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

Presentation for TU-Darmstadt – March 12, 2009

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

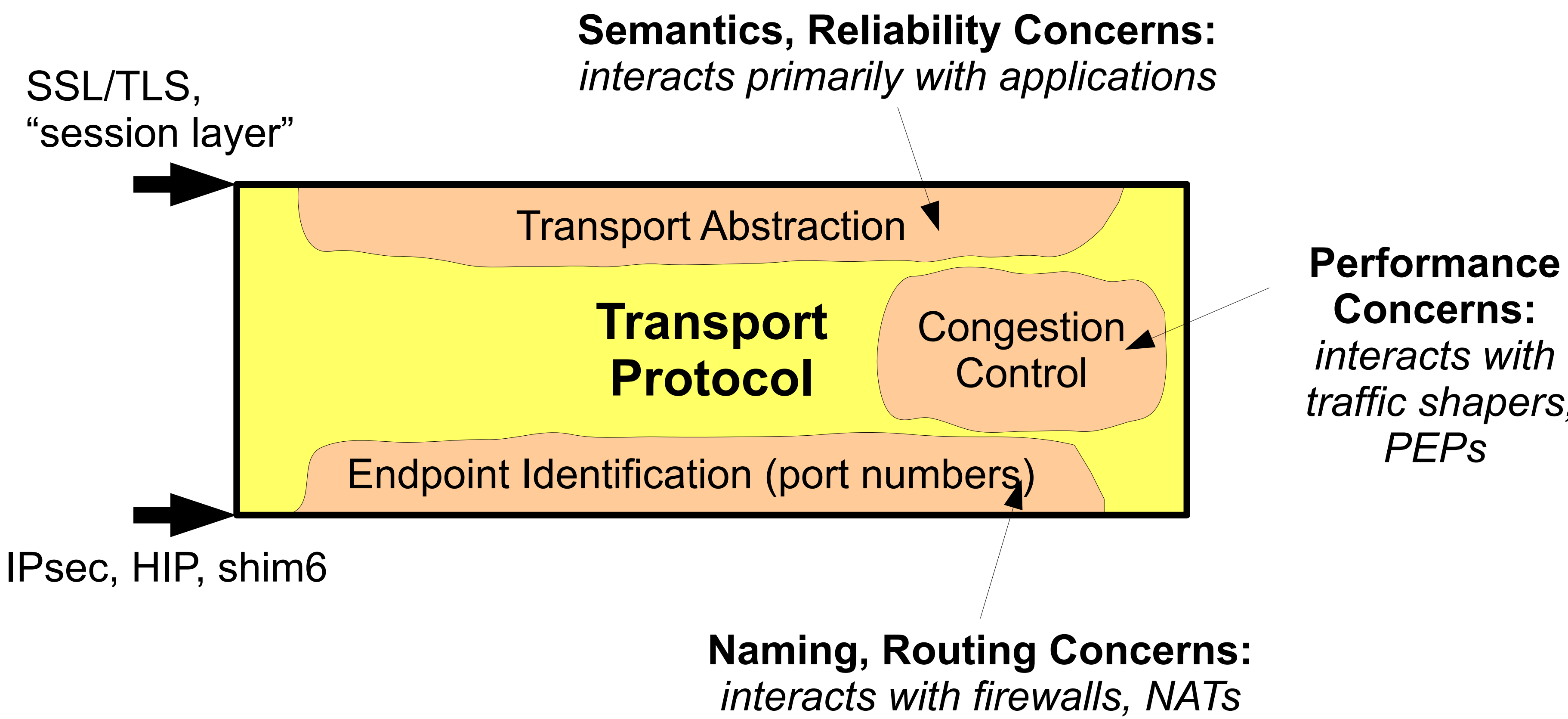
The Transport Layer is Stuck in an Evolutionary Logjam!

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

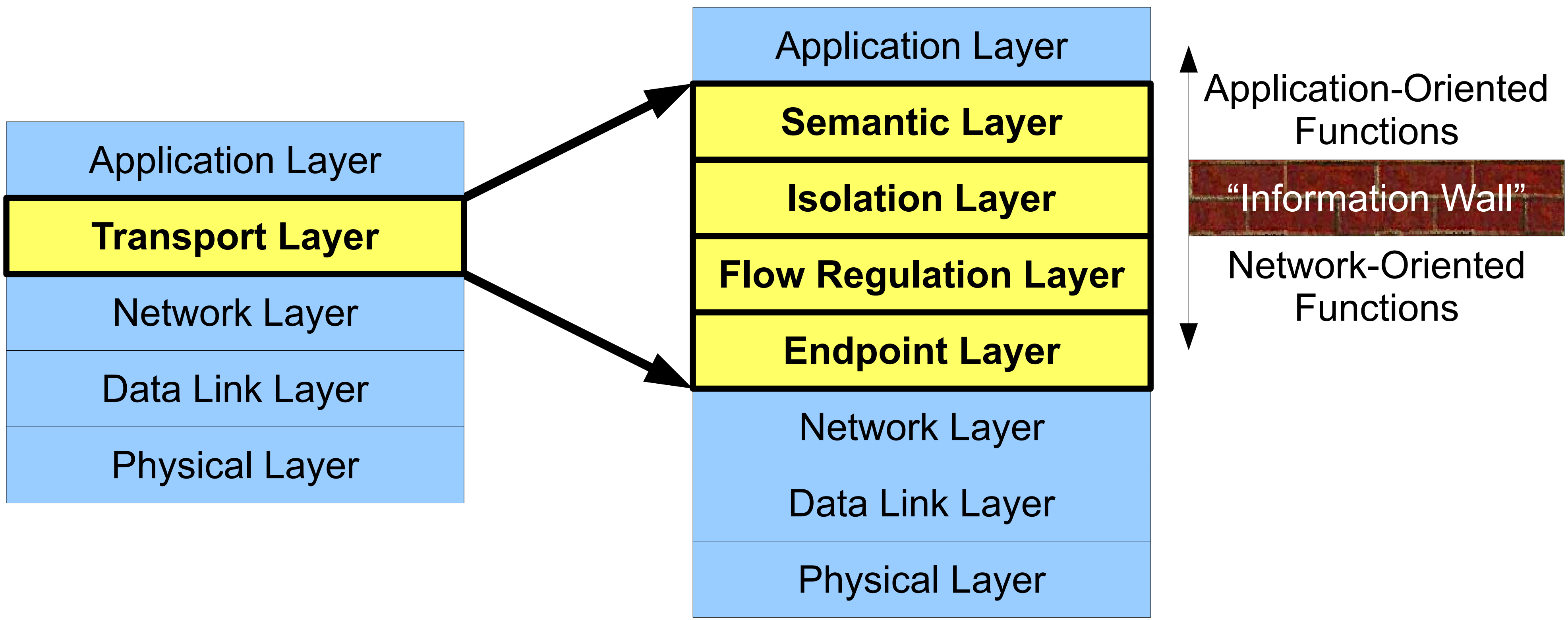
Current transports conflate **application-oriented** and **network-oriented** functions...



and where does security and location-independence go?

Our Proposal

Break up the Transport according to these functions:

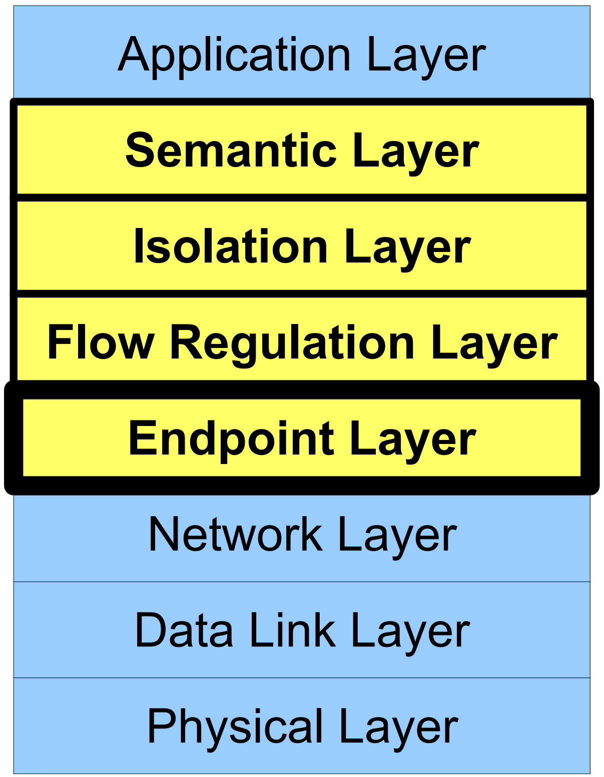


Layering Principles

- **Network Layer:**
  core routing needs to be simple, scalable, & stateless
- **Endpoint Layer:**
  edge routing needs richer endpoint info to enforce policy
- **Flow Regulation Layer:**
  performance tuning at technology & admin boundaries
- **Isolation Layer:**
  clean, enforceable separation between apps & network
- **Semantic Layer:**
  E2E reliability & semantics is purely app-driven

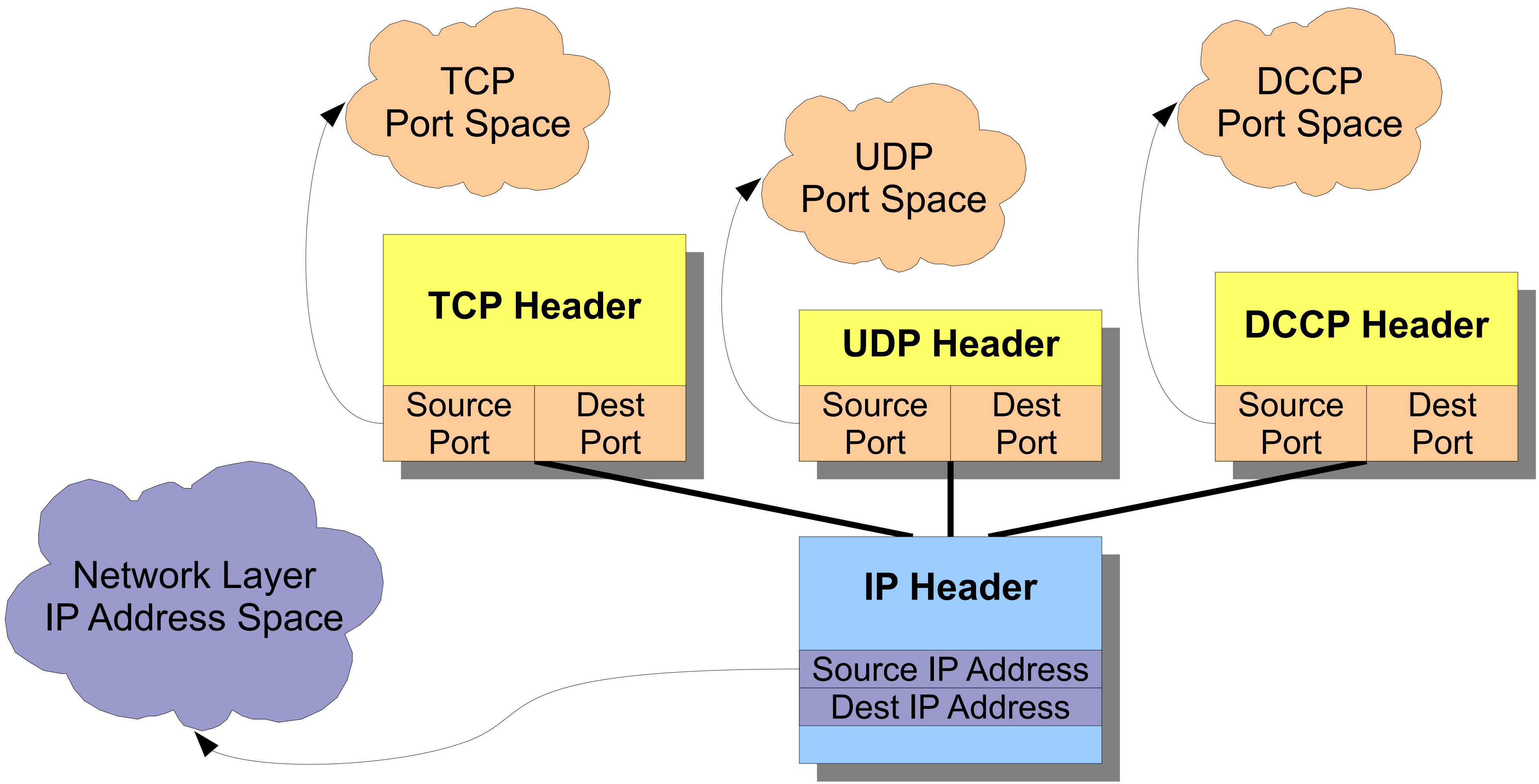
Endpoint Layer

edge routing needs
richer endpoint information
to enforce policy



Endpoint Identification via Ports

Each transport traditionally has a **port space**



Addresses Versus Endpoints

IP Address $\Longrightarrow$ identifies host (Inter-Host Routing)

Endpoint $\Longrightarrow$ identifies socket (Intra-Host Routing)

- Traditionally (IP Address, Port Number)
- Port numbers overloaded:
  - Well-known ports identify Applications
  - Dynamic ports identify Transport Sessions

Why the Network Needs to See Ports

Internet design assumes network needs *only* IP address

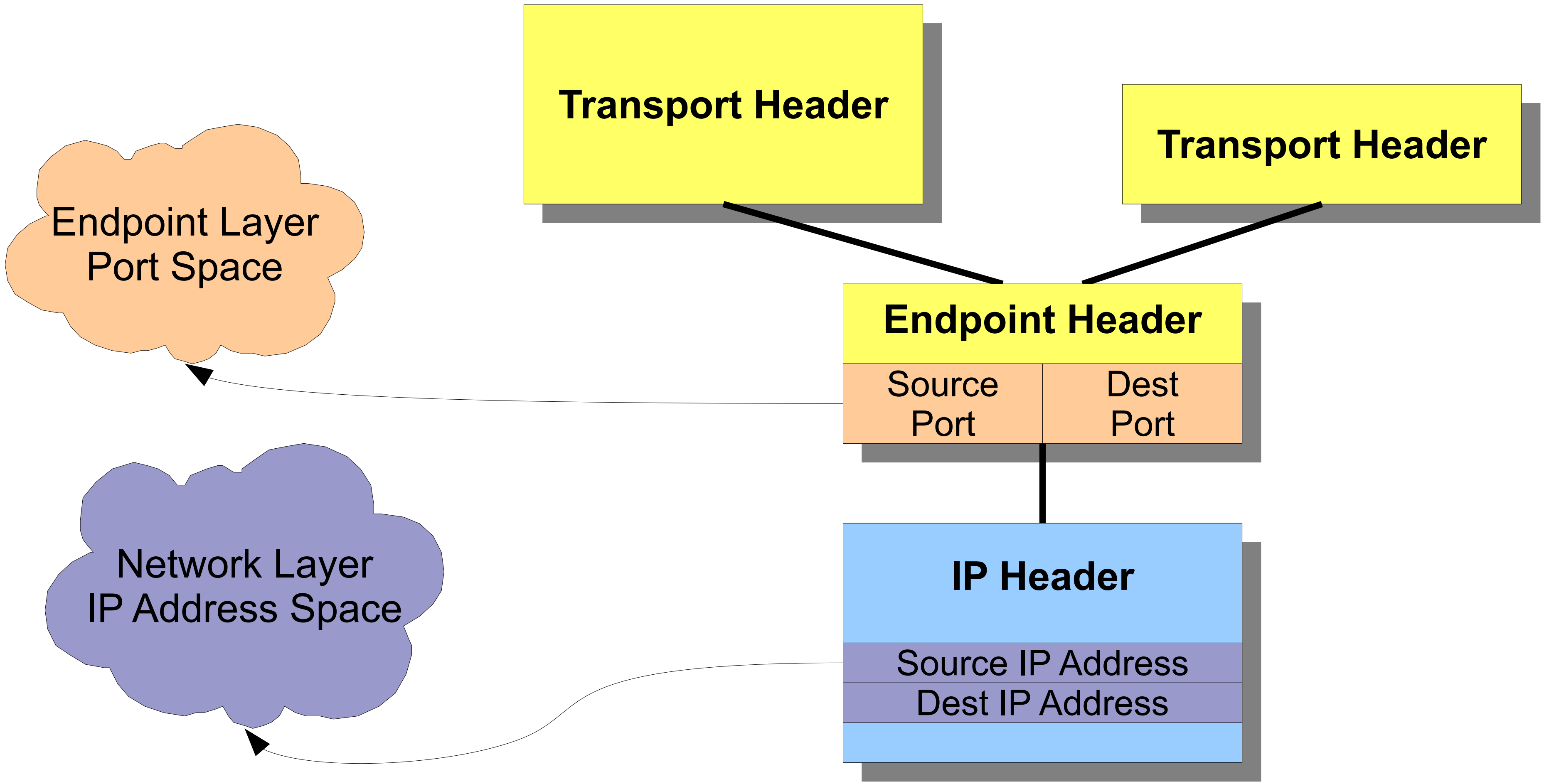
- (e.g., only IP address appears in every fragment)

Assumption has proven wrong!

- **Firewalls, traffic shapers** need to see them
  - to enforce connectivity policies, usage policies
- **NATs** need to see & transform them
  - IPv4: ports increasingly just "16 more IP address bits"
  - DHCP port borrowing/sharing [Despres, Bajko, Boucadair]
- **All** must understand transport headers
  - $\Longrightarrow$ only TCP, UDP can get through now

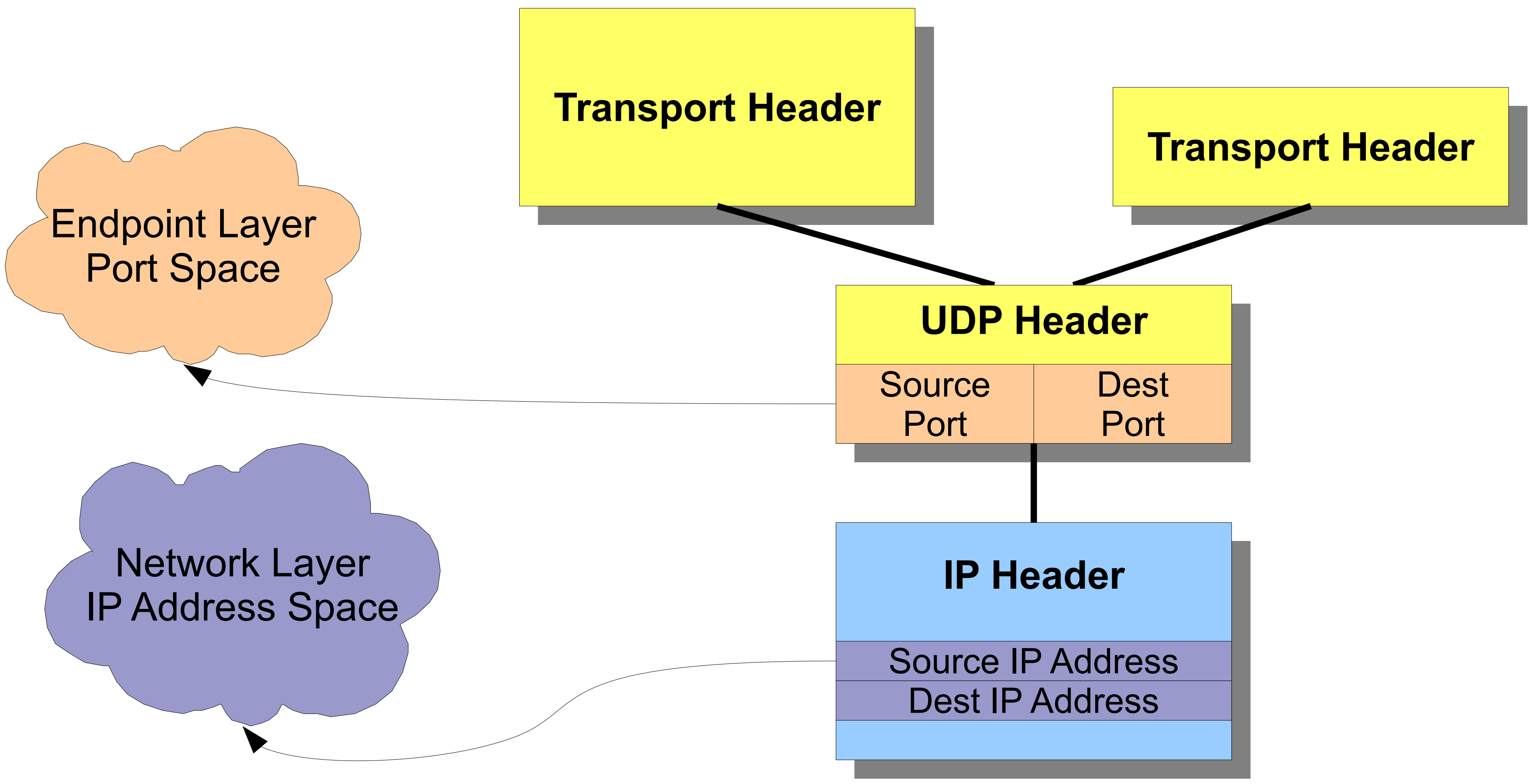
A Layering Solution

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**...

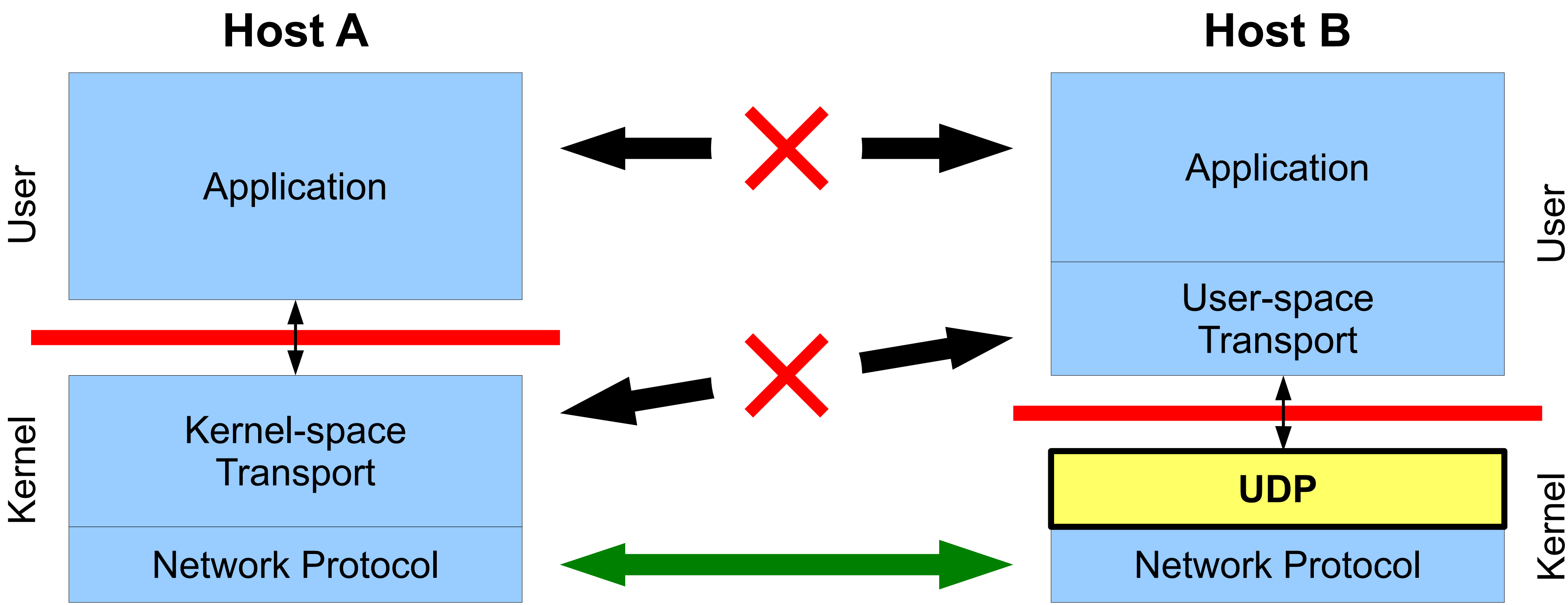
Practical Benefits

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 ...?

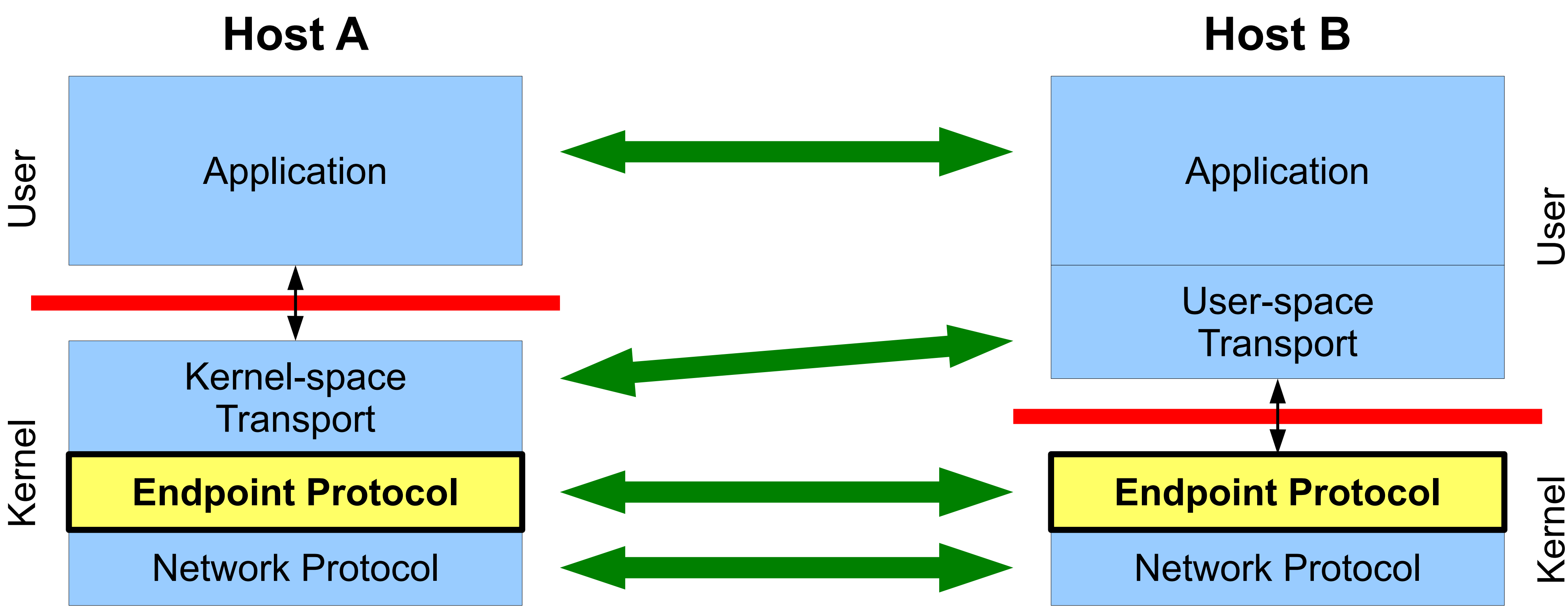
Kernel/User Transport **Non-**Interoperability

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

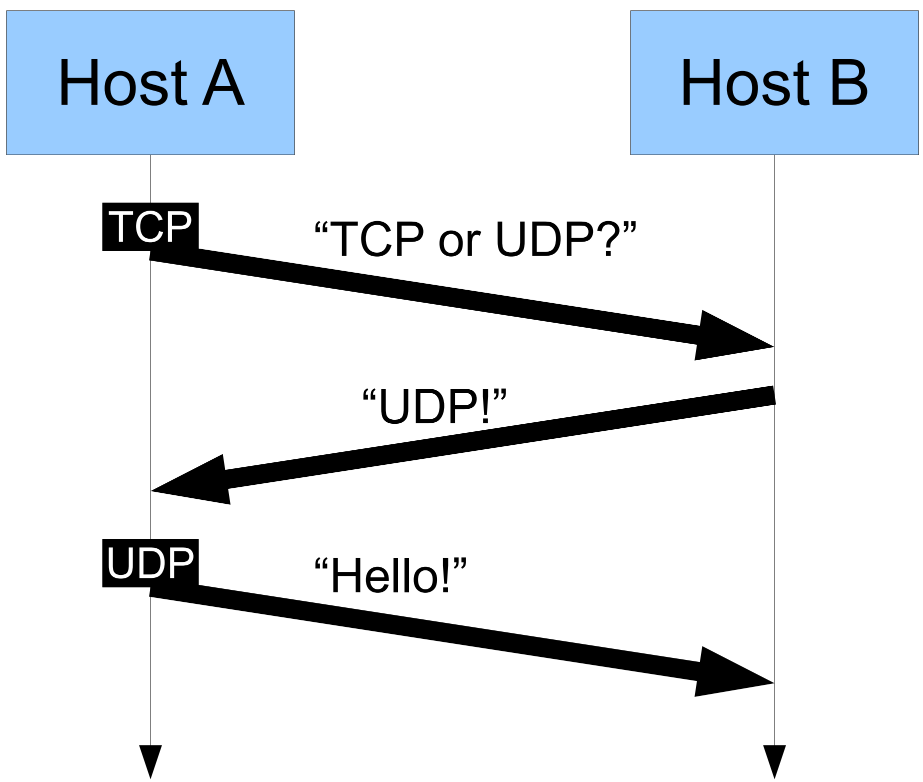
Endpoint layer provides **full interoperability**, user-space transports require **no special privileges**



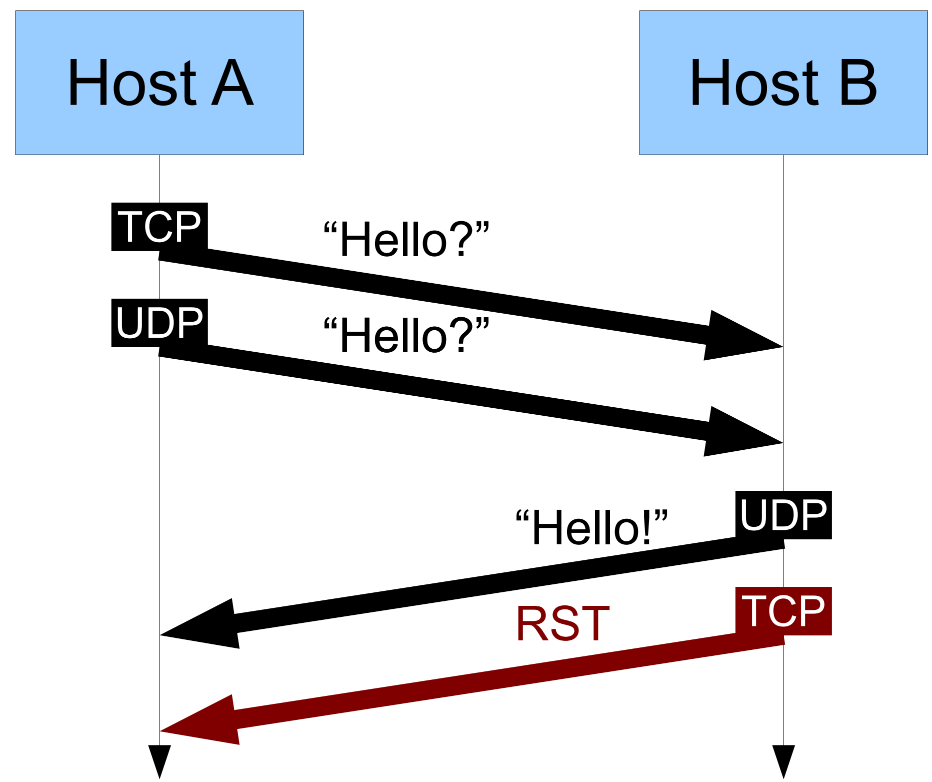
Transport Negotiation

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

"Cautious Negotiation"

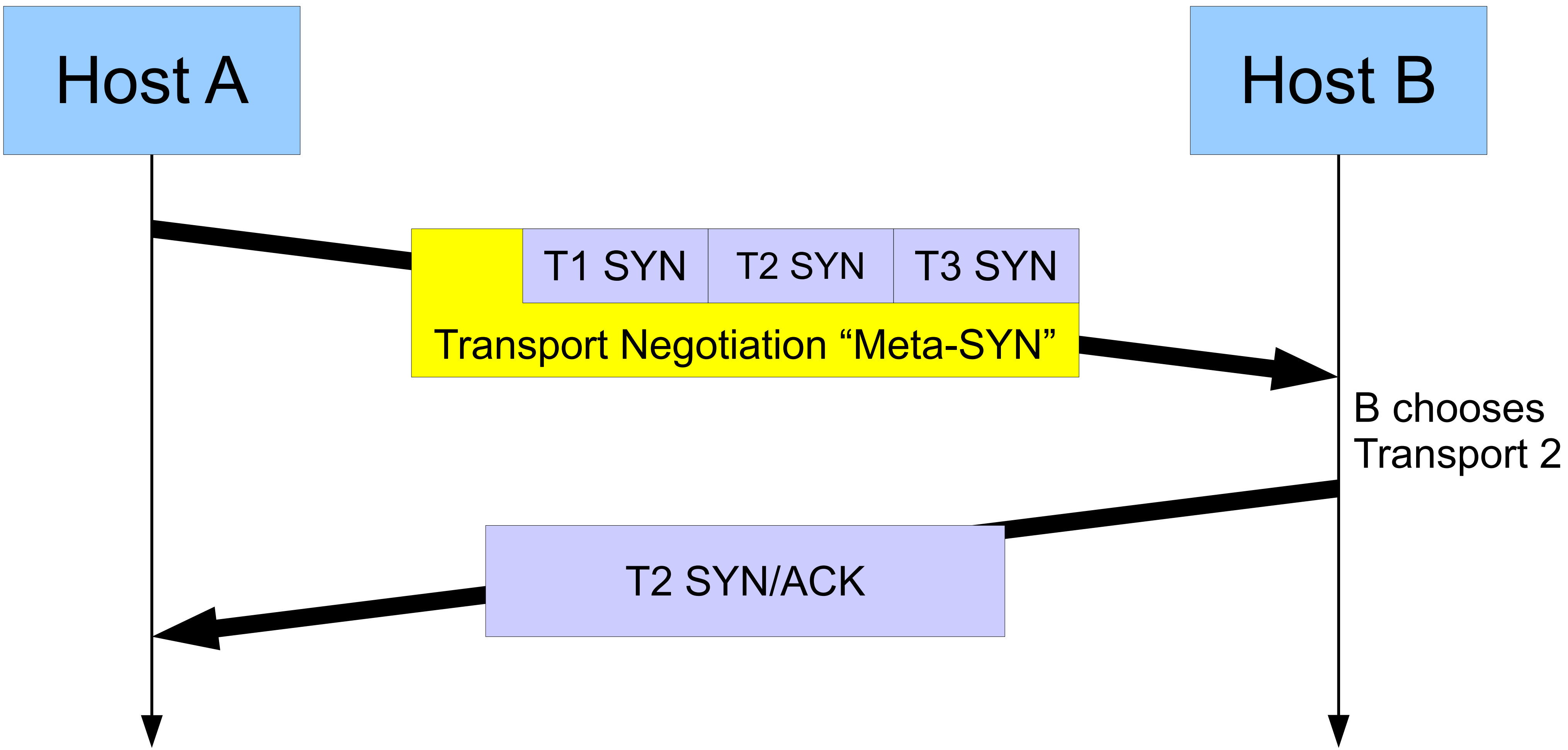


"Shotgun Negotiation"



"Zero-RTT" Transport Negotiation

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



Why the Network Will **Always** Need to See Endpoint Info

Imperfect world
\+ Administratively diverse network
\+ <u>Existence of attackers</u>
= **Need for Border Control**

Expecting middleboxes to enforce network policy
based only on IP address
is like expecting national border guards to ask only
what cities you're traveling to/from

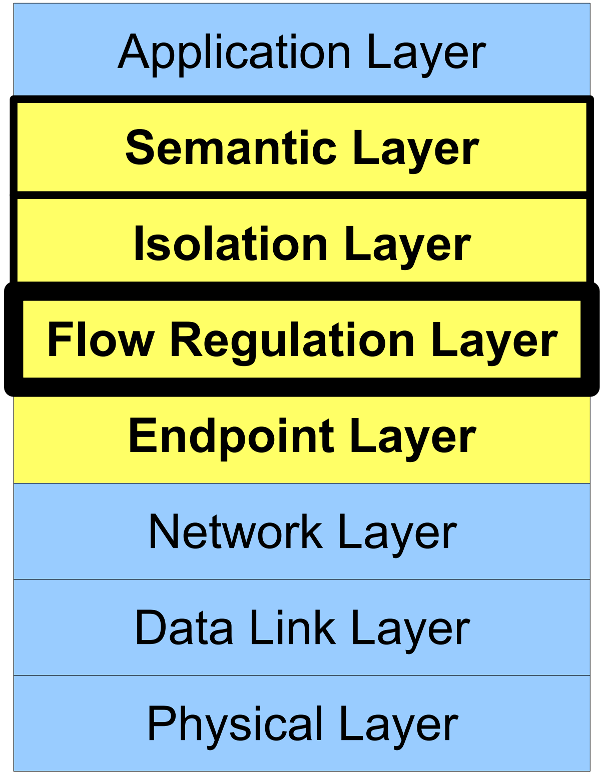
Further Endpoint Layer Evolution

"Next-Generation Endpoint Layer" needs to:

- Remain backward-compatible with UDP
  - Use same port space, fall back on UDP transparently
- Extend endpoints with more policy-relevant info
  - Port names [Touch 06], user names, service names, network customer accounts, ...
- Proactively advertise endpoints [Woodyatt-ALD]
  - Enable cleaner solutions to "NAT signaling" mess? [UPnP, NAT-PMP, MIDCOM, NSIS, …]
- Likely design inspiration: [TRIAD, NUTSS]

Flow Layer

performance tuning required at technology & administrative boundaries



Traditional "Flow Regulation"

Transports include end-to-end **congestion control**

- regulates flow transmission rate to network capacity

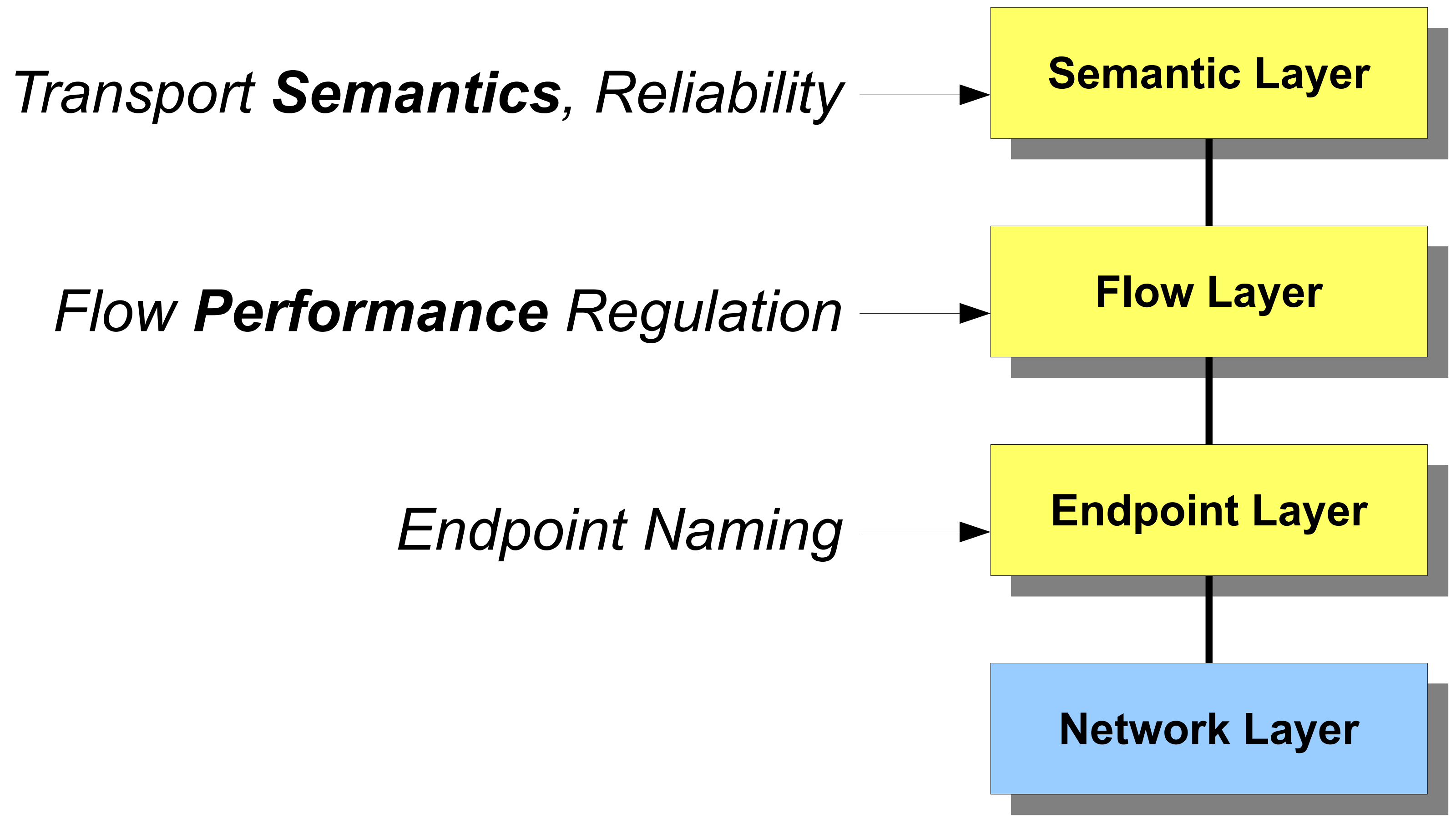
But one E2E path may cross **many**...

- 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]

A Layering Solution

Factor flow regulation into underlying **Flow Layer**

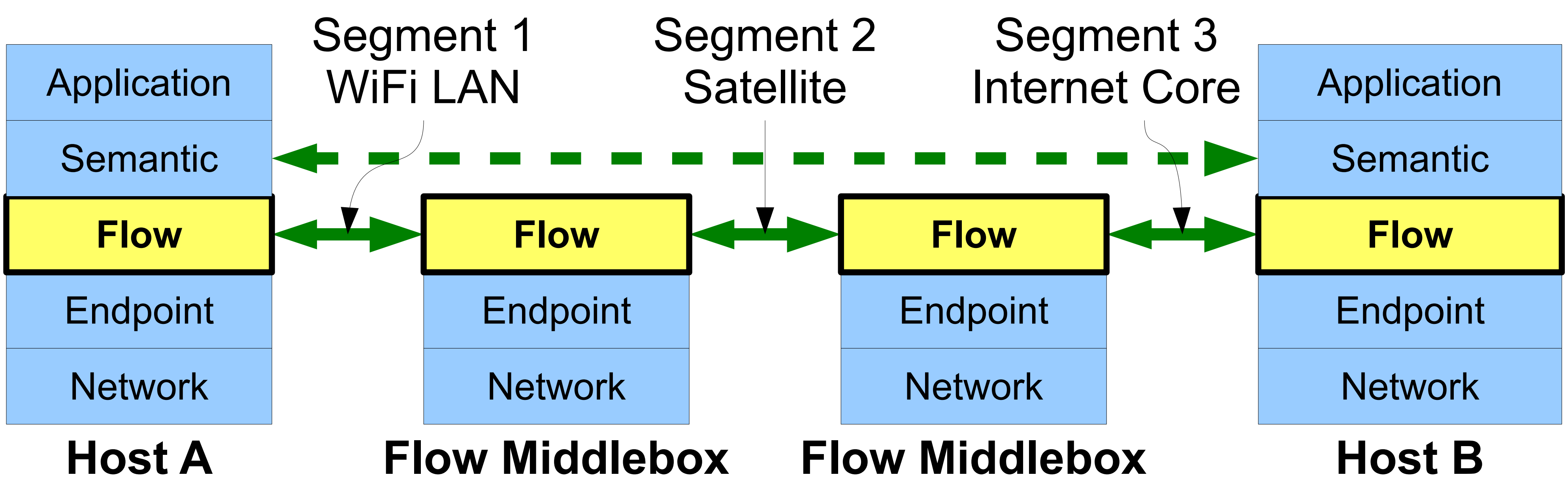


Main Practical Benefit

Can split E2E flow into separate CC *segments*

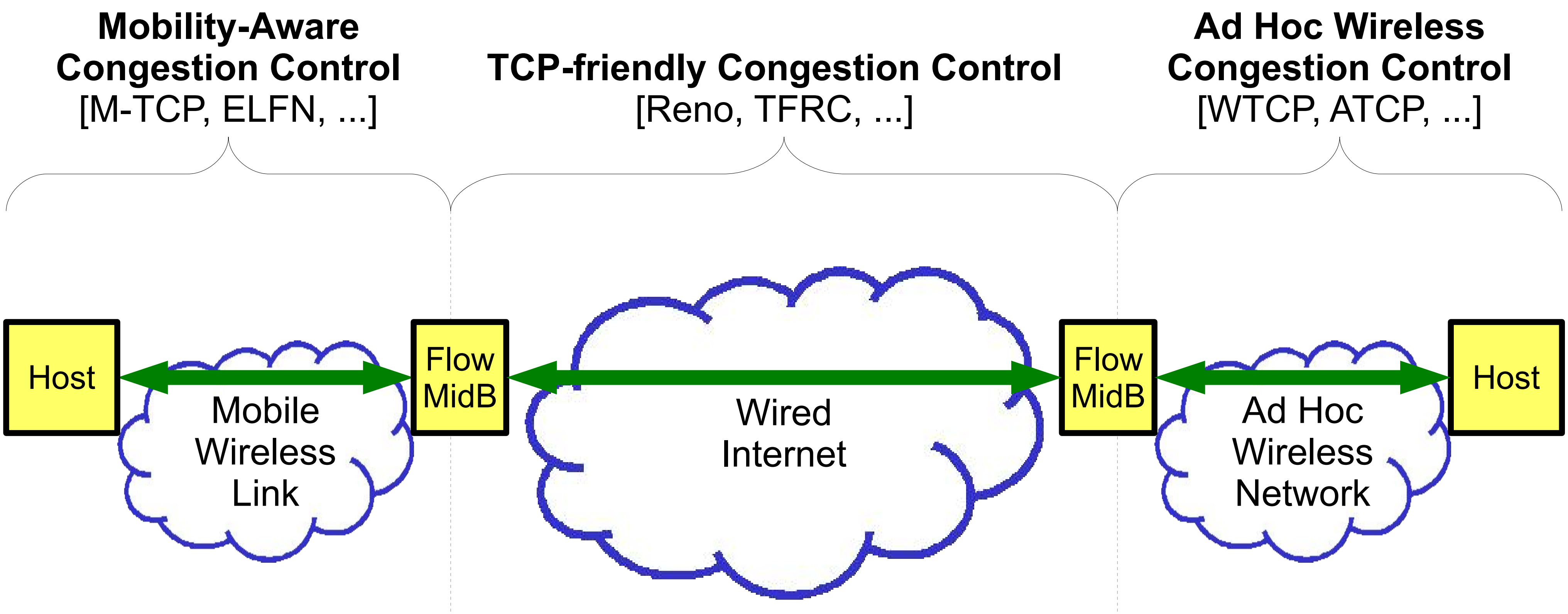
- Specialize CC algorithm to **network technology**
- Specialize CC algorithm within **admin domain**

… without interfering with E2E transport *semantics*!

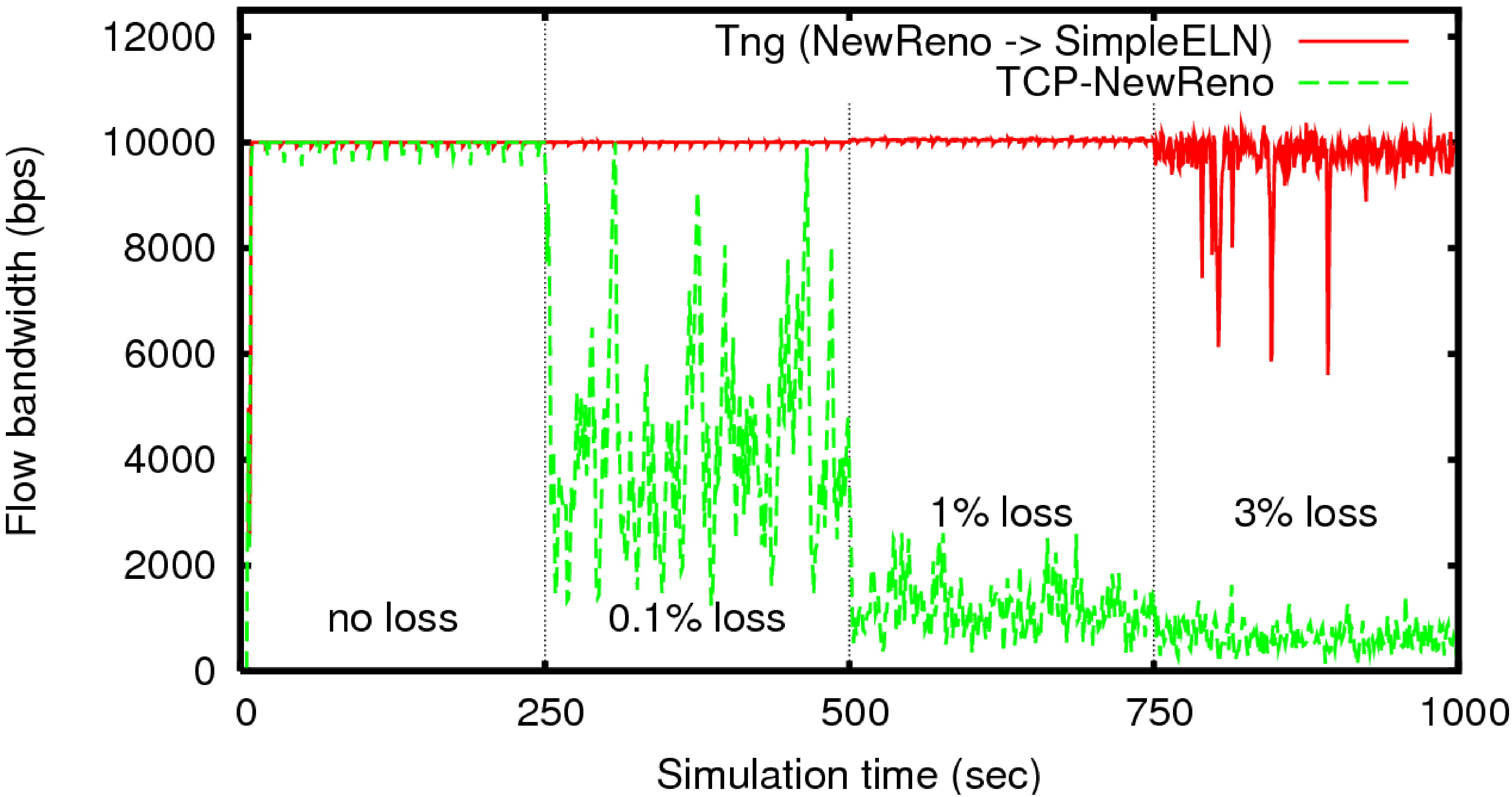


Example Scenarios

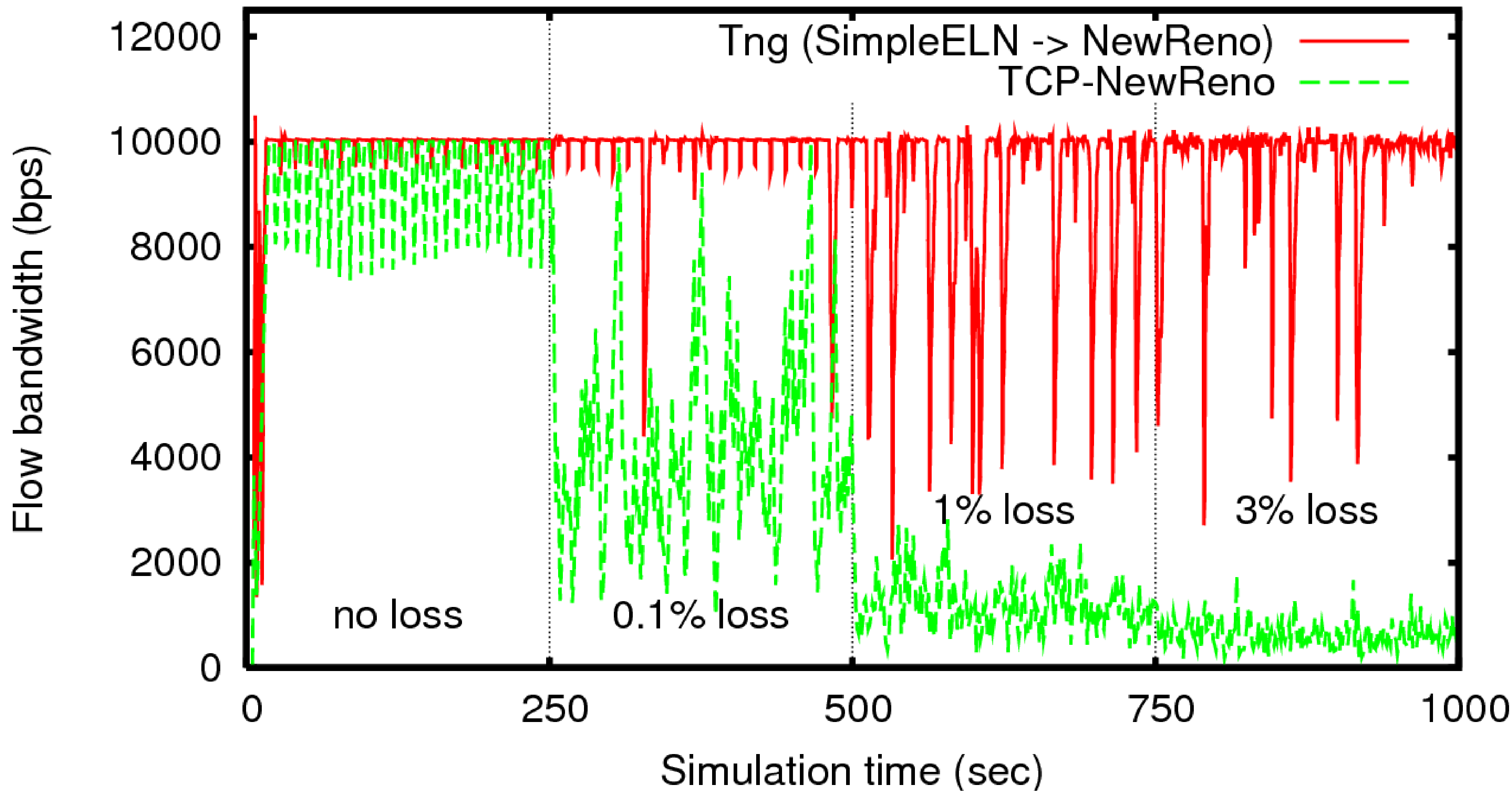
(1) Last-mile proxies for wireless/mobile links



Simulation: Download over Lossy Link

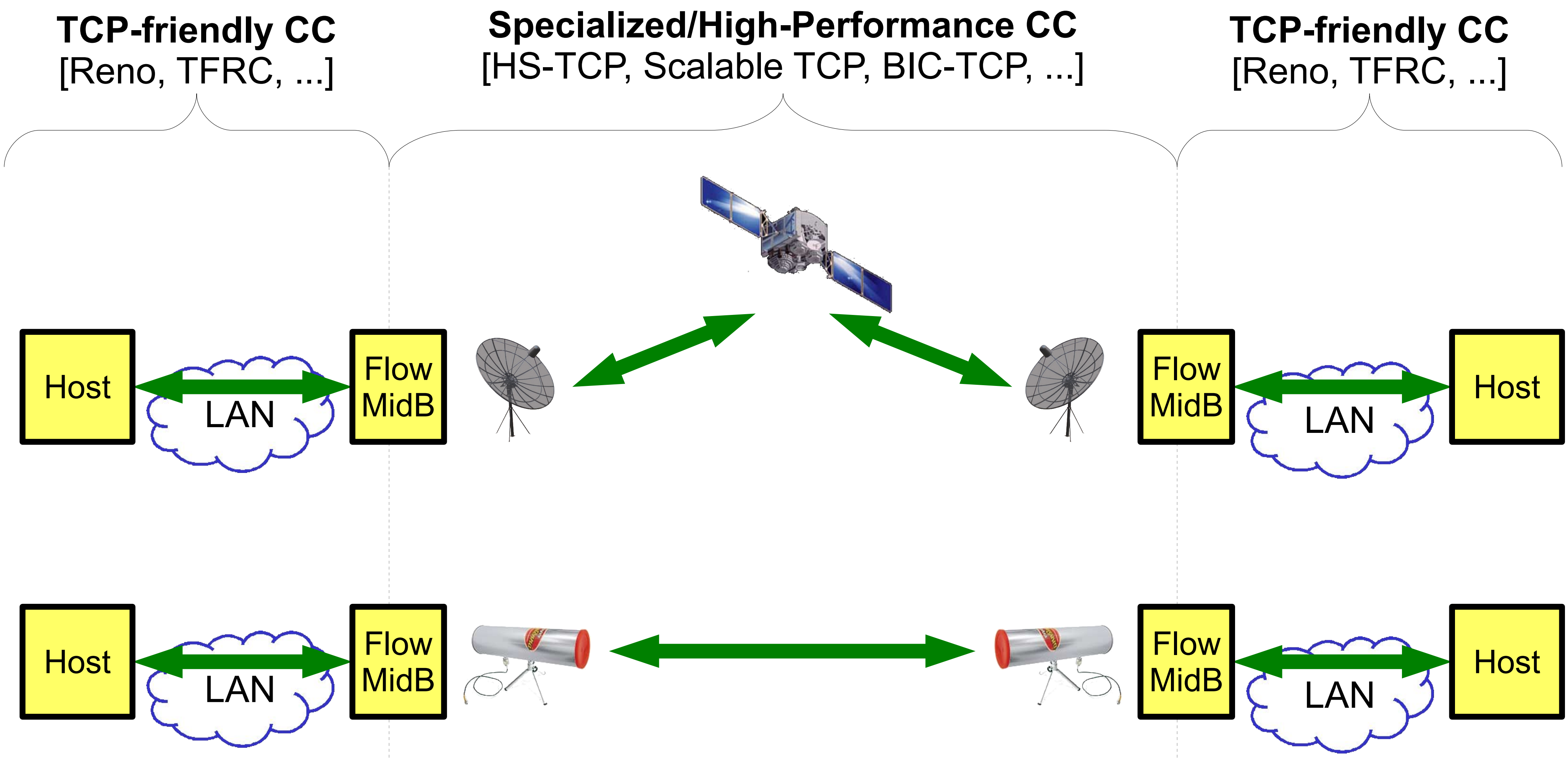


Simulation: Upload over Lossy Link

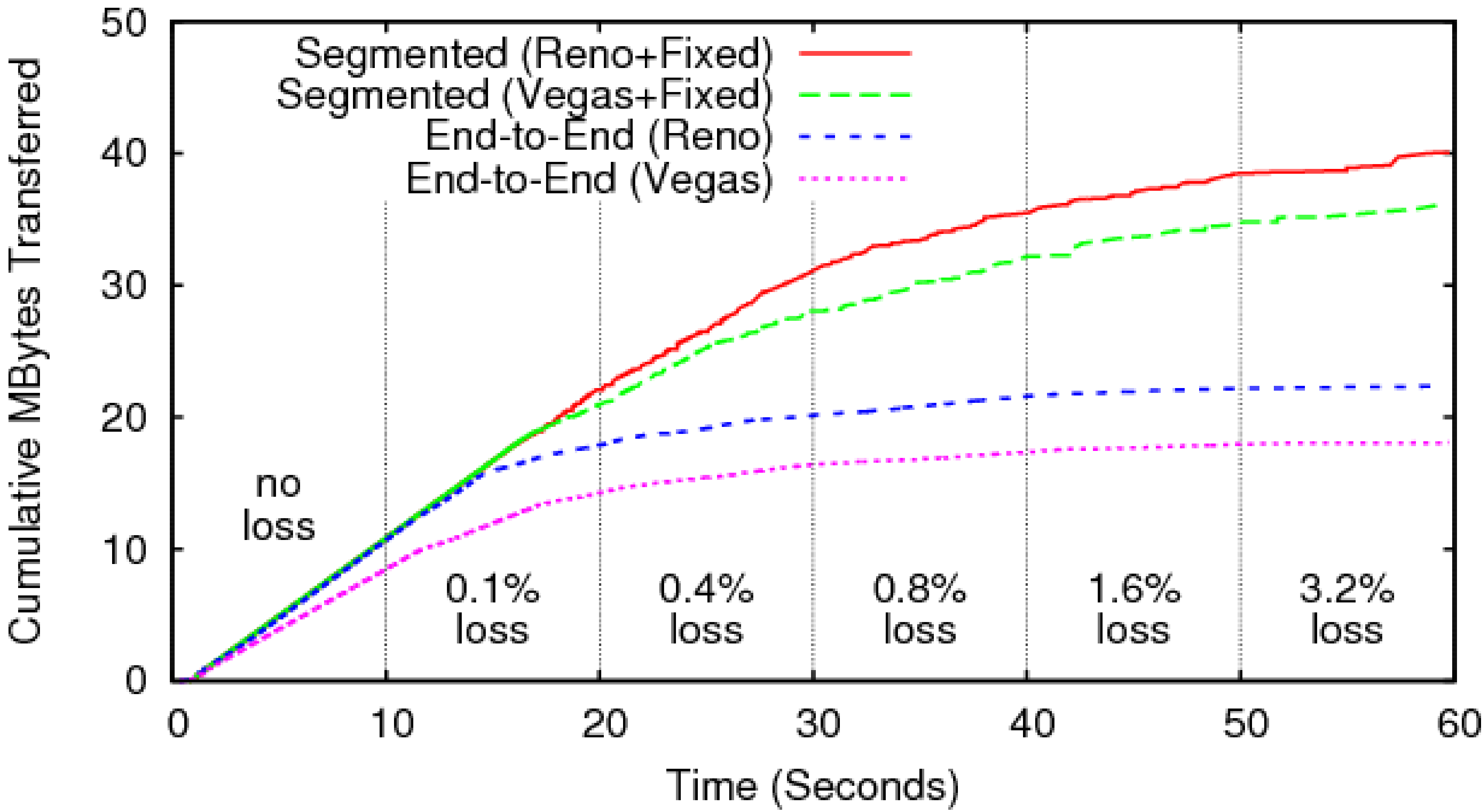


Example Scenarios

(2) Lossy Satellite or Long-Distance Wireless Links

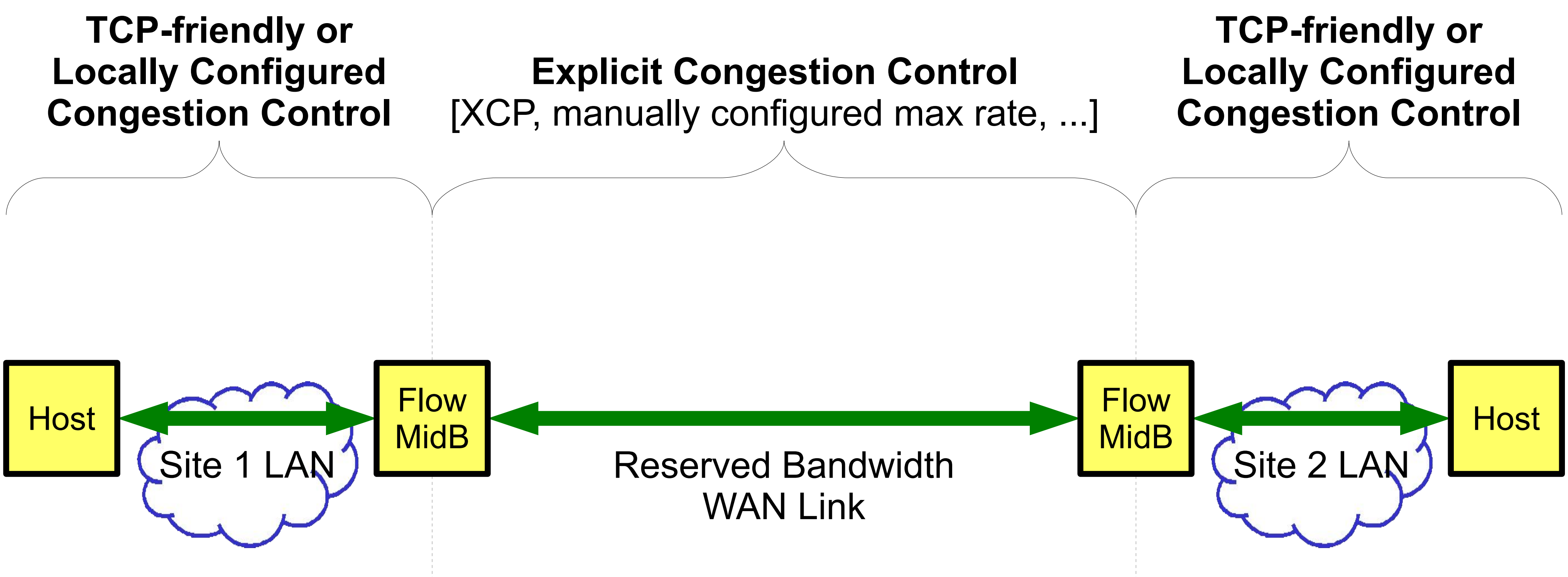


Simulation: Transfer over Satellite Link



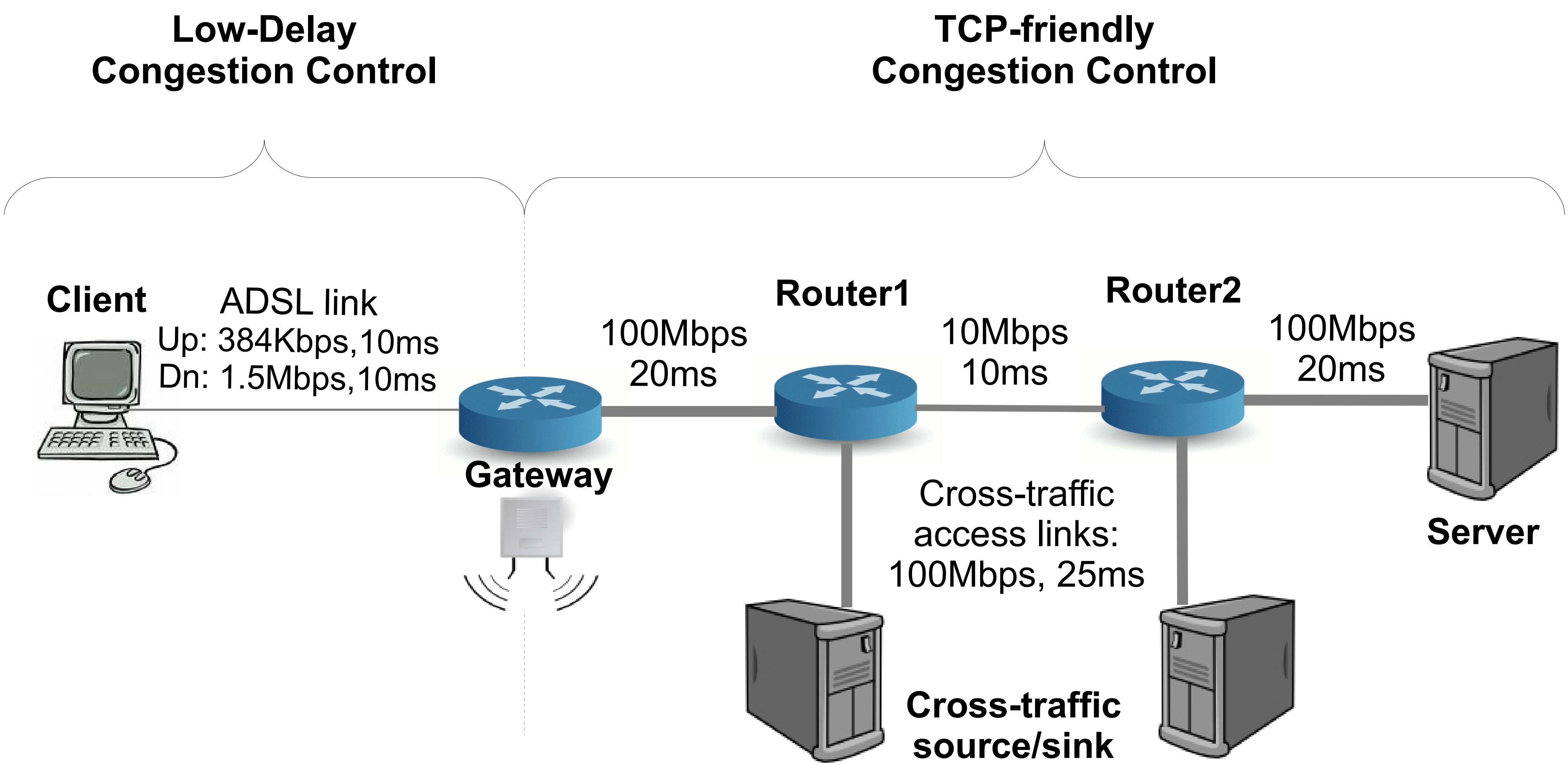
Example Scenarios

(3) Inter-Site WAN Links in Corporate Networks

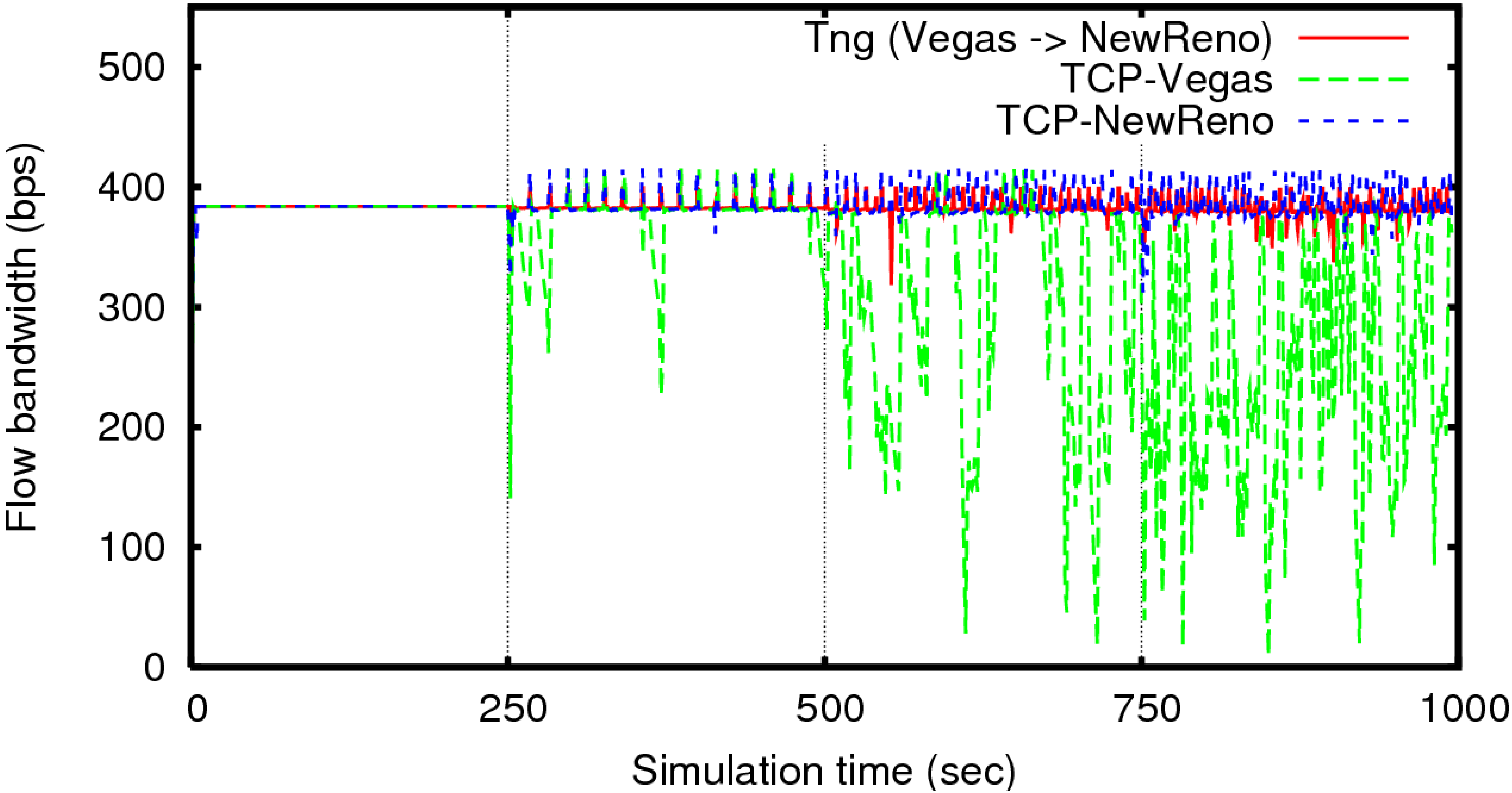


Example Scenarios

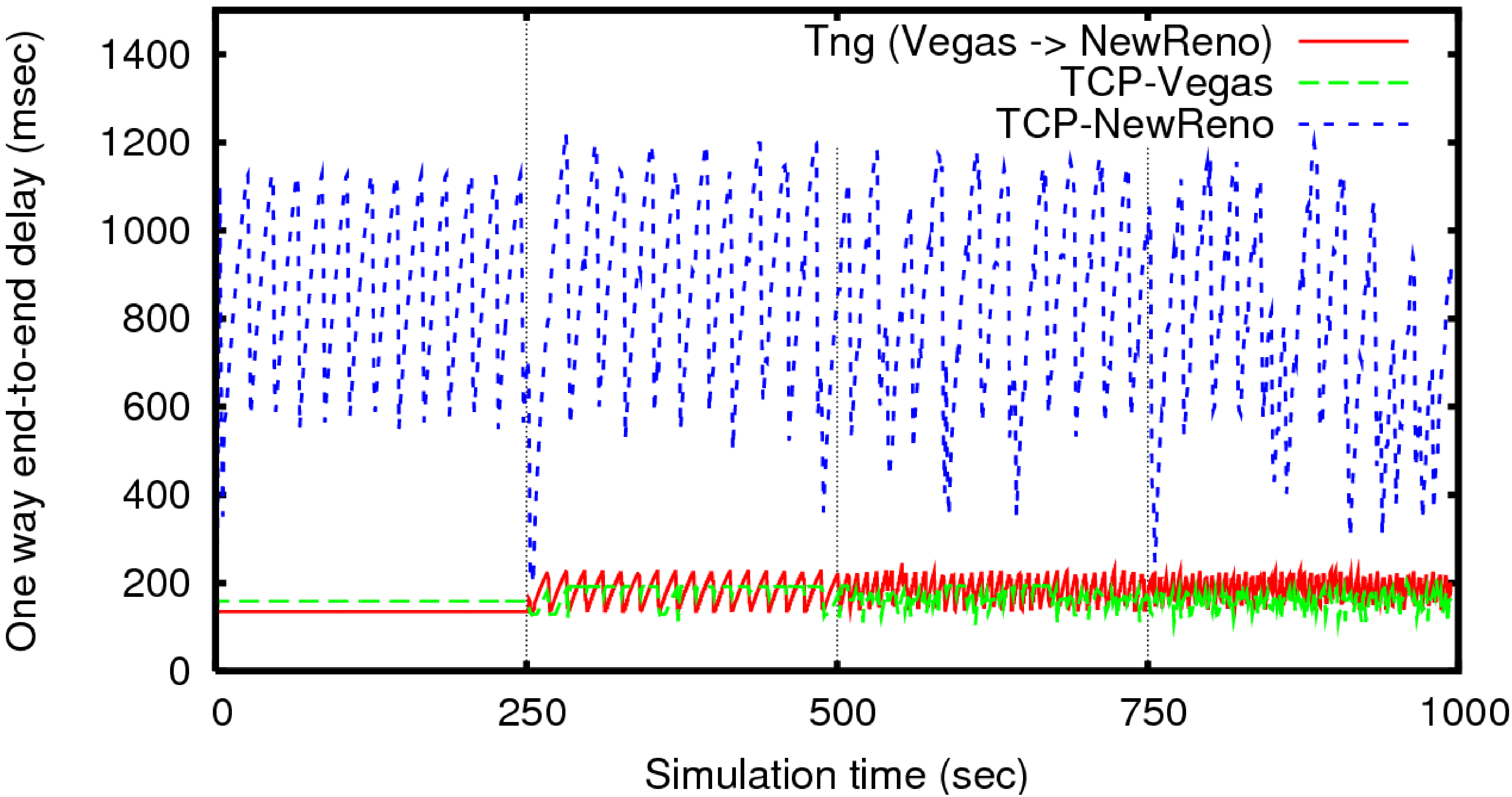
(4) Delay-Sensitive Use of DSL/Cable Links



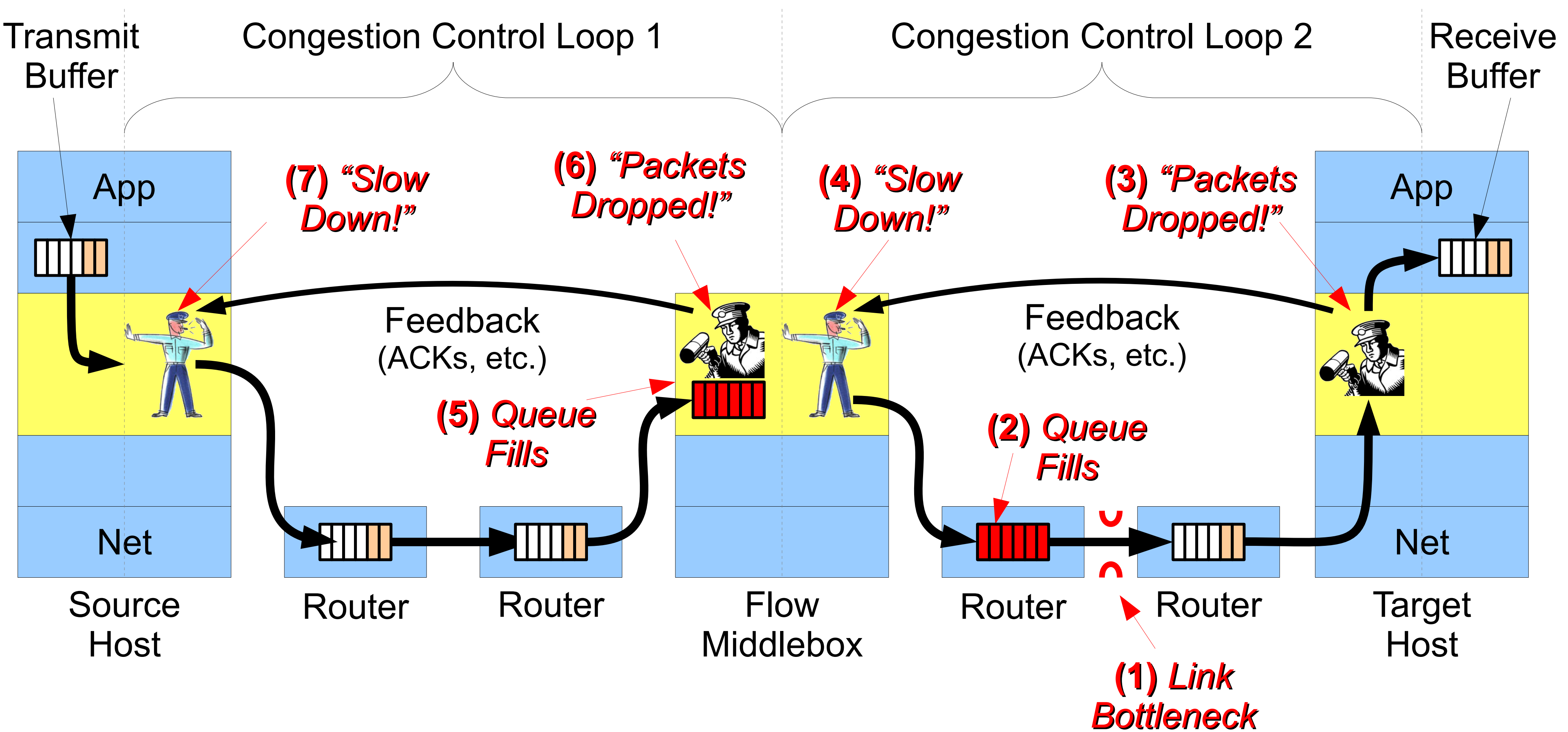
Simulation: DSL Upload – Bandwidth



Simulation: DSL Upload – Latency



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

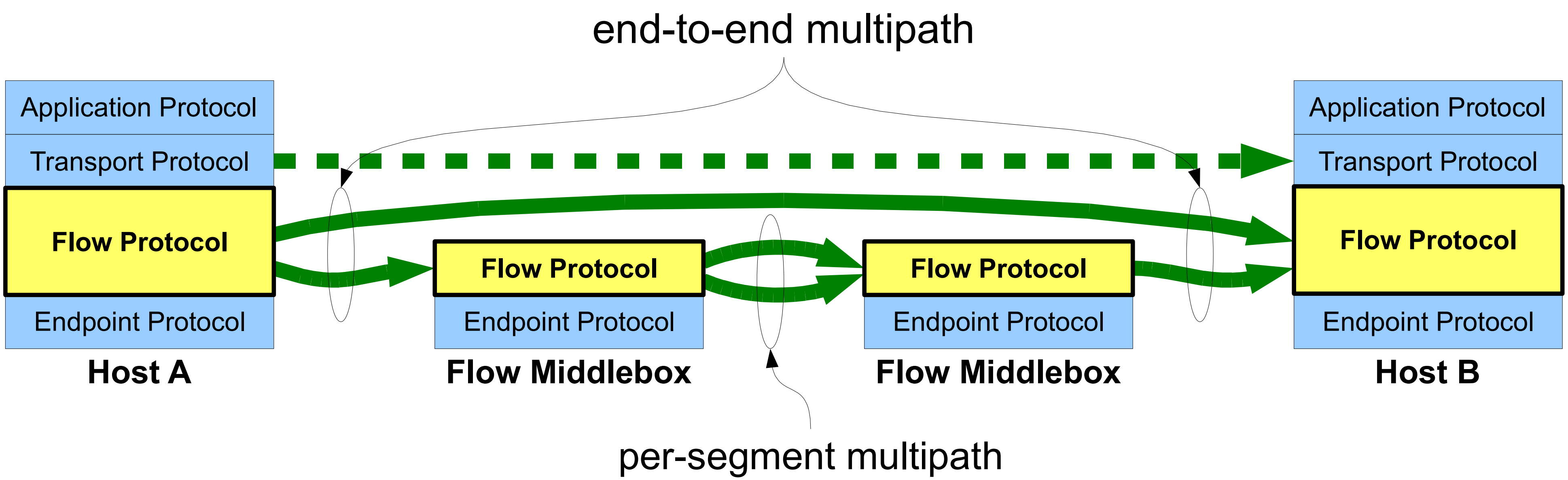


Other Practical Benefits (1/2)

Incrementally deploy performance enhancements

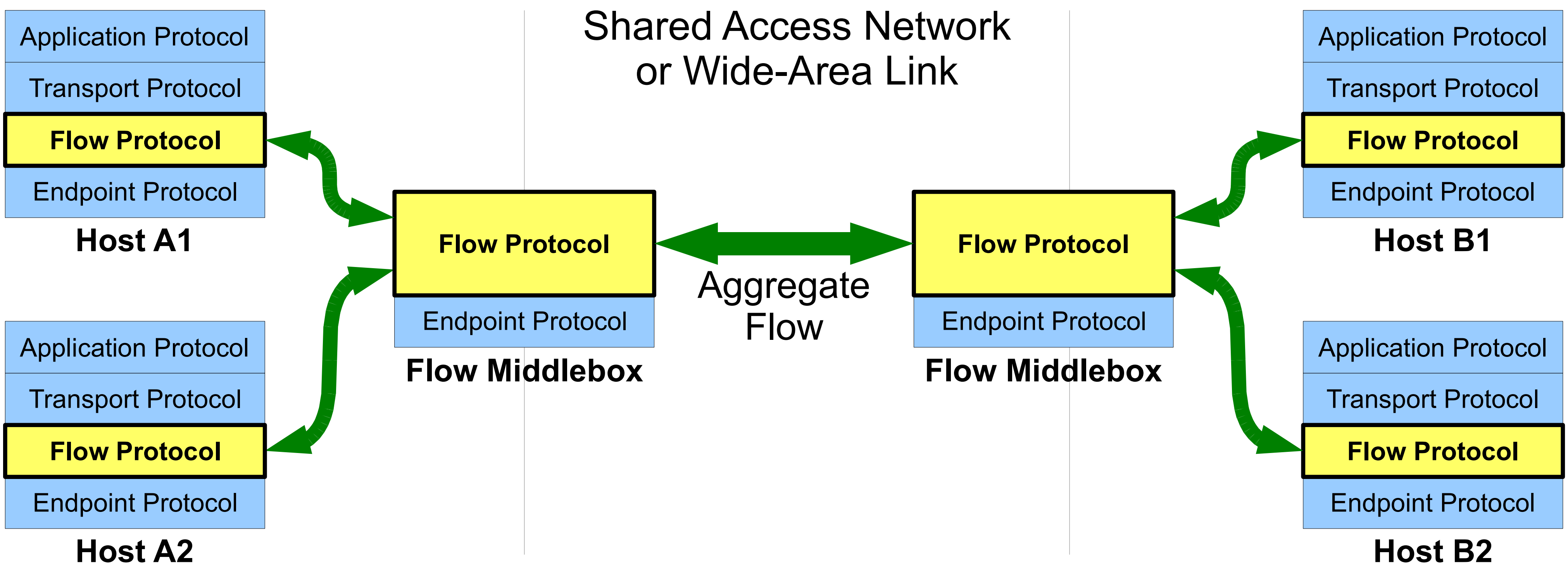
- multihoming [RFC 4960], multipath [Lee 01], dispersion [Gustafsson 97], aggregation [Seshan 97], ...

… without affecting E2E transport semantics!



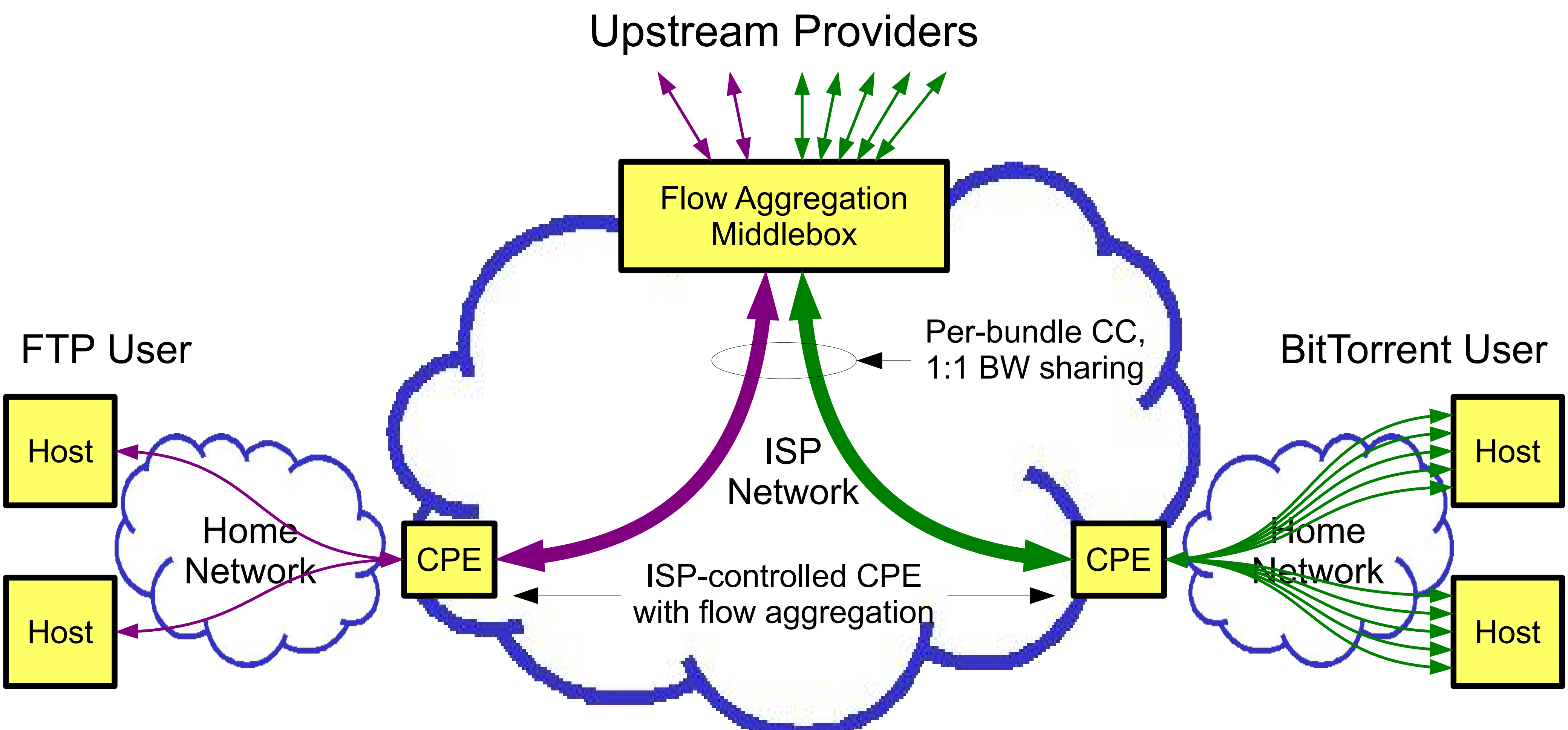
Other Practical Benefits (2/2)

- 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

- Several "starting points" exist:
  - Congestion Manager [Balakrishnan99]
  - DCCP [Kohler06]
    (just stop thinking of it as a "transport")
  - SST Channel Protocol [Ford07]

- Continuing work areas:
  - Support for flow middleboxes, path segmenting
  - Interfaces between (new) higher & lower layers

Will We **Always** Need a Flow Layer?

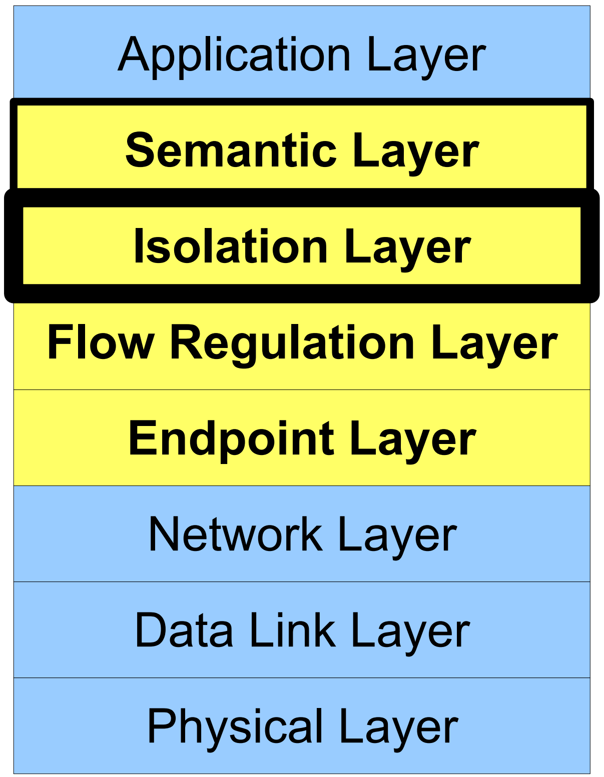
Not if everyone can agree on & universally deploy *one sufficiently powerful congestion control scheme*

- ECN? Re-ECN? XCP? RCP? …

Even if possible,
we'll need a flow layer to get there!

Isolation Layer

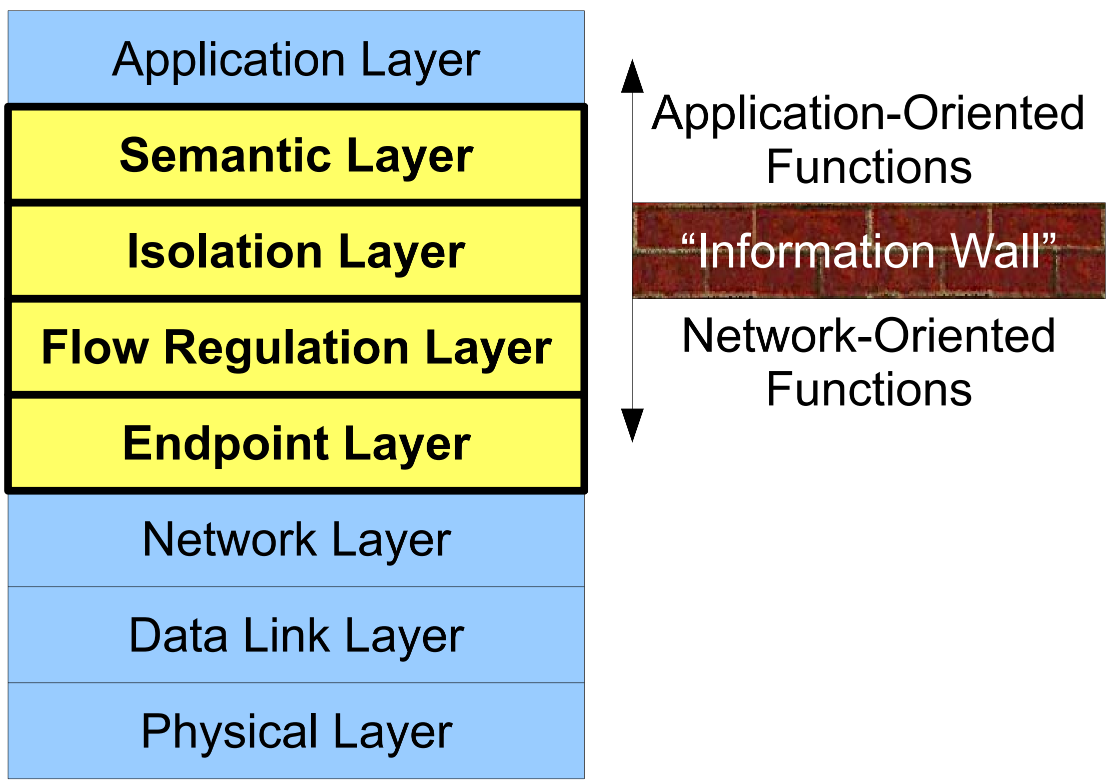
need clean, enforceable separation between apps & network



Purpose

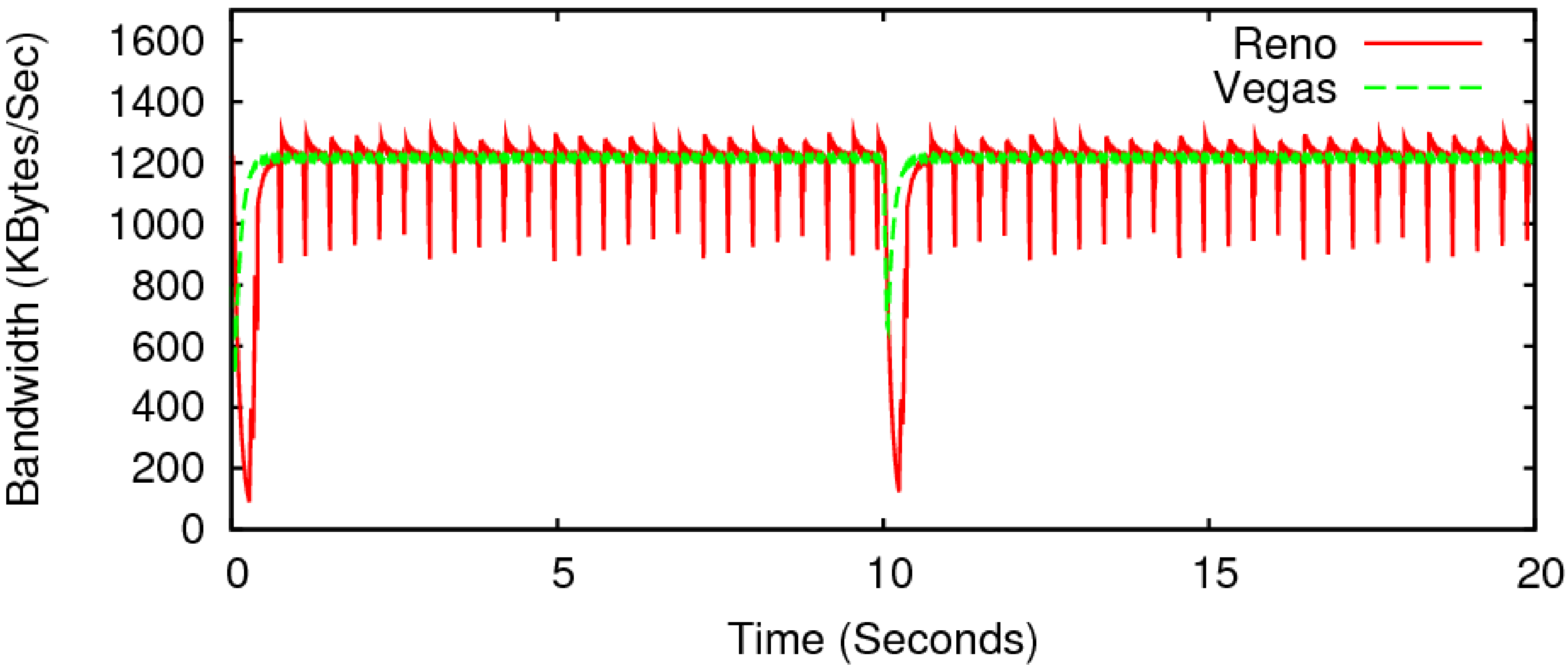
Isolate the E2E application flow from the network:

- By separating host identity from network location [HIP, UIA]
- By authenticating and encrypting E2E communication [IPsec]



Mobility Example

Mobile host starts file transfer at time 0,
IP address changes at 10 sec.



Architectural Novelty

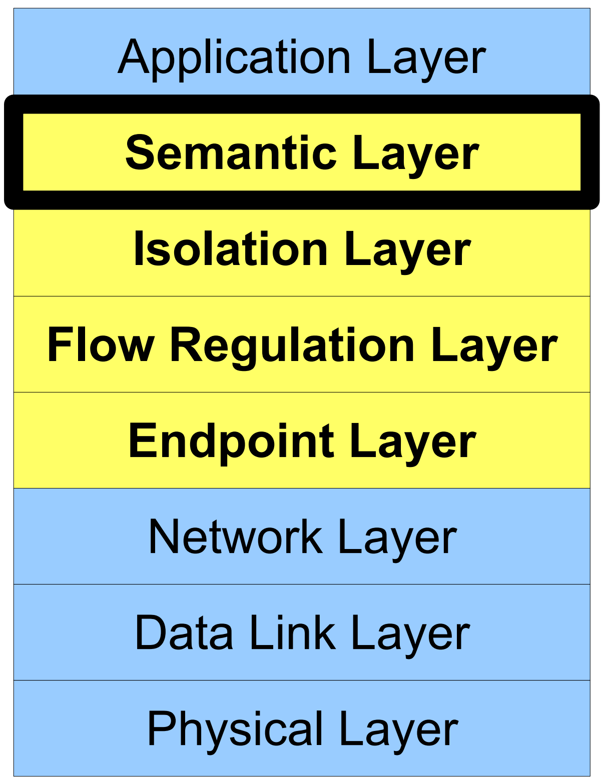
What's new about this?

Nothing about the isolation mechanisms themselves...
only there's finally **a clean place to put them!**

- **Above network-oriented functions:**
  doesn't interfere with firewalls, NATs, PEPs
- **Below application-oriented functions:**
  still transparent to applications like IPsec,
  doesn't require rewriting for each transport [DTLS]
  & integrating into each application

Semantic Layer

E2E reliability & semantics is purely app-driven



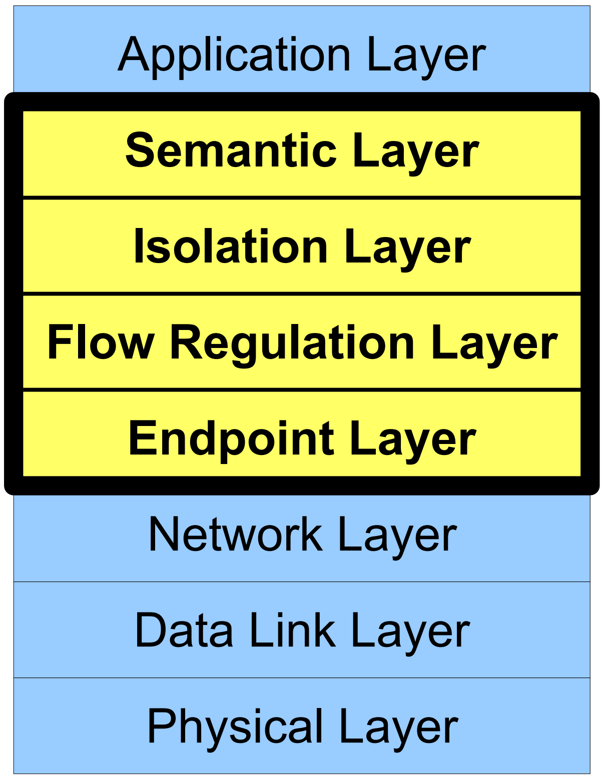
Semantic Layer

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
  - ...

Putting It All Together

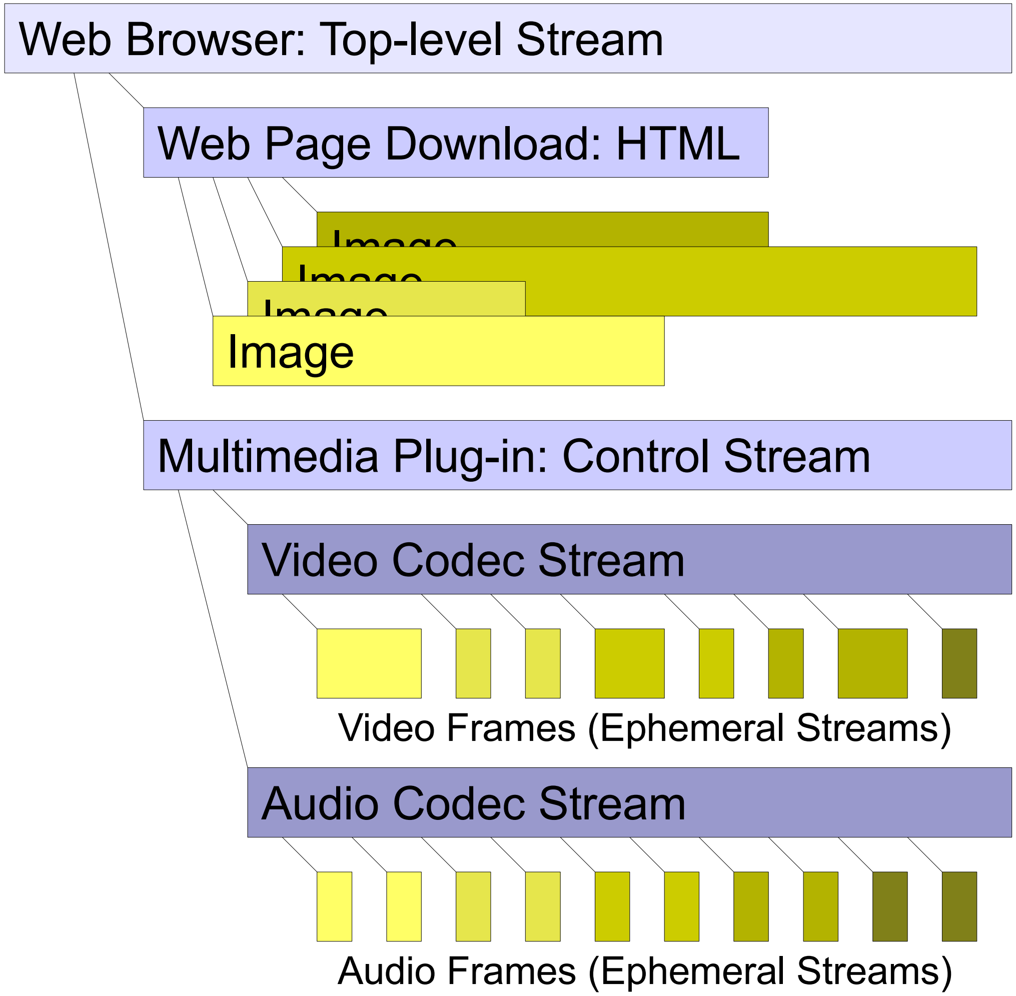
So how to build it?



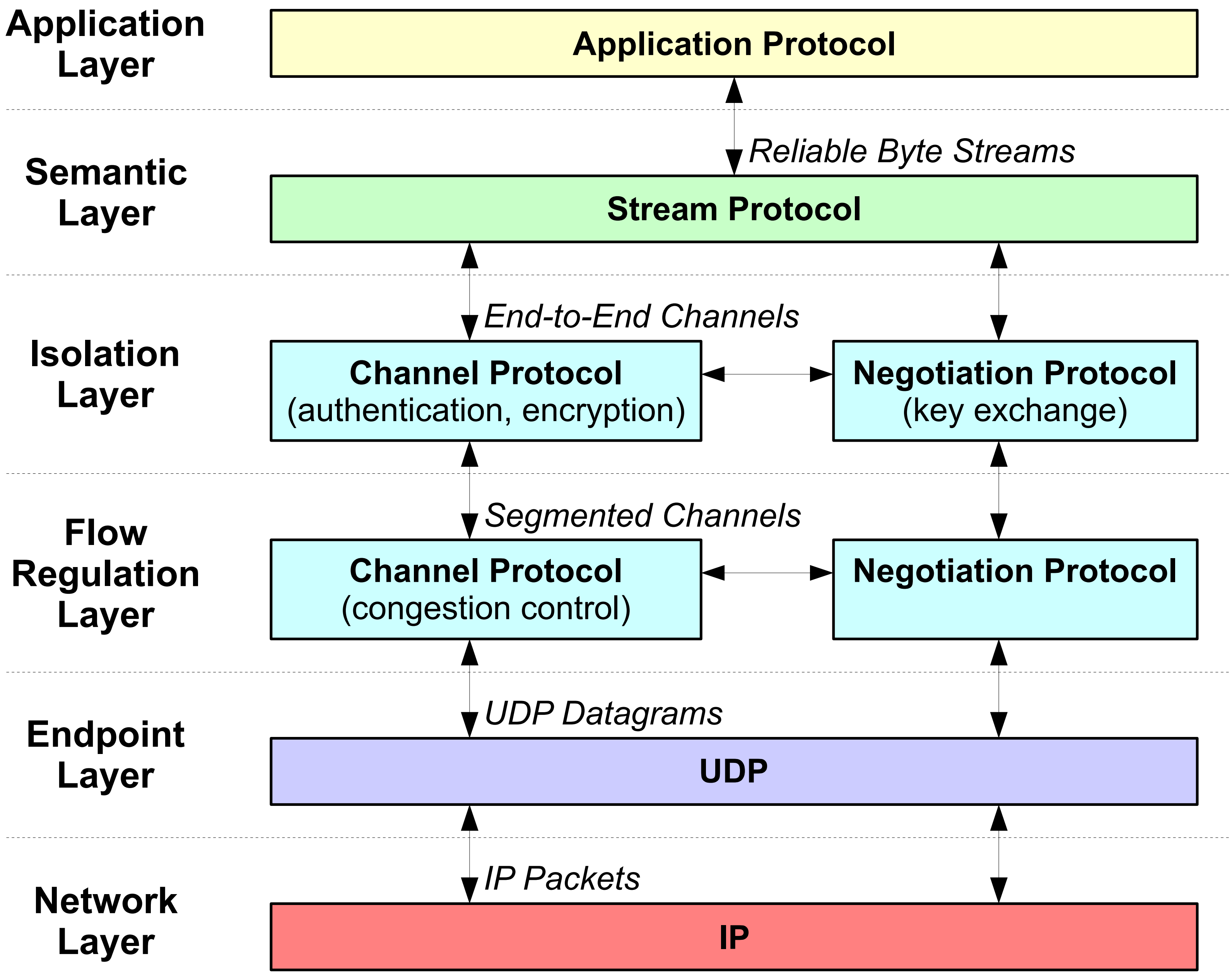
Working “Clean-Slate” Prototype

Based on **Structured Stream Transport** (SIGCOMM '07)

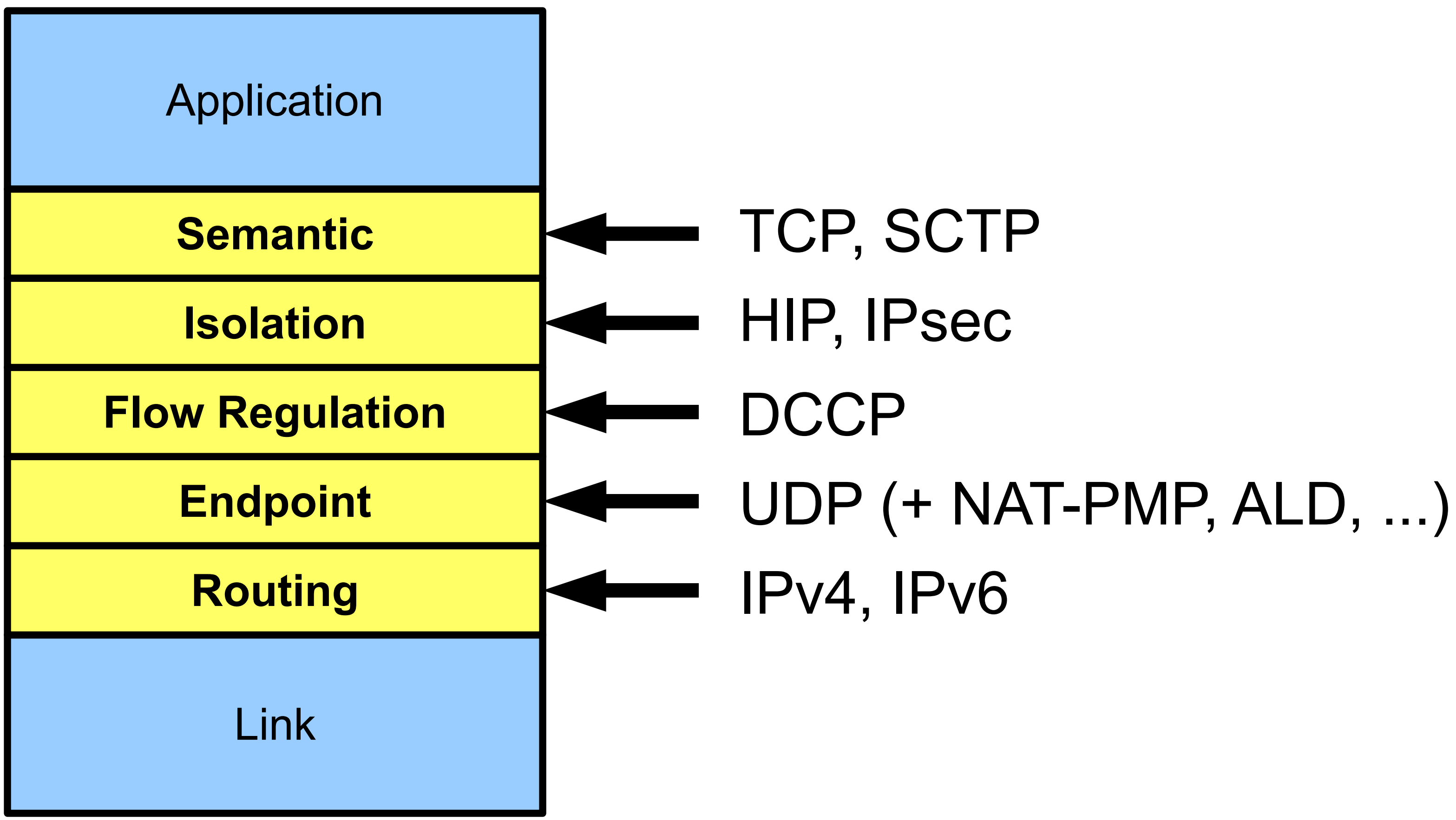
- TCP semantics
- Fast “stream fork”
- Fine-grained streams (e.g., per-transaction)
- Datagrams as streams



Prototype Structure



Alternative Structure, Reusing Existing Protocols



Per-Packet Header Overhead, Estimated Code Size Complexity

Current SST Prototype vs Equivalent Linux Protocols

– C++ vs C, prototype vs mature – *not a fair comparison!*

Layer	Protocols		Header Size		Code Size	
	SST	Legacy	SST	Legacy	SST	Legacy
Semantic	Stream	TCP	8	20	1600	5300
Isolation	Channel	ESP	24	32	930	5300
Flow	Channel	DCCP	12	16		2900
Endpoint	UDP	UDP	8	8	600	600
Total			52	76	3130	14100

Incremental Deployment

- XXX need more explicit story here

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 its functions:

- Endpoint naming/routing into separate **Endpoint Layer**
- Flow regulation into separate **Flow Layer**
- Place E2E security functions in **Isolation Layer**
- Place 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