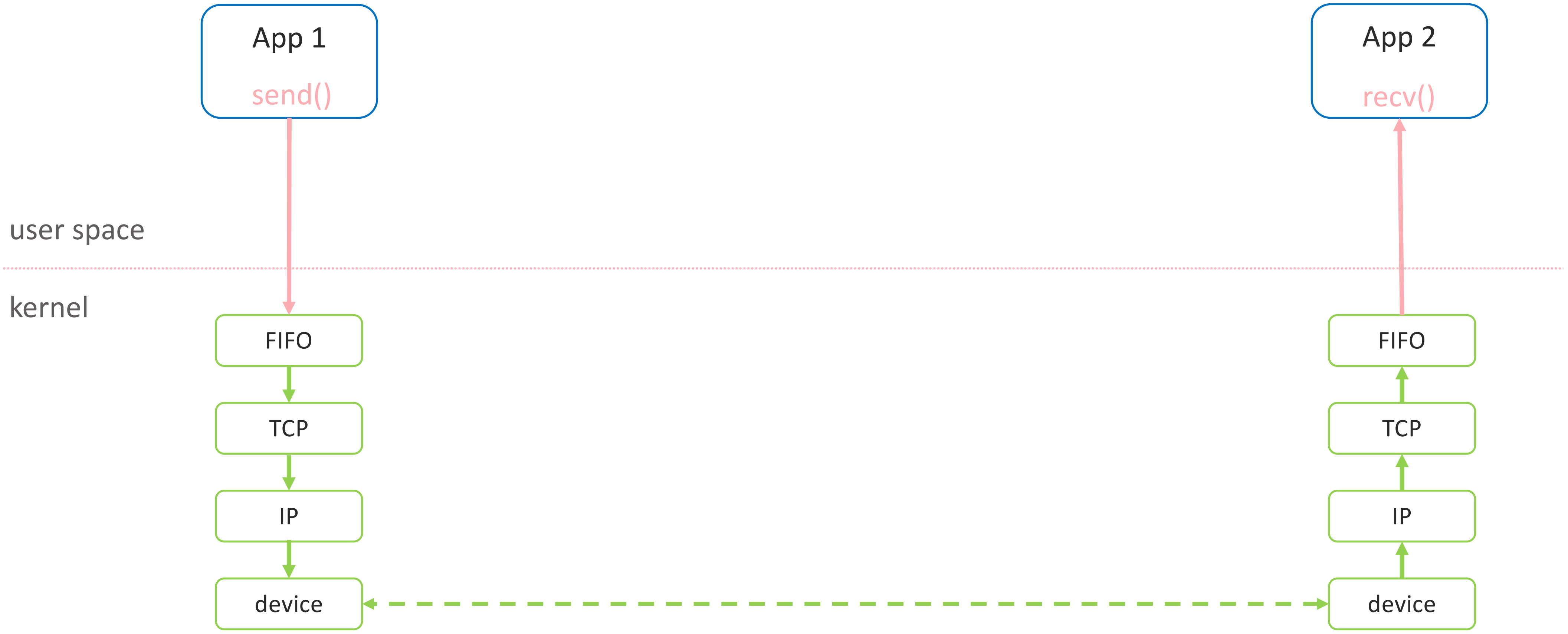
VPP Host Stack

Transport and Session Layers

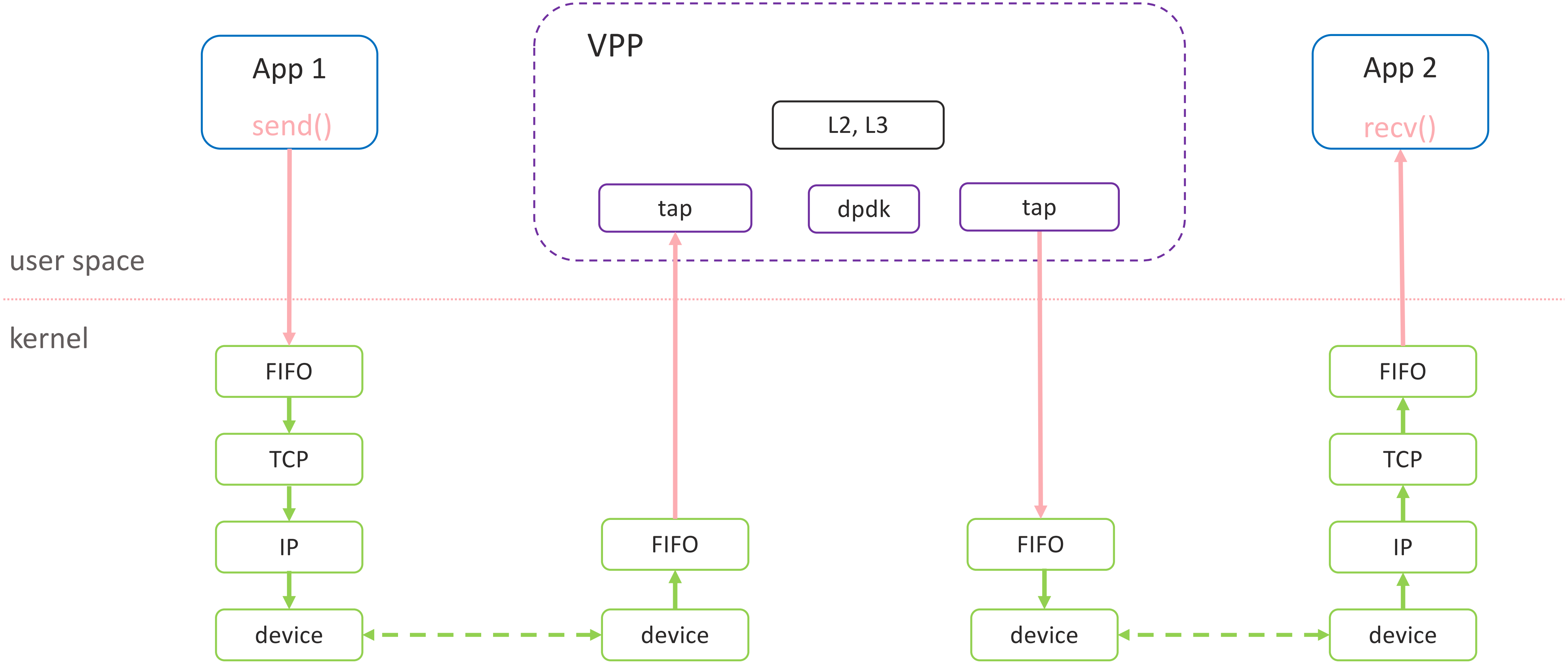
Florin Coras, Dave Barach

Motivation

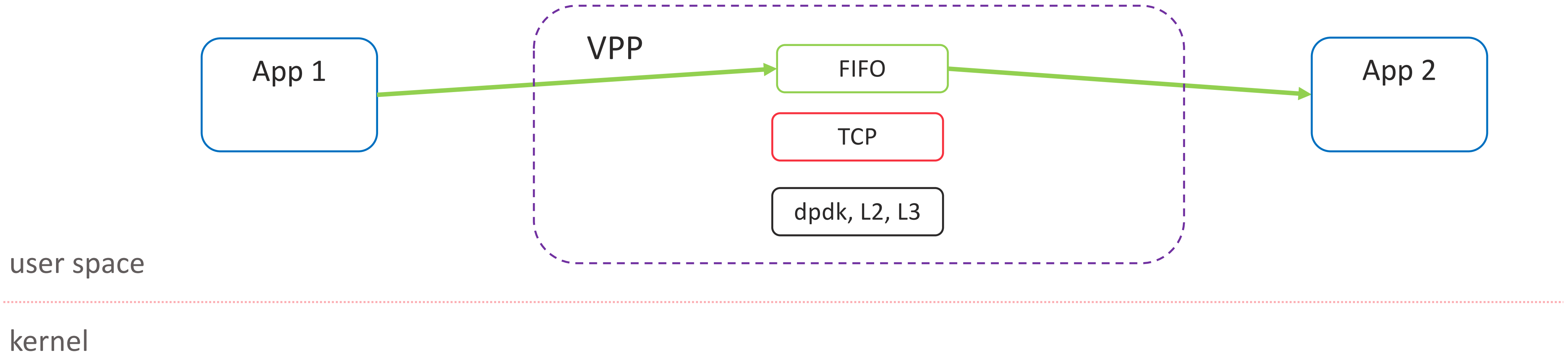


App 1
send()
App 2
recv()
user space
kernel
FIFO
TCP
IP
device
FIFO
TCP
IP
device

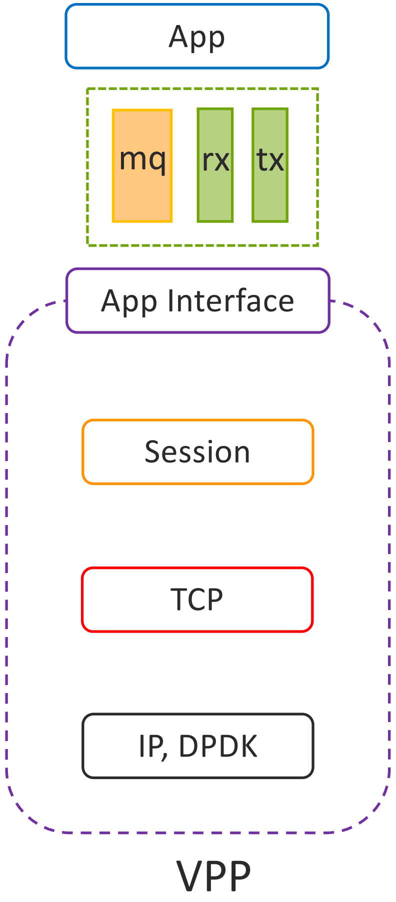
Motivation


VPP
App 1
send()
L2, L3
App 2
recv()
tap
dpdk
tap
user space
kernel
FIFO
FIFO
TCP
TCP
IP
FIFO
FIFO
IP
device
device
device
device

Why not this?

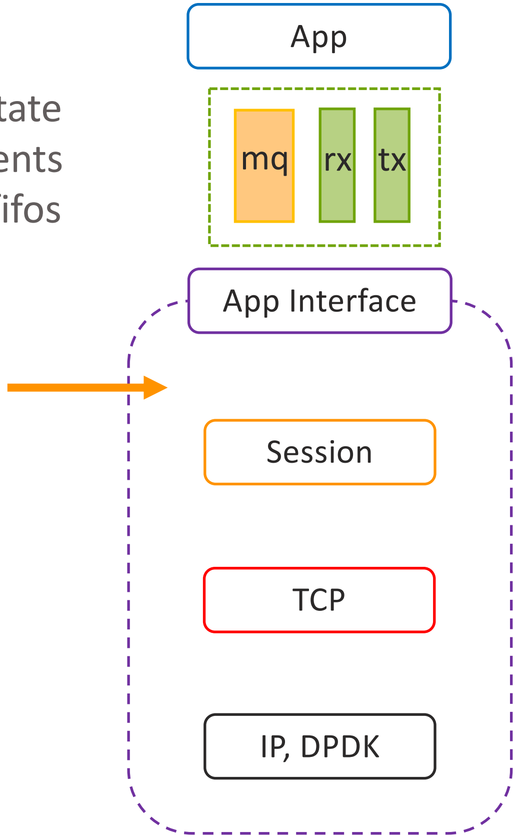


VPP Host Stack



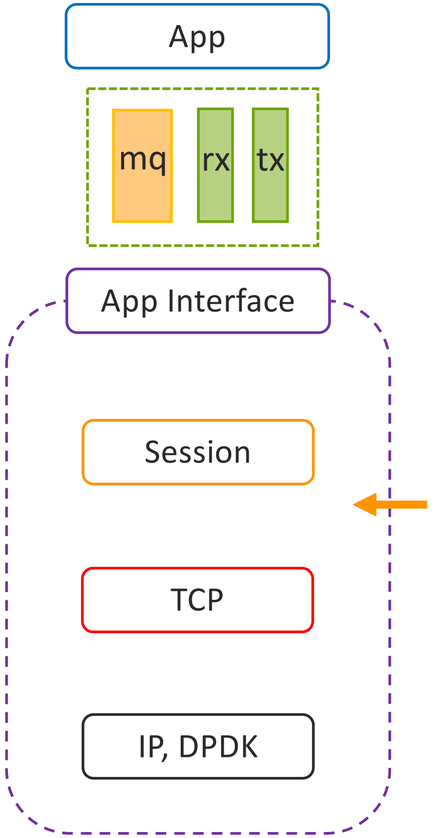
VPP Host Stack: Session Layer

- App-interface sub-layer maintains per app state and offers support for conveying session events
- Allocates and manages sessions/segments/fifos
- Handles segmentation of data into buffers before sending it to transport protocols
- Binary/native C API for external/builtin applications



VPP Host Stack: Session Layer

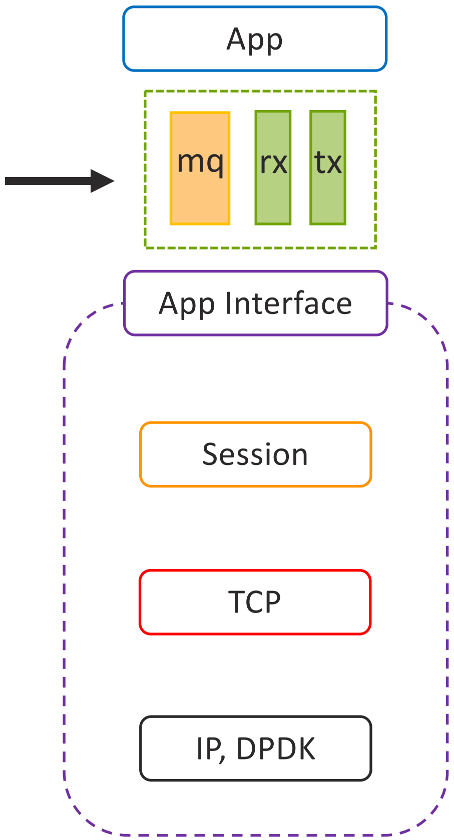
- App-interface sub-layer maintains per app state and offers support for conveying session events
- Allocates and manages sessions/segments/fifos
- Handles segmentation of data into buffers before sending it to transport protocols
- Binary/native C API for external/builtin applications



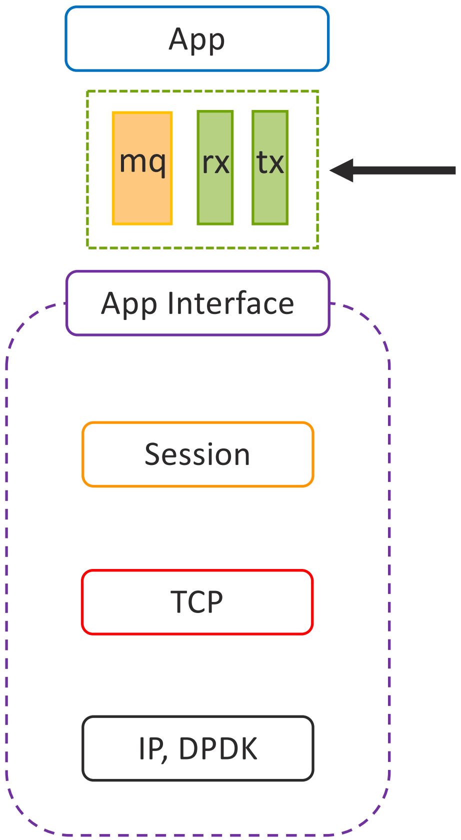
- Session lookup tables (5-tuple) and local/global session rule tables (filters)
- Support for pluggable transport protocols
- Can do tx-pacing if transport asks for it
- Offers API for enqueueing data for the apps
- Isolates network resources via namespacing

VPP Host Stack: SVM

- Shared memory segments:
  - Allocated by the app-interface sub-layer and mapped by applications
  - Preferred without file backing (memfd). Support for segments with file backing (shm) will be deprecated



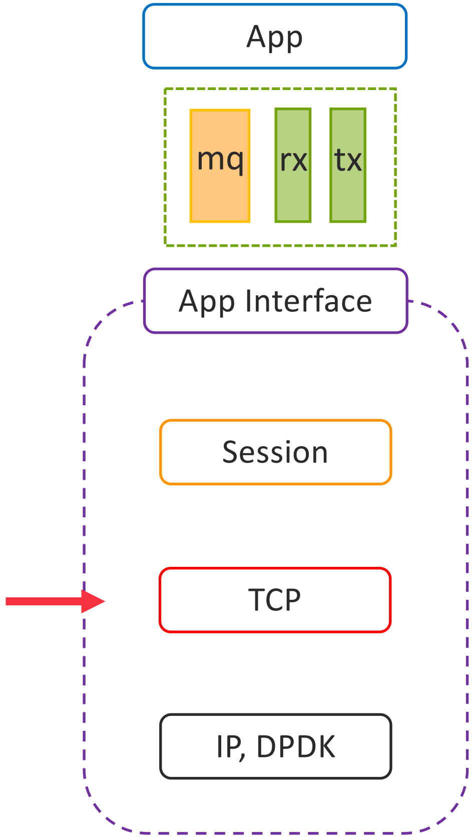
VPP Host Stack: SVM



- message queue
  - Allocated in first shared memory segment
  - Has two rings for control and io events from vpp to app
  - Supports condvar or eventfd notifications
- fifos
  - Allocated within shared memory segments
  - Fixed position and size
  - Lock free enqueue/dequeue but atomic size increment
  - Option to dequeue/peek data
  - Support for out-of-order data enqueues
  - Access data as segments

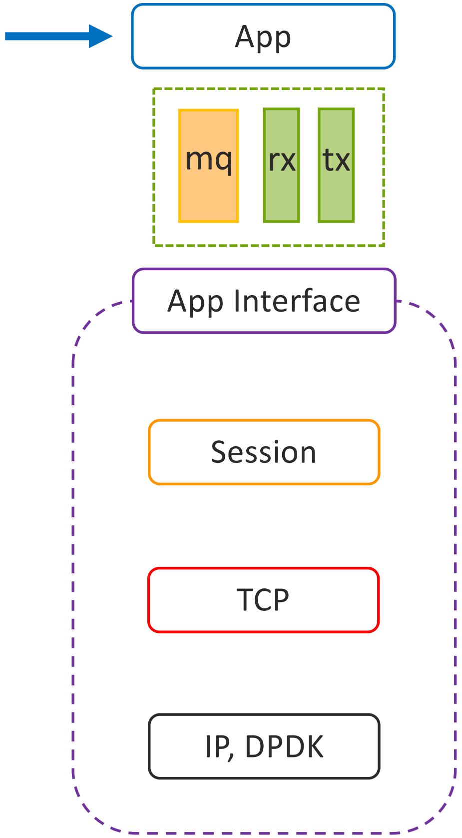
VPP Host Stack: TCP

- Clean-slate implementation
- “Complete” state machine implementation
- Connection management and flow control (window management)
- Timers and retransmission, fast retransmit, SACK
- NewReno and Cubic congestion control, SACK based fast recovery
- Tx pacing
- Checksum offloading
- Linux compatibility tested with IWL TCP protocol tester

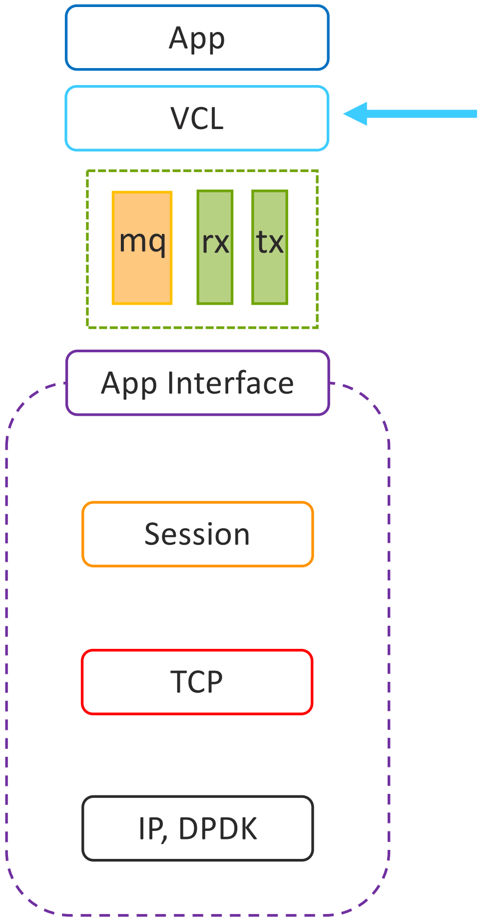


VPP Host Stack: Comms Library (VCL)

- Apps can directly use the raw session layer APIs but then need to:
  - Manage binary api and message queue interaction with vpp
  - Maintain session state, potentially deal with thread safety
  - Implement async communication mechanisms



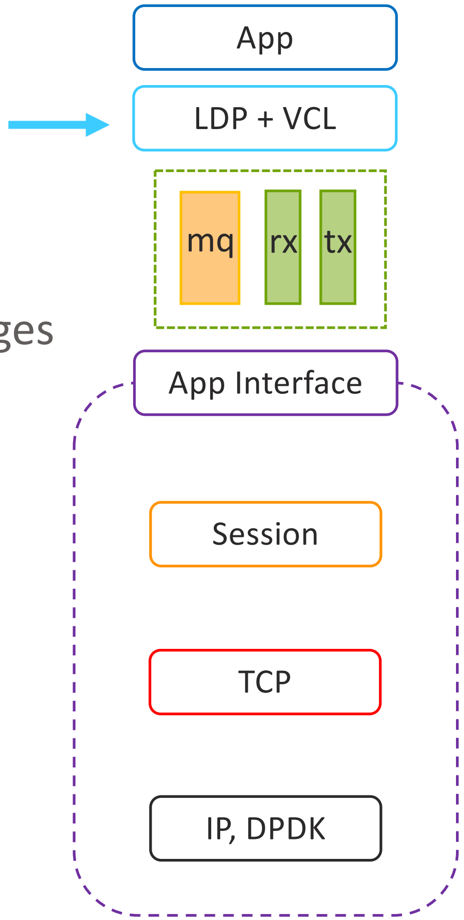
VPP Host Stack: Comms Library (VCL)



- VPP Comms library (VCL)
  - Manages interaction with session layer
  - Abstracts sessions to integer session handles
  - Exposes epoll/select/poll functions
  - Supports multi-threaded and multi-process applications
  - Can handle mq notifications with both mutex-condvar pair and eventfds

VPP Host Stack: Comms Library (VCL)

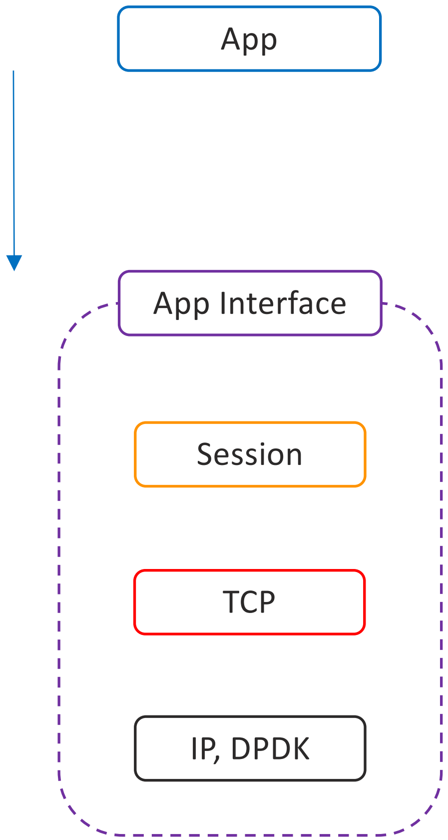
- LDP library
  - Uses LD_PRELOAD to intercept and redirect syscalls to VCL
  - Manages fd to session handle translation
  - When it works, it requires no changes to applications
  - Do not expect it to always work



- VPP Comms library (VCL)
  - Manages interaction with session layer
  - Abstracts sessions to integer session handles
  - Exposes epoll/select/poll functions
  - Supports multi-threaded and multi-process applications
  - Can handle mq notifications with both mutex-condvar pair and eventfds

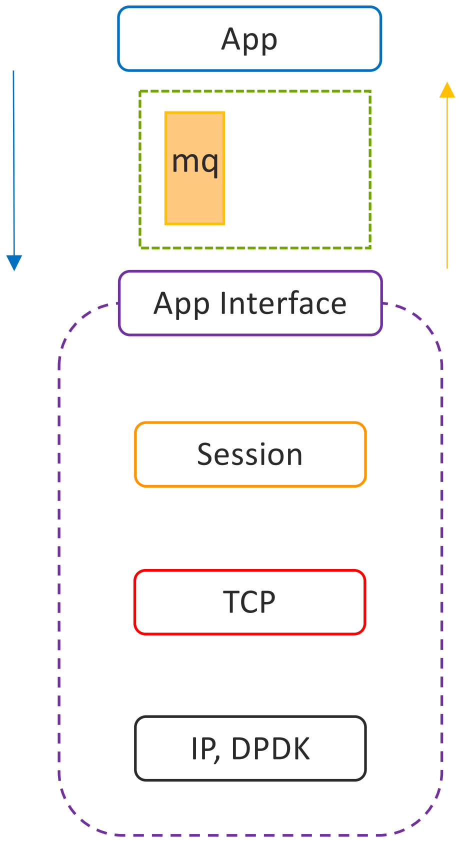
Application Attachment

- Connect to binary API
- Request attachment to session layer



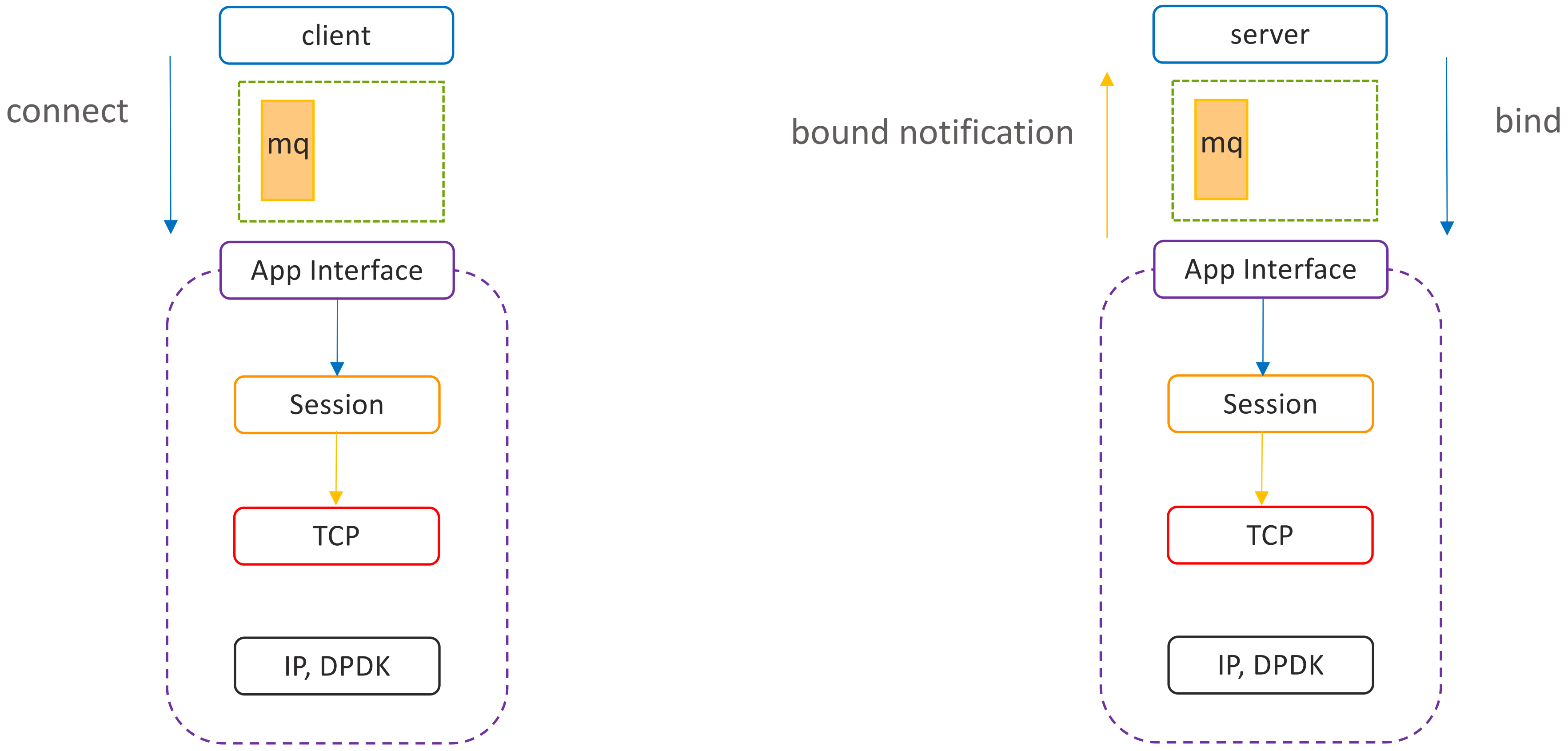
Application Attachment

- Connect to binary API
- Request attachment to session layer

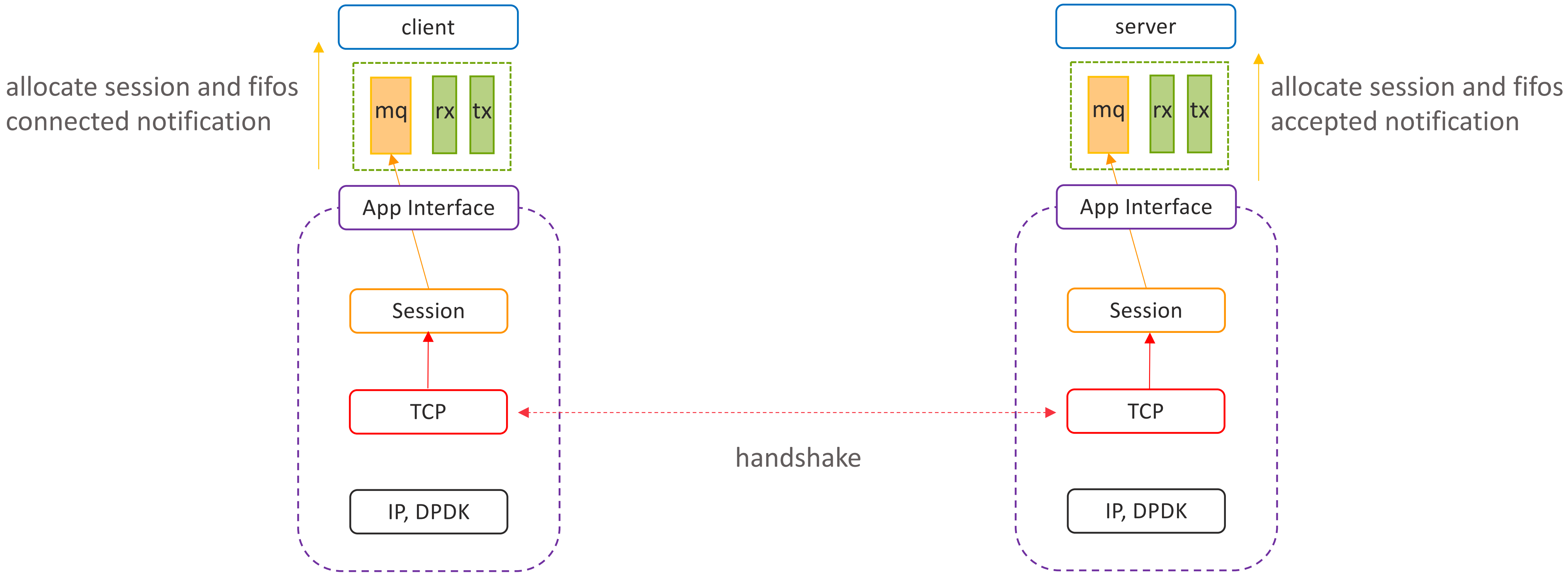


- Session layer allocates shared segment and message queue
- App mounts segment and starts listening for events on msg queue

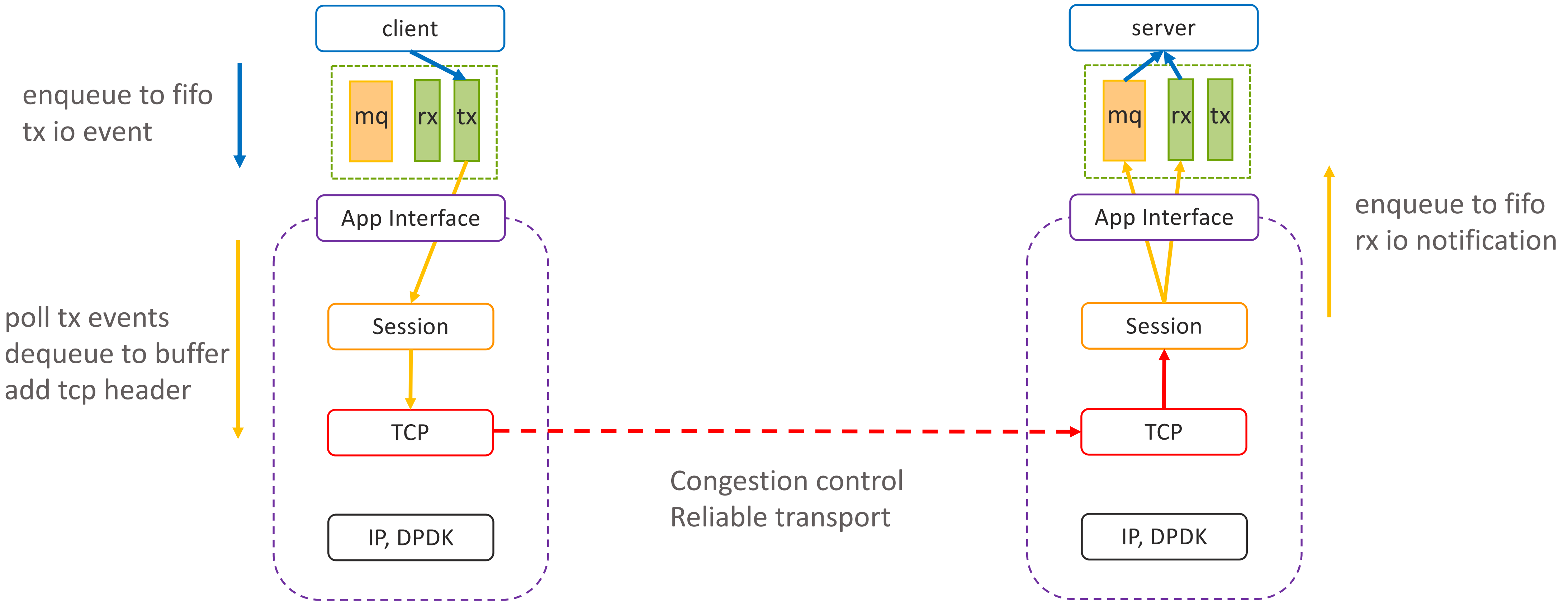
Session Establishment



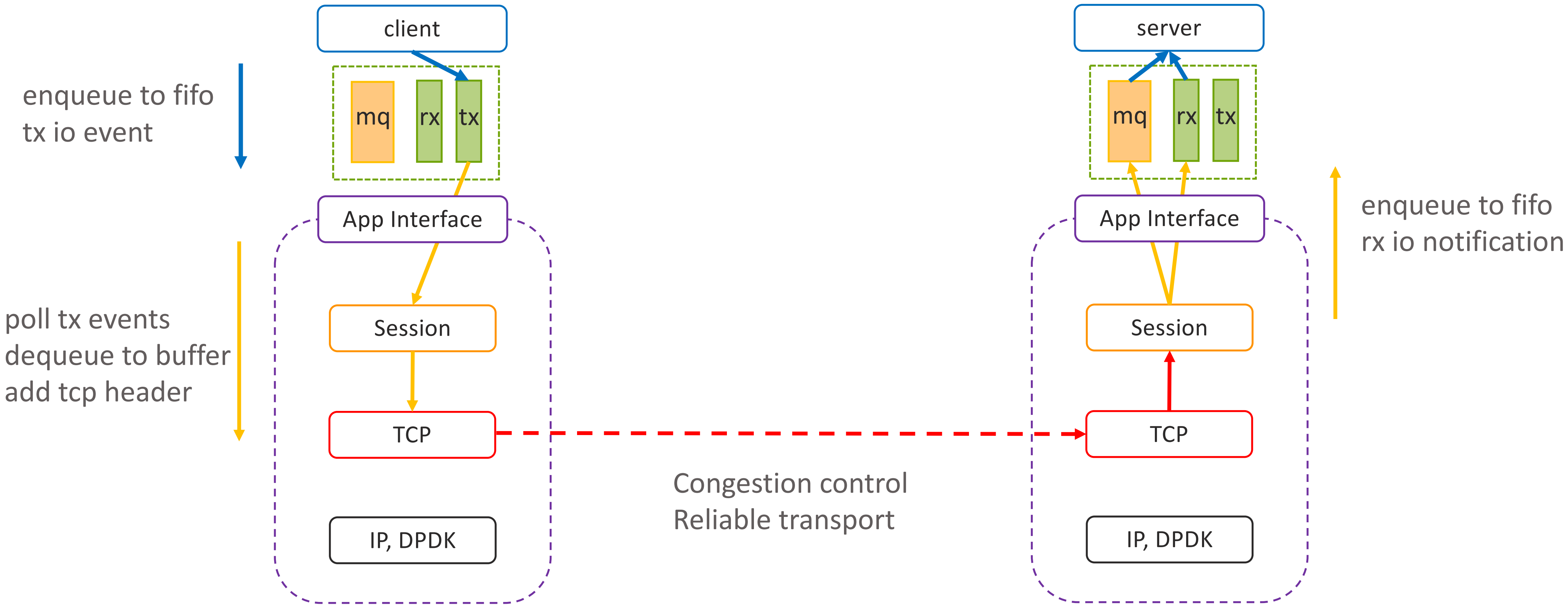
Session Establishment



Data Transfer

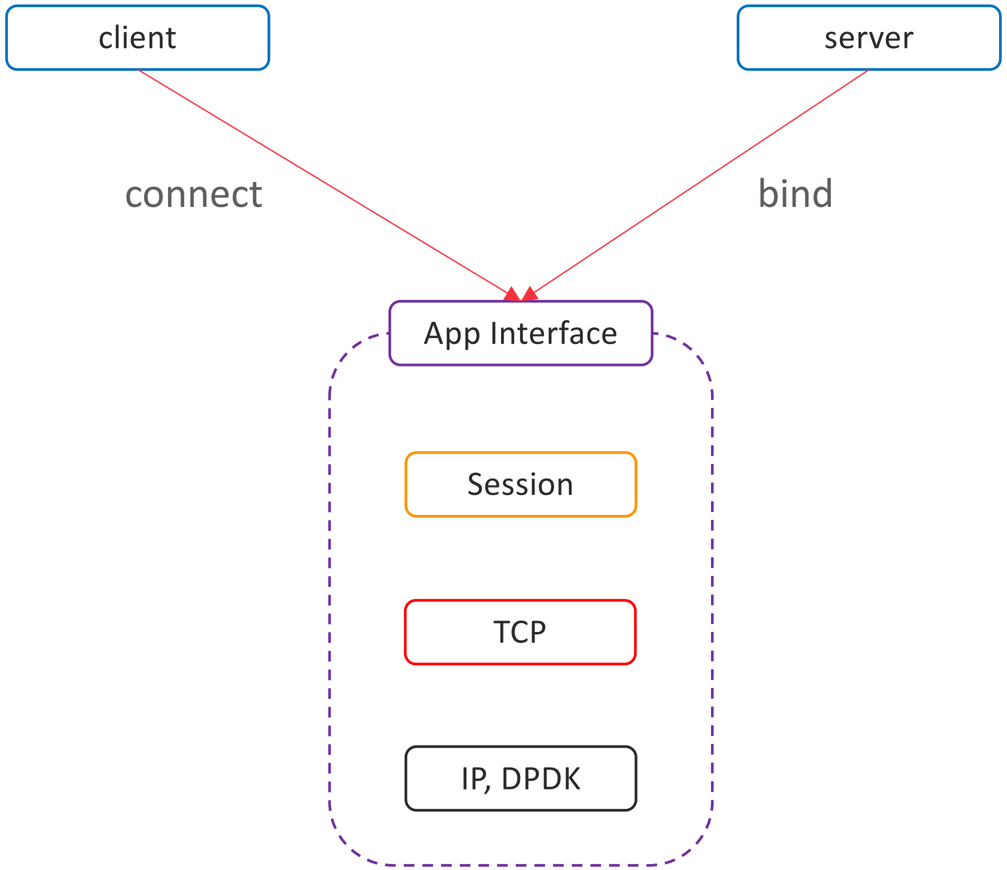


Data Transfer

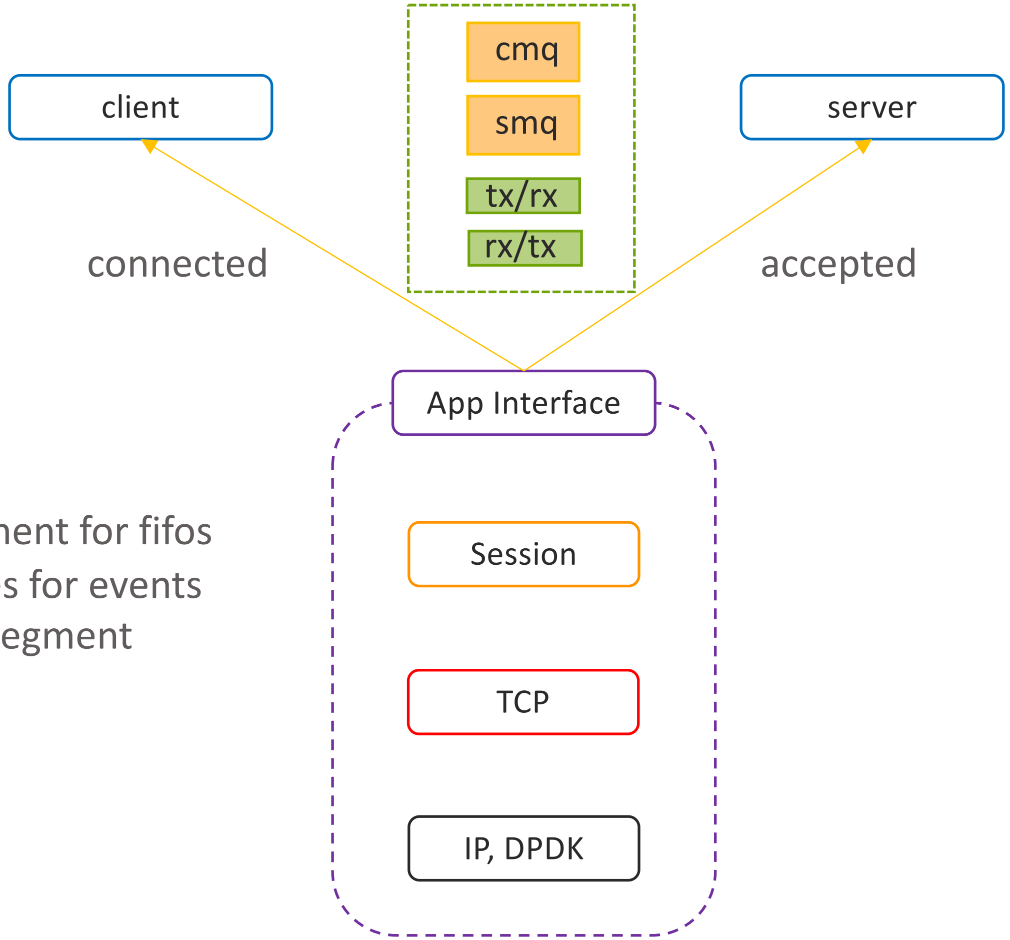


Some rough numbers on a E2699 w/XL710: ~36Gbps/core (1.5k MTU) half-duplex!

Redirected Connections (Cut-through)

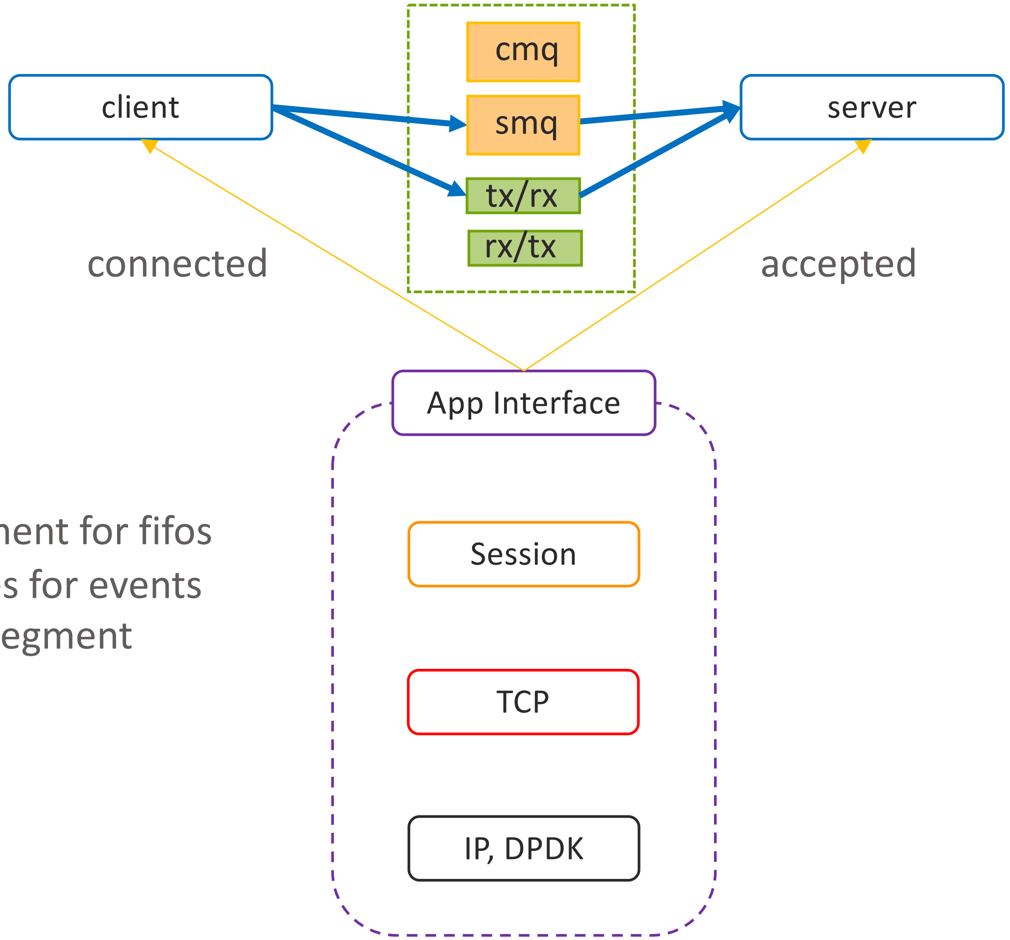


Redirected Connections (Cut-through)



- App interface sub-layer:
  - Tracks the sessions
  - Allocates ssvm segment for fifos and message queues for events
  - Asks peers to map segment

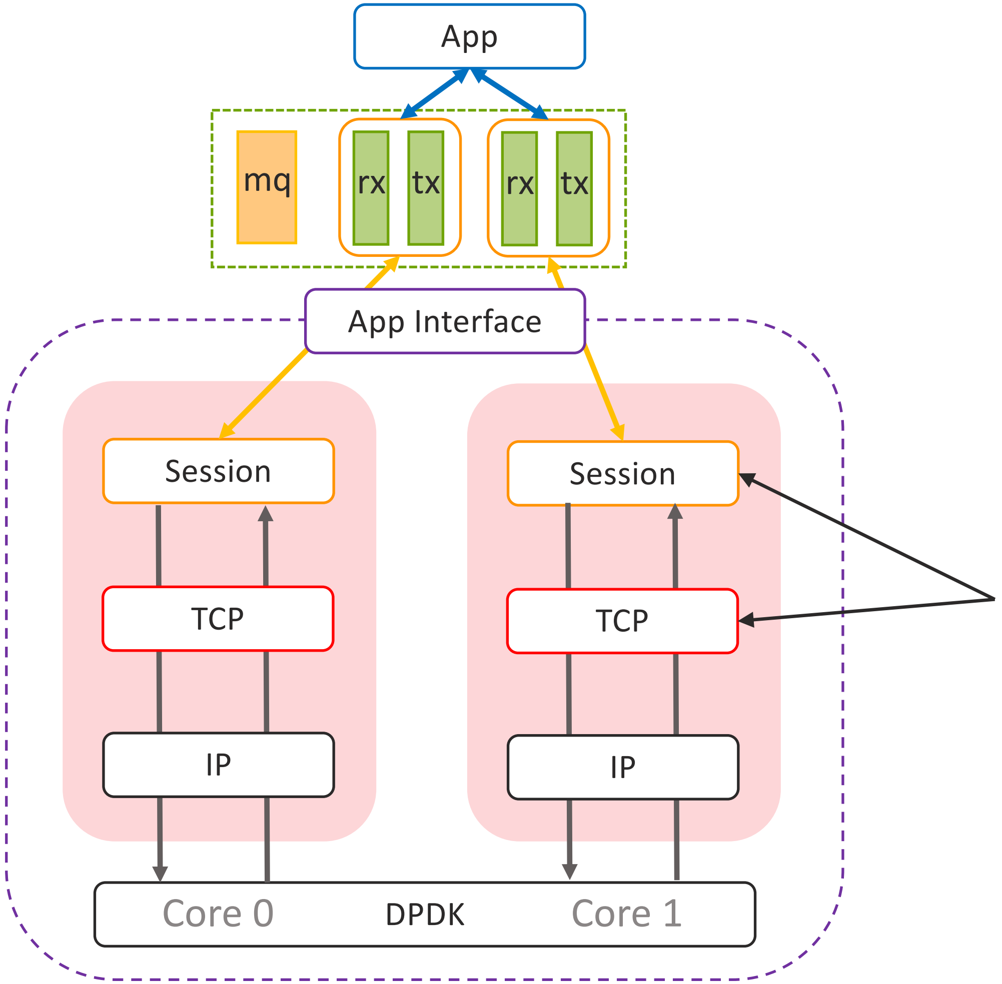
Redirected Connections (Cut-through)



- App interface sub-layer:
  - Tracks the sessions
  - Allocates ssvm segment for fifos and message queues for events
  - Asks peers to map segment

Throughput is around ~120Gbps half-duplex if receiver does not touch the data!

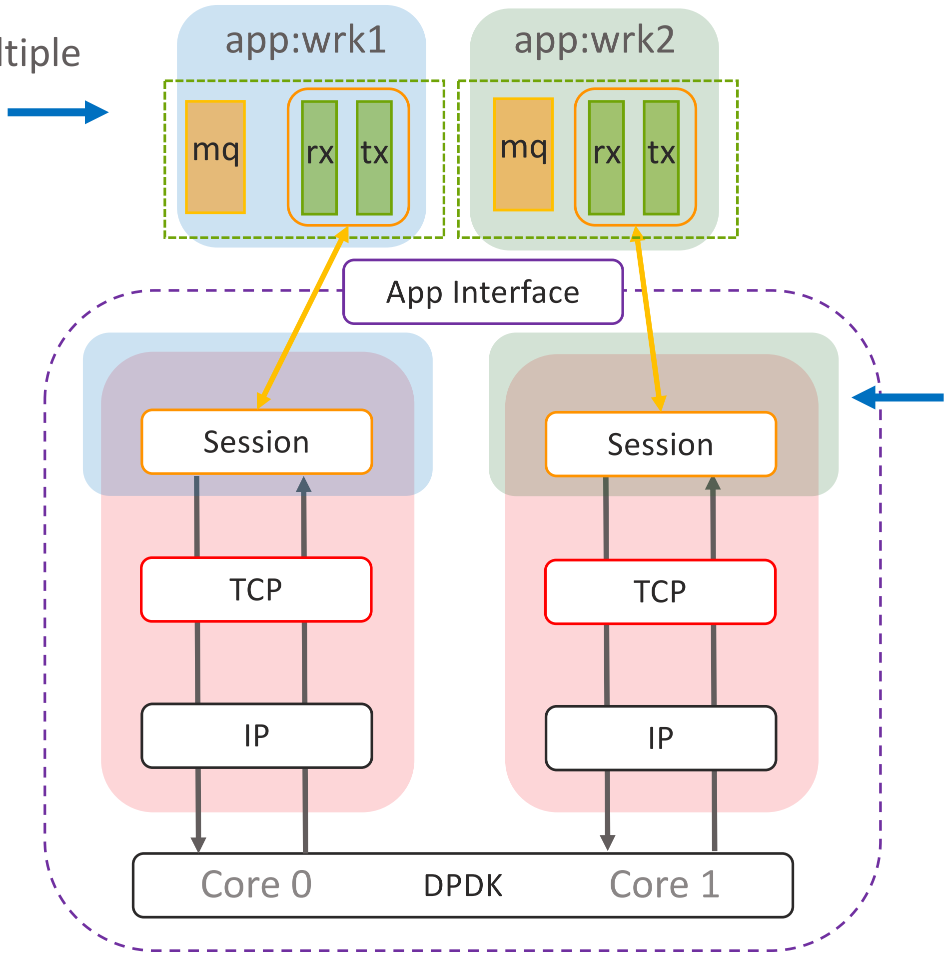
Multi-threading for stream connections



- Connections/sessions 'pinned' to a thread
- Per-thread data structures/state

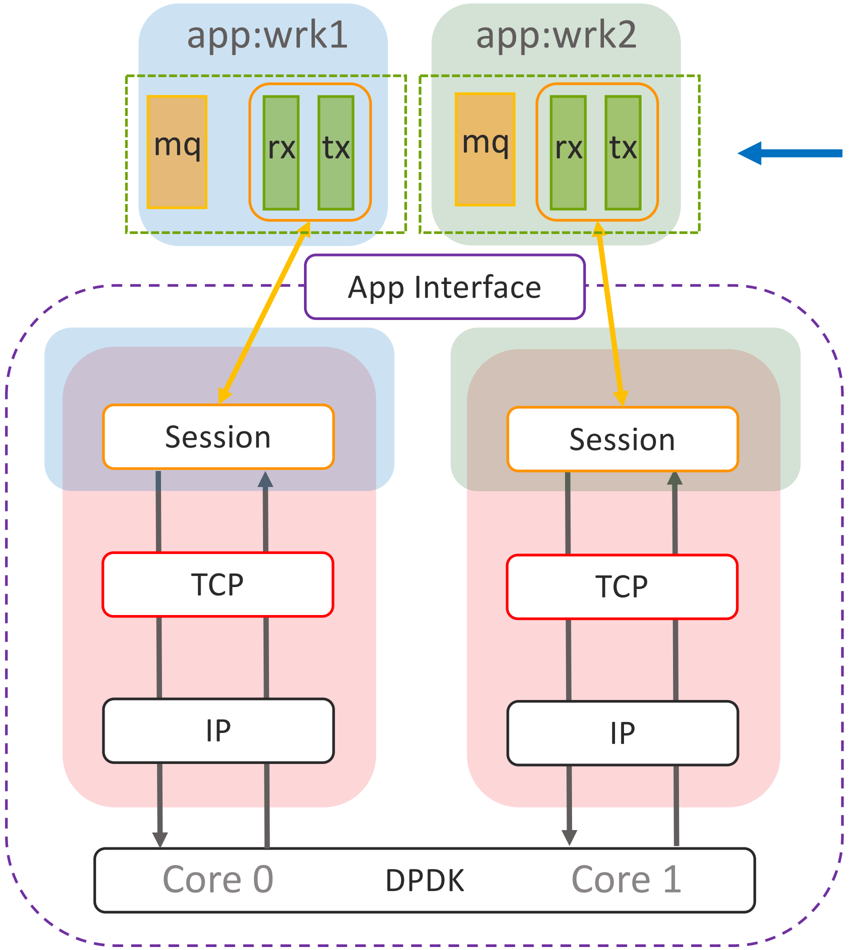
Apps with multiple workers

- Applications can register multiple "app-workers"
- Each worker gets its own message queue



- App interface sub-layer allocates app workers
- App workers have their own segment managers and message queues
- Sessions are associated to app-workers

Apps with multiple workers: VCL



- VCL has API for apps to register a new worker thread
- Sessions are pinned to a worker
- New workers are automatically registered on app fork
- Forked children automatically "share" the parent's sessions

Next steps – Get involved

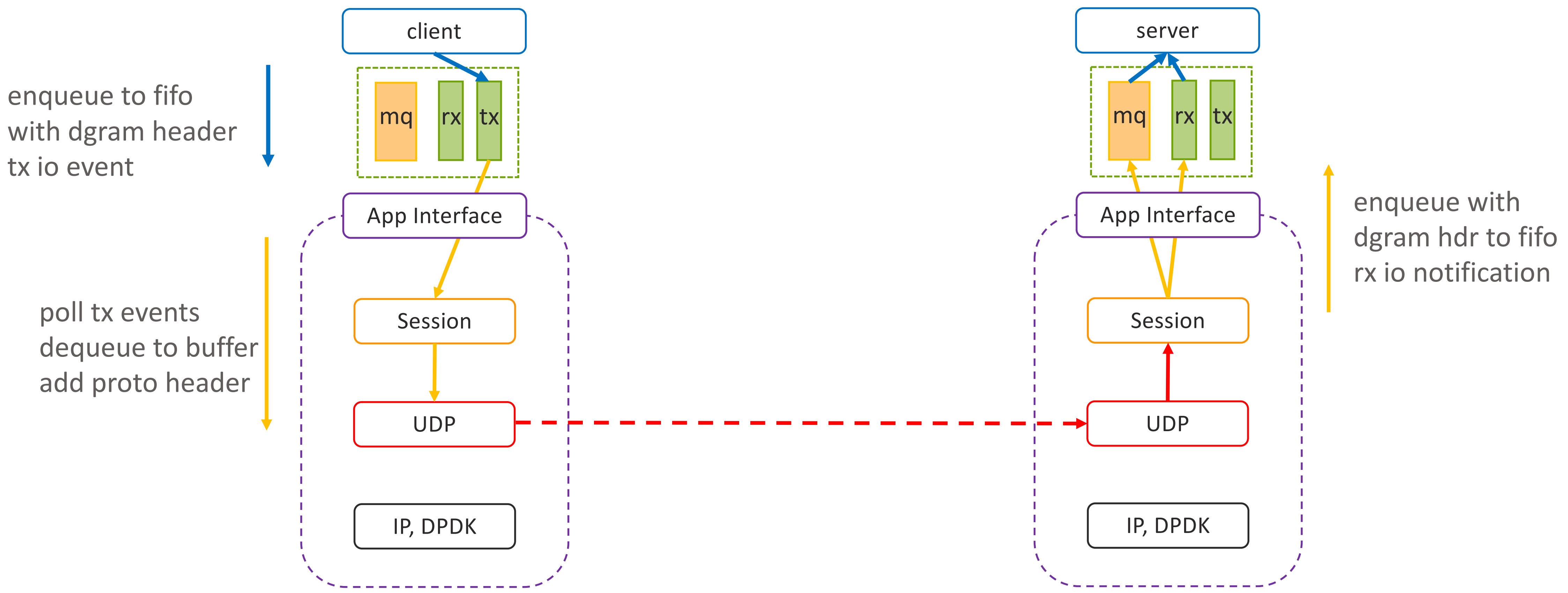
- Get the Code, Build the Code, Run the Code
  - Session layer: src/vnet/session
  - TCP: src/vnet/tcp
  - SVM: src/svm
  - VCL: src/vcl
- Read/Watch the Tutorials
- Read/Watch VPP Tutorials
- Join the Mailing Lists

Thank you!

?

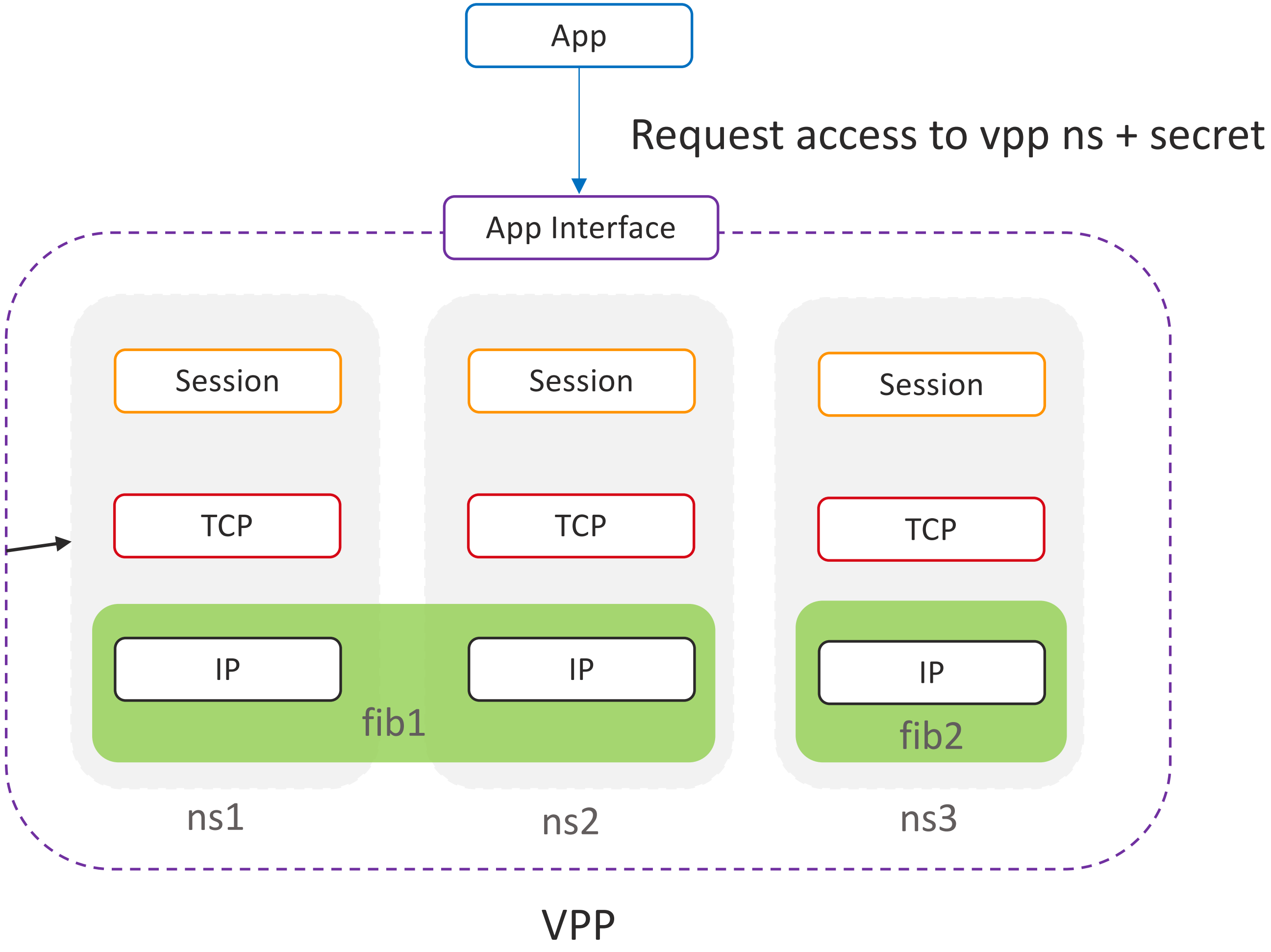
Florin Coras
email: fcoras@cisco.com
irc: florinc

Data Transfer: Dgram Transports

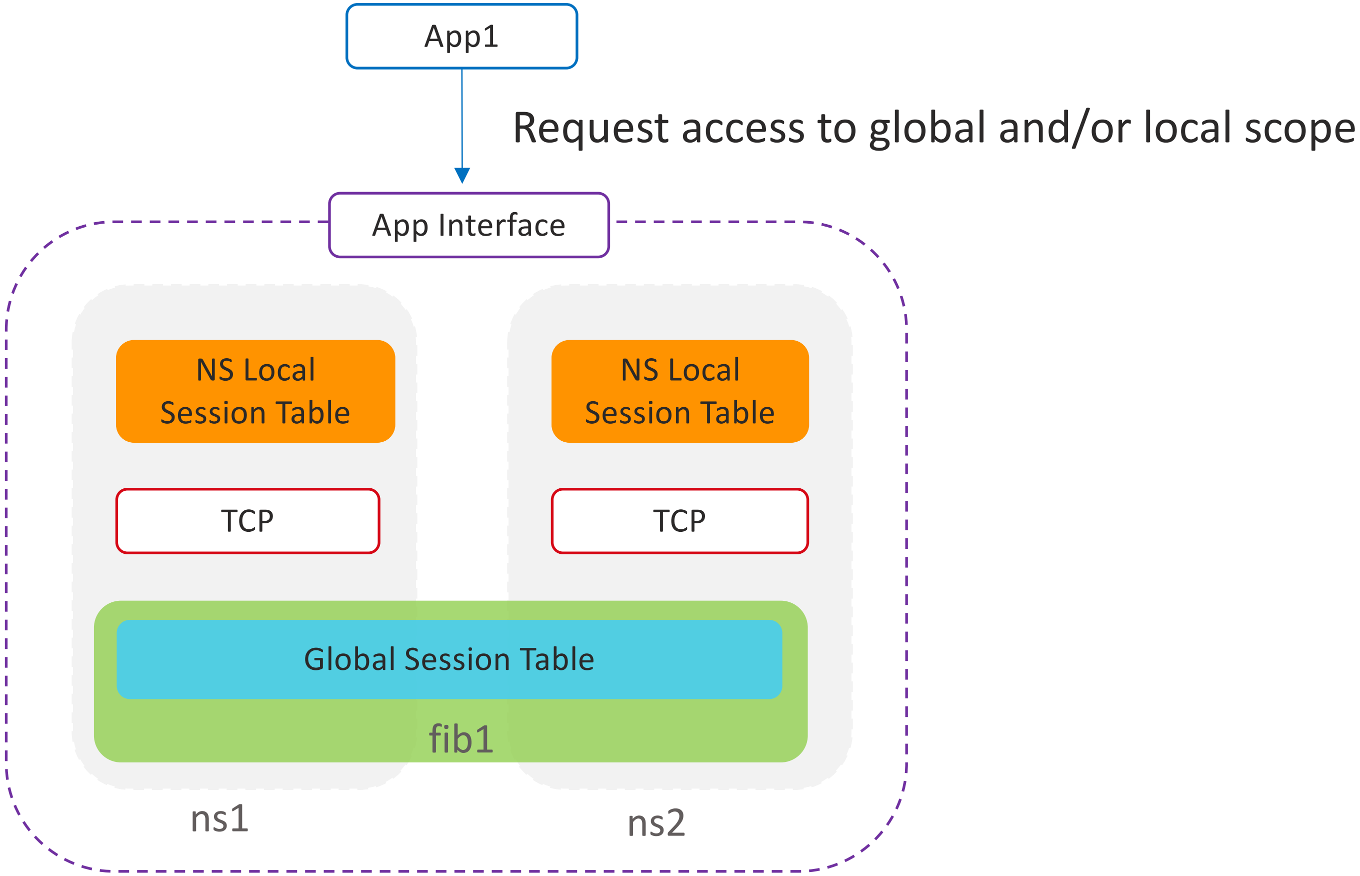


Features: Namespaces

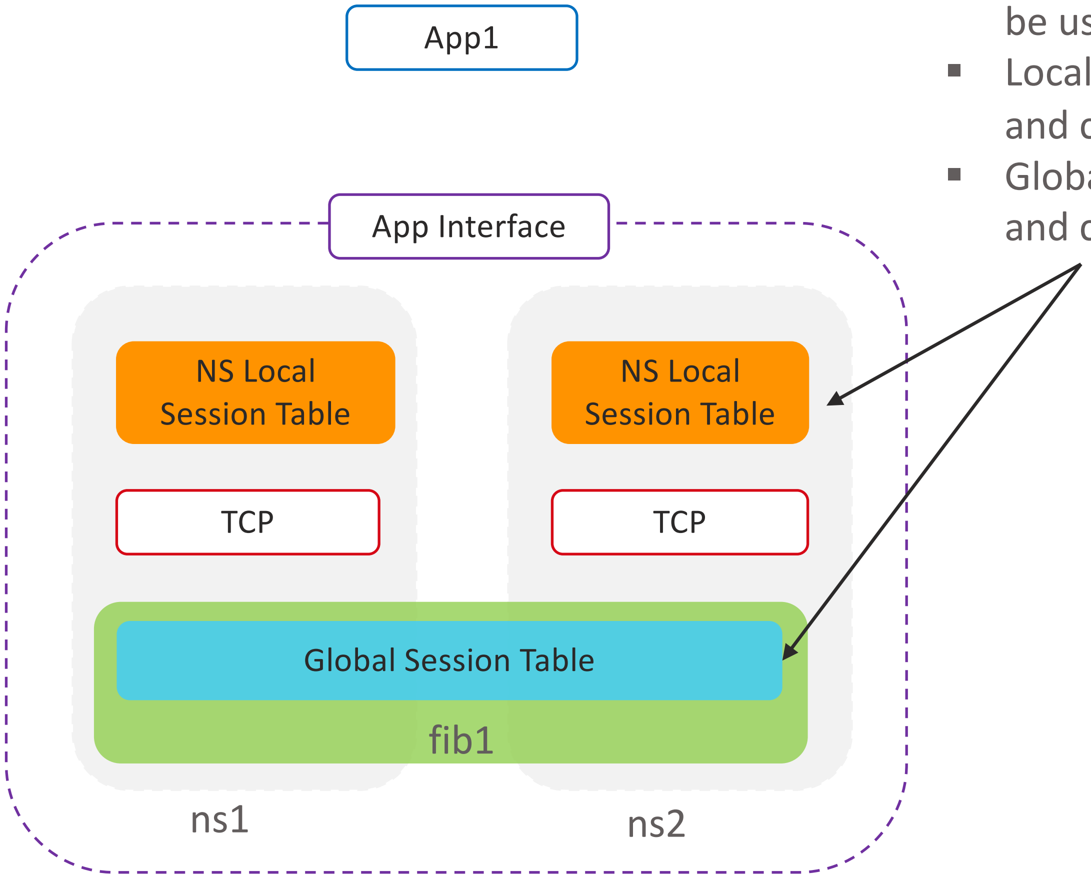
Namespaces are configured independently and associate applications to network layer resources like interfaces and fib tables



Features: Session Tables

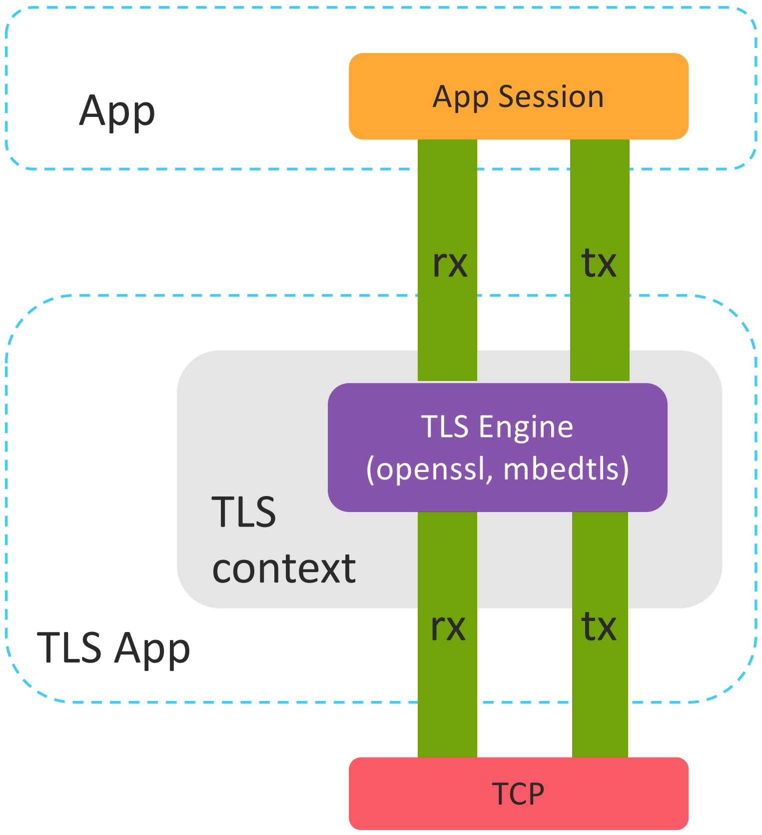


Features: Session Tables



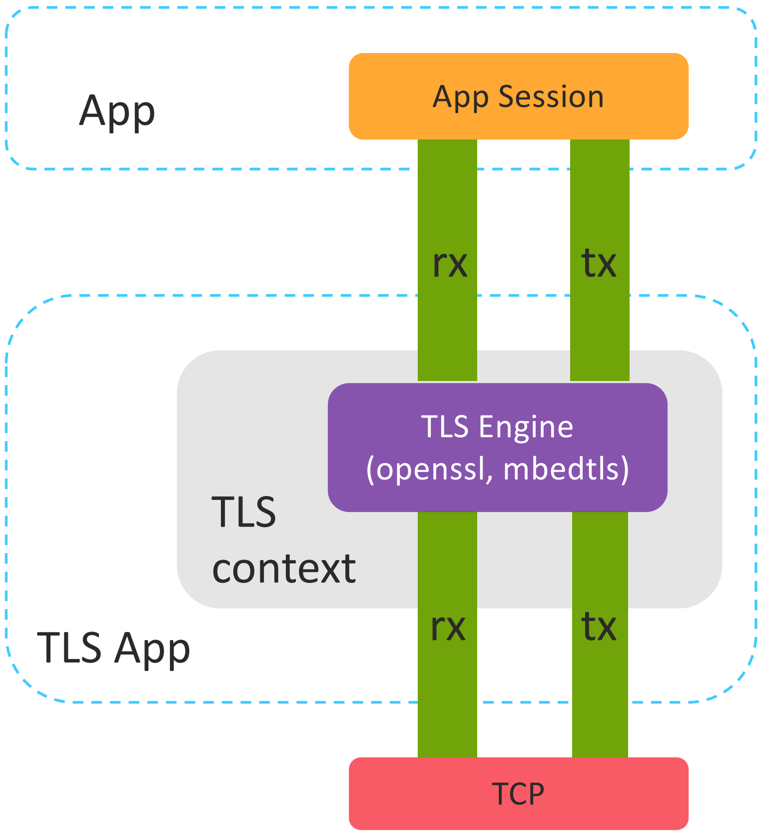
- Both table have “rules table” that can be used for filtering
- Local tables are namespace specific and can be used for egress filtering
- Global tables are fib table specific and can be used for ingress filtering

TLS App



- TLS App registers as transport at VPP init time
- TLS protocol implementation handled by plugin “engines”. We support openssl and mbedtls
- Client app registers key and certificate via api and requests tls as session transport
- CA certs read at TLS app init time. Defaults to reading /etc/ssl/certs/ca-certificates.crt
- Ping and Ray from Intel working on accelerating the openssl engine with QAT cards

TLS App



- TLS App registers as transport at VPP init time
- TLS protocol implementation handled by plugin "engines". We support openssl and mbedtls
- Client app registers key and certificate via api and requests tls as session transport
- CA certs read at TLS app init time. Defaults to reading /etc/ssl/certs/ca-certificates.crt
- Ping and Ray from Intel working on accelerating the openssl engine with QAT cards

Some rough OpenSSL numbers on a E2699: ~1Gbps/core (no hw accel)