Motivation

App 1
send()

user space

FIFO
TCP
IP
device

kernel

FIFO
TCP
IP
device

App 2
recv()
Motivation

App 1
- send()

VPP
- L2, L3
- tap
- dpdk
- tap

App 2
- recv()

user space

FIFO
TCP
IP
device

kernel

FIFO
TCP
IP
device

FD.io Mini-Summit at KubeCon North America 2018
Why not this?

App 1 → VPP → FIFO → TCP → dpdk, L2, L3 → App 2

user space

kernel
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 namespaces
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
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

FD.io Mini-Summit at KubeCon North America 2018
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

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

FD.io Mini-Summit at KubeCon North America 2018
Session Establishment

FD.io Mini-Summit at KubeCon North America 2018
Session Establishment

allocate session and fifos
connected notification

allocate session and fifos
accepted notification

handshake
Data Transfer

TCP/IP, DPDK

Session

Congestion control
Reliable transport

enqueue to fifo

enqueue to fifo

rx io notification

C

App Interface

client

mq

rx

tx

Tx io event

Poll tx events

dequeue to buffer

App Interface

server

mq

rx

tx

enqueue to fifo

rx io notification

Ip, DPDK

Ip, DPDK

FD.io Mini-Summit at KubeCon North America 2018
Data Transfer

enqueue to fifo

tx io event

poll tx events
dequeue to buffer

add tcp header

enqueue to fifo

rx io notification

TCP

IP, DPDK

Session

App Interface

client

mq

rx

tx

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

server

mq

rx

tx

App Interface

Session

TCP

IP, DPDK

FD.io Mini-Summit at KubeCon North America 2018
Redirected Connections (Cut-through)

client

connect

server

bind

App Interface

Session

TCP

IP, DPDK

FD.io Mini-Summit at KubeCon North America 2018
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
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

```
App Interface

mq  rx  tx  mq  rx  tx

Session
TCP
IP
Core 0  DPDK  Core 1
```

app:wrk1

app:wrk2
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

- **enqueue to fifo with dgram header tx io event**
- **poll tx events dequeue to buffer add proto header**
- **enqueue with dgram hdr to fifo rx io notification**

**App Interface**
- **Session**
- **UDP**
  - IP, DPDK

**client**
- mq
- rx
- tx

**server**
- mq
- rx
- tx

**FD.io Mini-Summit at KubeCon North America 2018**
Namespaces are configured independently and associate applications to network layer resources like interfaces and fib tables.
Features: Session Tables

App1

Request access to global and/or local scope

App Interface

NS Local Session Table

TCP

Global Session Table

fib1

ns1

ns2

FD.io Mini-Summit at KubeCon North America 2018
Features: Session Tables

- Both tables 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 registers as transport at VPP init time
TLS protocol implementation handled by plugin “engines”. We support openssl and mbdtls
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)