Square Pegs in a Round Pipe: Wire-Compatible Unordered Delivery In TCP and TLS

Jana Iyengar[*], Bryan Ford[+]
Syed Obaid Amin[*+], Michael F. Nowlan[+], Nabin Tiwari[*]

[*]Franklin & Marshall College

[+]Yale University

Transports come and transports go ...

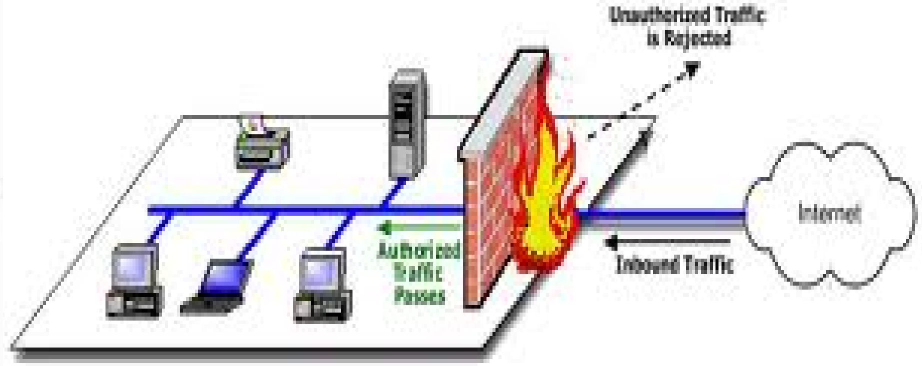
- SCTP
  - multistreaming, message boundaries, multihoming, partial reliability, unordered delivery
  - RFCs 4960, 3257, 3309, 3436, 3554, 3758, 3883 …
  - NAT behavior: draft-stewart-behave-sctpnat
- DCCP
  - Unreliable, congestion-controlled, datagram service
  - RFCs 4336, 4340, 4341, 4342, 5238, 5634, …
  - NAT behavior: RFC 5597

… but the Internet remains loyal!

- TCP and/or UDP get through most middleboxes
  - Only TCP gets through *all* middleboxes
  - ...often only to port 80 (HTTP) or port 443 (HTTPS)!

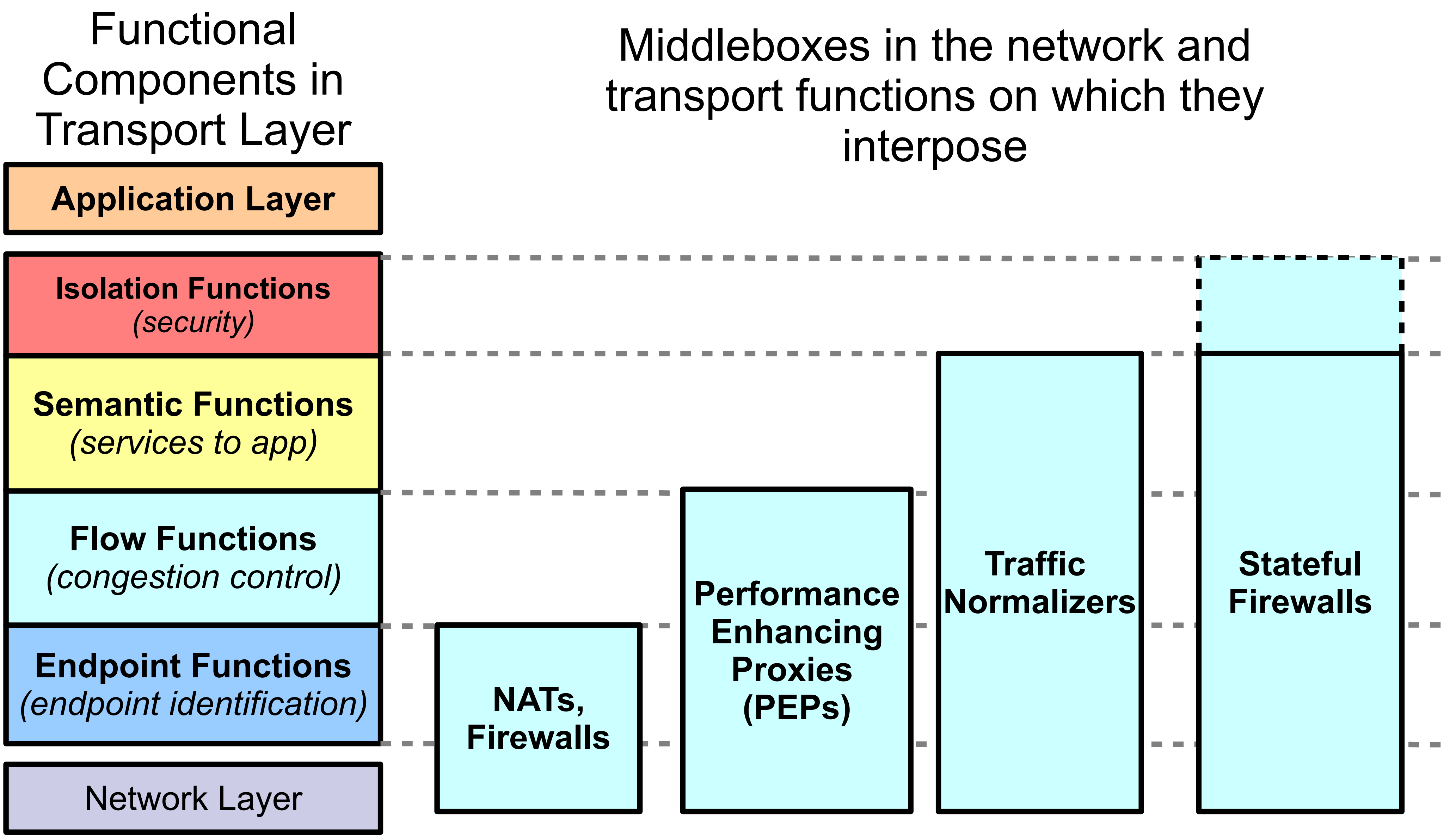
- New & unknown transports *rarely* get through
  - SCTP and DCCP not supported by middleboxes
  - Make it almost impossible to deploy new transports

How deep does this loyalty run?

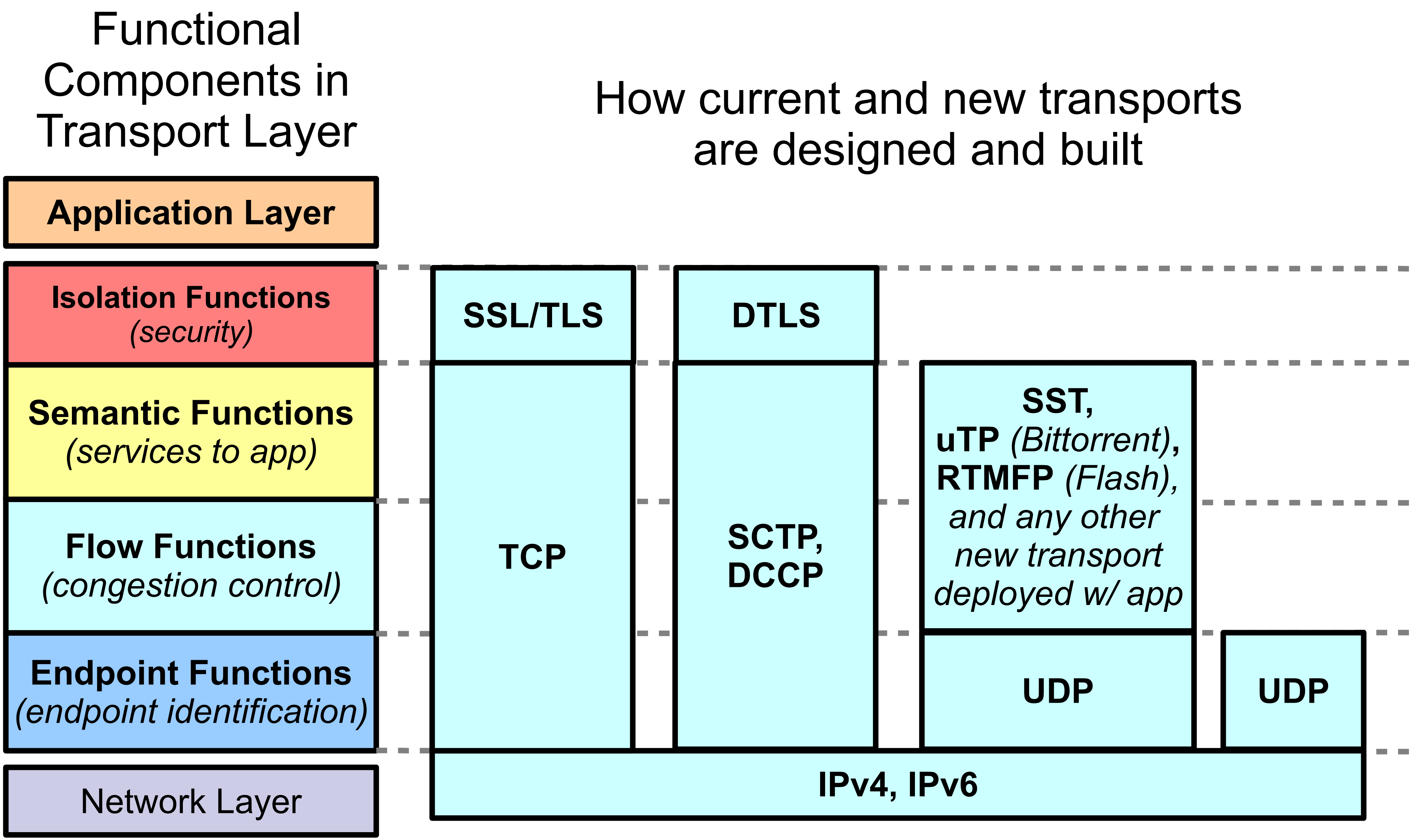


- Network Address Translators (NATs)
  - Cheap and ubiquitous, entrenched in the network
- Firewalls
  - Rules based on TCP/UDP port numbers; often DPI
- Performance Enhancing Proxies (PEPs)
  - Transparently improve TCP (*not* UDP!) performance

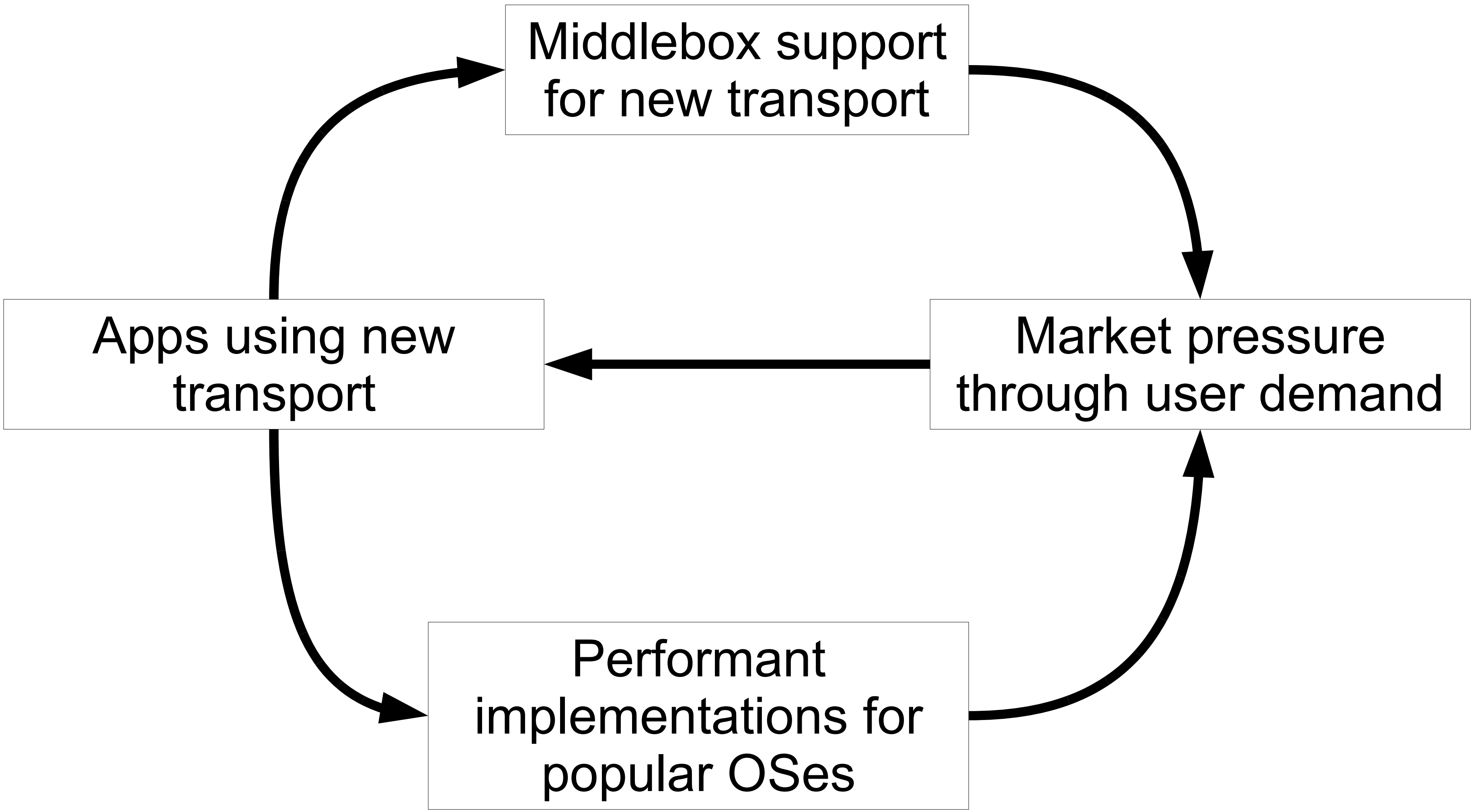
A taxonomy of transport functions



Why does this taxonomy matter?



Deployment Impossibility-Cycles



What have we done so far?

- "NATs are evil. We won't care about them."
- "It will all change with IPv6." ← **Denial**
- "Don't design around middleboxes, that will only encourage them!" ← **Anger**
- "Alright, we'll specify how middleboxes *ought* to behave with different protocols. But they still have to behave." ← **Bargaining**
- "Why build a new transport?? It won't get deployed anyways." ← **Depression***

**Kübler-Ross model: Five stages of grief*

The final stage: Acceptance

- Design assumptions for new transport services:
  - New transport services should *require* modifications *only* to end hosts
  - Middleboxes are here to stay
- Consequences:
  - New end-to-end services *should not require* changes to middleboxes.
  - New end-to-end services must use protocols that appear as legacy protocols on the wire.
- Eg: MPTCP

The Minion Suite

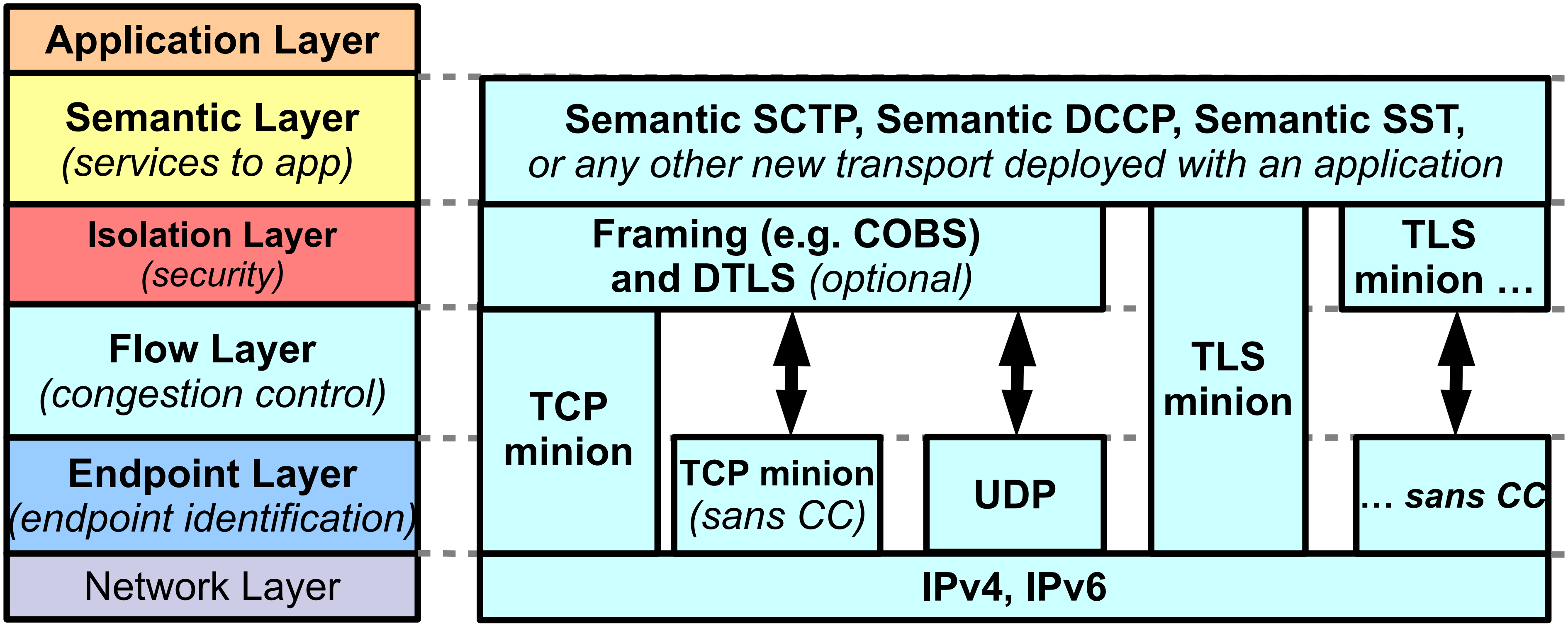
A “packet packhorse” for deploying new transports

- ***Uses legacy protocols …***
  - TCP, UDP
- ***… as a substrate …***
  - turn legacy protocols into *minions* offering unordered, unreliable datagram service
- ***… for building new services that apps want***
  - multistreaming, message boundaries, unordered delivery, app-defined congestion control
  - *(working on: stream-level receiver-side flow control, priority streaming, multipath, partial reliability)*

Outline

- Minion: a packet packhorse for new transports
  - Carry new transports over Internet's rough terrain
- TCP Minion: unordered delivery in TCP
  - Making datagram service look like a TCP stream
- TLS Minion: unordered delivery in SSL/TLS
  - Making datagrams *indistinguishable* from HTTPS
- Next steps

What's in the Minion Suite?

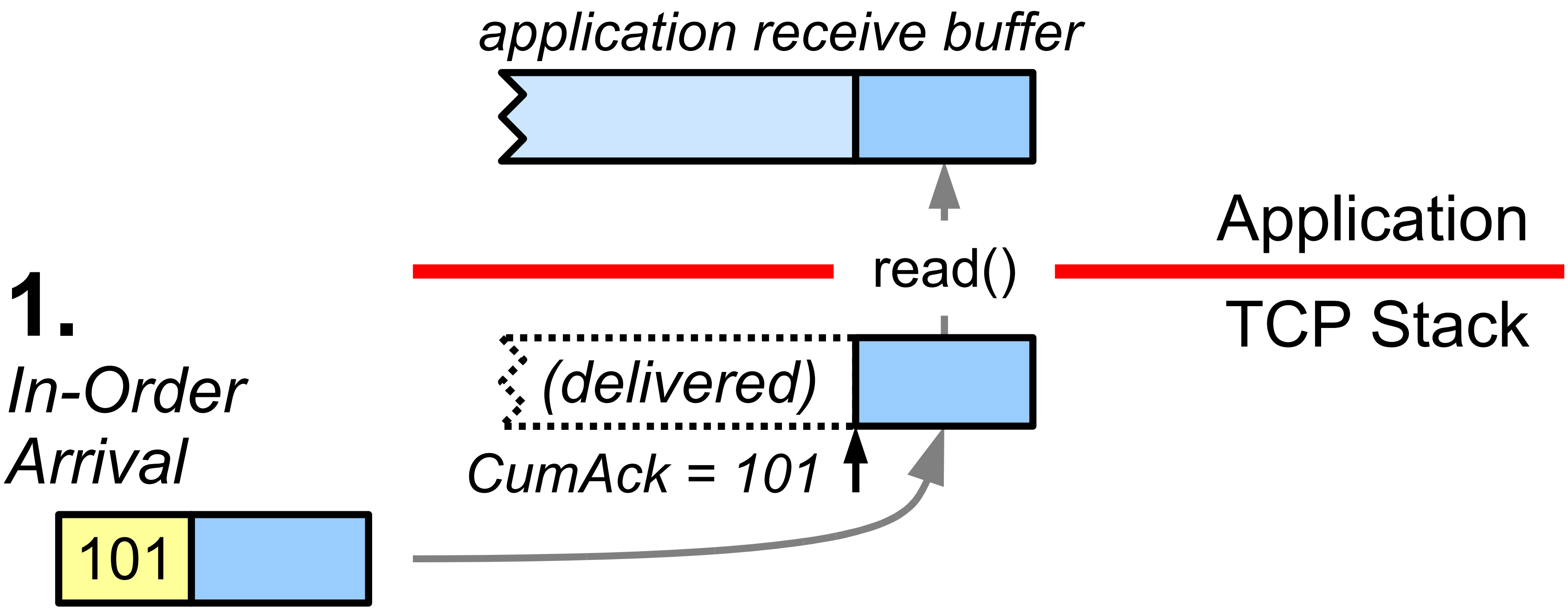


- Break up the functions of the legacy transport layer
  - “Breaking Up the Transport Logjam”, HotNets '08
- Use legacy protocols as compatible building blocks
- We'll focus here on TCP minion (and a summary of TLS)

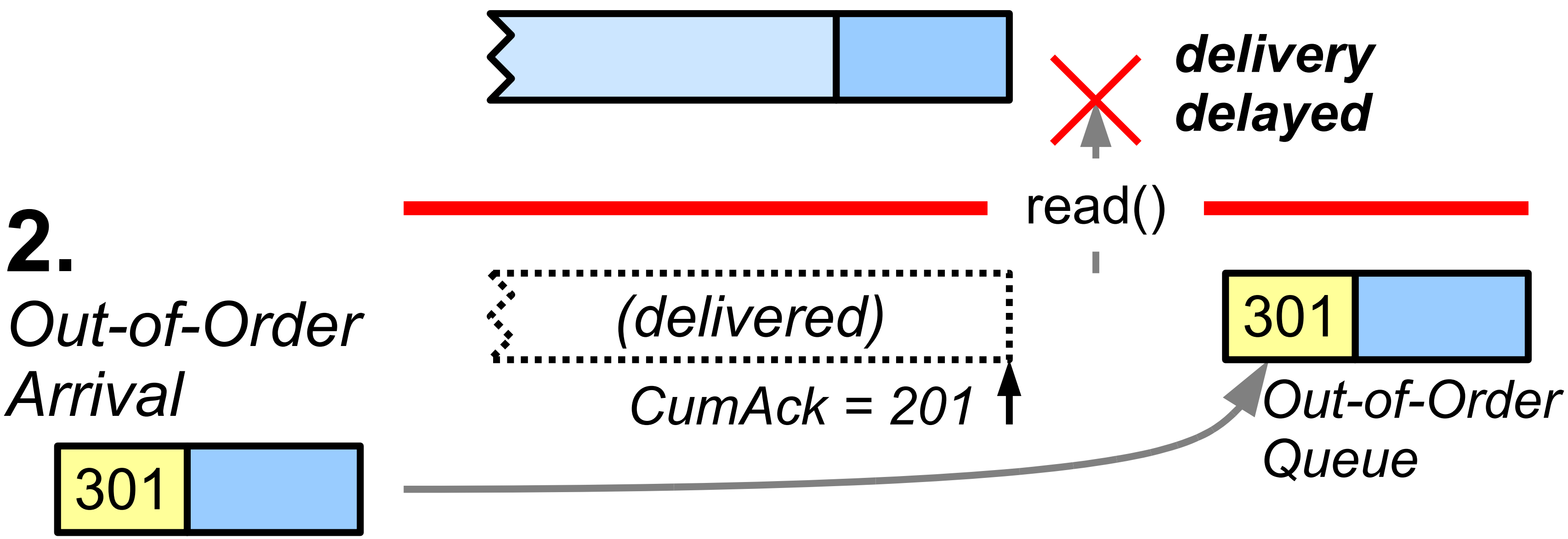
TCP Minion

- Retain TCP protocol semantics on the wire
  - Middleboxes *cannot* distinguish from normal TCP
  - ...*except* by looking into application payload
    - we'll address this “except” later in TLS Minion
- Offer datagram service to apps, new transports
  - Out-of-order delivery
    - Minimize delay for latency-sensitive applications: e.g., voice/videoconferencing, VPN tunneling, ...
  - Eliminate nasty “TCP-on-TCP” tunneling effects
    - No broken connections due to “retransmission overload”
- By adding 1 new TCP socket option...

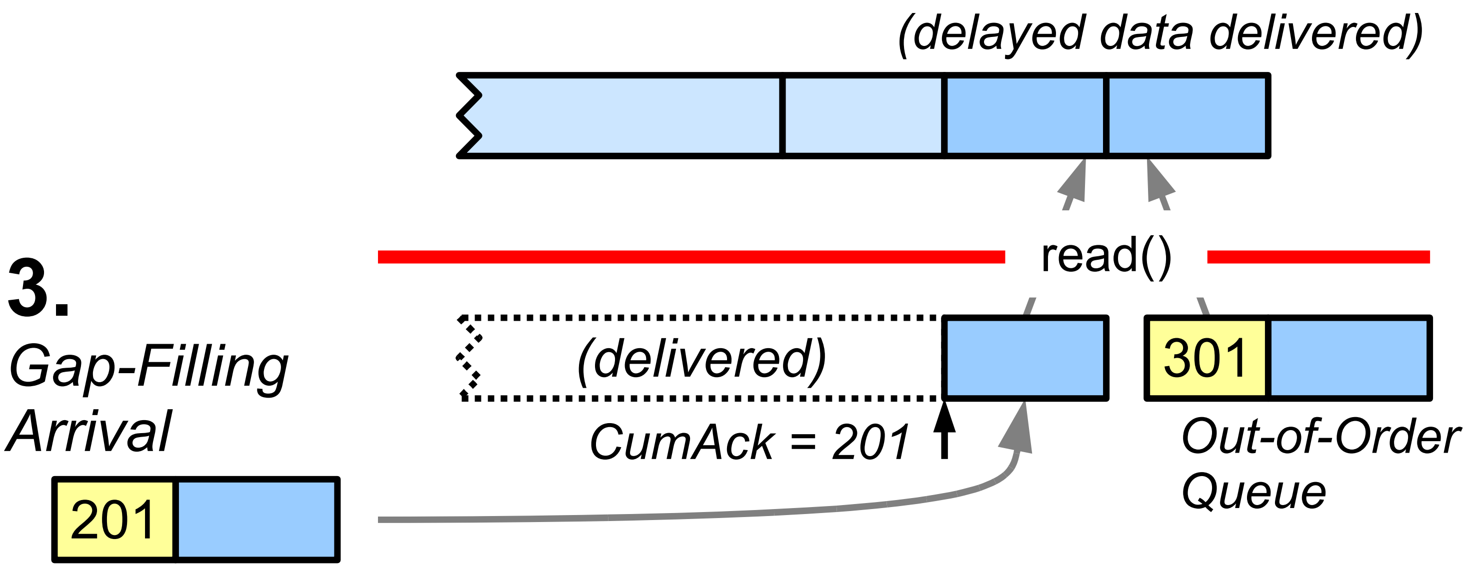
Delivery in Standard TCP



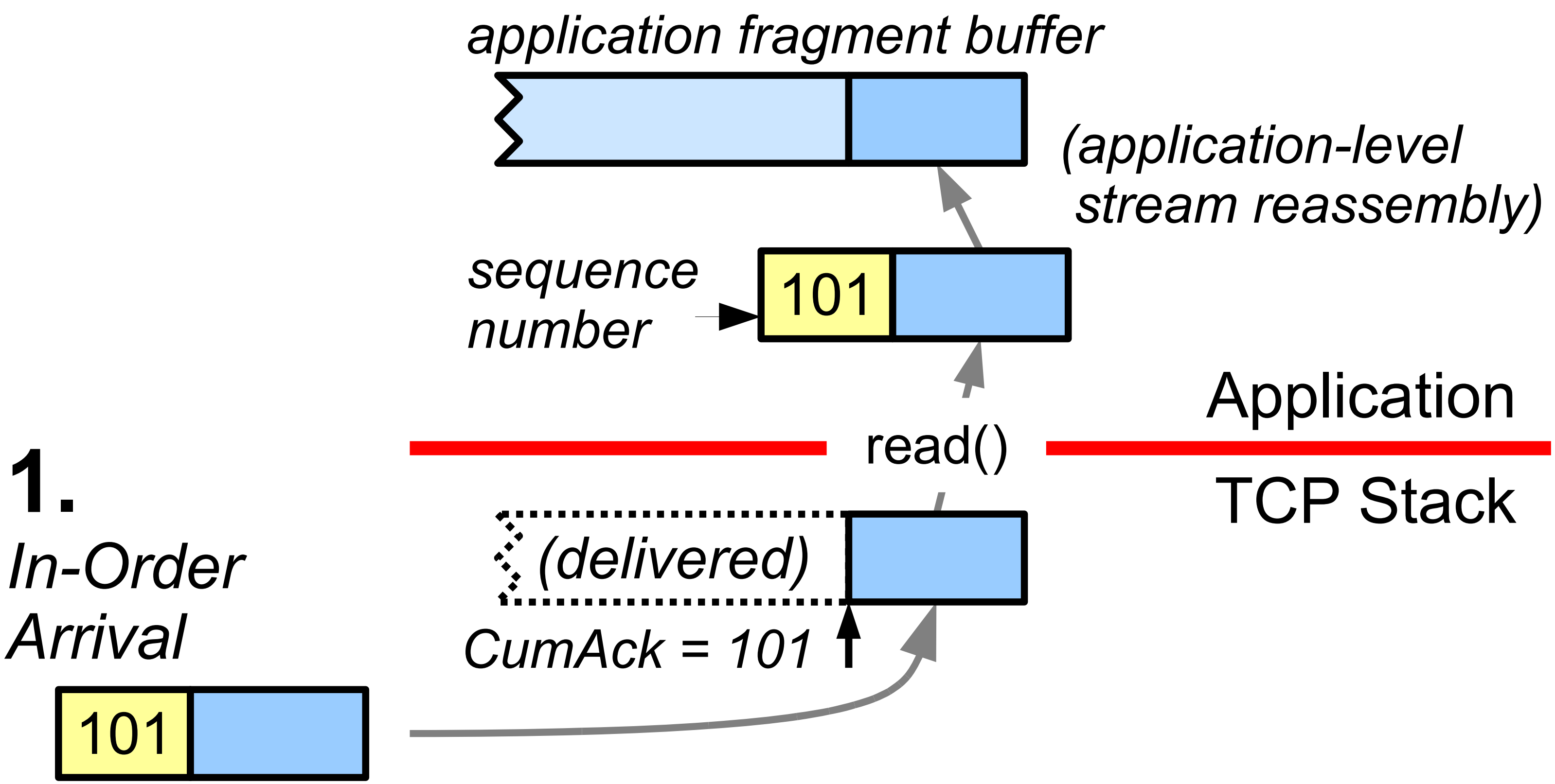
Delivery in Standard TCP



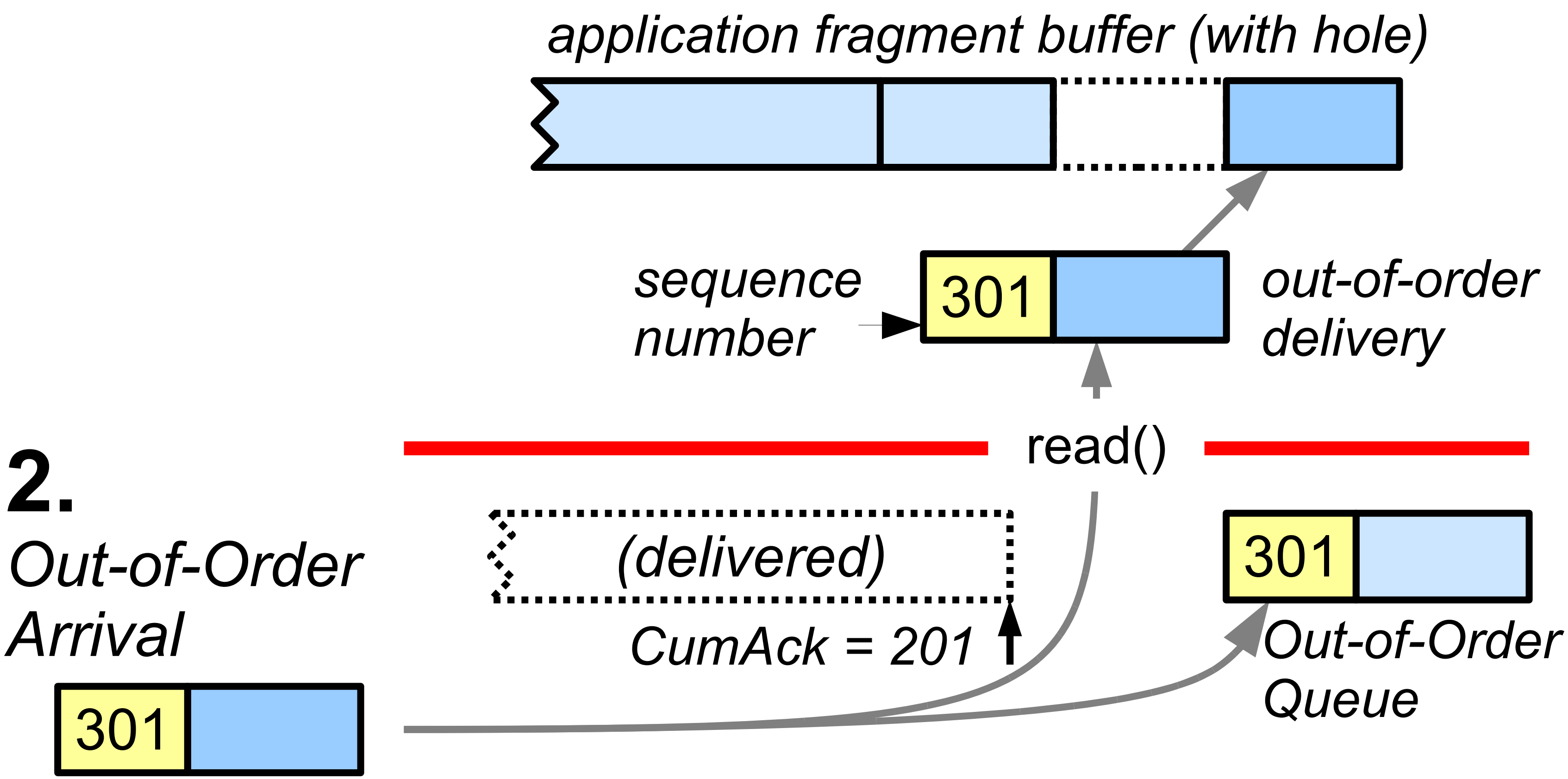
Delivery in Standard TCP



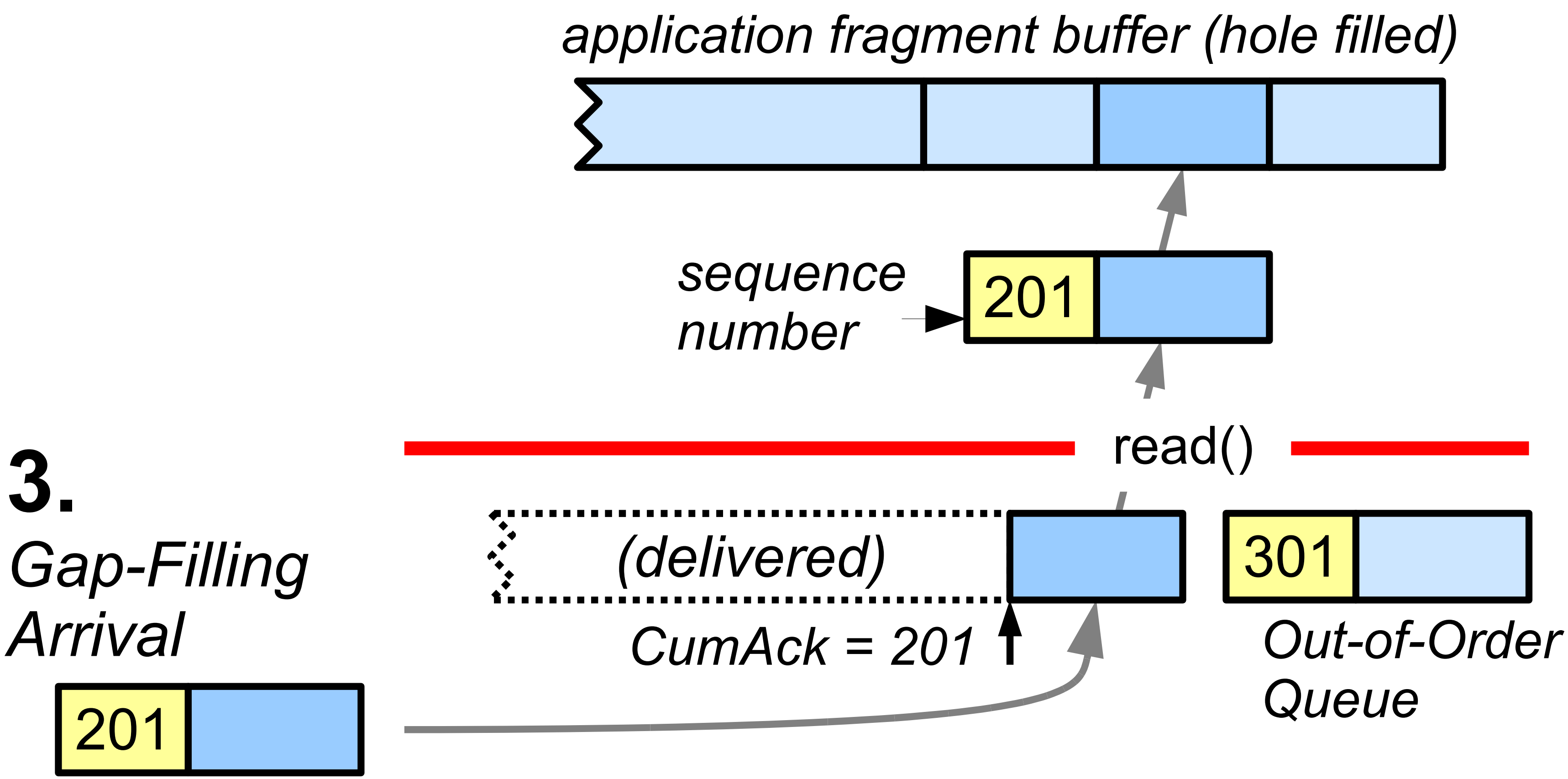
Delivery in TCP Minion



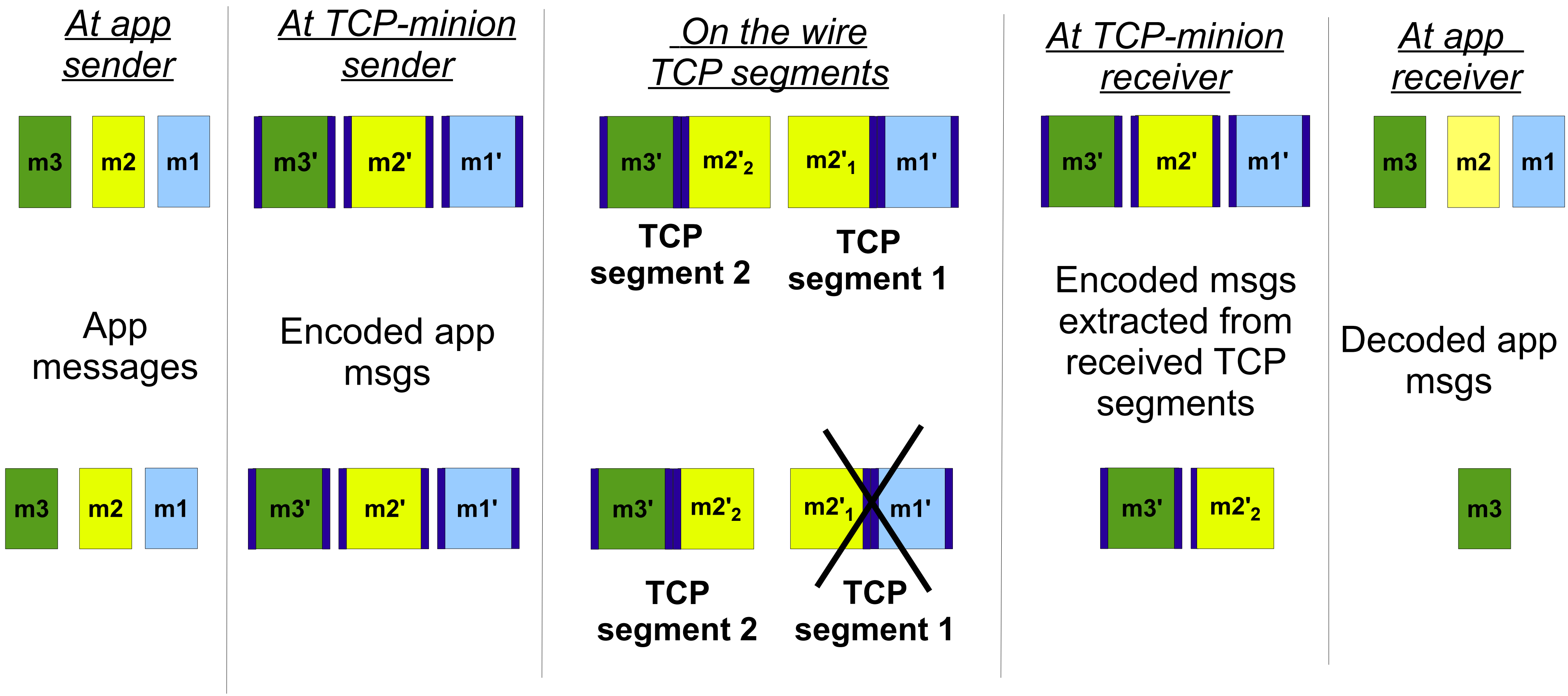
Delivery in TCP Minion



Delivery in TCP Minion



Problem: Network Resegmentation

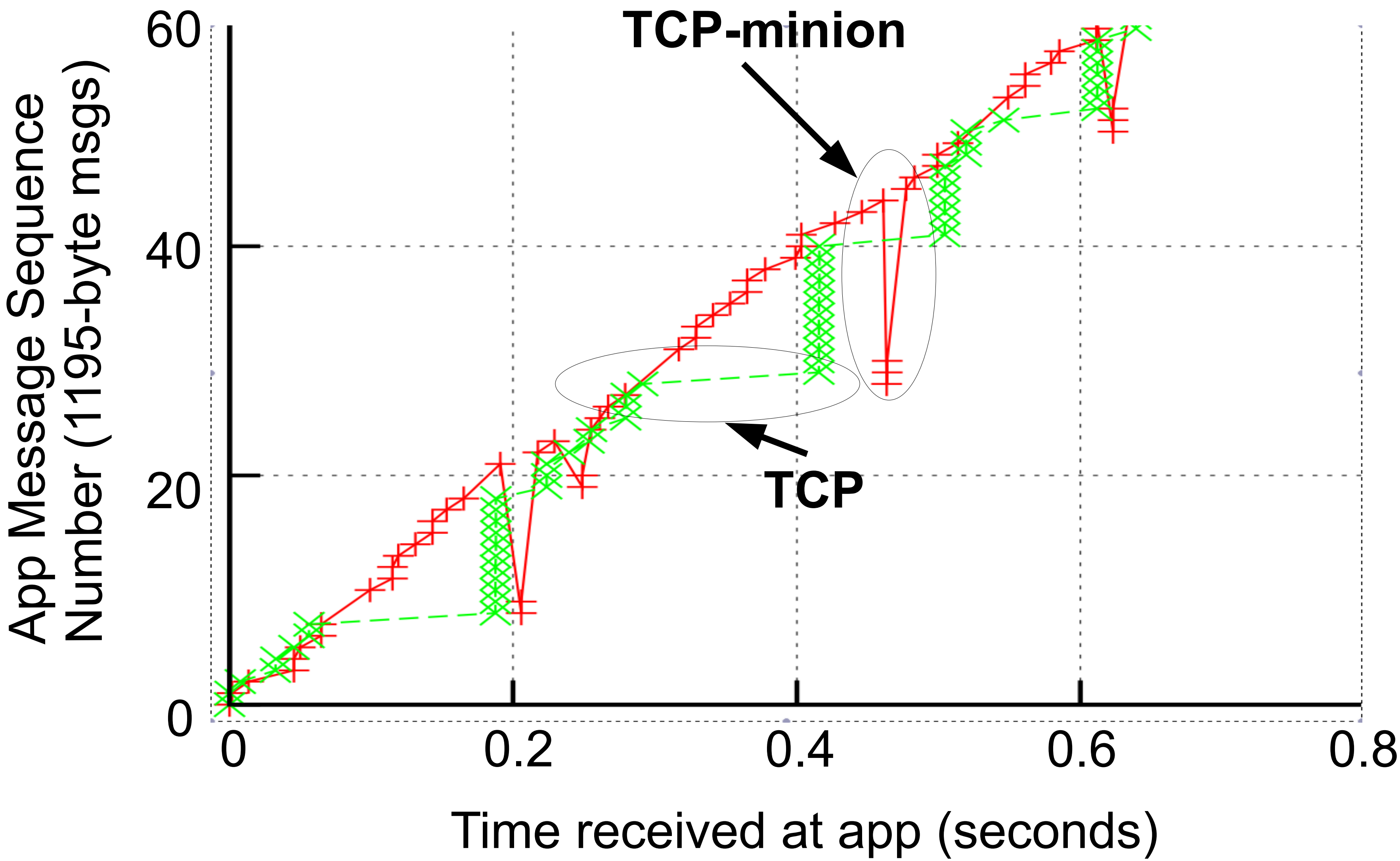


COBS encoding

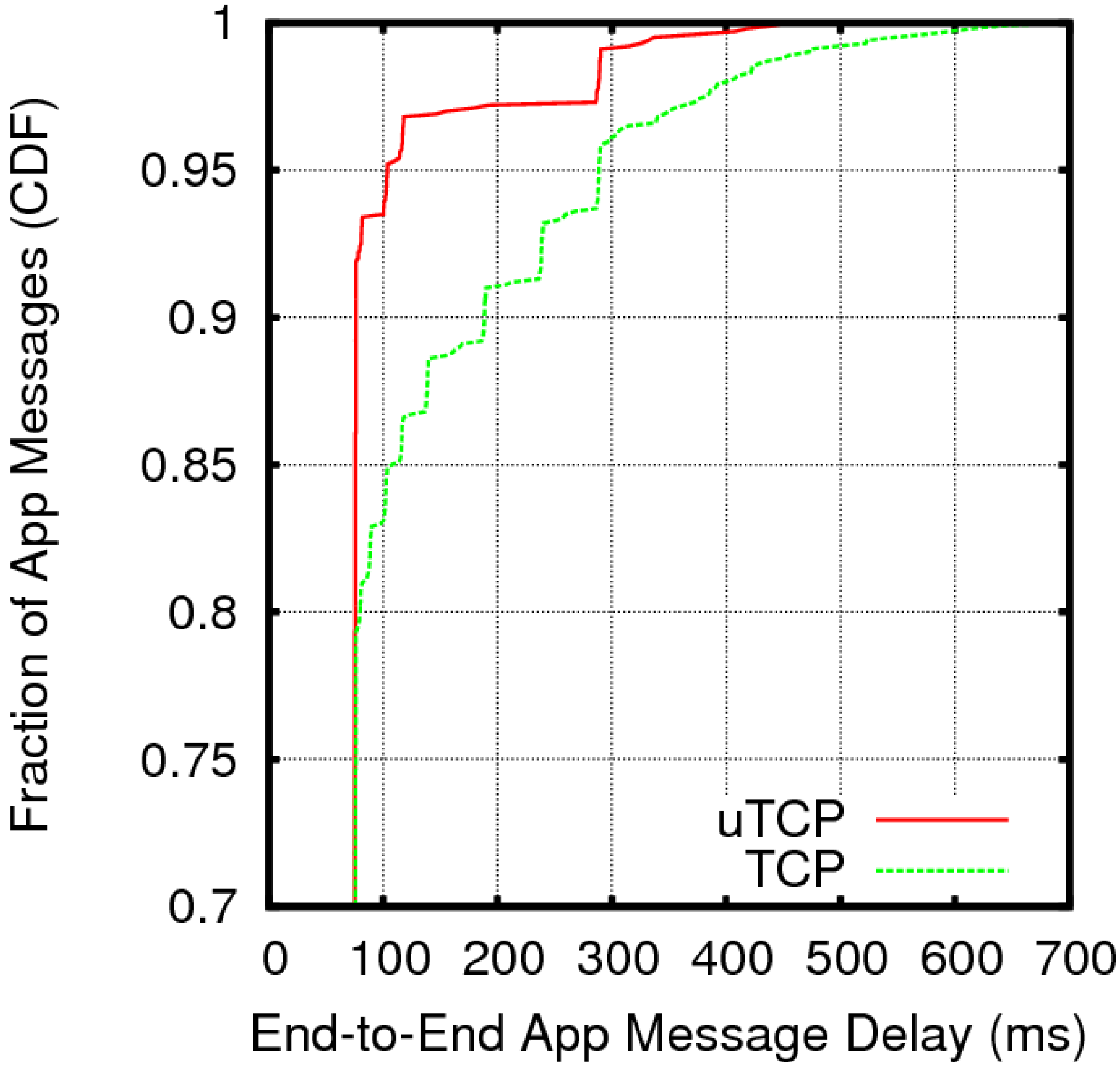
- Size-preserving encoding that eliminates all occurrences of delimiter byte from payload
  - Max overhead of 0.4% (6 bytes for 1448-byte msg)
  - Delimiter byte then inserted between messages
  - Receiver extracts messages, decodes, delivers up

- We make one modification
  - We insert delimiter byte both before *and* after msg
    - Increases max overhead to 0.8%
  - To deal with common cases for apps
    - App sends only one message (eg: HTTP GET req)
    - Each app msg gets encap'd in its own TCP segment

App messages with TCP (TLV encoding) vs. TCP-minion



App-Observed Delay Distribution

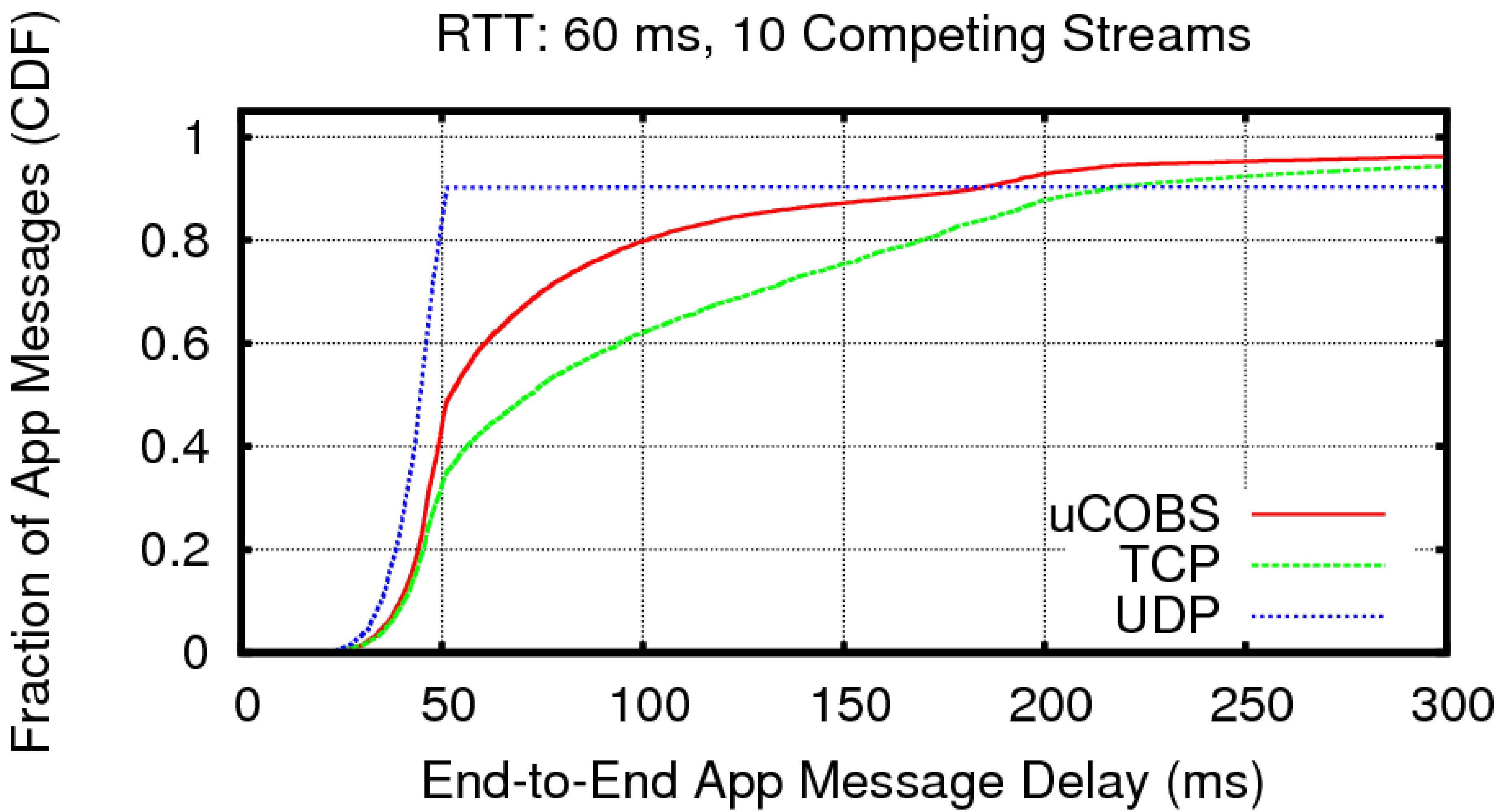


Impact on “Real Applications”

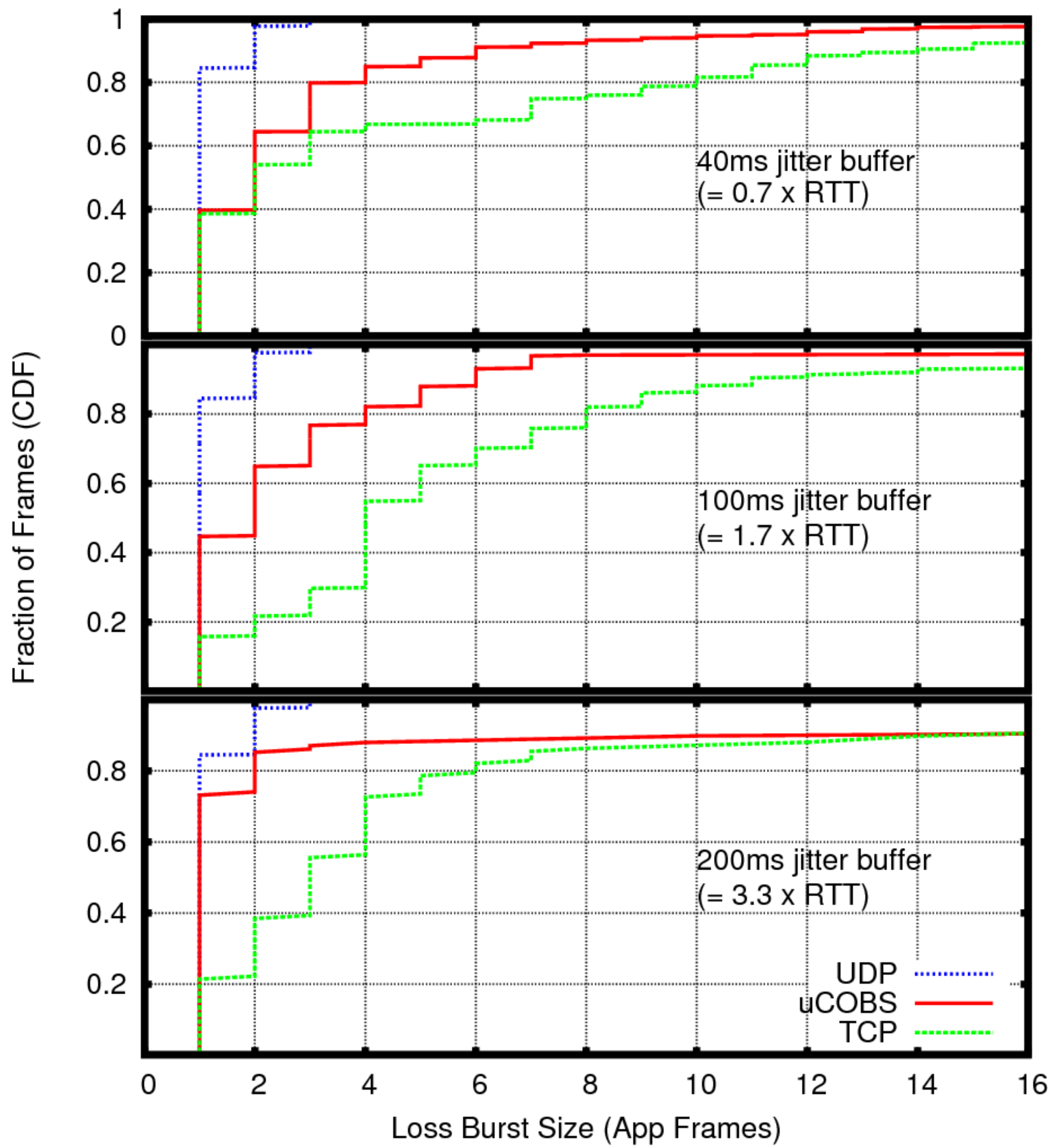
Example: Voice-over-IP (VoIP)

- Voice/videoconferencing is delay-sensitive
  - Long round-trip delays perceptible, frustrate users
- Modern VoIP codecs tolerate *individual* losses
  - Interpolate over 1 or 2 lost packets
- But are highly sensitive to *burst* losses
  - Can't interpolate when many packets lost/delayed!

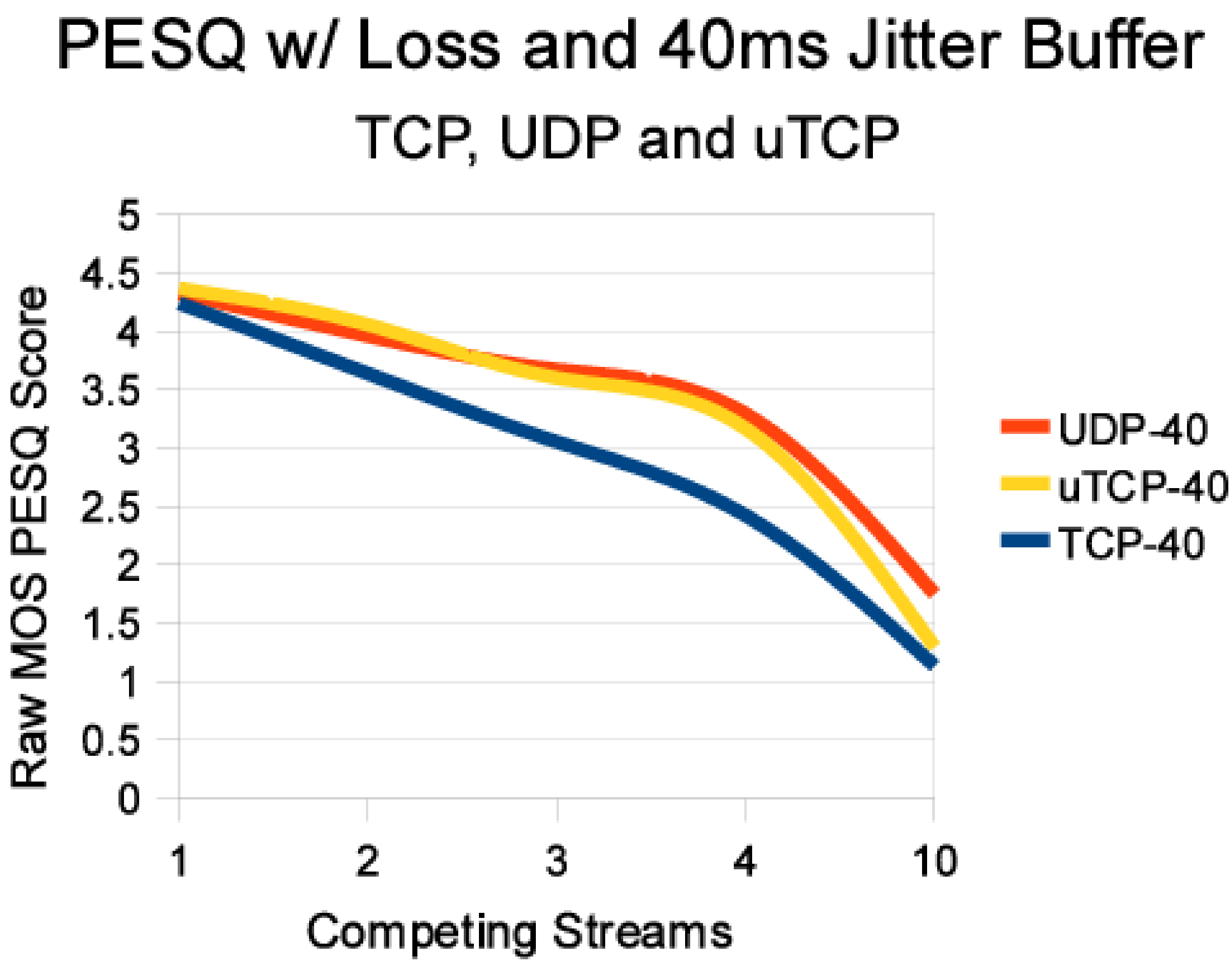
VoIP application: observed delay



VoIP: distribution of burst loss/delay



VoIP: perceptual quality impact

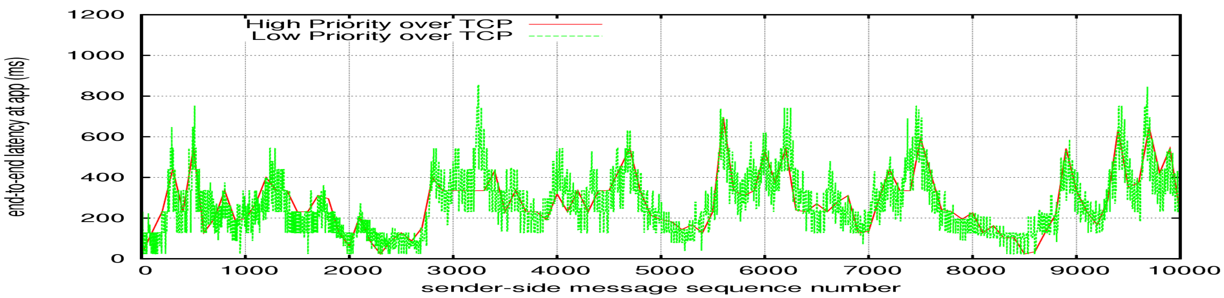


TCP Minion: What's next

- Better control over sender-side buffering
  - Work in progress
  - Initial Linux-based prototype allows priority-queueing of app messages within socket buffer.

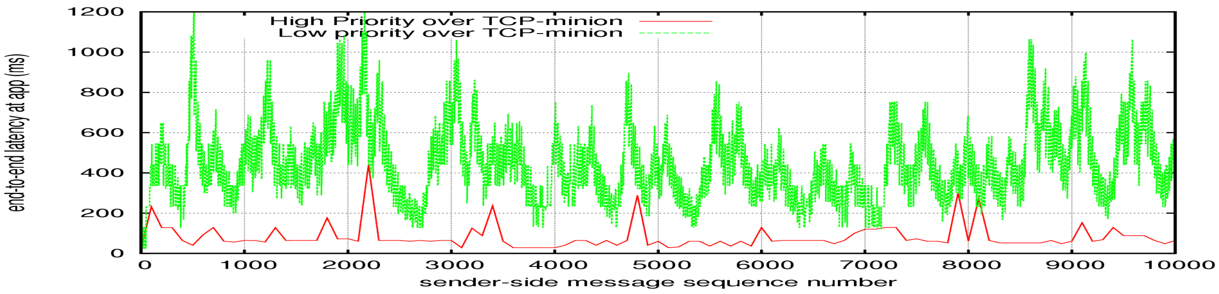
- Testing underway to measure effects with both sender- and receiver-side modifications

App with message priorities



- 1000-byte messages
- every 100^{th} message is high priority
- 100ms RTT
- 1% loss at bottleneck

App with message priorities



1000-byte messages

every 100th message is high priority

100ms RTT

1% loss at bottleneck

TLS Minion (Summary)

- TLS-minion protects end-to-end signaling and data
  - appears as SSL/TLS on the wire, *but*
  - provides out-of-order datagram service
- Makes stream indistinguishable from, e.g., HTTPS
  - even to middleboxes that inspect *all* app payloads!
  - only *encrypted* content affected
- Technical Challenges:
  - TLS records not encoded for out-of-order decoding
  - Ciphersuites chain encryption state across records
  - MACs use implicit record counter, hard to recover

Our implementation of the minions

- Some inside Linux kernel
  - Added SO_UNORDERED sockopt to SOCK_STREAM
  - On receiver-side:
    - subsequent read()s results in a *contiguous byteblock* being returned, without regard to order
    - TCP sequence number returned with byteblock
    - Only one kernel change required
  - On sender-side:
    - write() now includes *msgid* for queueing message by kernel
- Userspace library for rest of TCP- and TLS-minion
  - reassembles fragmented streams
  - extracts message, decodes, and delivers to app
  - library → can ship as part of apps

In Conclusion

- **TCP, TLS work on the Internet**
  - mature, performant implementations
  - *workhorses* of the Internet
  - but in-order delivery bad for delay

- **We can fit square pegs (packets) through a round pipe (TCP, TLS)**
  - Eliminates in-order delivery delays
  - Most mods deployable with apps
  - Turn workhorses into *packhorses*!

Minion encourages adoption of new transports

- Minion allows new *services* to be created and deployed in a legacy environment.
  - Does not prevent native deployment of new protocols.
  - Encourages adoption of new protocols by middleboxes and OSes through use of new services by apps *before* middlebox/OS support is available.
- WIP: Ends need to detect *protocol-graph* supported by endpoints *and by middleboxes*
  - Negotiation Service (HotNets '09)
  - “Happy Eyeballs” on steroids