Flow Splitting in Tng, a Next-Generation Transport Architecture

Bryan Ford

Max Planck Institute for Software Systems and Yale University

baford@mpi-sws.org

Janardhan Iyengar

Franklin & Marshall College

jiyengar@fandm.edu

http://bford.info/tng/

Presentation for IETF 75 – July 27, 2009

Relevant Documents

Papers/Drafts:

"Breaking Up the Transport Logjam"

– HotNets '08: *http://bford.info/pub/net/logjam.pdf*

"Flow Splitting with Fate Sharing"

– Research draft: *http://bford.info/pub/net/flowsplit.pdf*

"A Next Generation Transport Services Architecture"

– Internet-Draft: *draft-iyengar-ford-tng-00.txt*

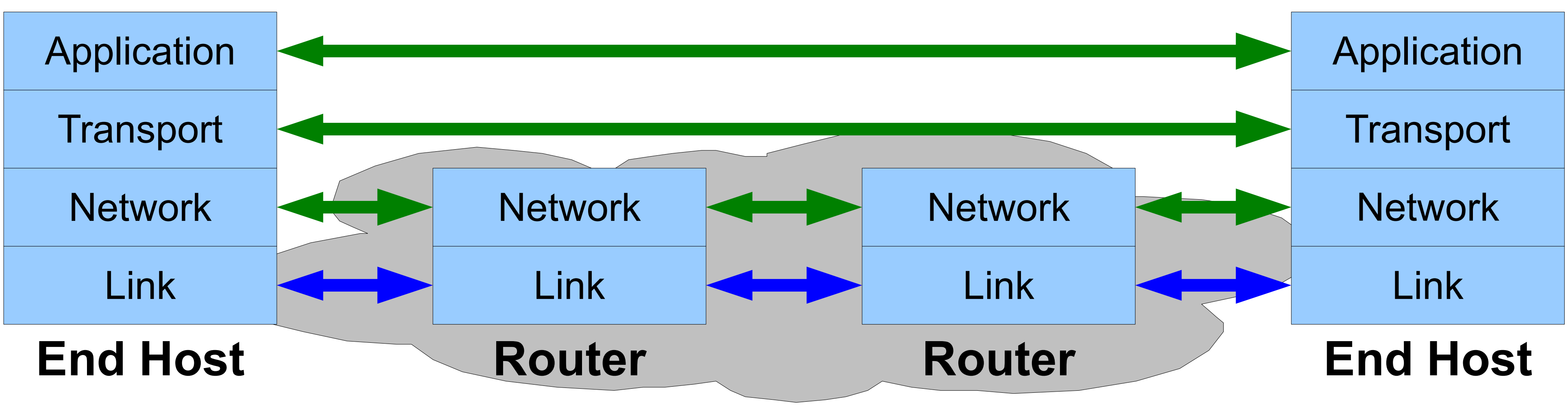
(Current) Project Web Page:

– http://bford.info/tng/

The End-to-End Principle

In TCP/IP's original design, **only the end hosts**

- see past a packet's Network Layer (IP) header
  - **Generality:** network carries *any payload*
- maintain "hard state" whose loss visibly impacts the user
  - **Fate Sharing:** transports retransmit E2E, can recover from failures in intermediate nodes



The Rise of the Middle

Internet scaling and diversity have led operators to place ever more **intelligence in the middle**

- **Firewalls:** enforce network access policies
- **Traffic shapers:** manage network bandwidth & delay
- **Network Address Translators (NATs):** alleviate IPv4 address scarcity by sharing IP addresses
- **Performance enhancing proxies (PEPs):** optimize performance in problematic situations, e.g., high-speed, high-delay, or wireless links [RFC3135]

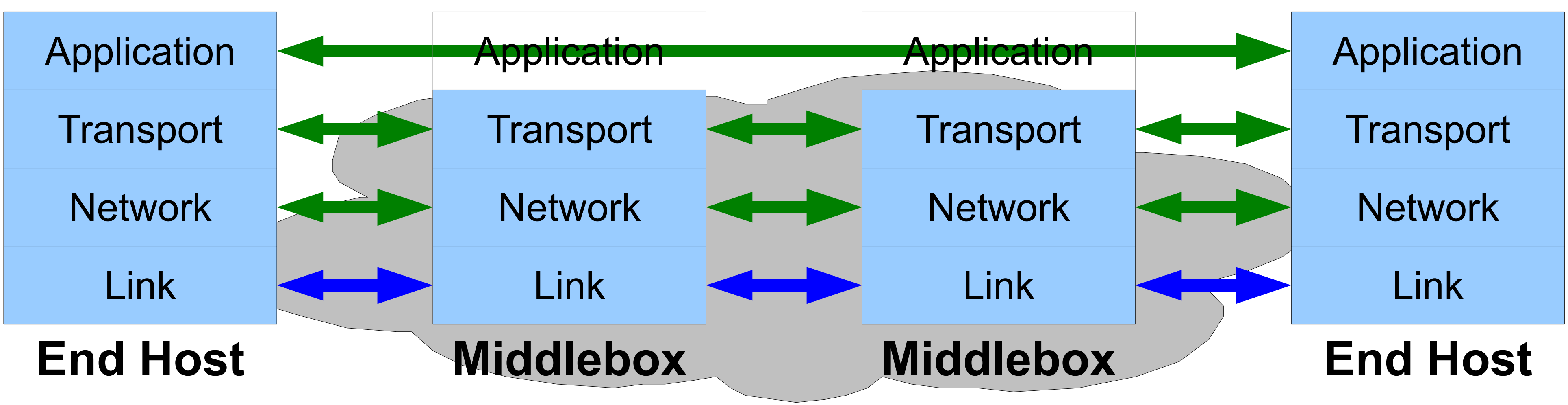
This Talk's Focus

Eroding End-to-Endness of Transports

Middleboxes **need to** interact with Transport Layer

- **Firewalls, traffic shapers:** to differentiate between applications via TCP/UDP port numbers
- **NATs:** to modify IP addresses & port numbers
- **PEPs:** to monitor & affect TCP congestion control

Result: the Transport Layer is no longer "End-to-End"

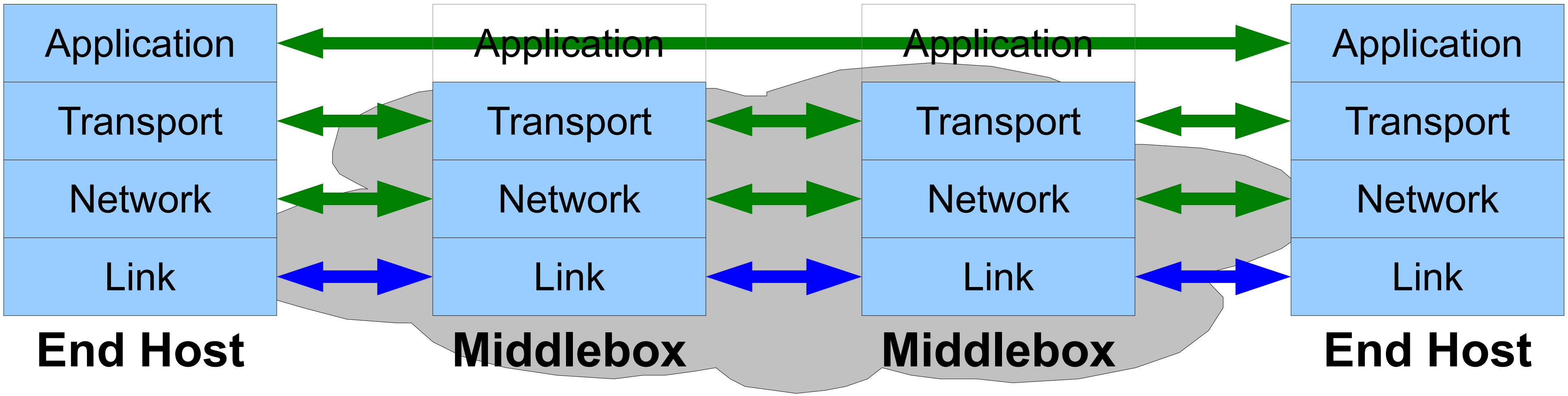


The Transport Layer's Lost Purity

Along with transport end-to-endness, we also lose:

- **Generality:** new transports can't pass → undeployable
- **Fate sharing:** middlebox failures → hard TCP failures
- **Security:** can't use transport-neutral security (IPsec)

Transports are still *designed to*, but now *fail to*, provide reliable end-to-end communication services

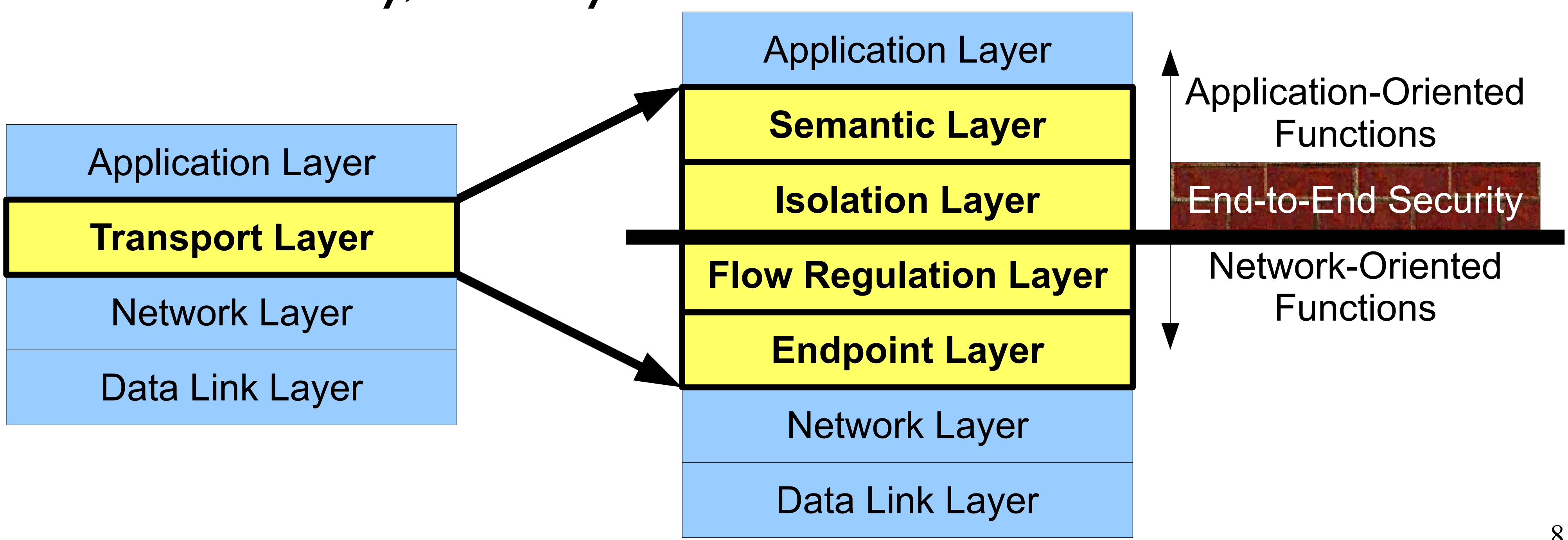


The Transport Layer is Stuck in an Evolutionary Logjam!

Tng: *Transport next-generation*

Refactor transport layer to match reality

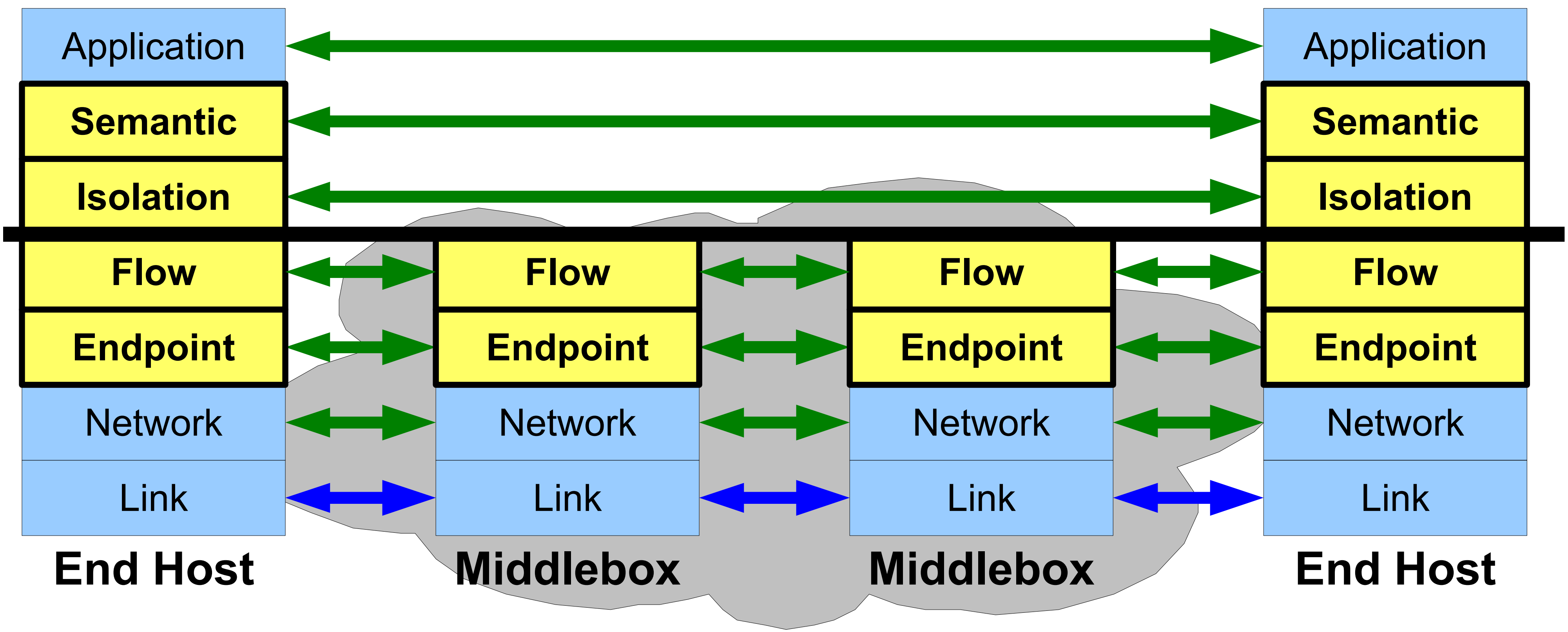
- **Network-oriented** functions of interest to middleboxes
  - Endpoints (ports); flow regulation (congestion control)
- **Application-oriented** functions serving the endpoints
  - Reliability, security



End/Middle Coexistence

Tng's Key Benefit: enable middleboxes to

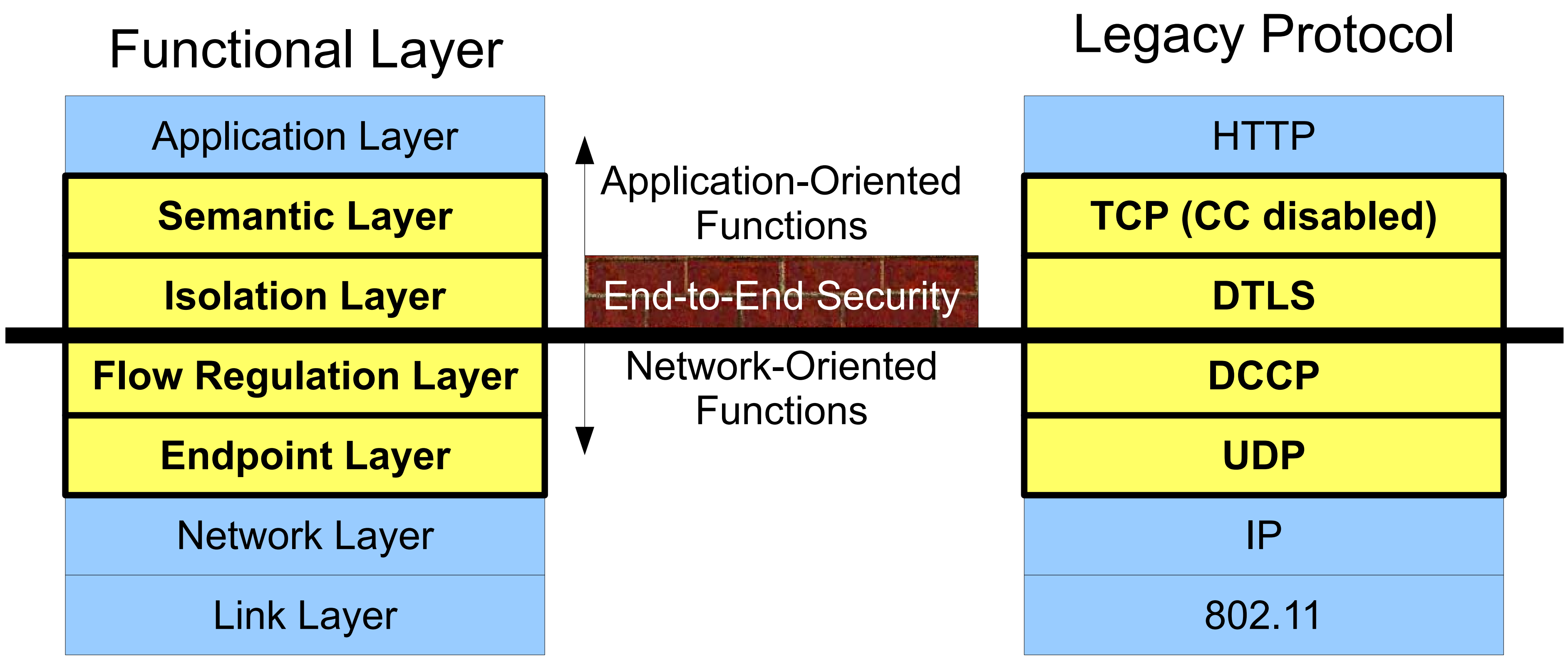
- interact cleanly with network-oriented functions
- avoid interfering with E2E application-oriented functions



Example Tng Protocol Stack

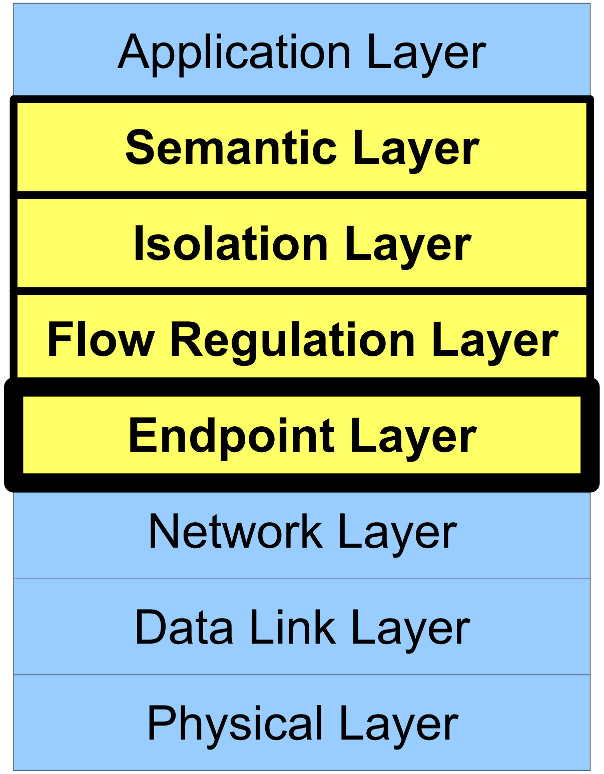
Can implement Tng using only "legacy" protocols

– Workable design; not ideal in function or efficiency



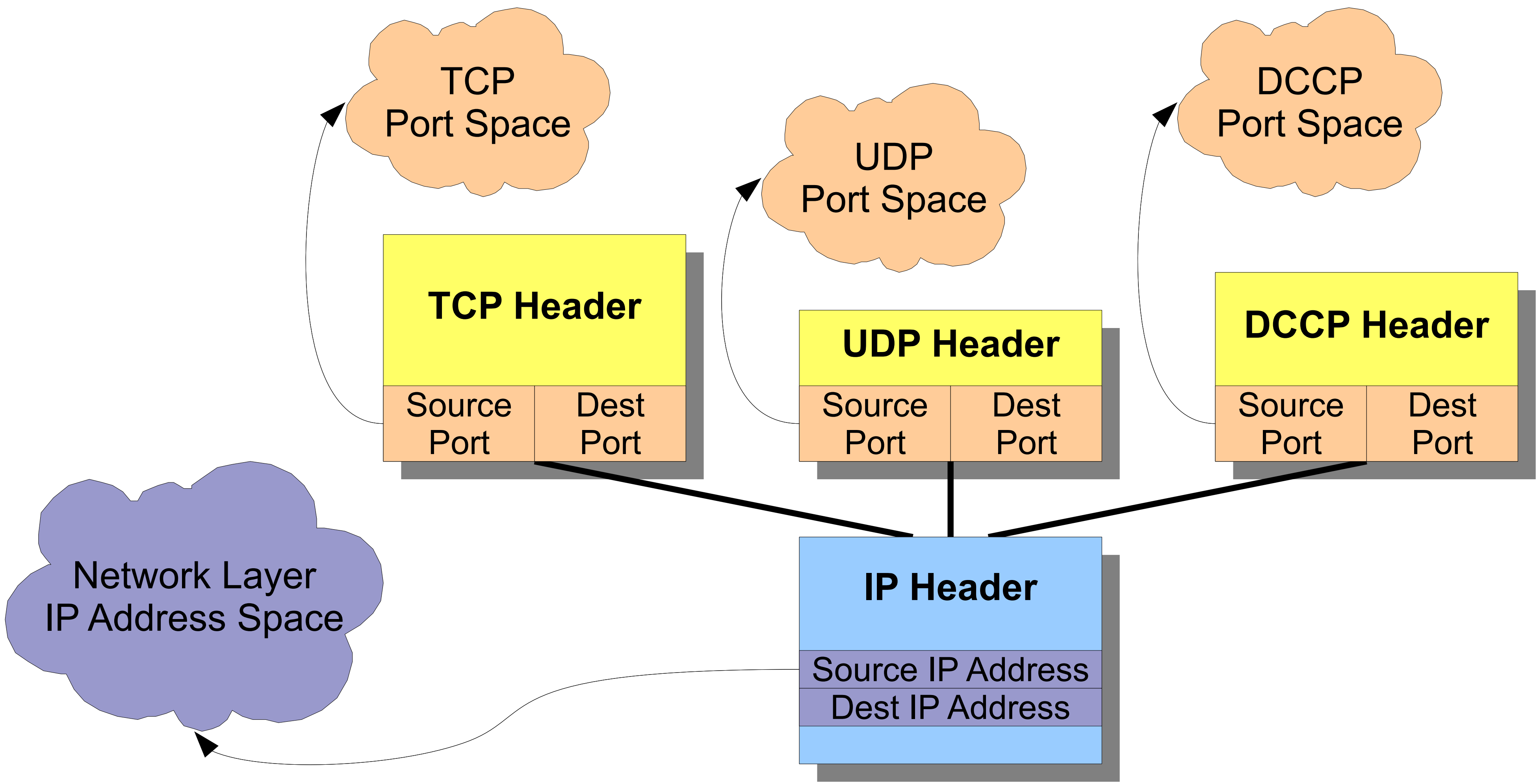
Endpoint Layer

edge routing needs richer endpoint information to enforce policy



Endpoint Identification via Ports

Each transport traditionally has its own **port space**



Why the Network Needs to See Ports

Internet design assumes network needs *only* IP address

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

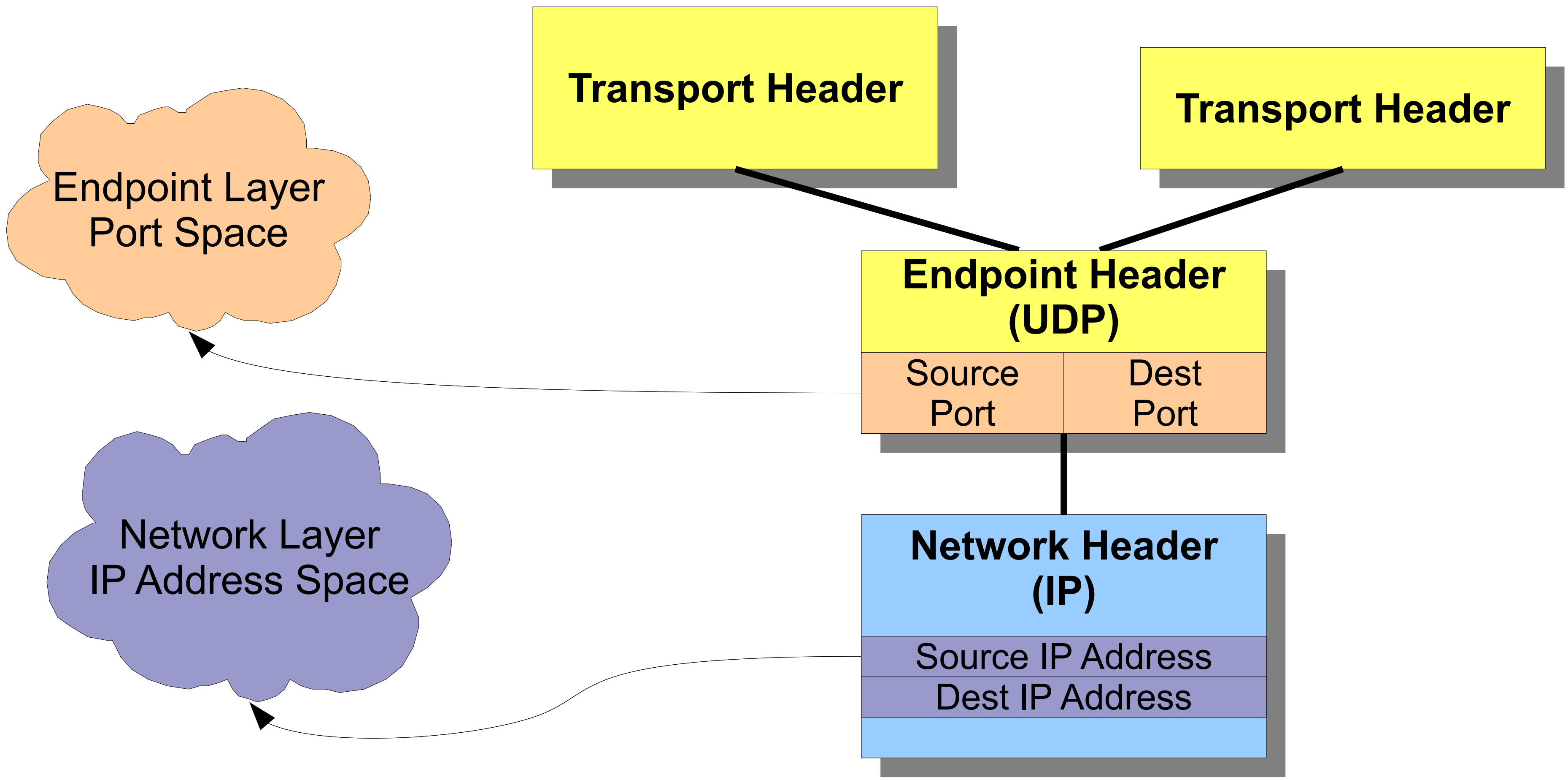
Assumption has proven wrong!

- **Firewalls, traffic shapers** need to see them
  - to enforce connectivity policies, need to know about not just *hosts* but also *protocols*, *applications*, *users*, ...
- **NATs** need to see & transform them
  - IPv4: ports increasingly just "16 more IP address bits"
- **All** must understand transport headers
  - $\Rightarrow$ only TCP, UDP get through now

Tng's Layering Solution

Factor endpoints into shared **Endpoint Layer**

– Starting point "Endpoint Layer" = UDP



Embrace the Inevitable

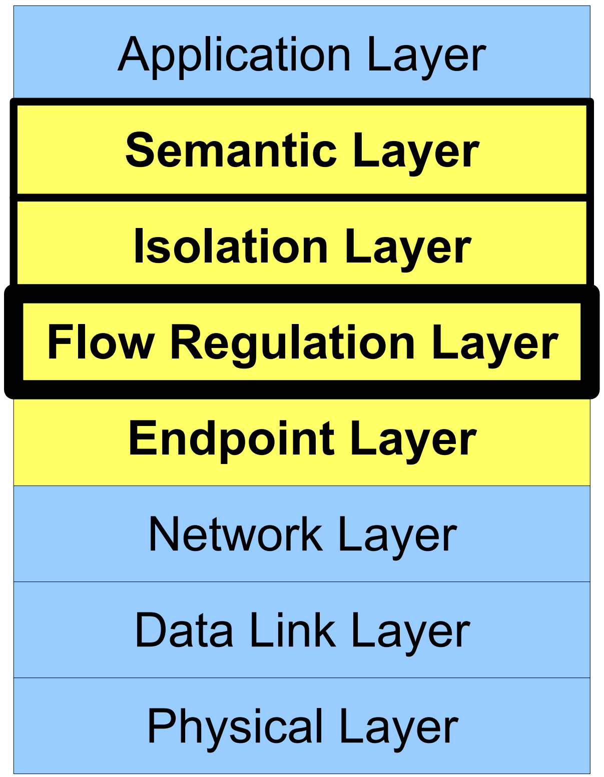
It's happening in any case!

- 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]
- A lot of non-transports do too
  - IPSEC [RFC 3947/3948], Mobile IP [RFC 3519], Teredo [RFC 4380], …

Other benefits: see *"Breaking Up the Transport Logjam"*

Flow Layer

performance tuning
at technology &
administrative boundaries



Congestion Control on a Diverse Internet

TCP congestion control traditionally "end-to-end"

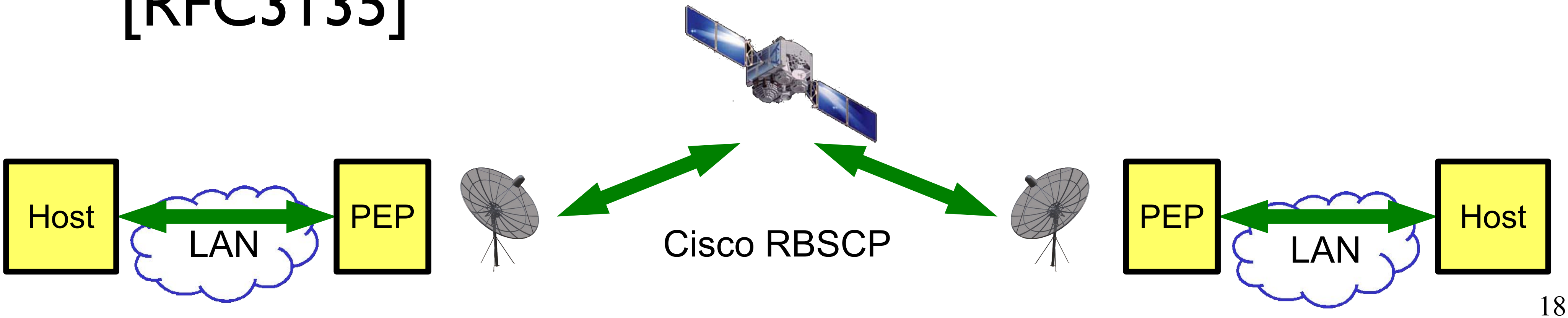
But one end-to-end path may cross **many**...

- different **network technologies**
  - Wired LAN, WAN, WiFi, Cellular, AdHoc, Satellite, …
  - Standard TCP performance sucks on many of these; needs specialized adaptation!
- different **administrative domains**
  - Each cares about CC algorithms in use, for fairness
  - May wish to deploy new CC schemes, e.g., XCP/RCP

Emerging Market Solution

Performance Enhancing Proxies (PEPs)

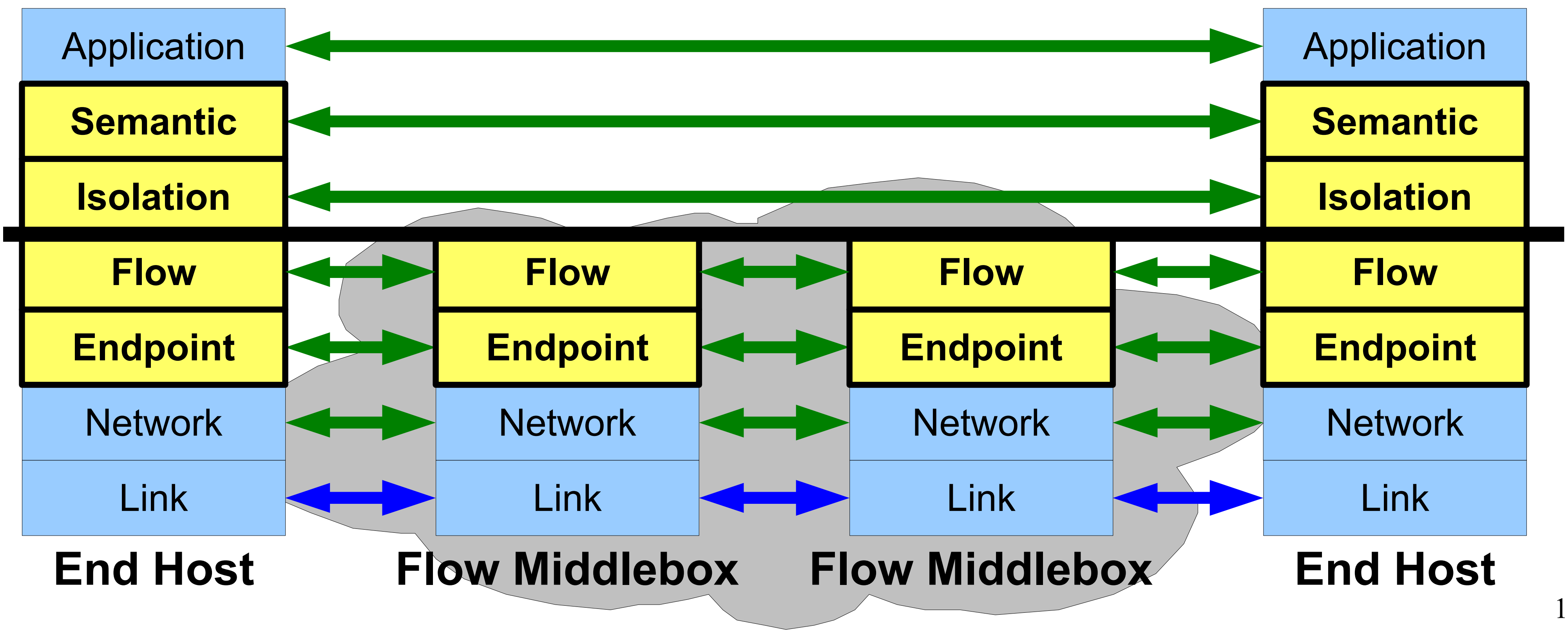
- Tune TCP performance within the network
- Increasingly pervasive; may be “the next NAT”:
  - $236 million market in 2005 [Hall 2006]
  - $1 billion market in 2009 [McGillicuddy 2009]
- **Breaks:** fate sharing, new transports, IPsec [RFC3135]



Tng Solution: *Flow Splitting*

Decompose congestion control **(Flow Layer)** from transport semantics **(Semantic Layer)**

- PEPs interpose on Flow Layer but not Semantic Layer



Technical Challenges

May (or may not) look easy; the devil's in the details:

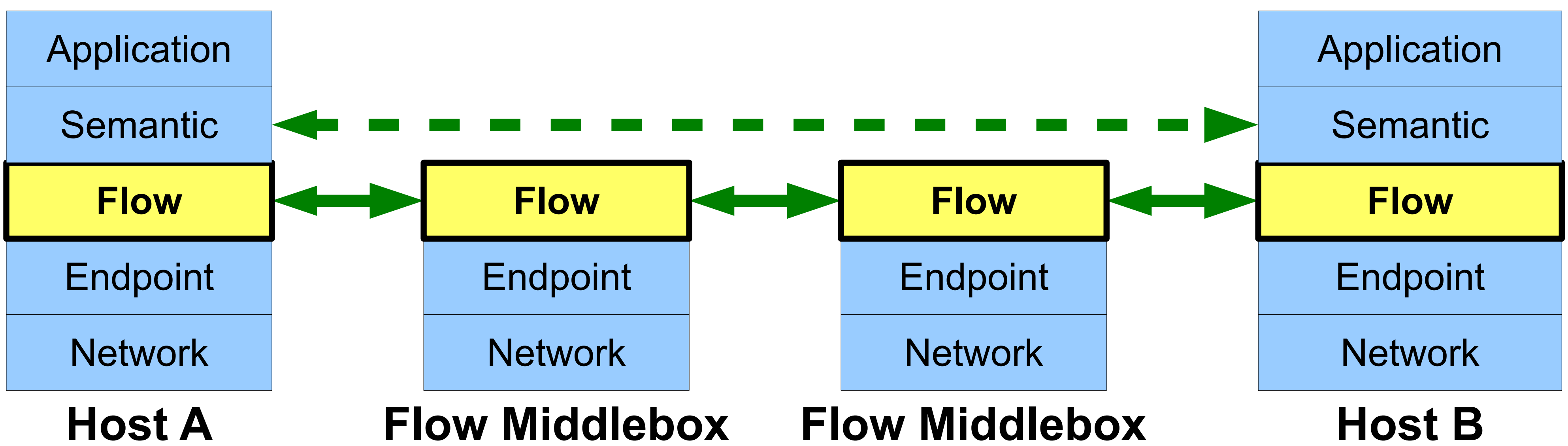
- **Joining:** how to join congestion-controlled path sections into E2E congestion-controlled path?
- **Compatibility:** how to deploy Tng incrementally, staying compatible with existing networks & PEPs?

How to Join Flow Segments to yield End-to-End Congestion Control?

Exploring two approaches:

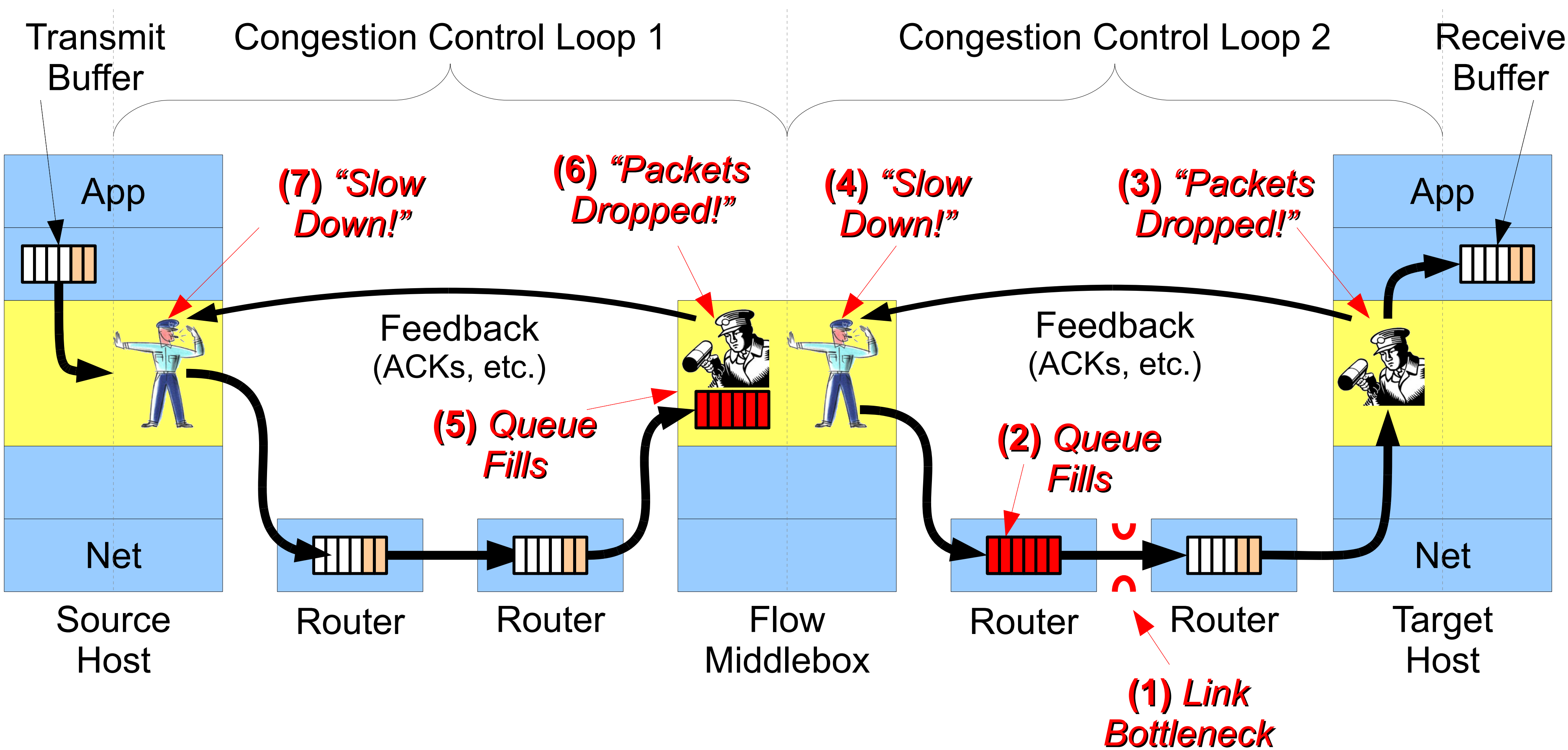
1) Queue sharing (implemented)

2) Congestion control stacking (WIP)



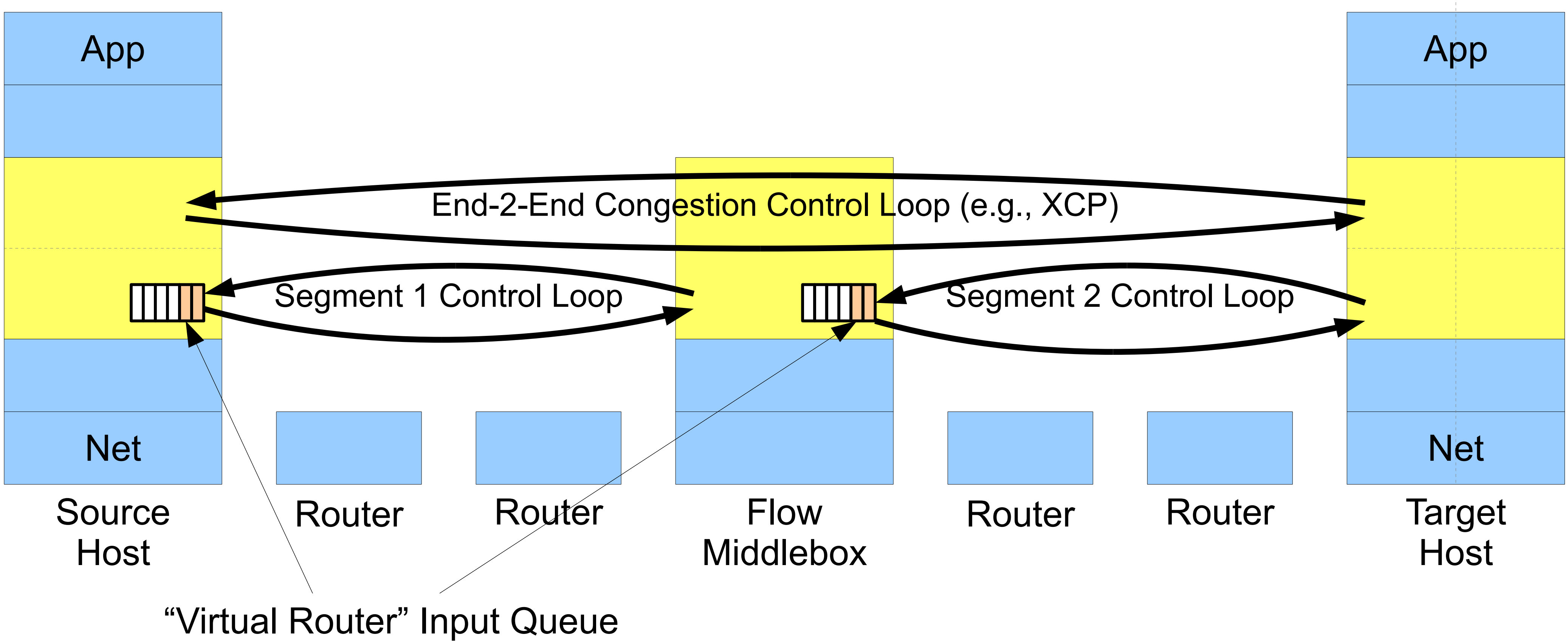
Queue Sharing

(implemented in NS2 simulation & working prototype)



Congestion Control Stacking

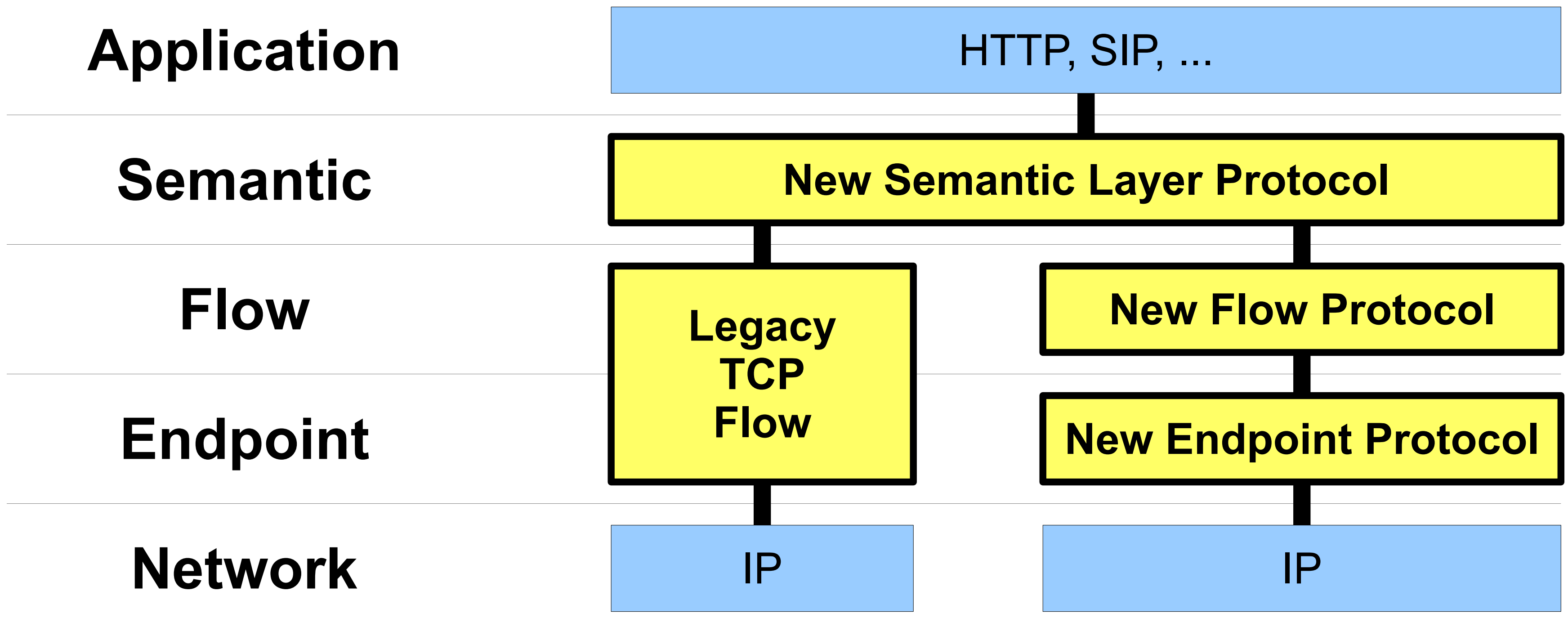
(work in progress)



Compatibility with Legacy PEPs

How to **deploy Tng incrementally**,
given prevalence of PEPs that know only TCP?

- Prefer DCCP-like protocol implementing Flow Layer...
- But fall back on TCP as "compatibility Flow Layer"



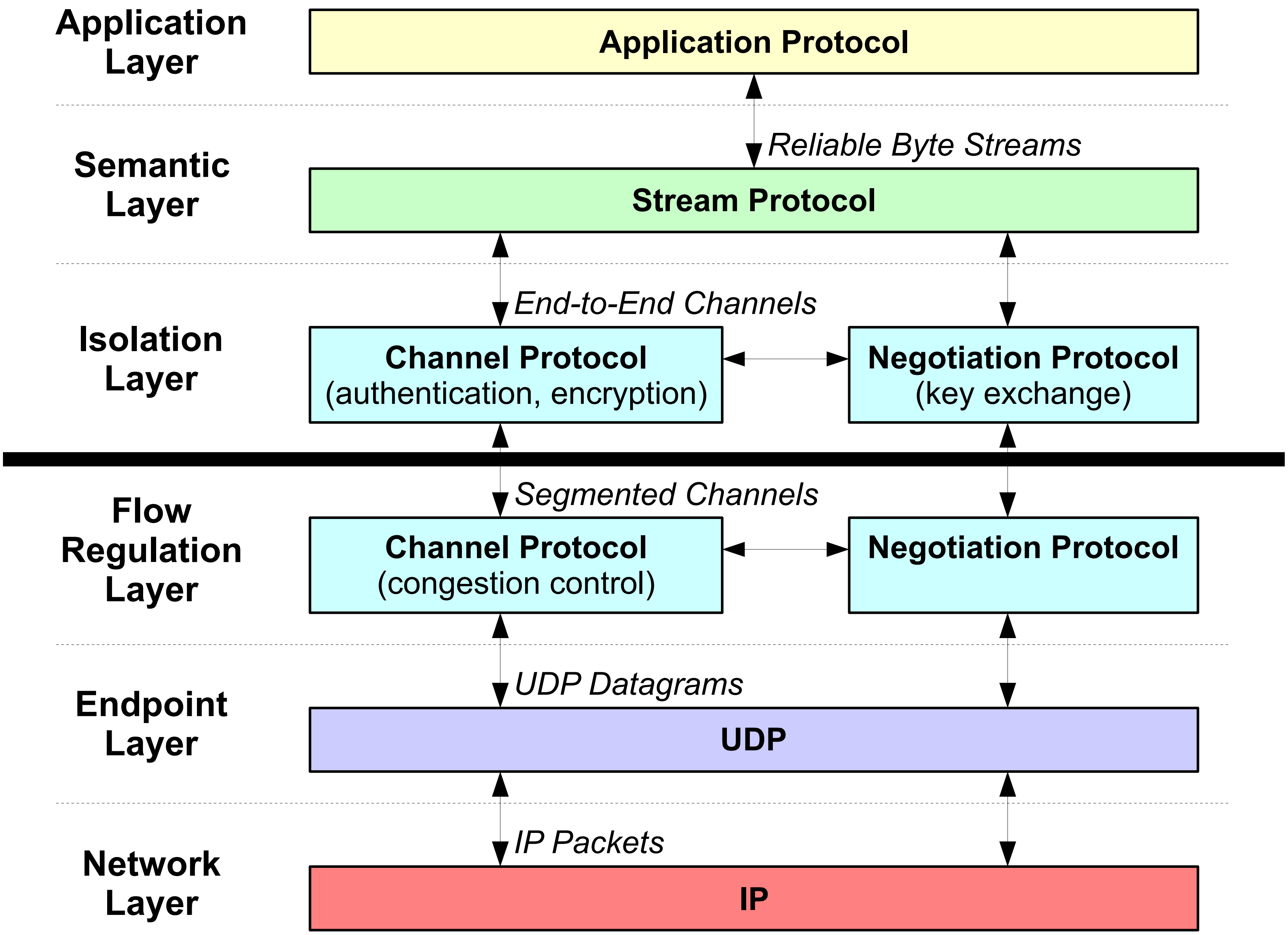
Evaluation

Using:

- NS2-based Simulations
  - Building on NS2's models of TCP congestion control
- Working prototype usable on real networks
  - Building on Structured Stream Transport (SST)

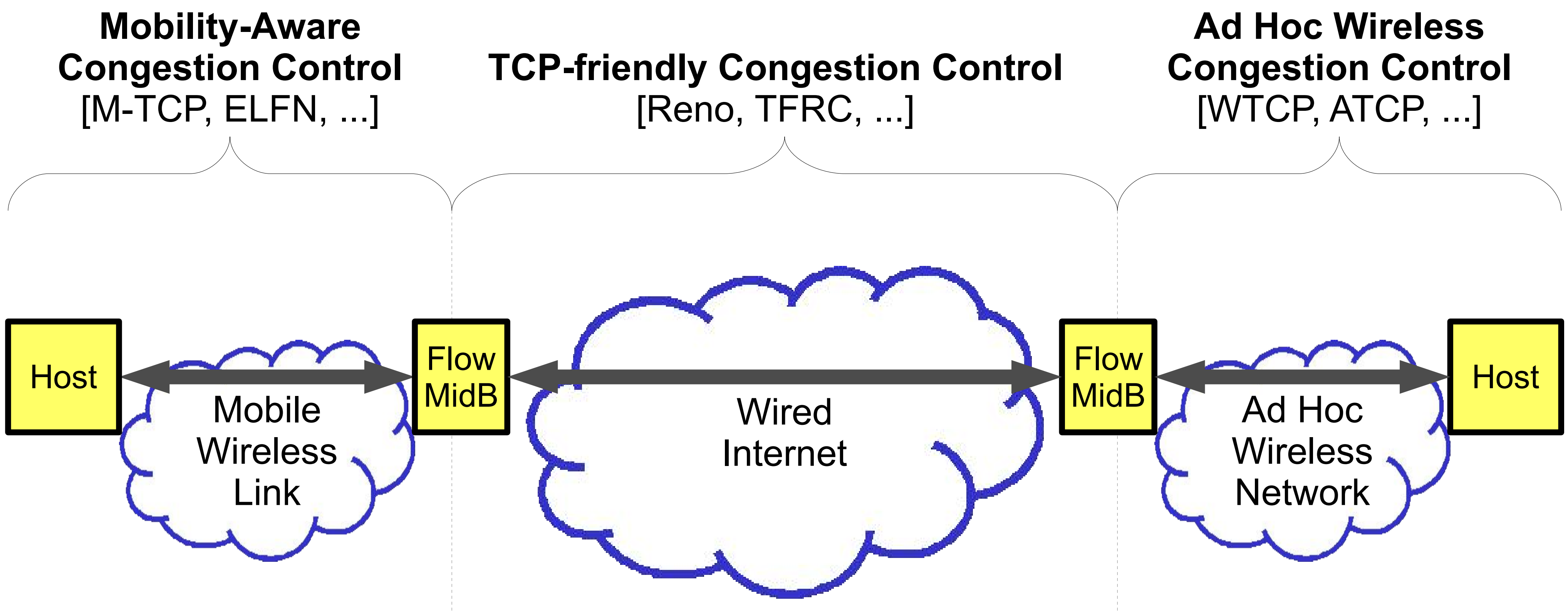
    Ford, "Structured Streams: a New Transport Abstraction", SIGCOMM 2007

SST-Based Prototype Structure



Simulation Scenario 1

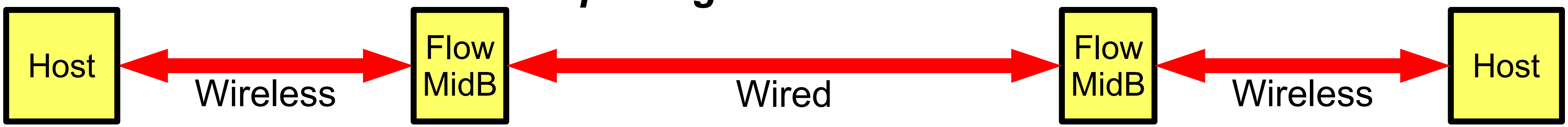
Last-mile proxies for wireless/mobile links



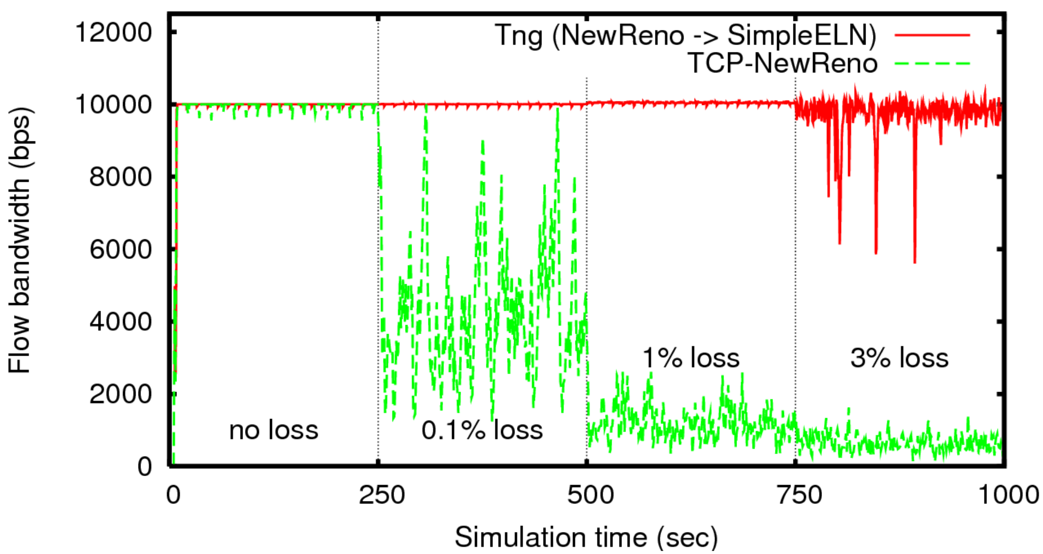
Simulation Scenario 1: Results

End-to-End: Wireless Loss Kills TCP Performance

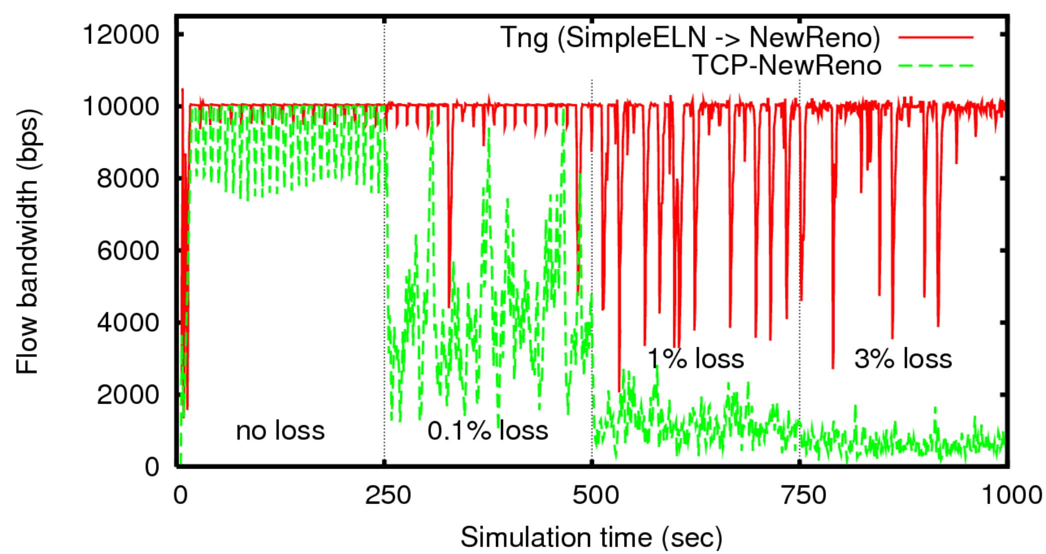
With Flow Splitting: Performance Maintained



Download

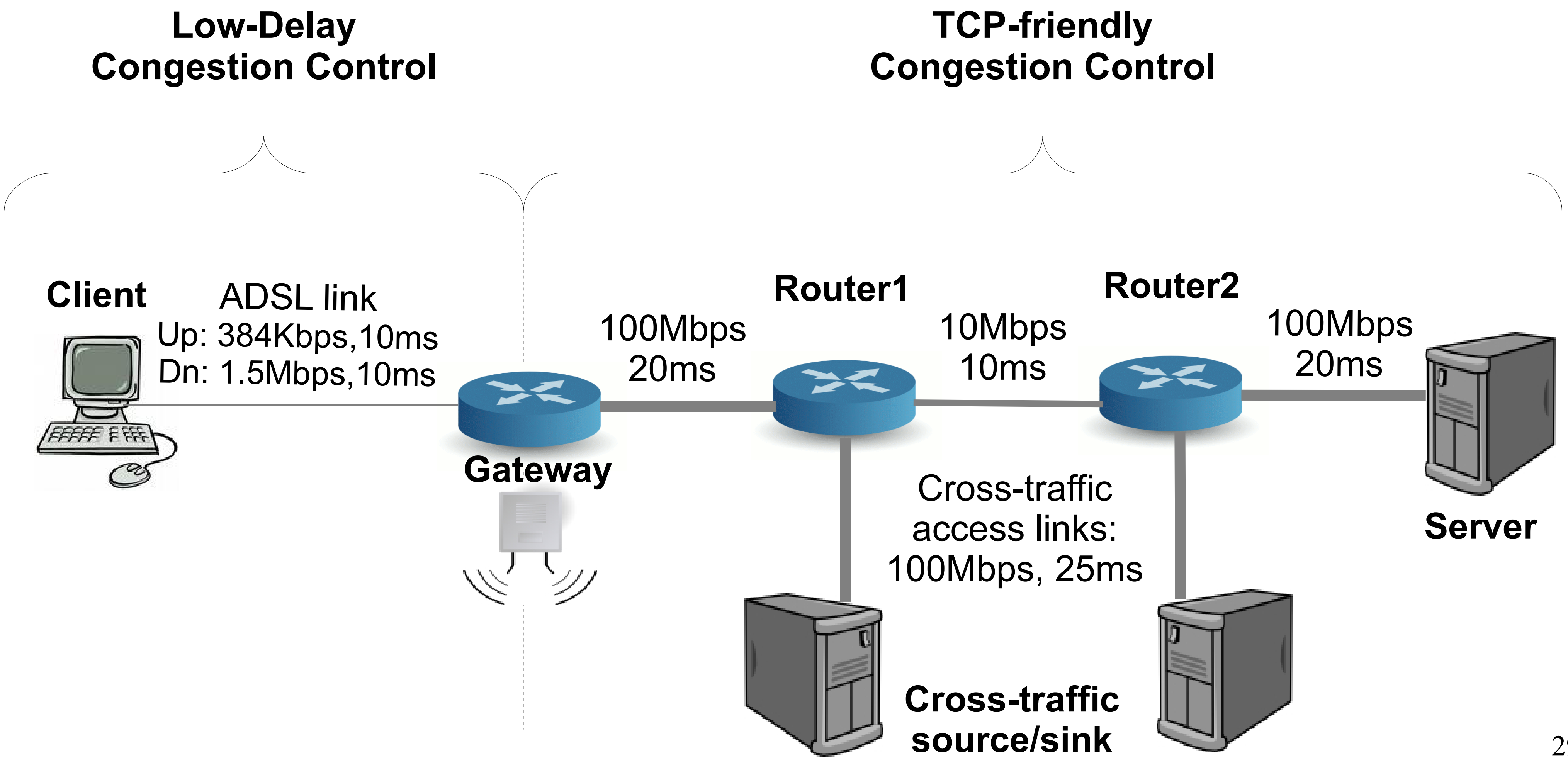


Upload

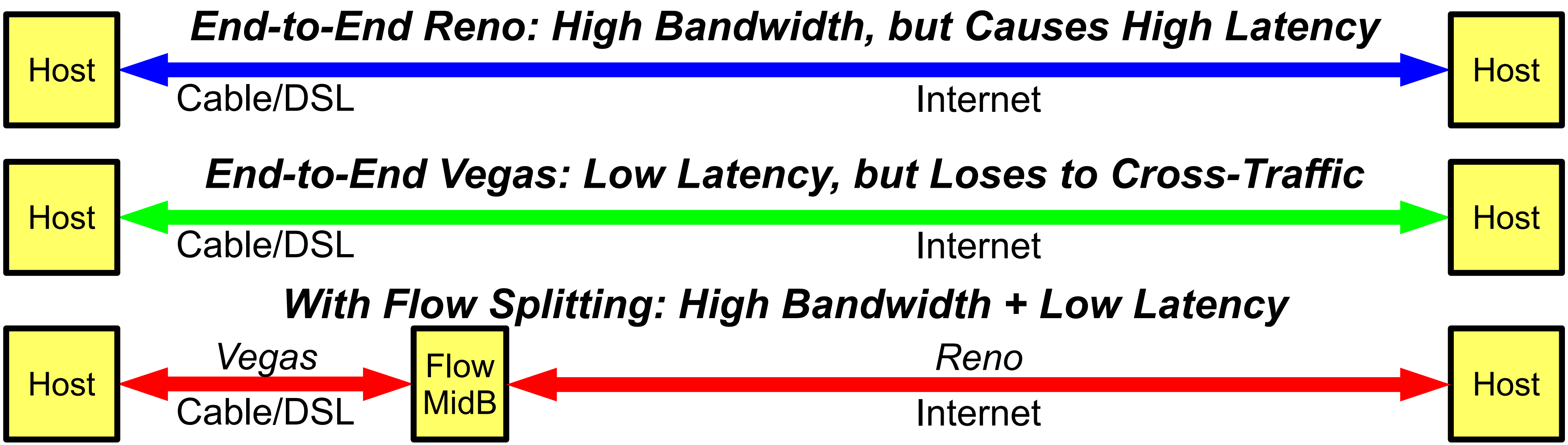


Simulation Scenario 2

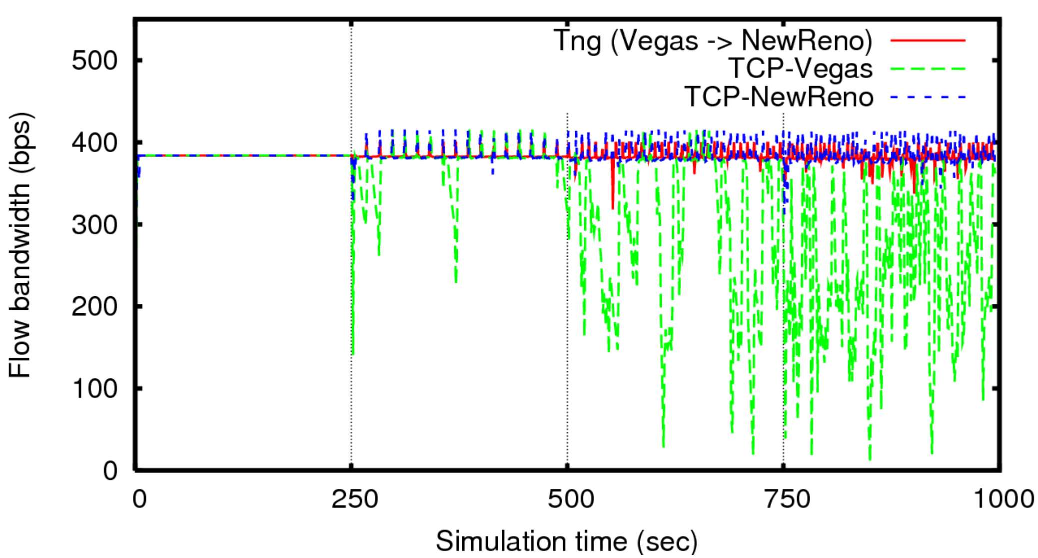
Delay-Sensitive Use of DSL/Cable Links



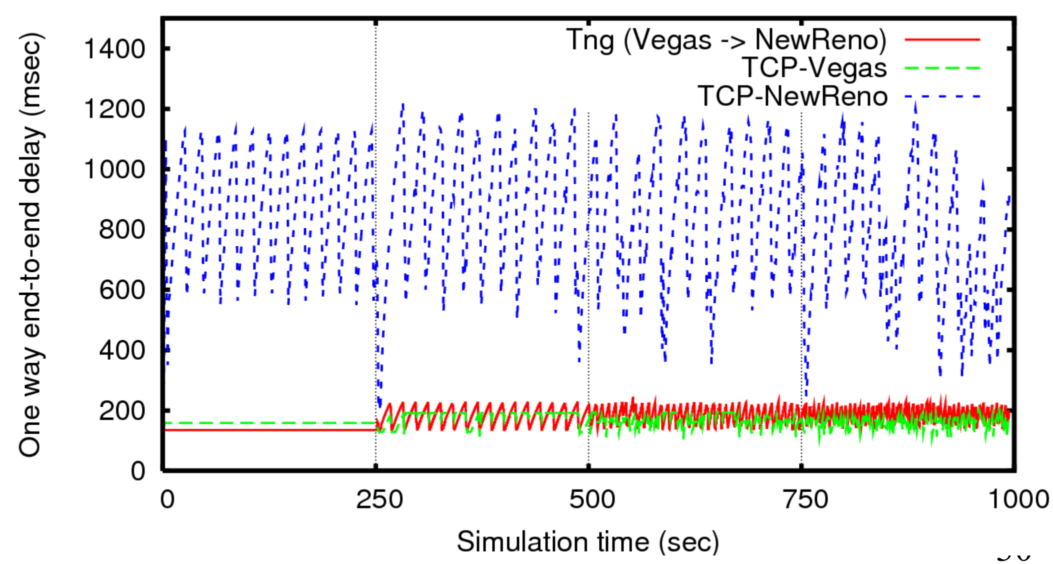
Simulation Scenario 2: Results



Upload Bandwidth

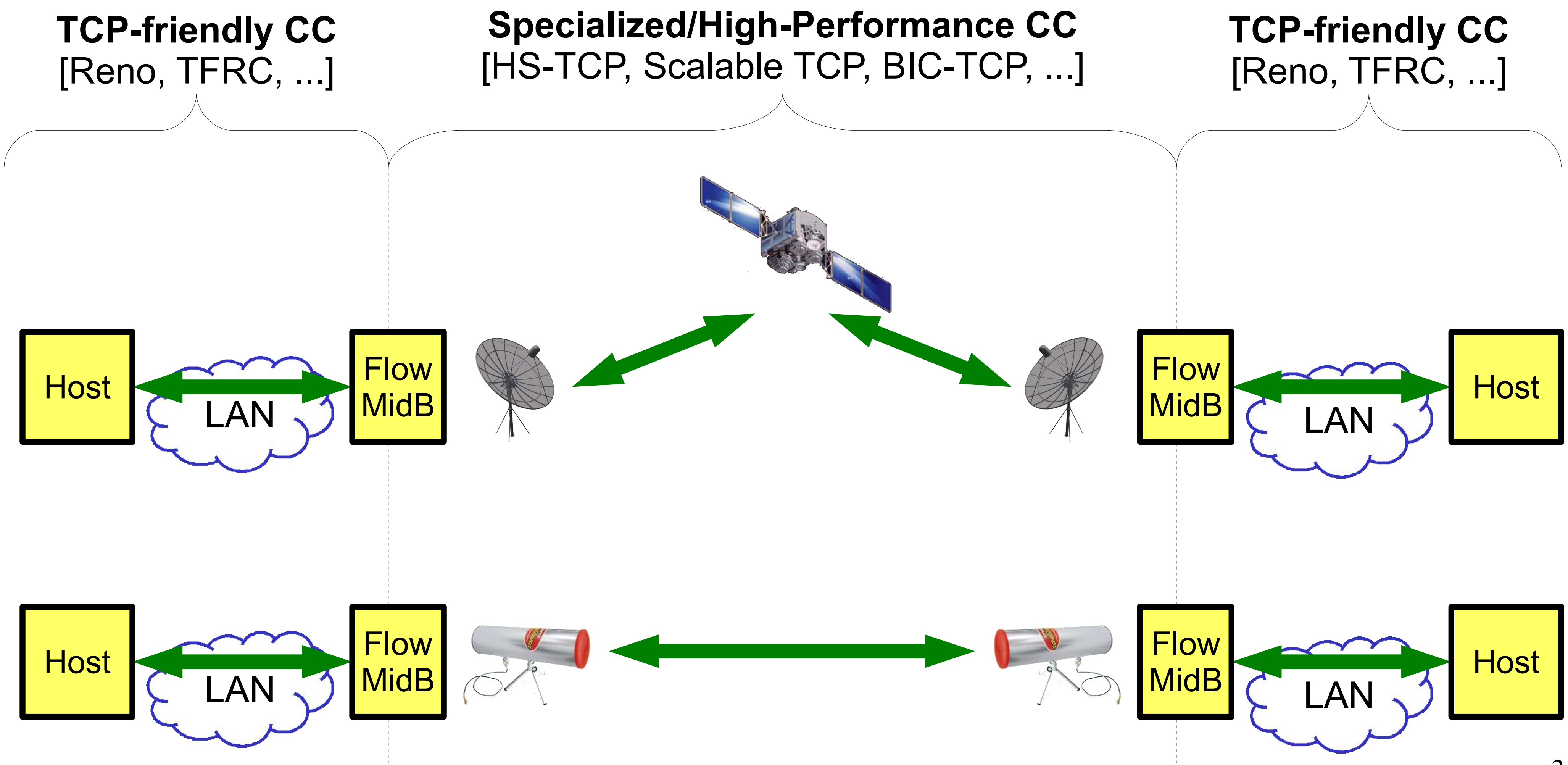


Upload Latency

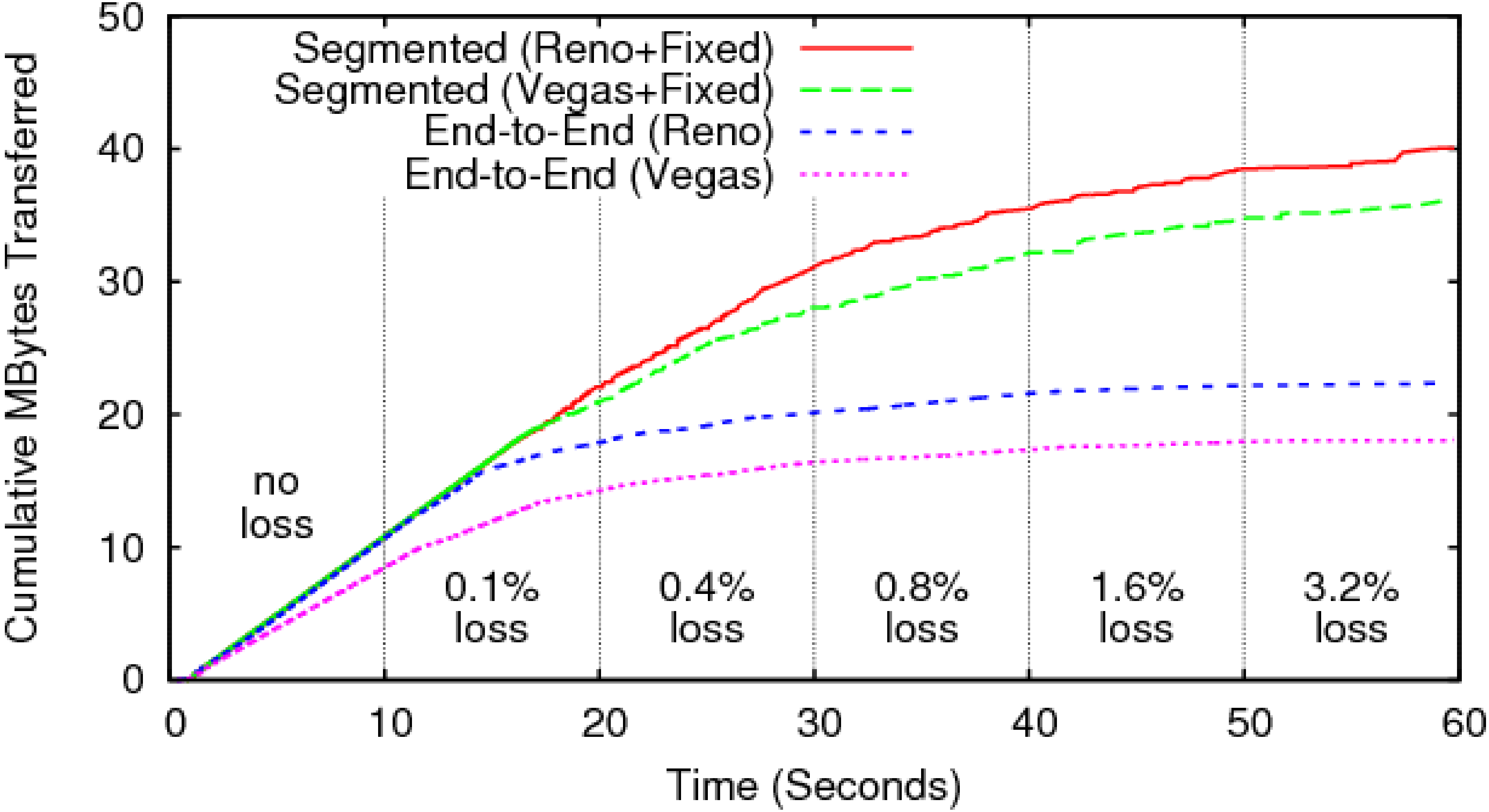


Prototype Test Scenario 1

Transfer over Lossy Long-Distance Satellite Link



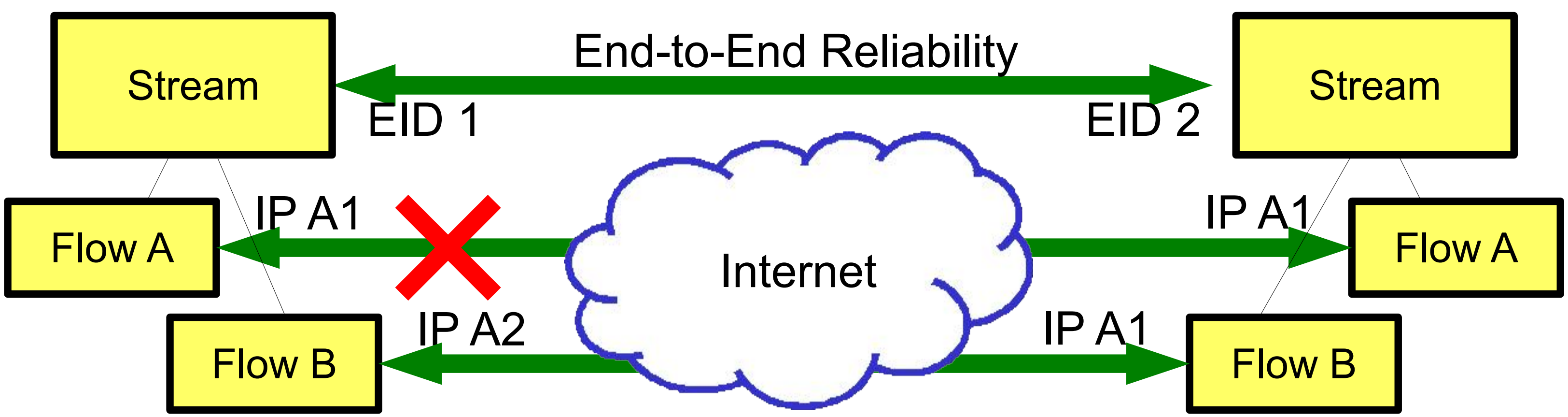
Reliable Transfer over Satellite Link



Prototype Test Scenario 2

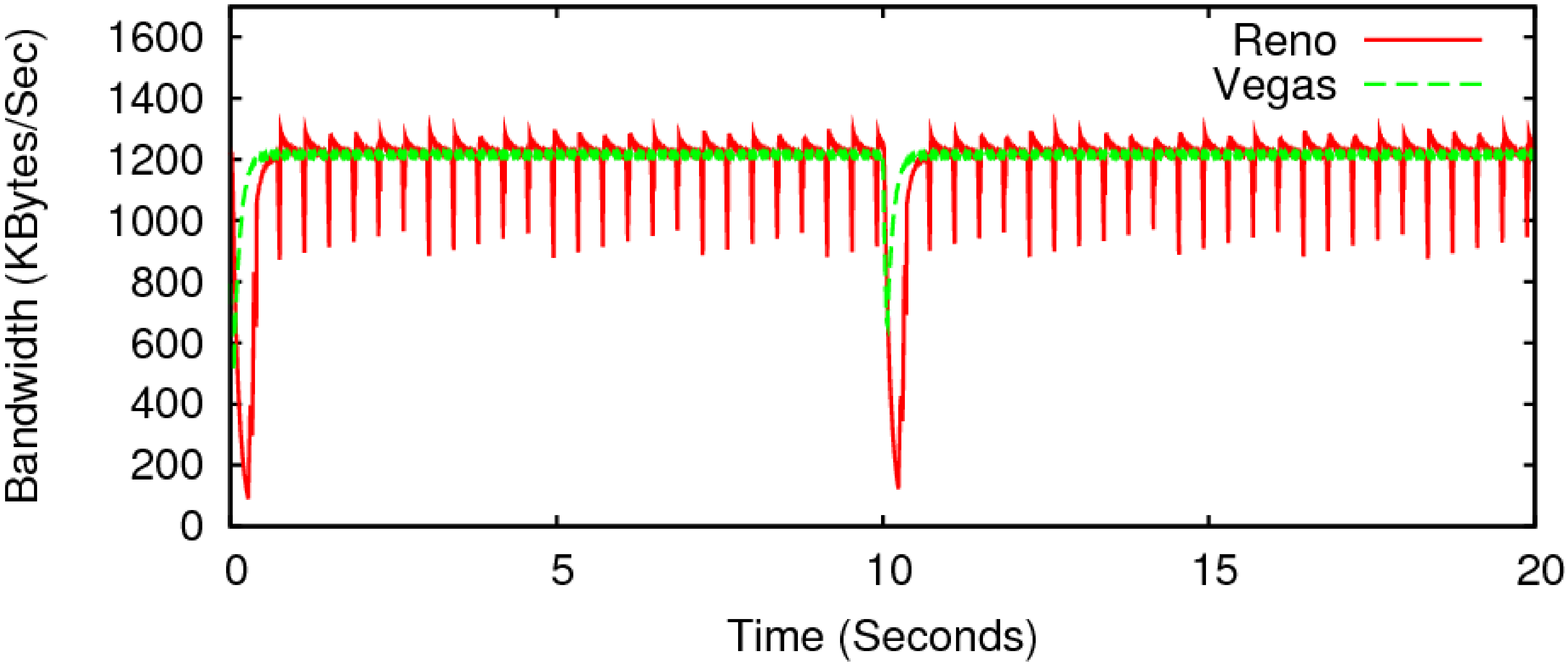
Fate Sharing:
recovery of end-to-end stream communication across flow layer failures

- SST Stream Protocol associates streams with stable cryptographic endpoint identities
- Underlying Flow fails if a host's IP address changes, but stream can (re)start and migrate to new flow



Prototype Test Scenario 2

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



"Clean Slate" versus "Legacy" Implementation of Tng Architecture

Code size and Protocol Overhead Comparison

- Current SST Prototype vs Equivalent Linux Protocols
- C++ vs C, prototype vs mature stacks – *not really fair!*

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

Conclusion

Transport evolution is **stuck!**

- **Lost:** transport evolvability, E2E security, fate sharing

To unstick, need to **refactor**:

- Enable middleboxes to function without interfering with end-to-end transport functions

Tng allows **performance enhancing proxies** to **split flows** and **tune congestion control** while **preserving end-to-end semantics**

Further information: http://bford.info/tng/