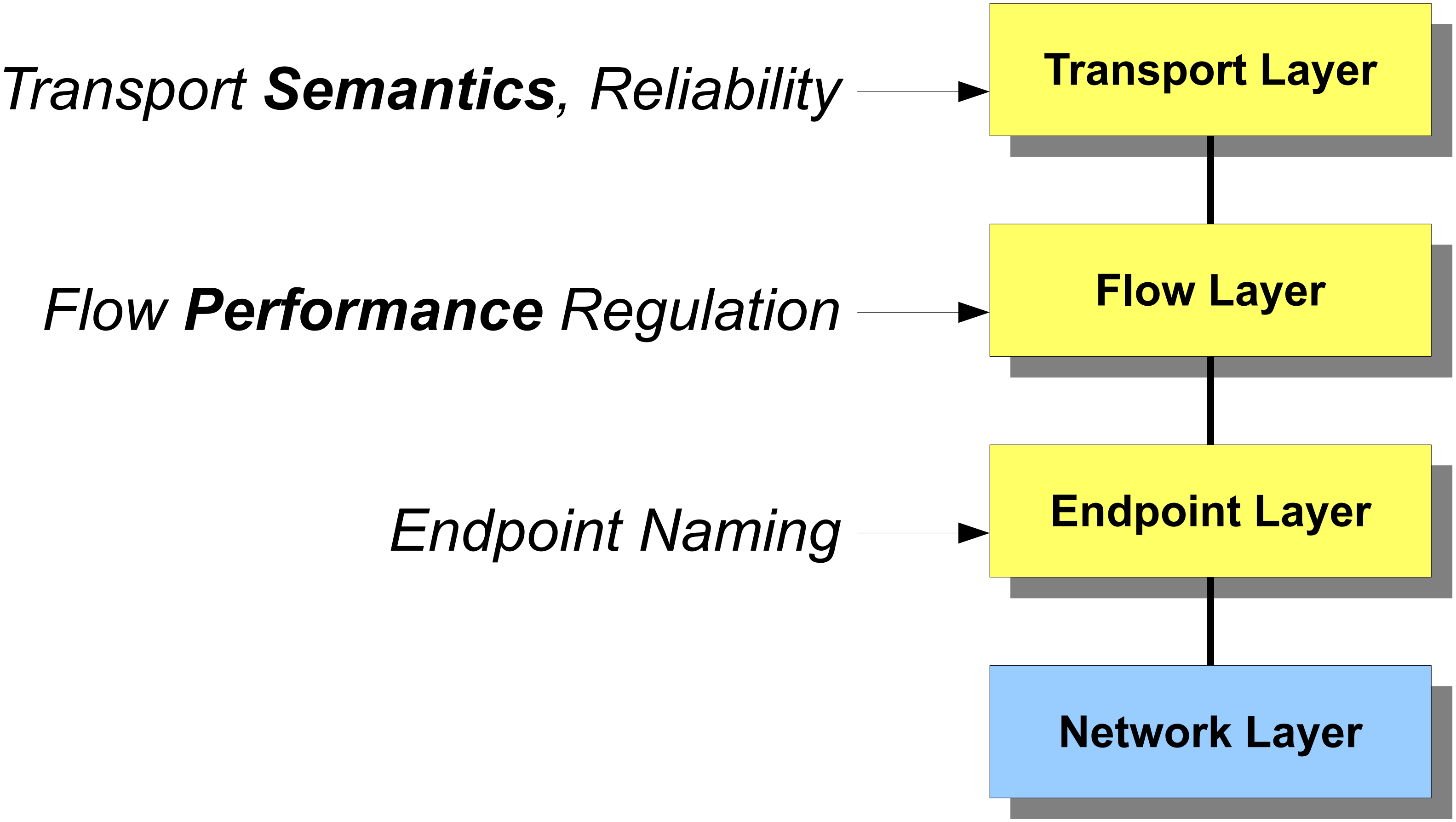
But one E2E path may cross **many**...

- 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!

Can't **tune performance, fairness** in one domain w/o affecting other domains, E2E semantics [RFC3515]

Proposed Solution

Factor flow regulation into underlying **Flow Layer**

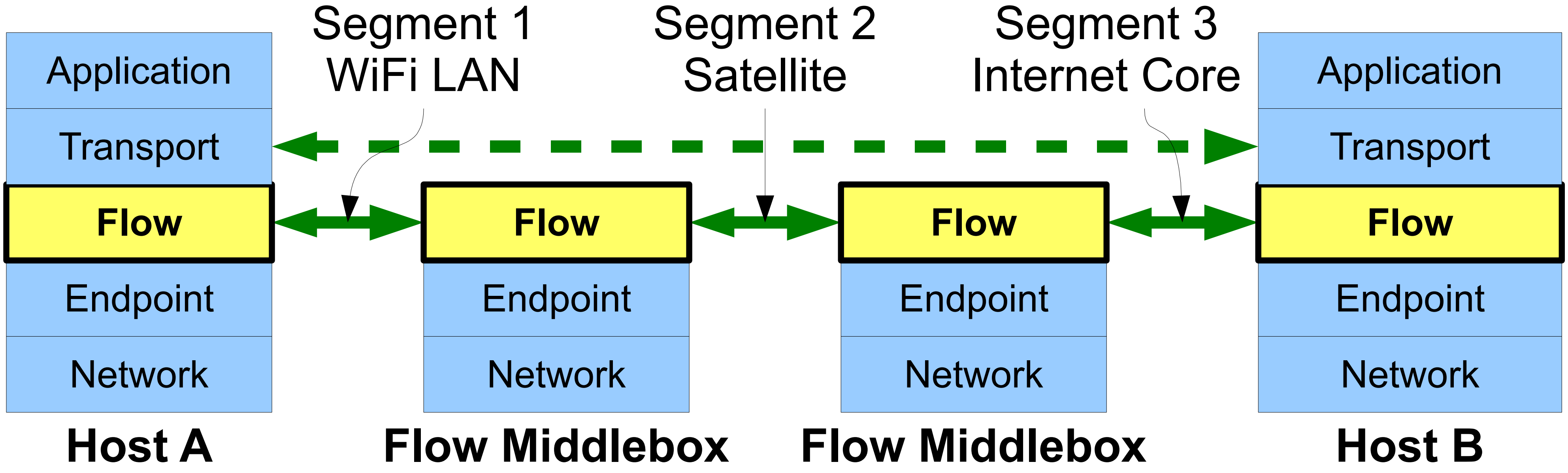


Practical Benefits (1/3)

Can split E2E flow into separate CC *segments*

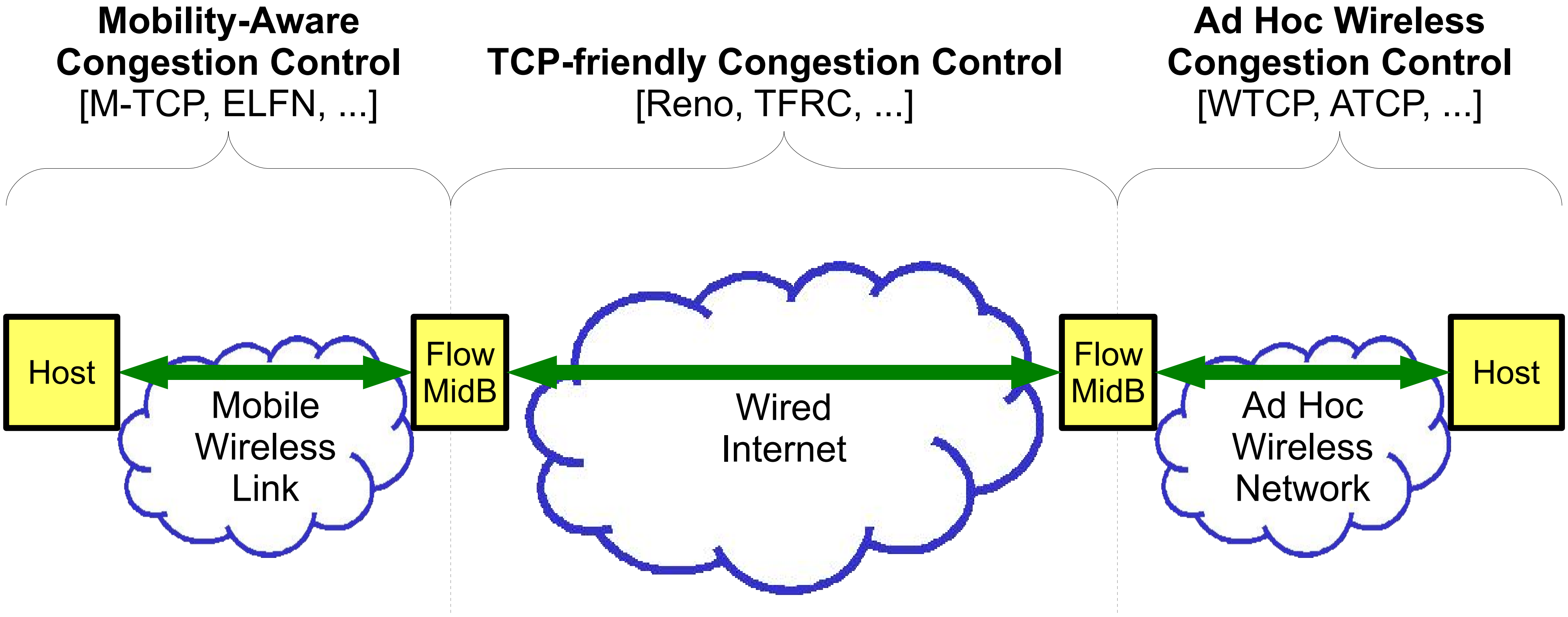
- Specialize CC algorithm to **network technology**
- Specialize CC algorithm within **admin domain**

… without interfering with E2E transport *semantics*!



Example Scenarios

(1) Last-mile proxies for wireless/mobile links



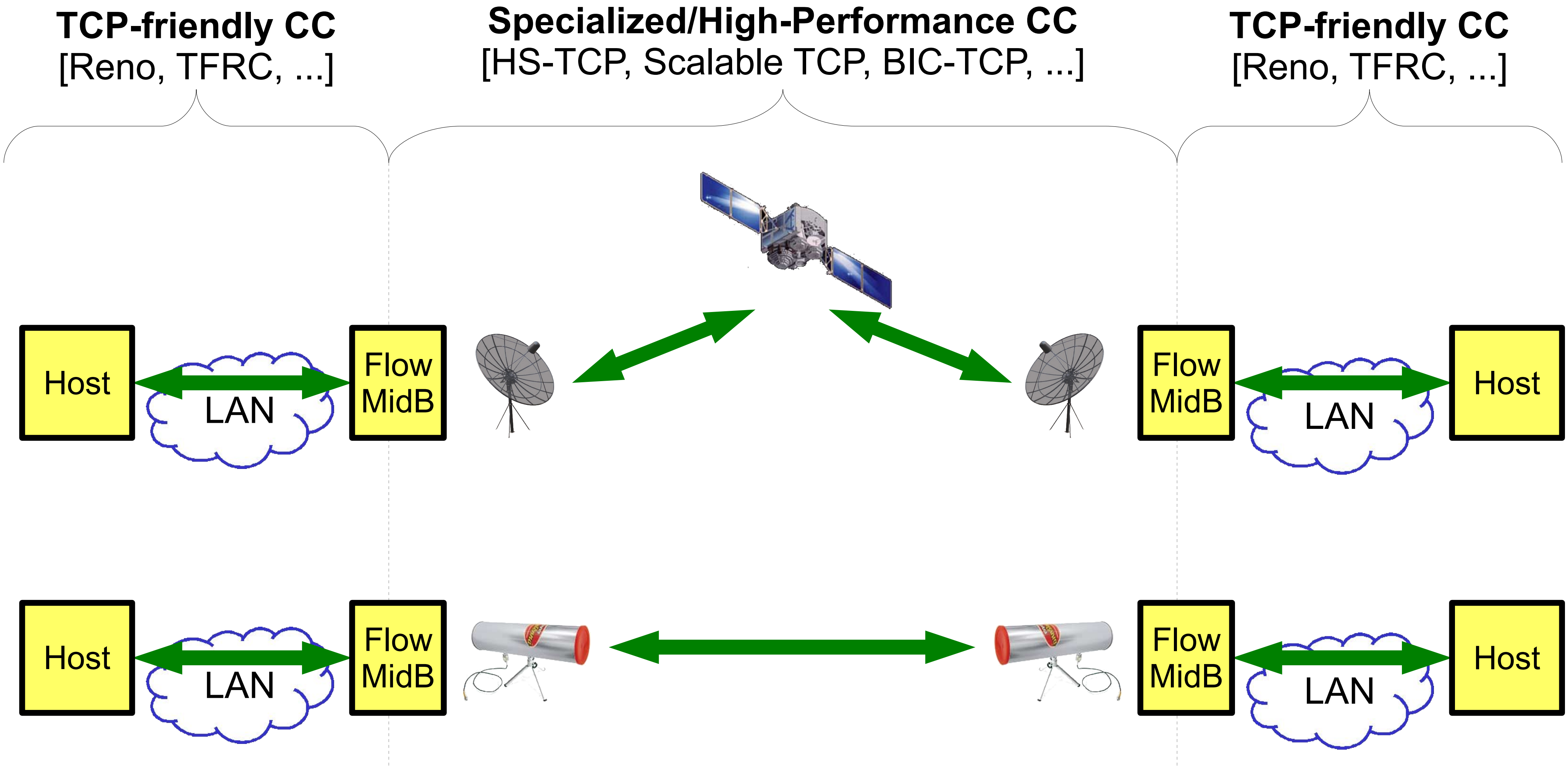
Mobility-Aware Congestion Control
[M-TCP, ELFN, ...]

TCP-friendly Congestion Control
[Reno, TFRC, ...]

Ad Hoc Wireless Congestion Control
[WTCP, ATCP, ...]

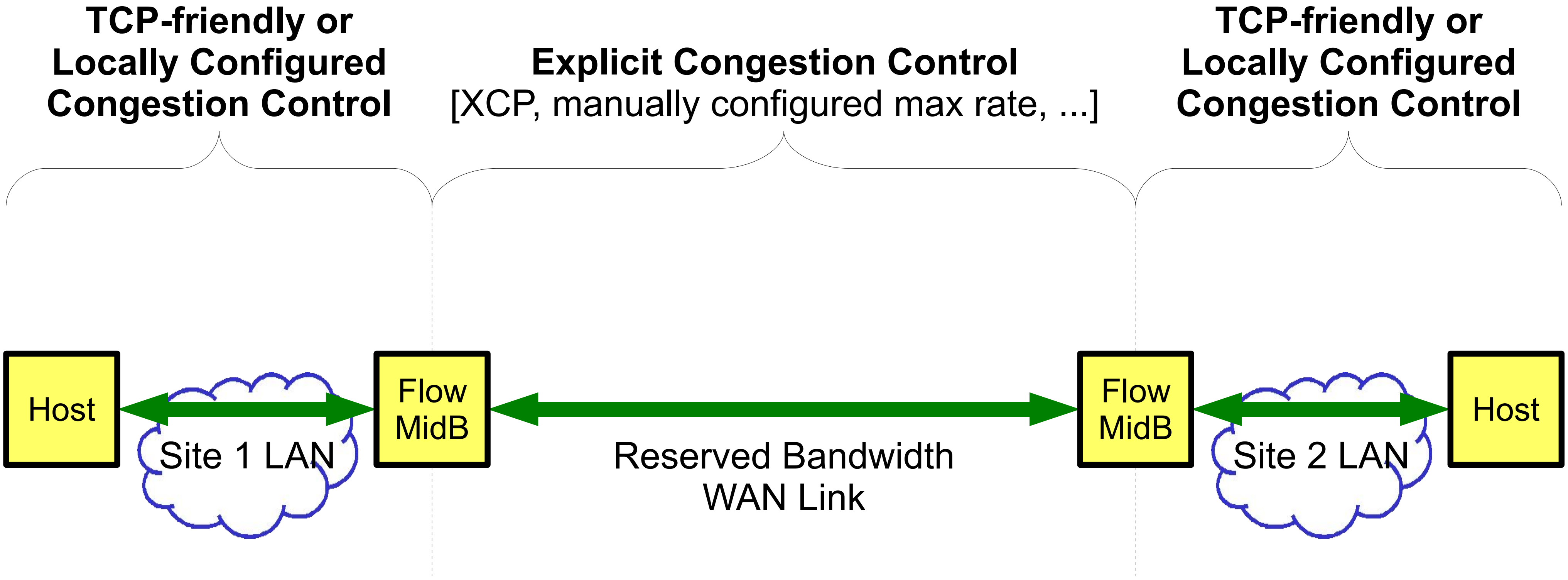
Example Scenarios

(2) Lossy Satellite or Long-Distance Wireless Links

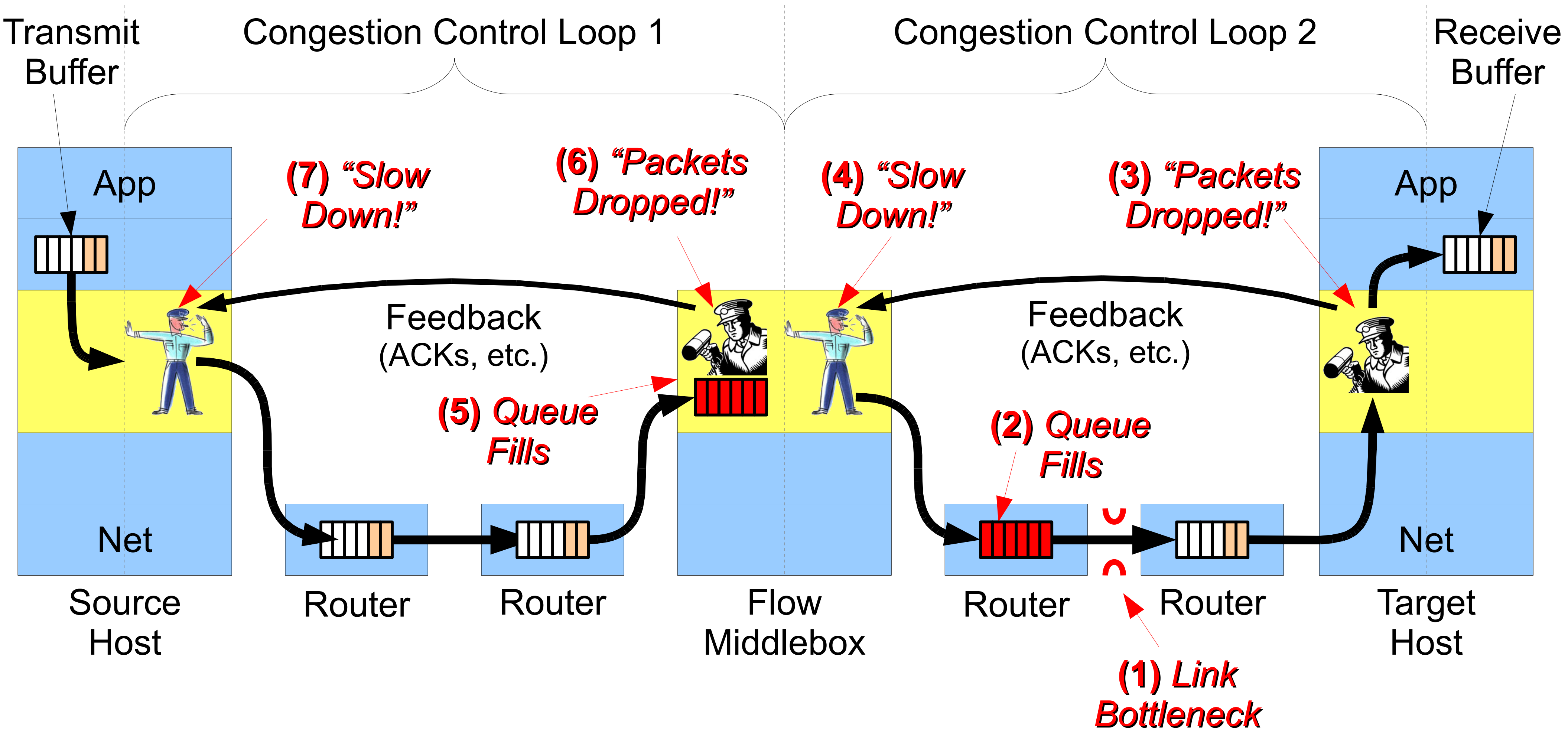


Example Scenarios

(3) Inter-Site WAN Links in Corporate Networks



End-to-End Congestion Control, One Segment at a Time

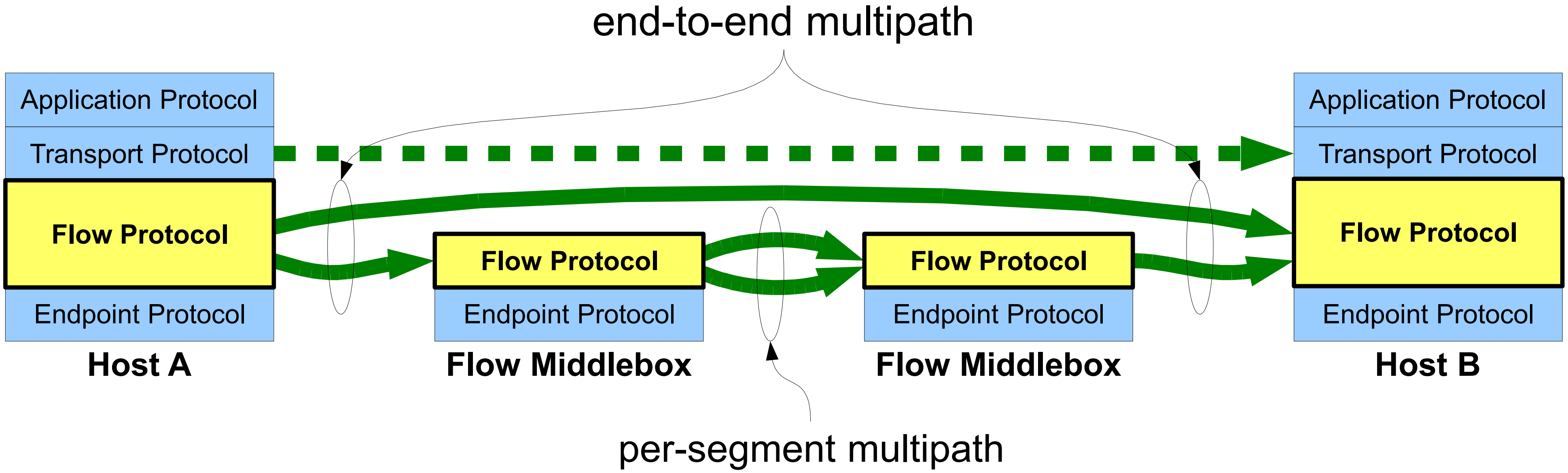


Practical Benefits (2/3)

Incrementally deploy performance enhancements

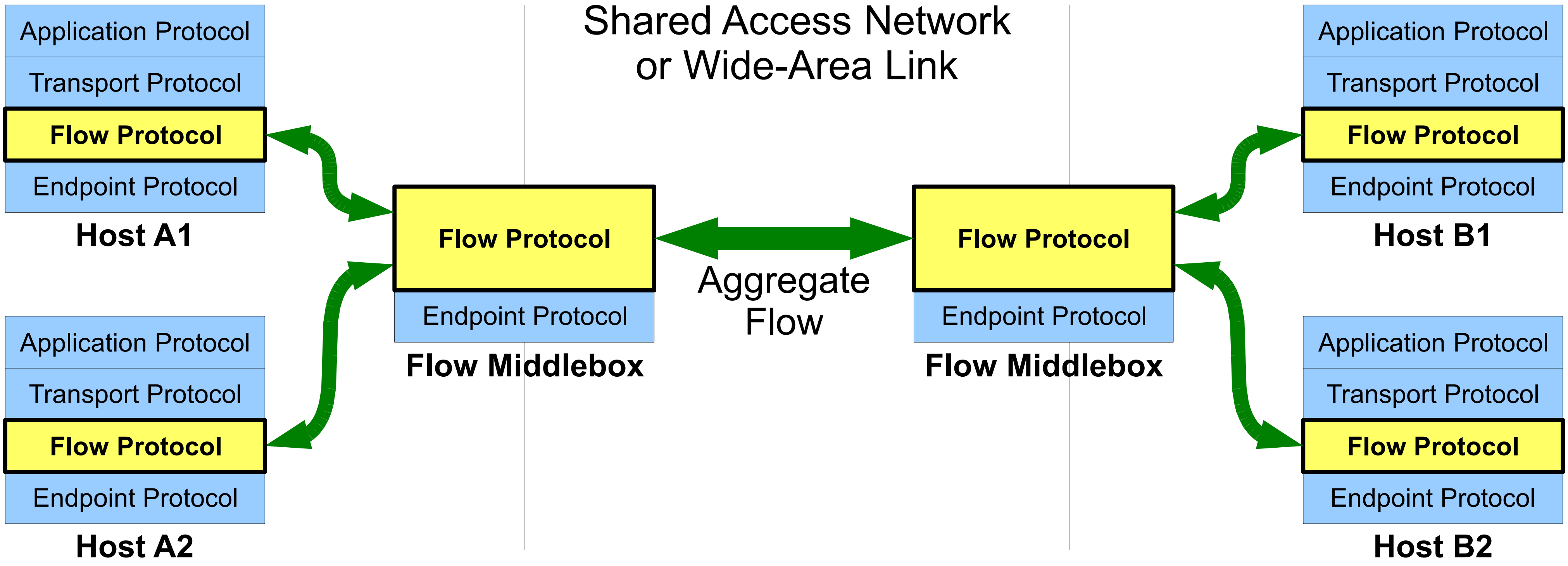
- multihoming [RFC 4960], multipath [Lee 01], dispersion [Gustafsson 97], aggregation [Seshan 97], ...

… without affecting E2E transport semantics!



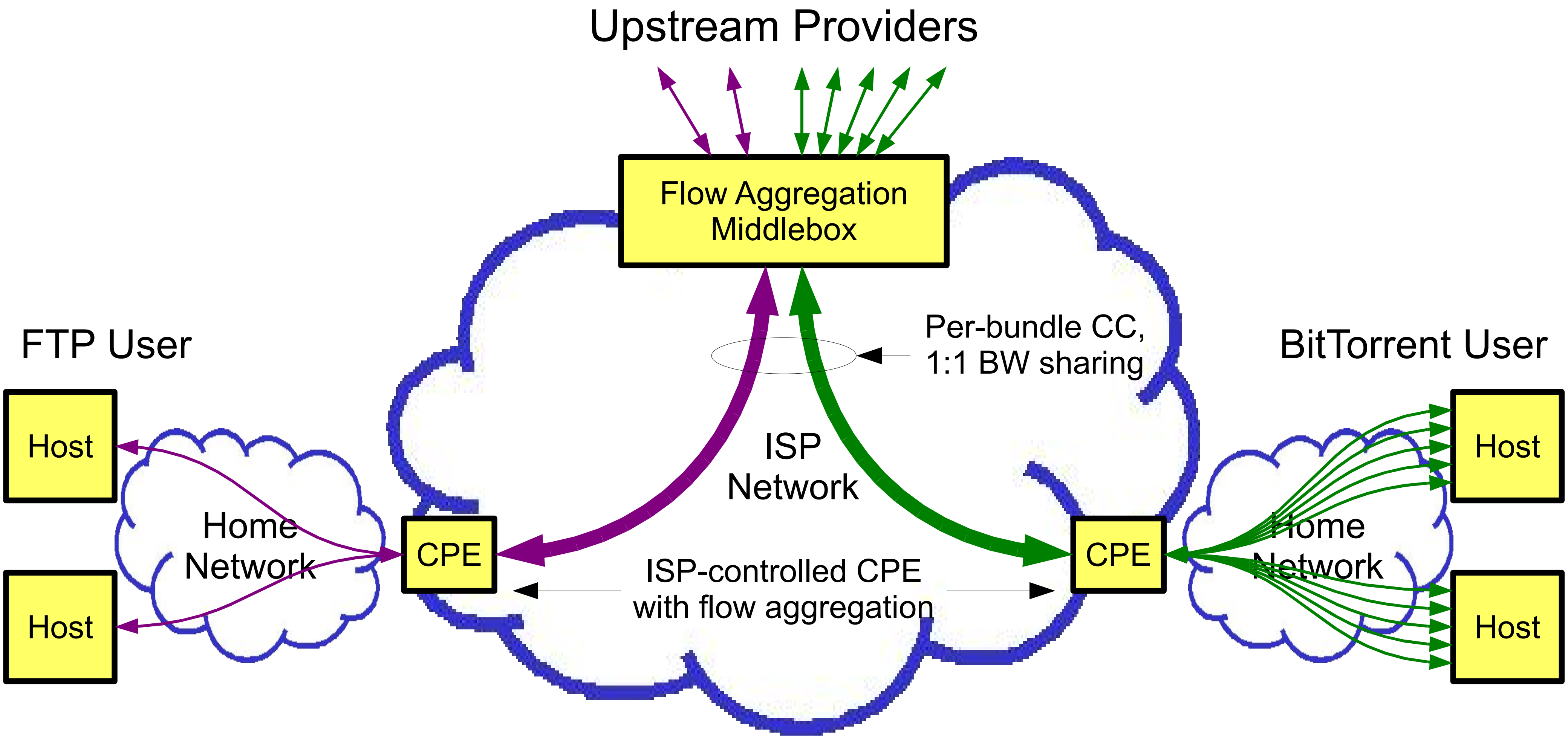
Practical Benefits (3/3)

- Can aggregate flows cleanly within domains for
  - Efficient traffic measurement, management
  - Fairness at "macro-flow" granularity



“Fairness Enhancing Middleboxes”

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



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
  - ...

Epilogue

The Transport Logjam Revisited

- New transports ~~un~~**deployable**
  - Can traverse NATs & firewalls
  - Can deploy interoperably in kernel or user space
  - Apps can negotiate efficiently among transports
- New congestion control schemes ~~un~~**deployable**
  - Can specialize to different network types
  - Can deploy/manage within administrative domains
- Multipath/multiflow enhancements ~~un~~**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
- We are interested in learning about other relevent applications/scenarios

Conclusion

Transport evolution is **stuck**

To unstick, need to separate functions:

- 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