Oh End-to-End, Where Art Thou?

Jana Iyengar
Franklin & Marshall College

Acknowledgments

- Syed Obaid Amin (F&M, Yale)
- Fitz Nowlan (Yale)
- Nabin Tiwari (F&M)
- Jeff Wise (F&M)
- Bryan Ford (Yale)

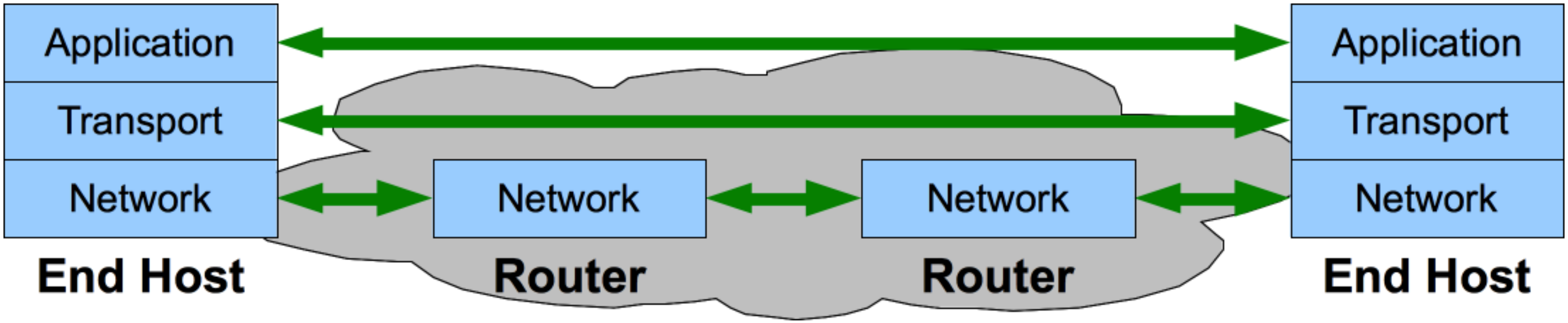
What is the Internet?

Why is it so different from other networks we've seen before?

The End-to-End Principle

Only end hosts

- see past a packet's IP header **(Generality)**
- maintain “hard state” **(Fate Sharing)**



Rise of the Middle

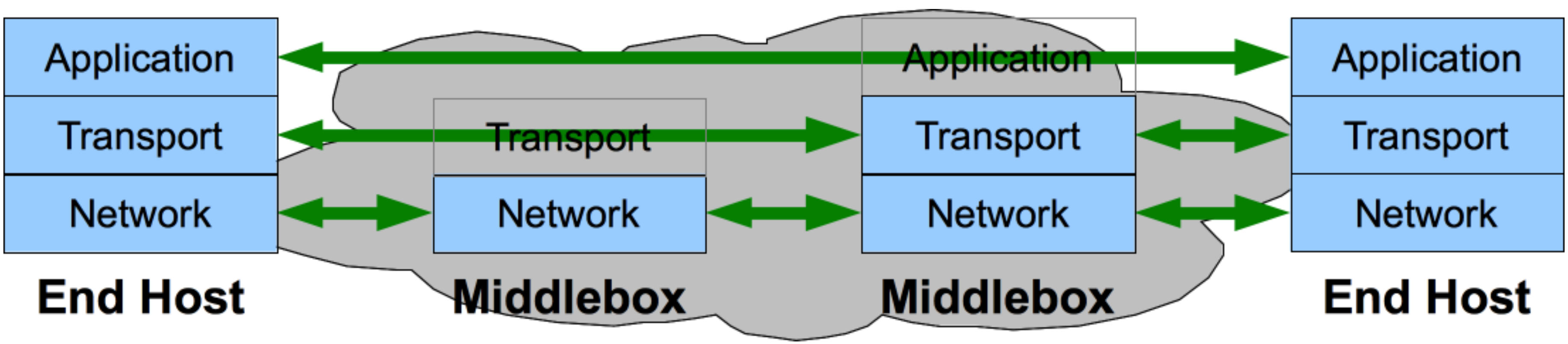
- NATs: IP address scarcity
- Firewalls: protection and policy
- Traffic Shapers: manage bandwidth and delay
- PEPs: optimize performance

All solve important problems for operators

Eroding Transport End-to-Endness

Middleboxes **need to** interact with Transport

- NATs: port number, checksum
- Others: most TCP header fields, state machine



Eroding Transport End-to-Endness

We lose:

- Generality: only TCP, UDP
- Fate Sharing: hard state inside network

What is "Reachability" now?

Middleboxes are new architectural control points …

(control what the Internet *can* do)

… but they are accidental control points!

Accidental Control Points

RFC 1631 (NAT):

> “The two most compelling problems facing the IP Internet are IP address depletion and scaling in routing. […] Until the long-term solutions are ready an easy way to hold down the demand for IP addresses is through address reuse.”

> “NAT has several negative characteristics that make it inappropriate as a long term solution, and may make it inappropriate even as a short term solution.”

A Flaky Internet!

- Packet blackholes
  - Will a particular packet will get through?
  - On what paths?

- Cheap boxes are common
  - And buggy!
  - Almost all our traffic goes through these boxes!

Do we need new Transports?

Isn't TCP enough?

Consider e2e multipath

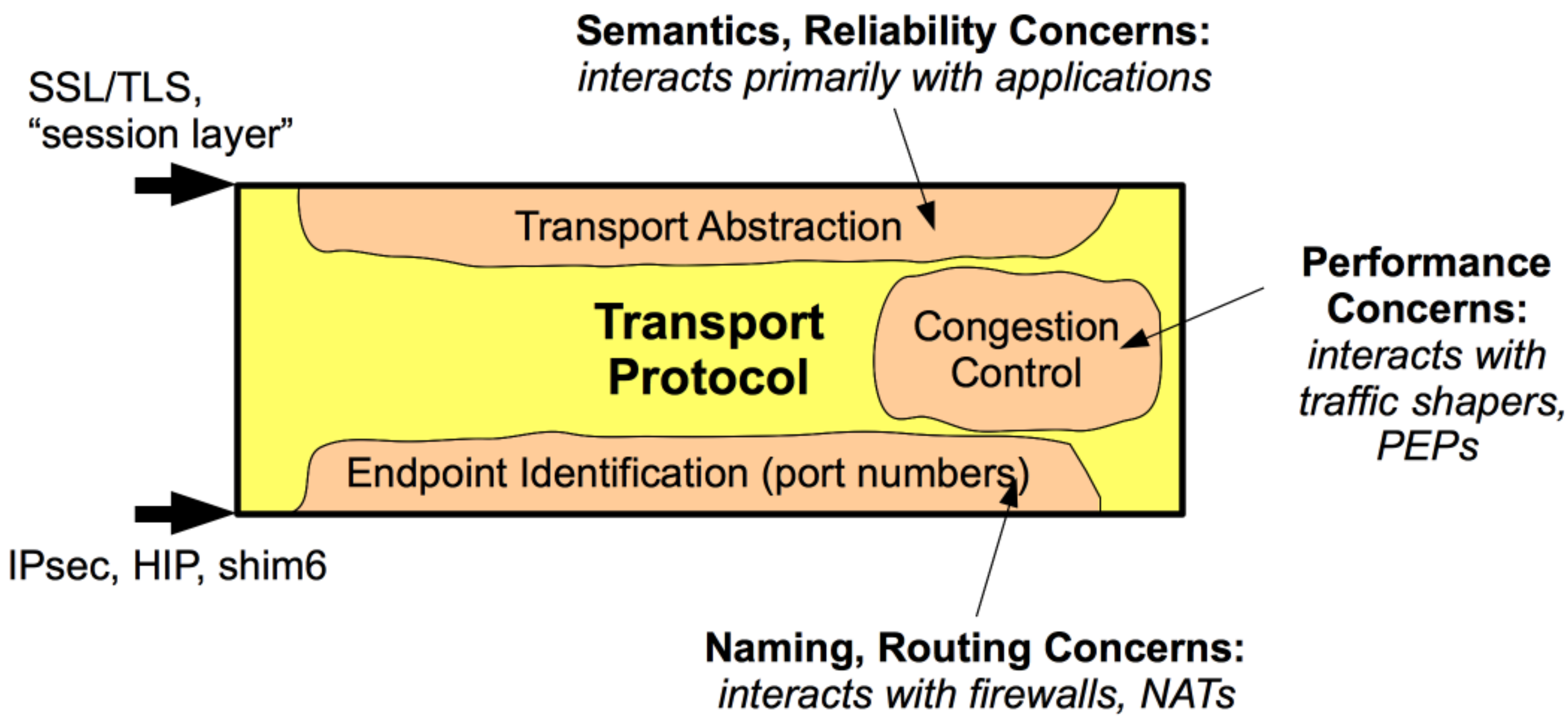
- SCTP: Stream Control Transmission Protocol
- RFC 2960 (circa 2000)
- MPTCP: Multipath with TCP (circa 2012)

Apps build transport abstractions atop TCP, UDP

- Eg: Adobe Flash (RTMFP)

The Transport Layer is Stuck in an Evolutionary Logjam!

The Problem



TCP conflates network-oriented and app-oriented functions

(Thought Experiment:

Design an Internet with middleboxes as first-class citizens)

Design Assumptions For New Transport Services to be Deployable

New end-to-end services should not *require* changes to middleboxes

Consequence: New end-to-end services must appear as legacy protocols on the wire

The Minion Suite

A "packet packhorse" for deploying new transports

- ***Uses legacy protocols …***
  TCP, TLS

- ***… as a substrate...***
  turn legacy protocols into *minions* offering unordered datagram service

- ***… for building new services that apps want***
  multistreaming, message boundaries, unordered delivery, app-defined congestion control
  (may be extended to: stream-level receiver-side flow control, multipath, partial reliability)

The Minion Suite

Main goals:

- How far can we stretch the TCP protocol wire-format?
- TCP increasingly used as a substrate. Make it a good one!

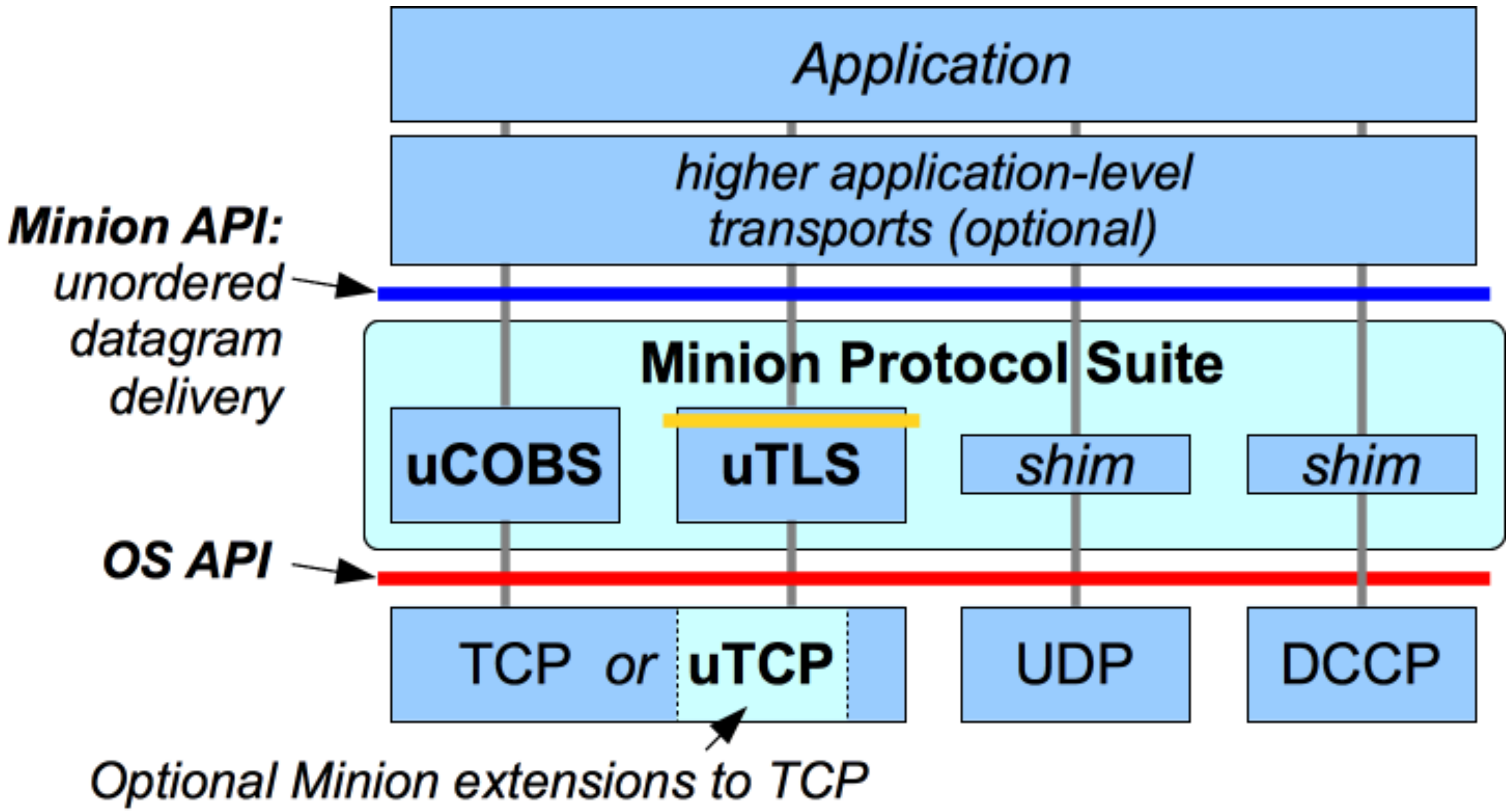
Minion's strength is that it works.

NSDI 2012 review:

A reasonably performing solution to what is (sadly) a real practical problem

[…] accepts the new "narrow waist" as TCP, and shows that with enough devious thought, one can manage to implement functionality at cross-purposes with TCP atop it.

What's in the Minion Suite?

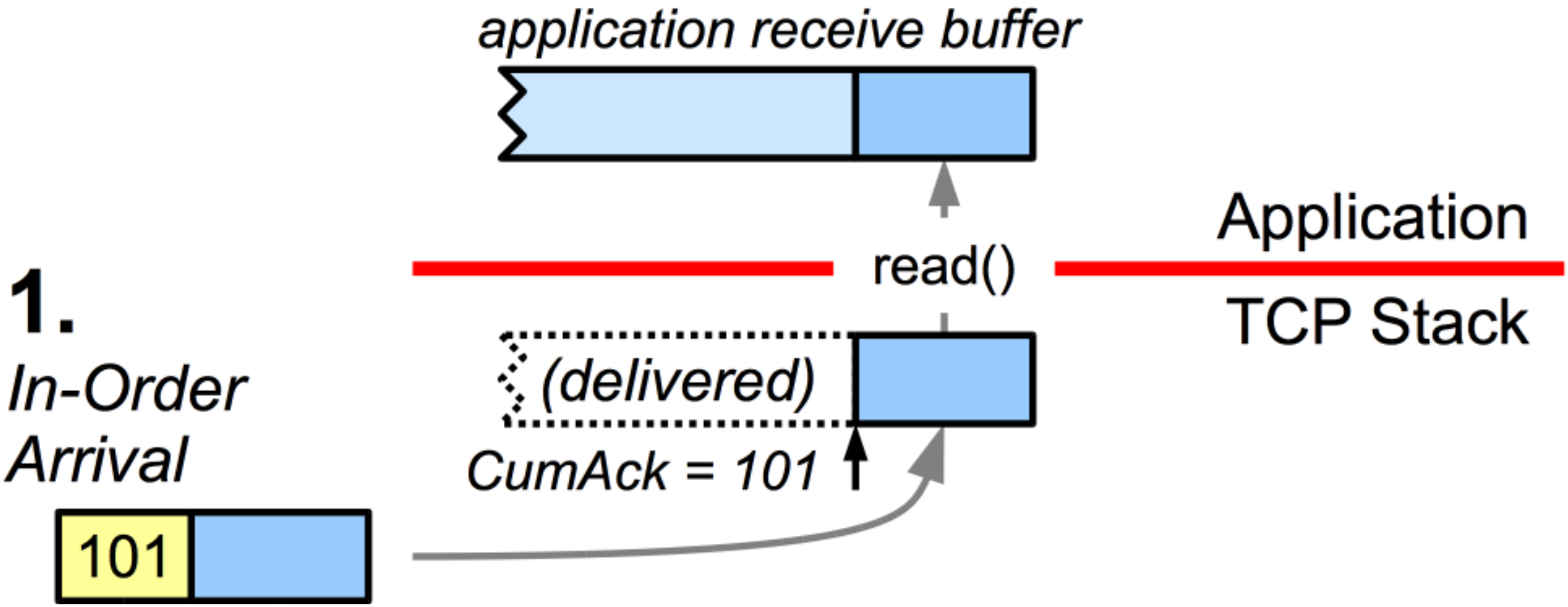


*u*TCP: unordered TCP

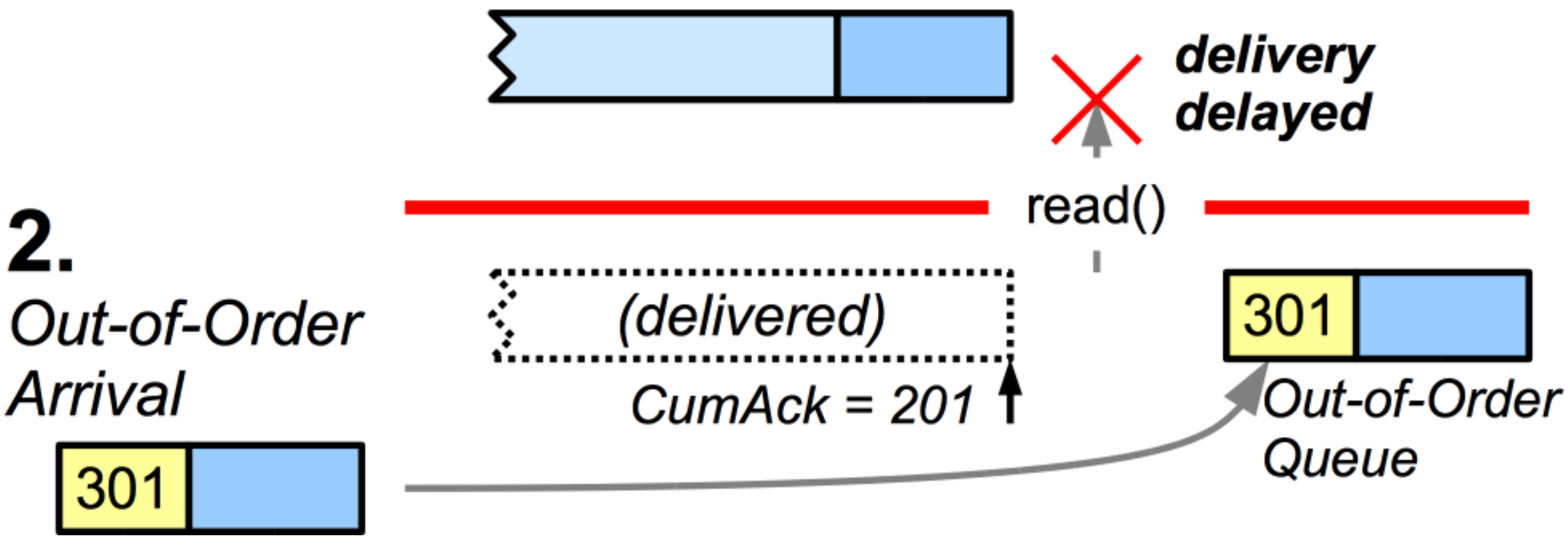
We introduce 2 new TCP socket options in Linux:

- **SO_UNORDERED_RCV**
  - kernel delivers incoming data immediately
- **SO_UNORDERED_SND**
  - accepts priority with every app message
  - message placed in a priority queue in kernel

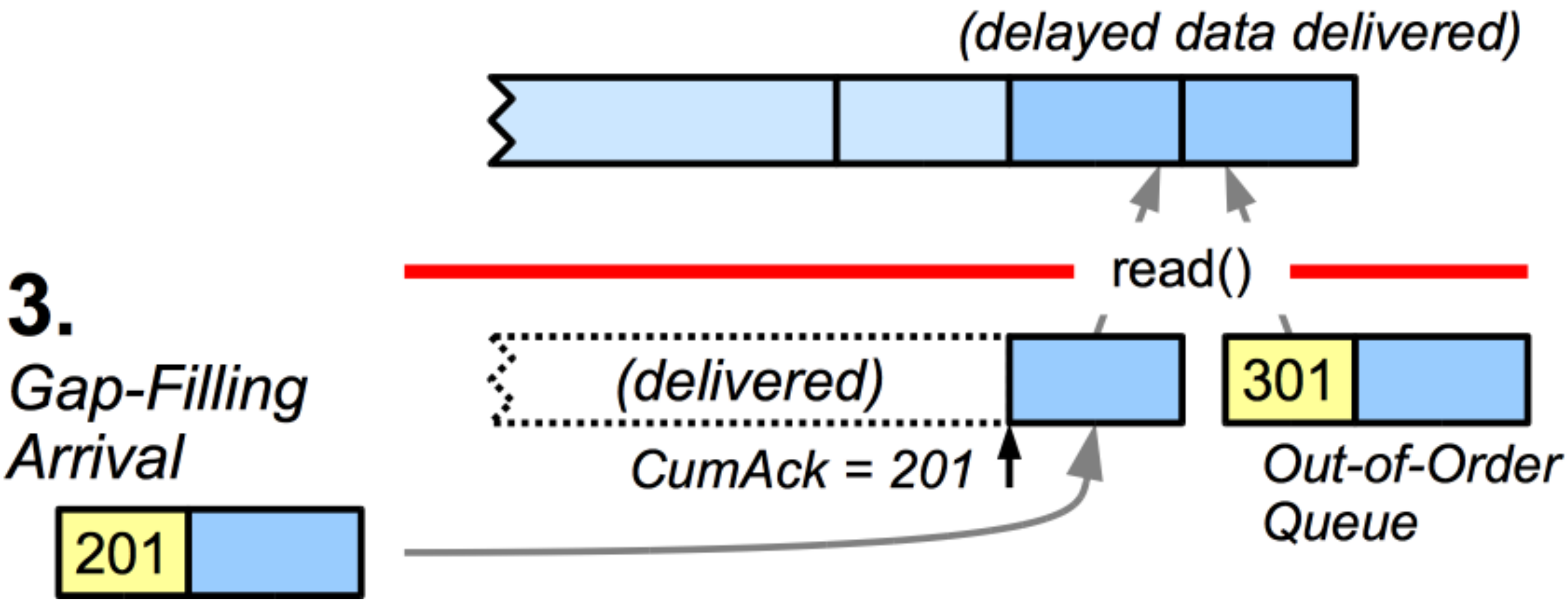
Delivery in Standard TCP



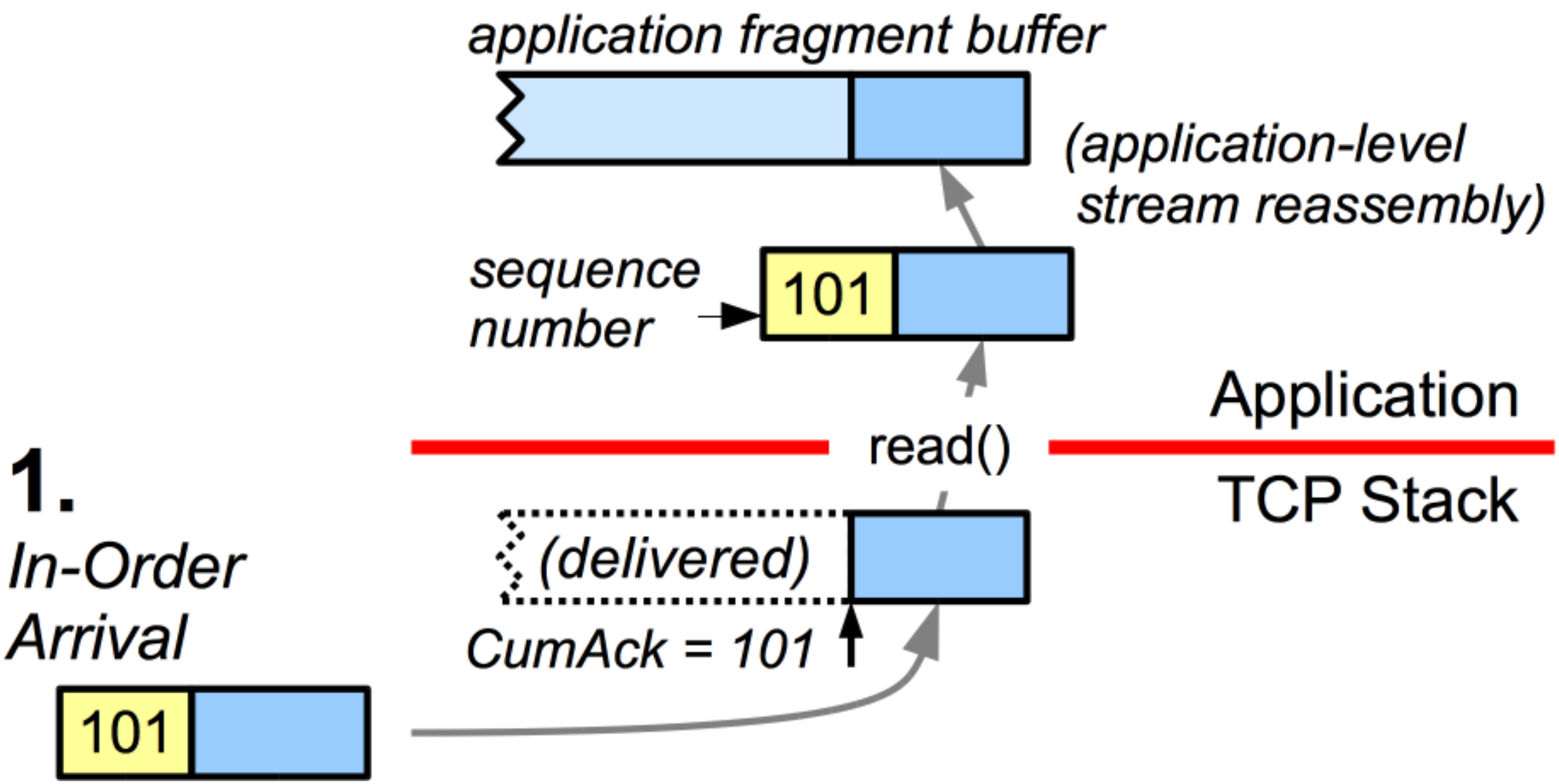
Delivery in Standard TCP



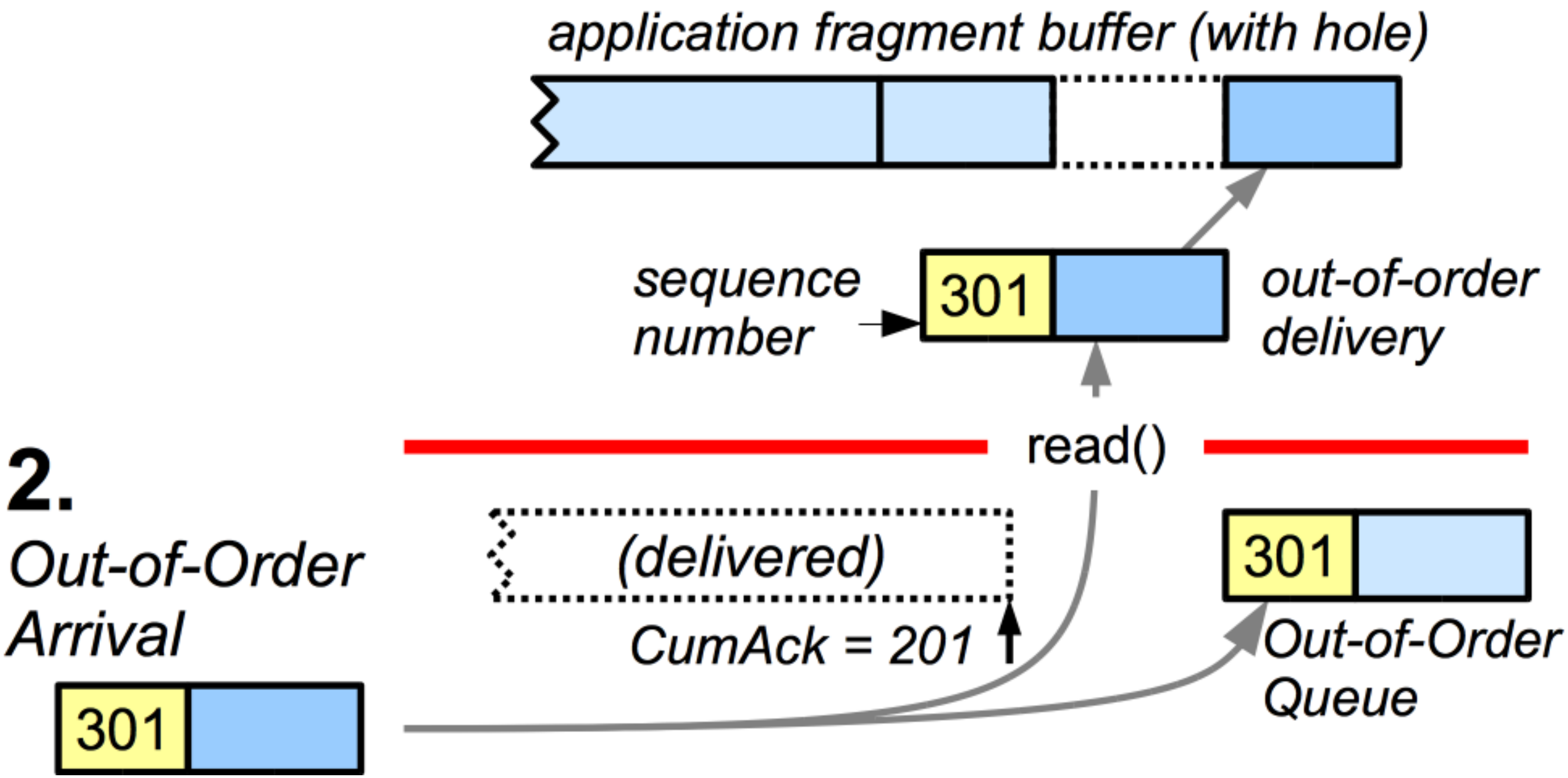
Delivery in Standard TCP



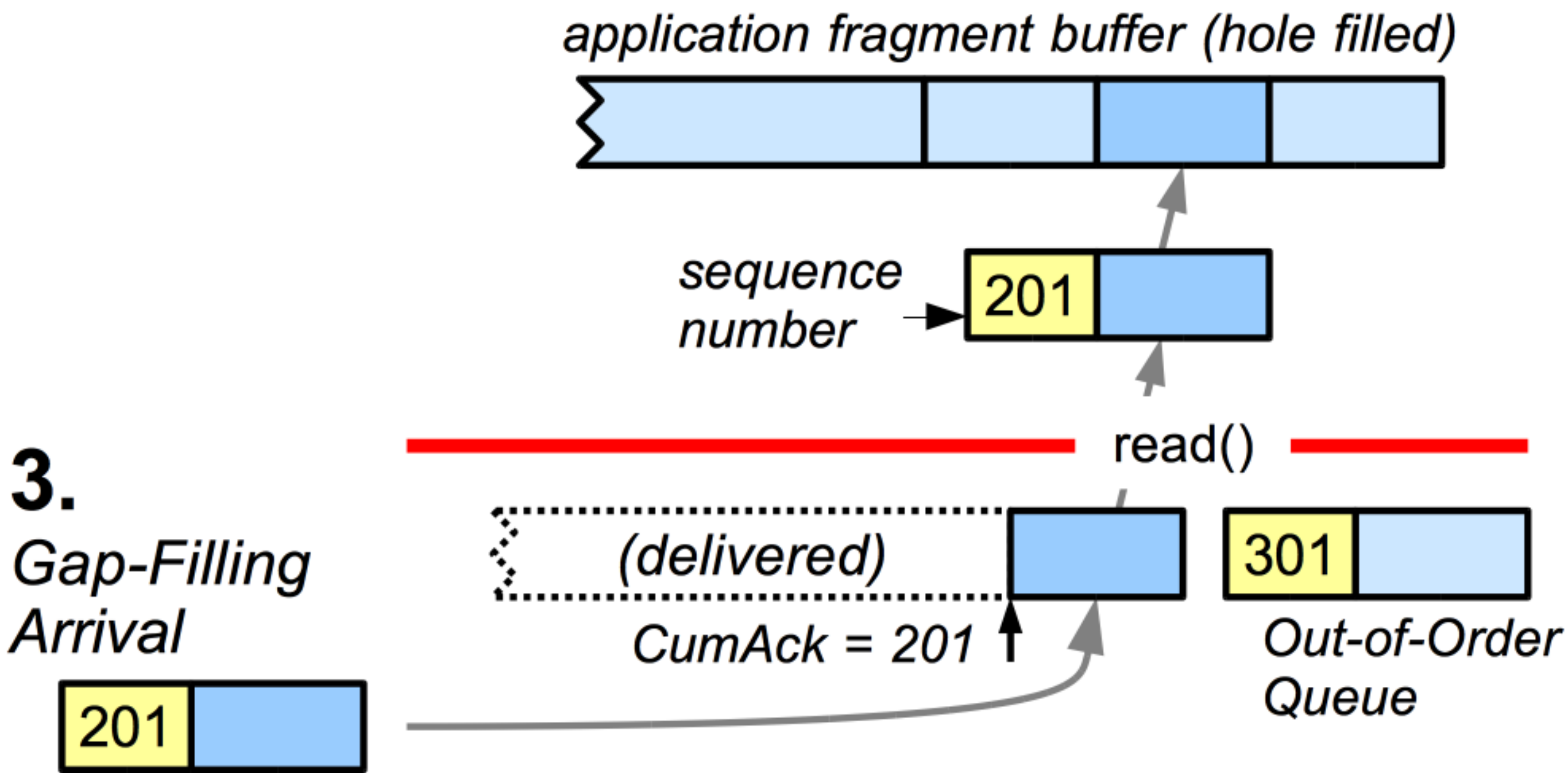
Delivery in *u*TCP



Delivery in *u*TCP



Delivery in *u*TCP



Dealing with Unordered Bytes

- **No inherent structure in bytestream**
  - *u*TCP receives and delivers arbitrary unordered fragments from the data stream

- ***Self-delimiting* framing with COBS**
  - *zero* added to both ends of an app message
  - COBS encoding eliminates zeros in orig data
  - guaranteed max bit-overhead: 0.4% (6 bytes for 1448-byte msg)

Using Minion

- ***u*COBS Sender**
  - COBS-encodes messages, sends them with *u*TCP

- ***u*COBS Receiver**
  - manages out-of-order data received from *u*TCP
  - extracts, decodes, delivers messages anywhere in received data bytes

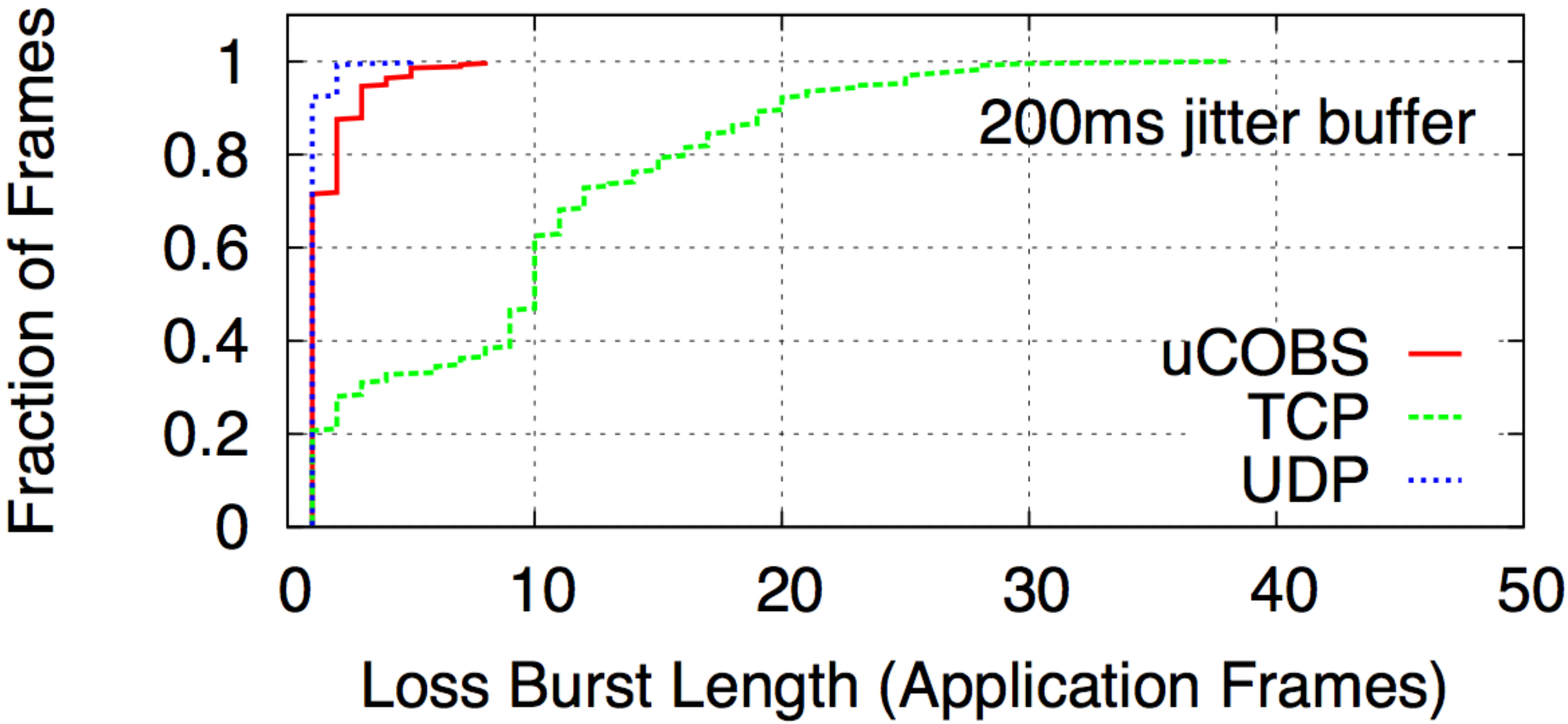
Impact on “Real Applications”

Example: Voice-over-IP (VoIP)

- Delay-sensitive
  - Long RTT delays perceptible, frustrate users
- VoIP codecs are highly sensitive to *burst* losses
  - Can't interpolate when many packets lost or delayed!

VoIP Application: Observed Delay

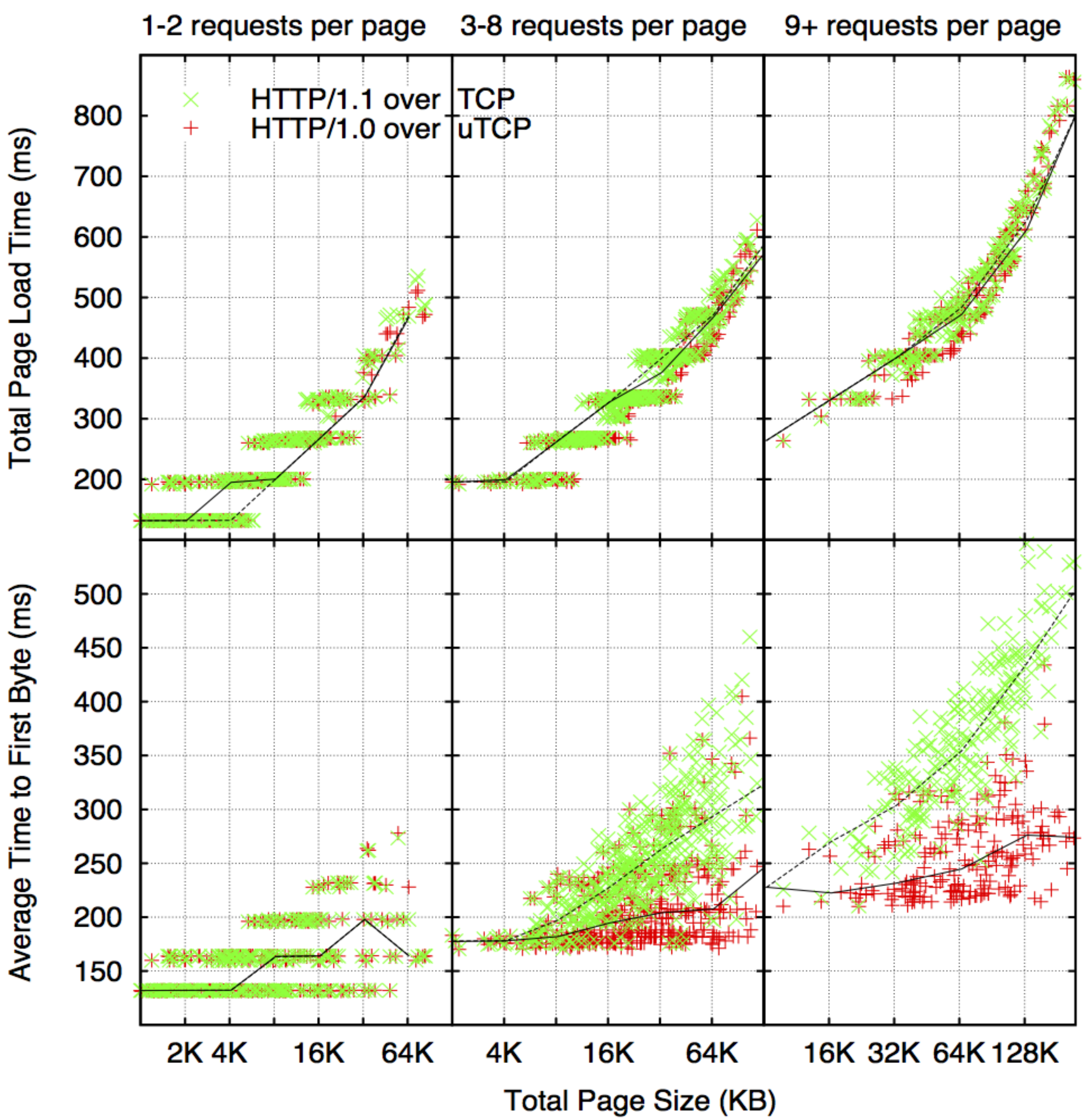
(3Mbps BW; 60ms RTT; 4 TCP flows in background)



Impact on “Real Applications”

- **Example: Web**
  - Independent objects in web pages
  - TCP: parallelism vs. throughput tradeoff
- Multistreaming with Minion
  - ordered streams on top of *u*COBS, 1 per object
  - no Head-of-Line blocking at receiver across streams

User-Perceived Web Latency



Trace-driven, over a network; BW of 1.5Mbps, RTT of 60ms RTT

Much work remains to be done

- Minion goes after transport abstraction issue
  - in an admittedly hack-ish way
  - but aligns incentives well

- Work areas:
  - How to tell reachability between endpoints?
  - Make middleboxes part of the architecture (Align incentives with incremental deployment)
  - Build a middlebox map of the Internet

Papers, etc.

Look for:

T*ng* (Transport Next Generation), **Minion**

http://www.fandm.edu/jiyengar

http://dedis.cs.yale.edu