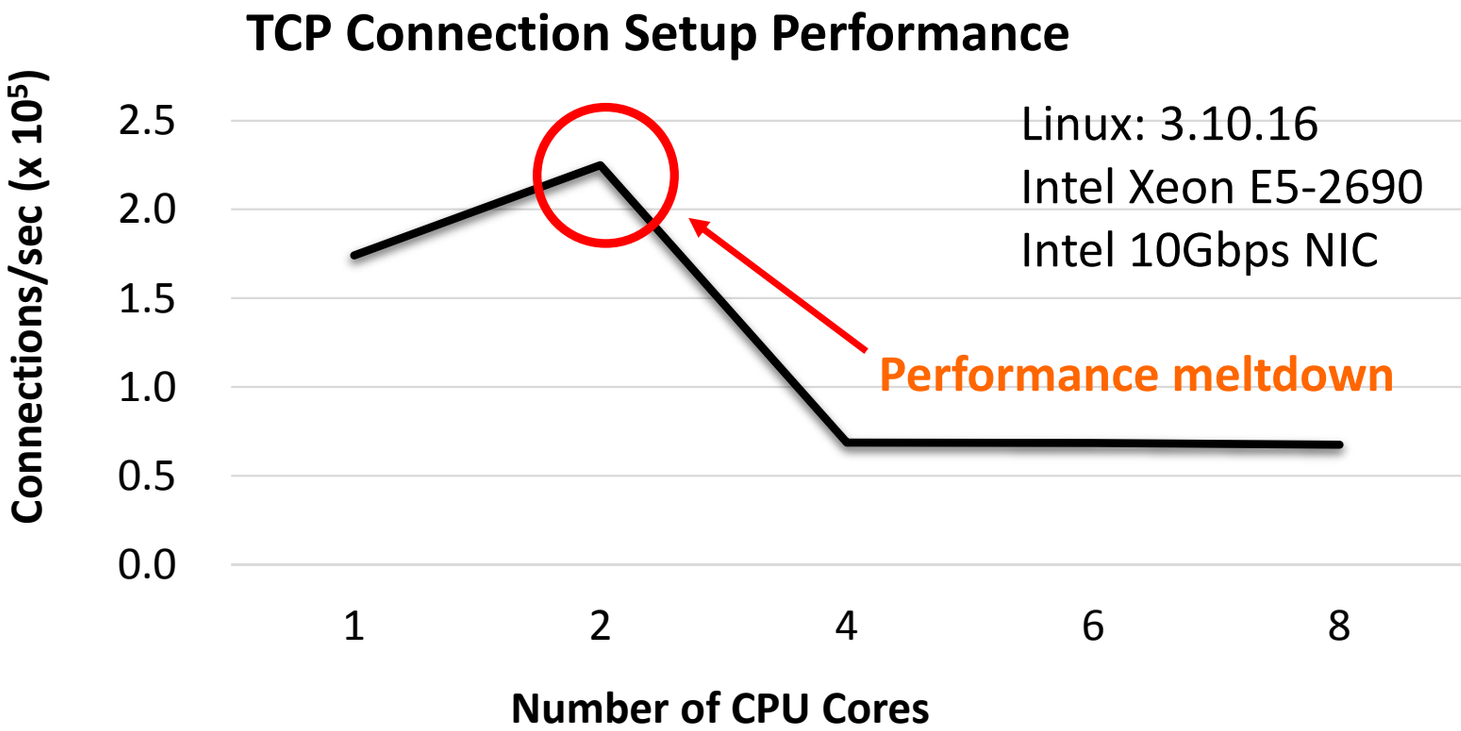
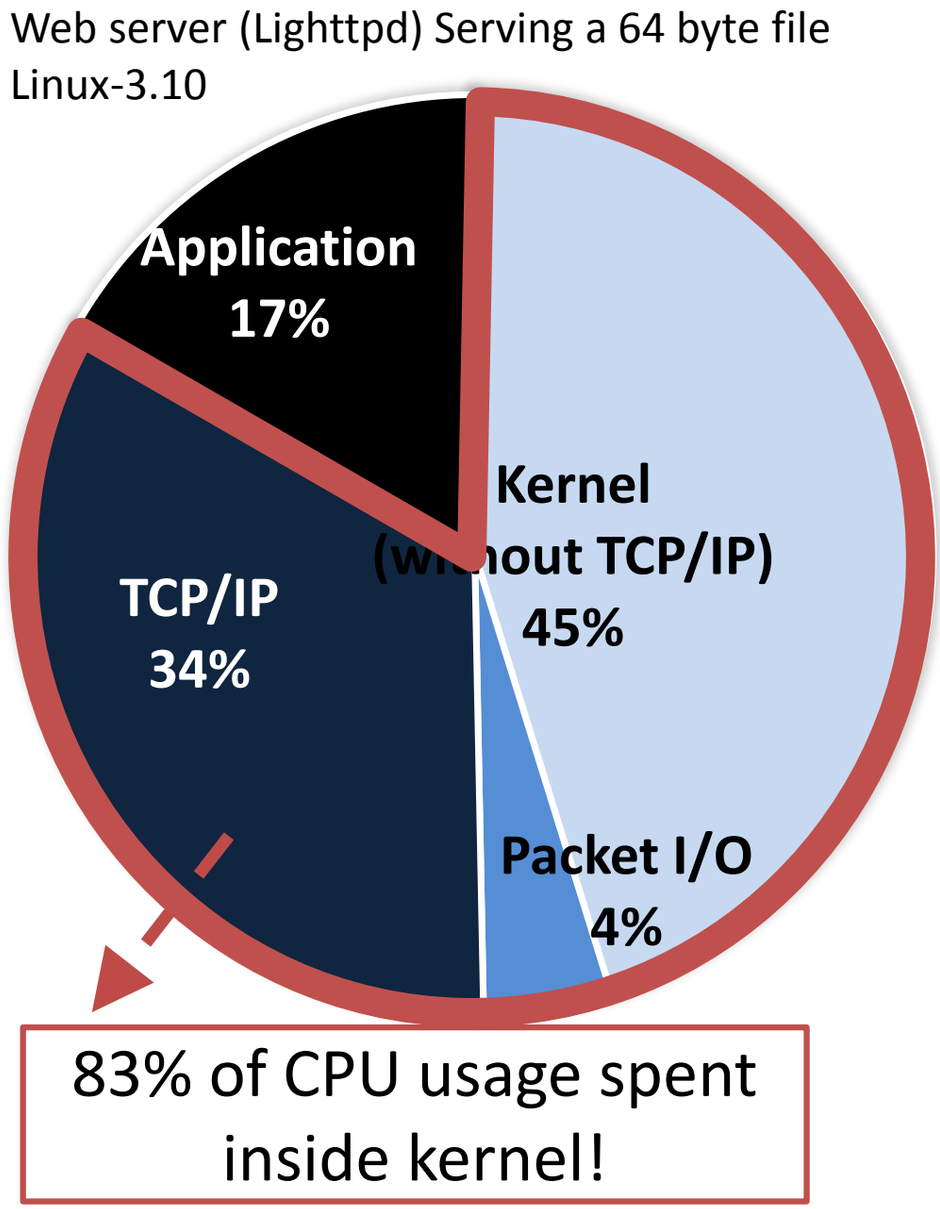
User-space Stacks

TCP in Linux

Linux TCP stack is not designed for high performance

- Especially for short flows
- Poor scalability, bad locality, etc
- Same problems we saw with DPDK





Figures from Jeong's mTCP talk at NSDI 14

mTCP [Jeong, NSDI '14]

User space TCP stack

- Built on DPDK/netmap (and now OpenNetVM!)

Key Ideas:

- Eliminate shared resources by partitioning flows to independent threads
- Use batching to minimize overheads
- Epoll interface to support existing end-point applications

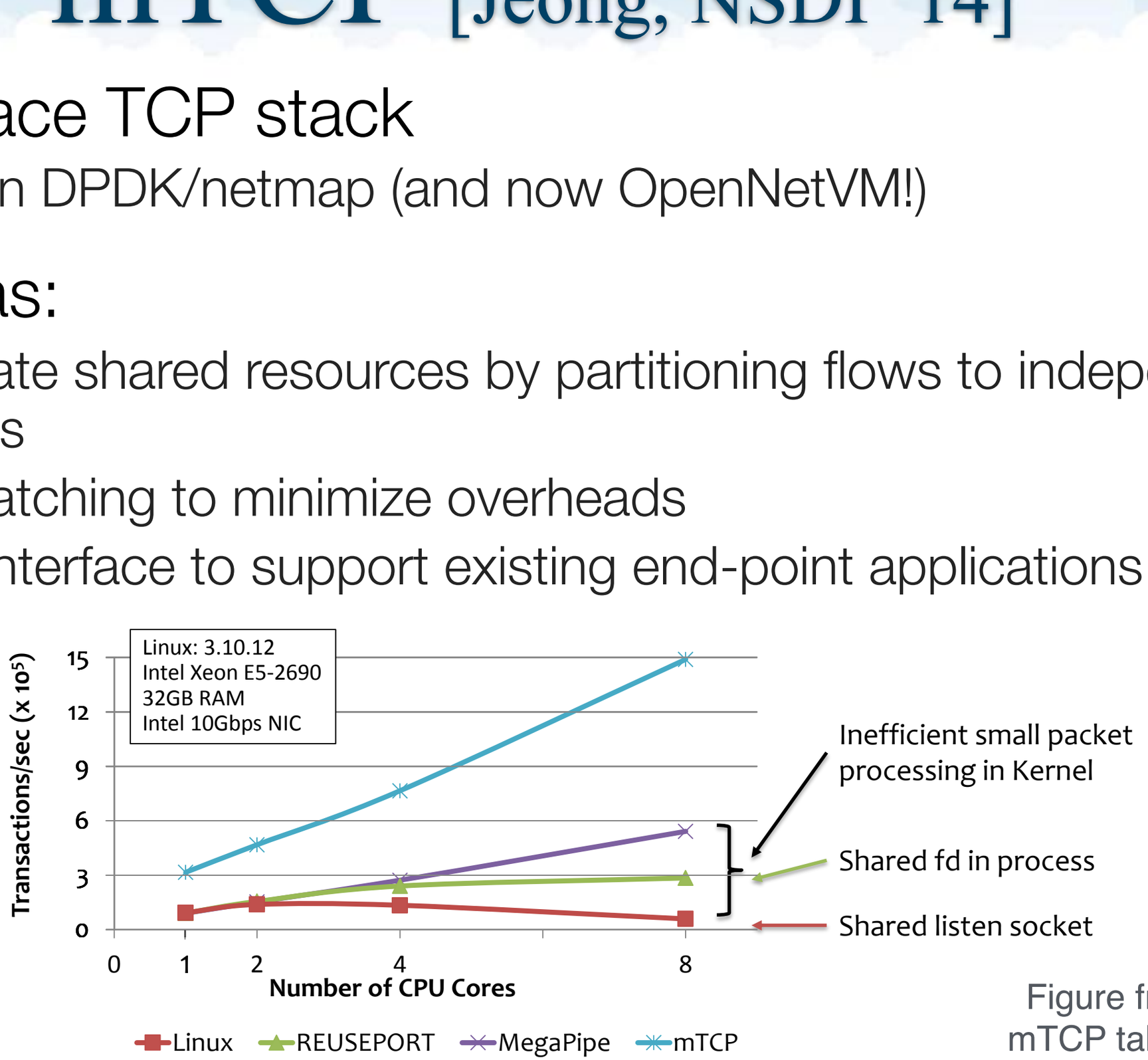


Figure from Jeong's mTCP talk at NSDI 14

mTCP Kernel Bypass

Responding to a packet arrival only incurs a context switch, not a full system call

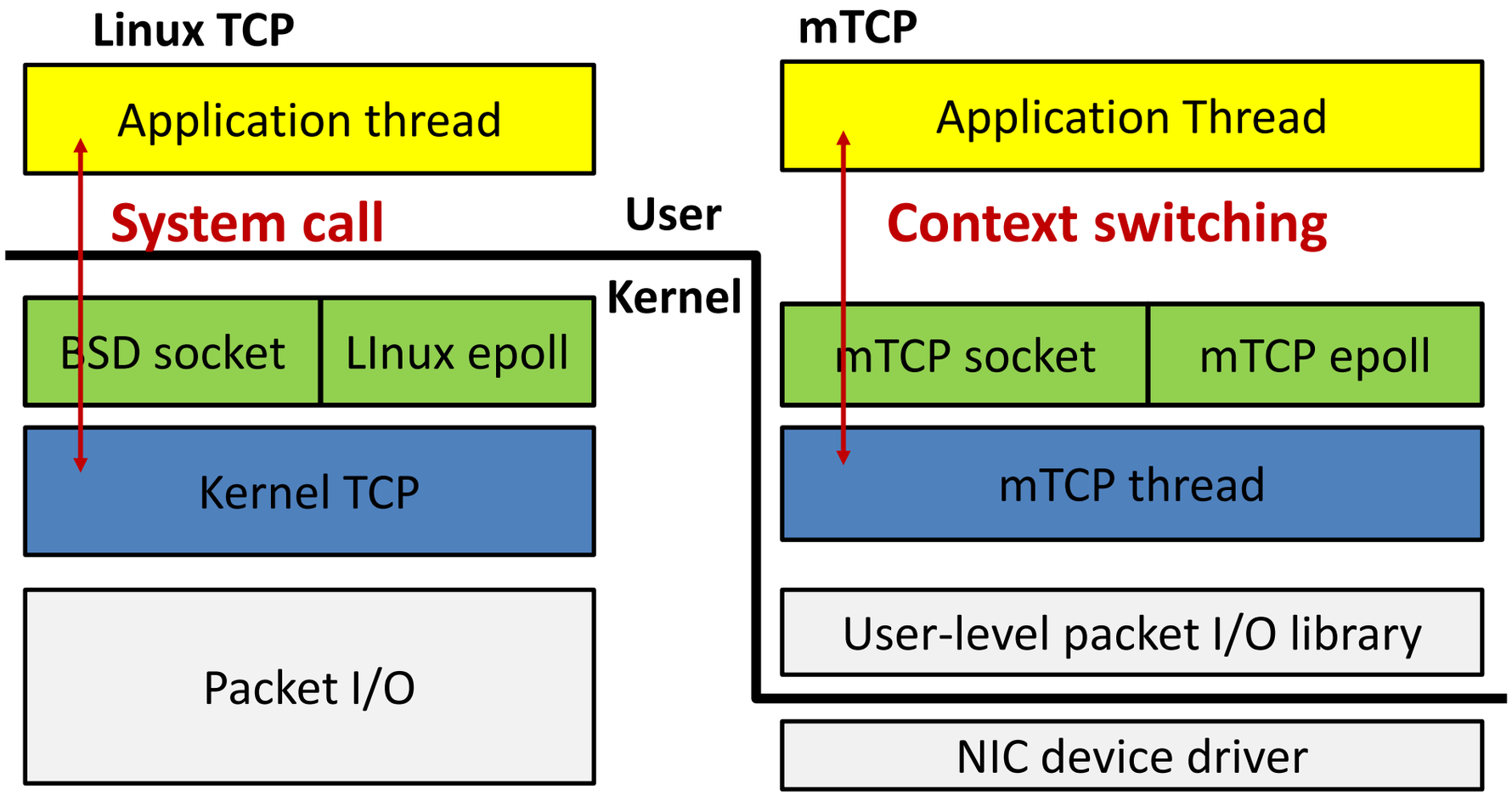
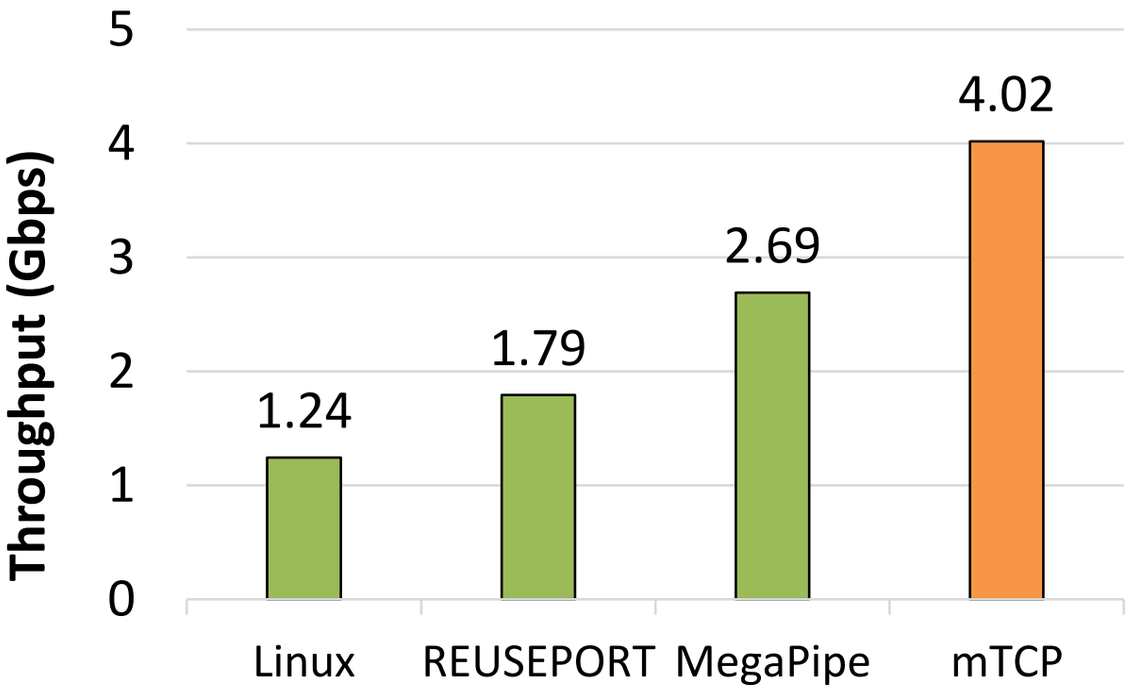


Figure from Jeong's mTCP talk at NSDI 14

Performance Improvement on Ported Applications

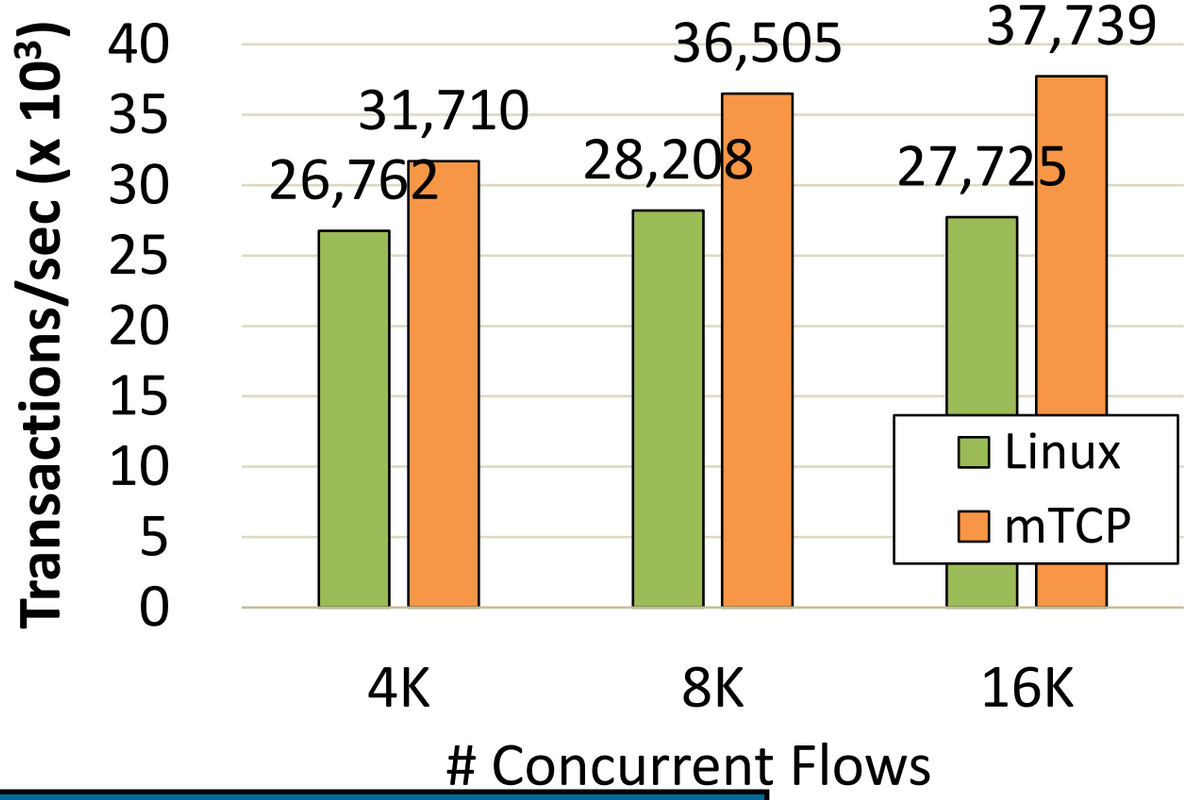
Web Server (Lighttpd)

- Real traffic workload: Static file workload from SpecWeb2009 set
- **3.2x** faster than Linux
- **1.5x** faster than MegaPipe



SSL Proxy (SSLShader)

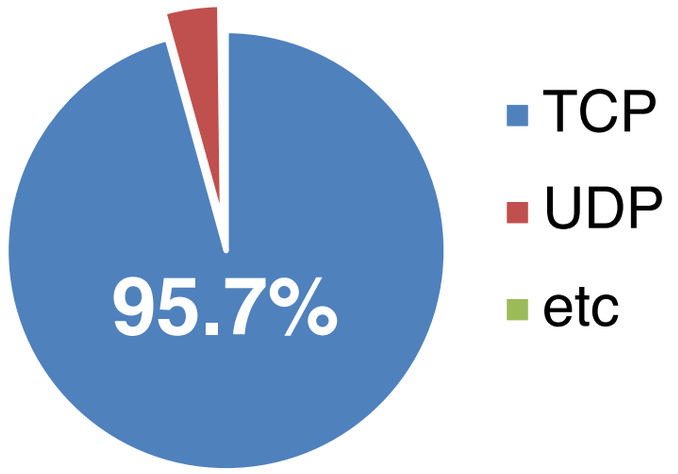
- Performance Bottleneck in TCP
- Cipher suite
  1024-bit RSA, 128-bit AES, HMAC-SHA1
- Download 1-byte object via HTTPS



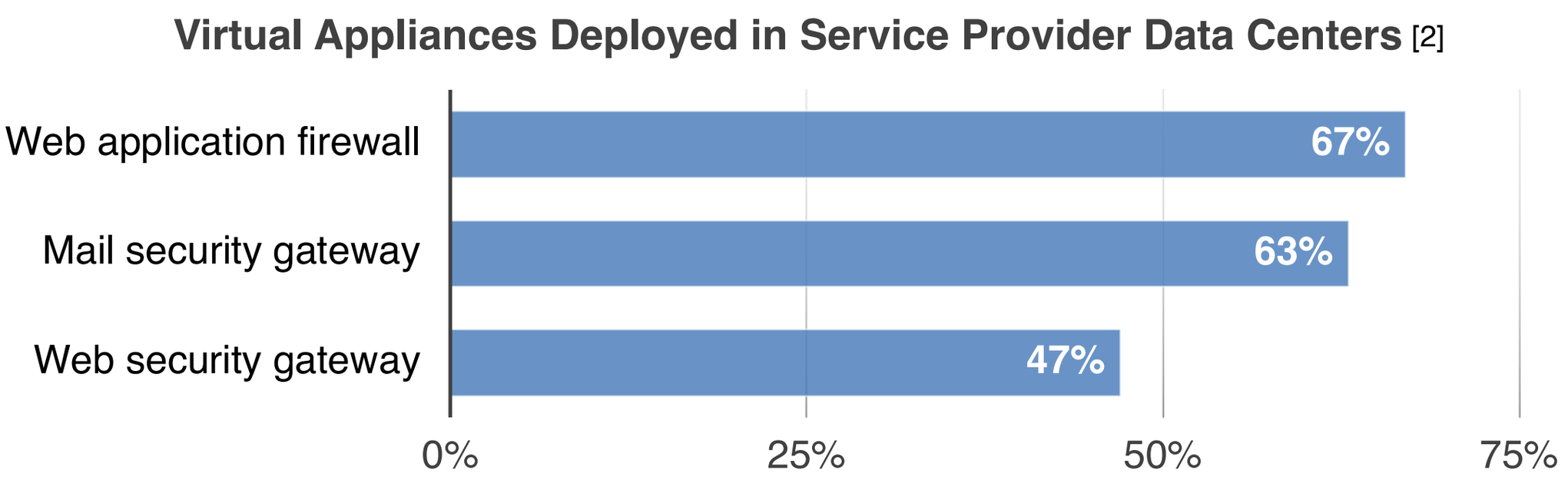
Slide from Jeong's mTCP talk, NSDI '14

Most Middleboxes Deal with TCP Traffic

- TCP dominates the Internet
  - 95+% of traffic is TCP [1]



- Top 3 middleboxes in service providers rely on L4/L7 semantics



[1] “Comparison of Caching Strategies in Modern Cellular Backhaul Networks”, ACM MobiSys 2013.

[2] IHS Infonetics Cloud & Data Center Security Strategies & Vendor Leadership: Global Service Provider Survey, Dec. 2014.

Slide from Jamshed’s mOS talk, NSDI ‘17

mOS [Jamshed, NSDI '17]

What if your middle box (not end point server) needs TCP processing?

Proxies, L4/L7 load balancers, DPI, IDS, etc

- TCP state transitions
- Byte stream reconstruction

Borrow code from open-source IDS (e.g., snort, suricata)	• 50K~100K code lines tightly coupled with their IDS logic
Borrow code from open-source kernel (e.g., Linux/FreeBSD)	• Designed for TCP end host • Different from middlebox semantics
Implement your own flow management code	• Complex and error-prone • ***Repeat*** it for every custom middlebox

Table from Jamshed's mOS talk, NSDI '17

mOS [Jamshed, NSDI '17]

Reusable protocol stack for middle boxes

Key Idea: Allow customizable processing based on flow-level "events"

Separately track client and server side state

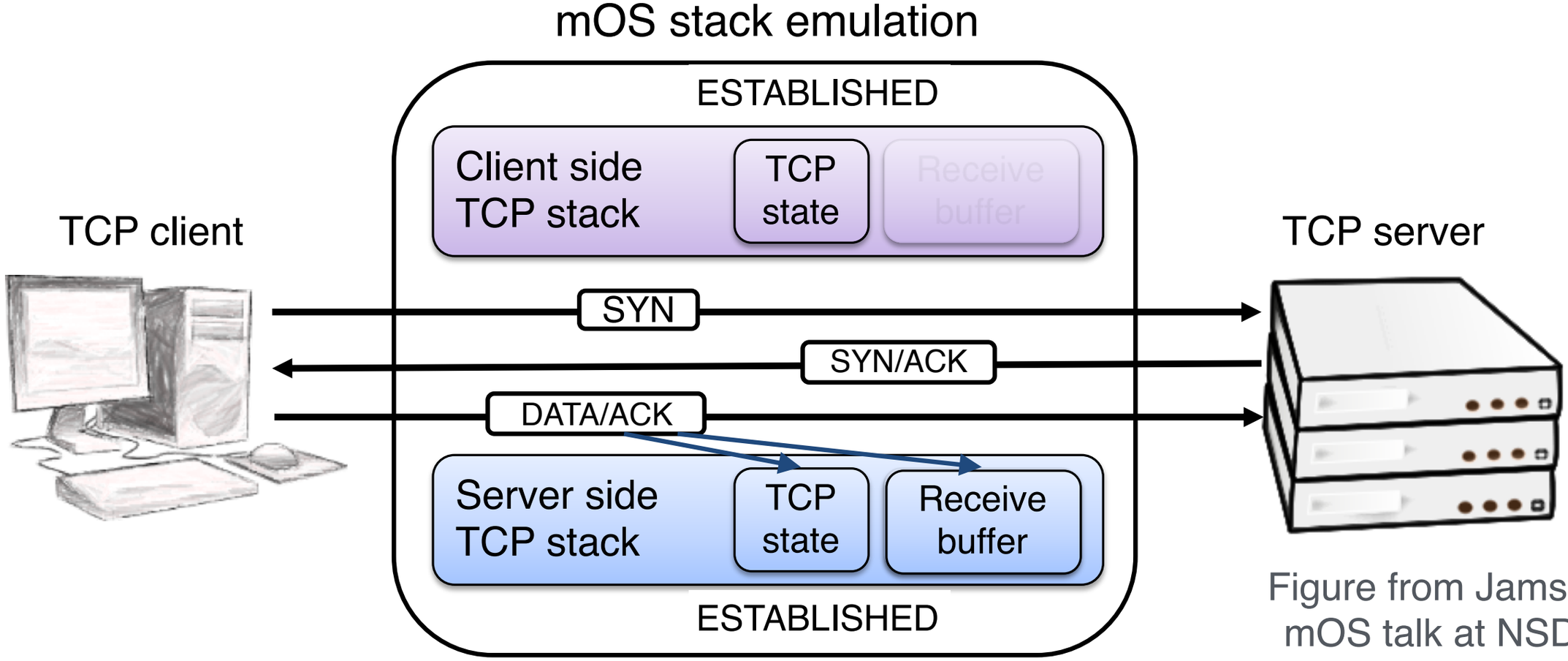


Figure from Jamshed's mOS talk at NSDI 17

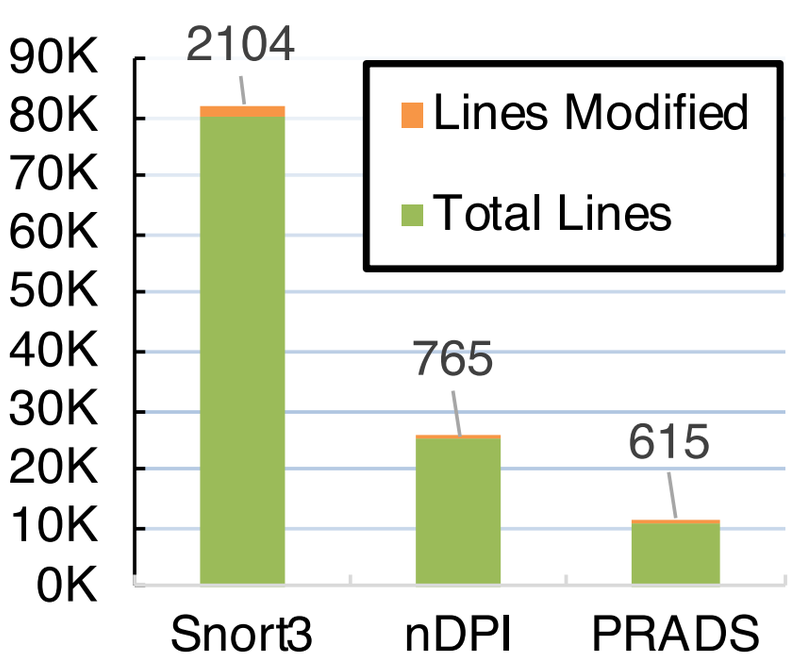
mOS [Jamshed, NSDI '17]

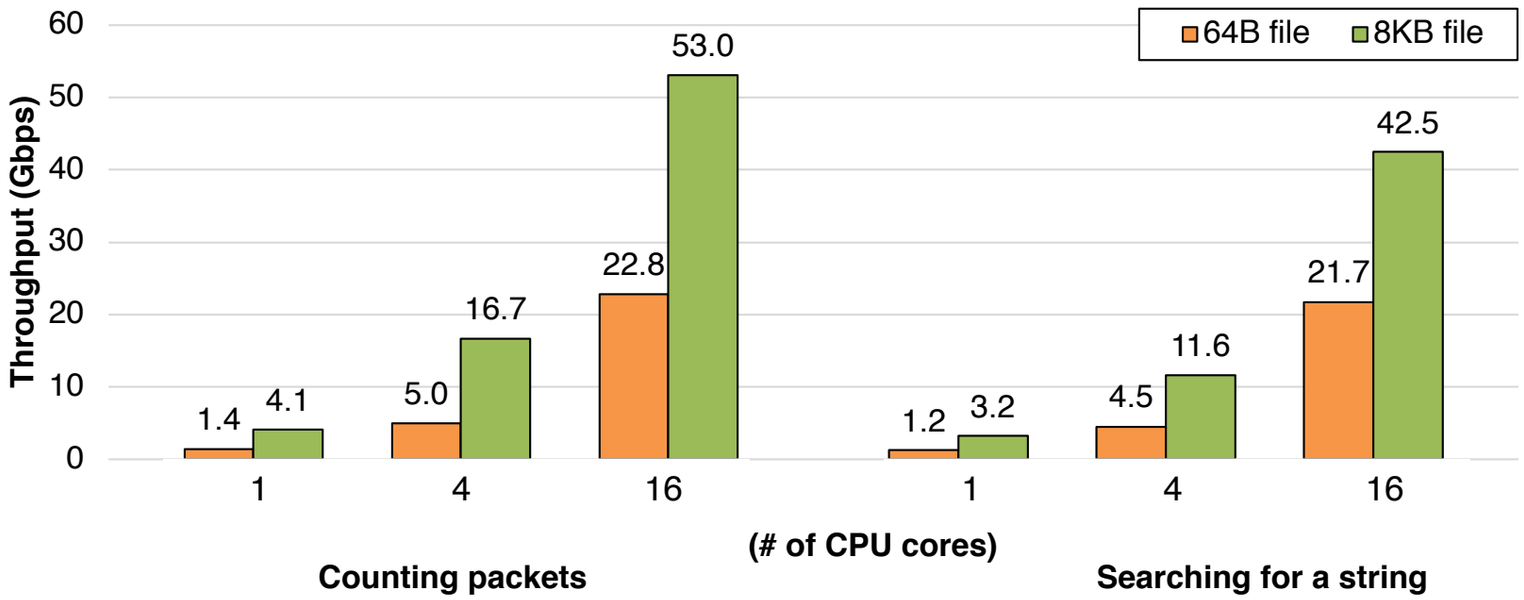
Base Events

- TCP connection start/end, packet arrival, retransmission, etc

User Events

- Base event + a filter function (executable code) run in mOS stack





Figures from Jamshed's mOS talk at NSDI 17

Microboxes [SIGCOMM 18]

- ✕ Redundant Stack Processing
- ✕ A Monolithic Stack
- ✕ Separate Stacks/Interfaces

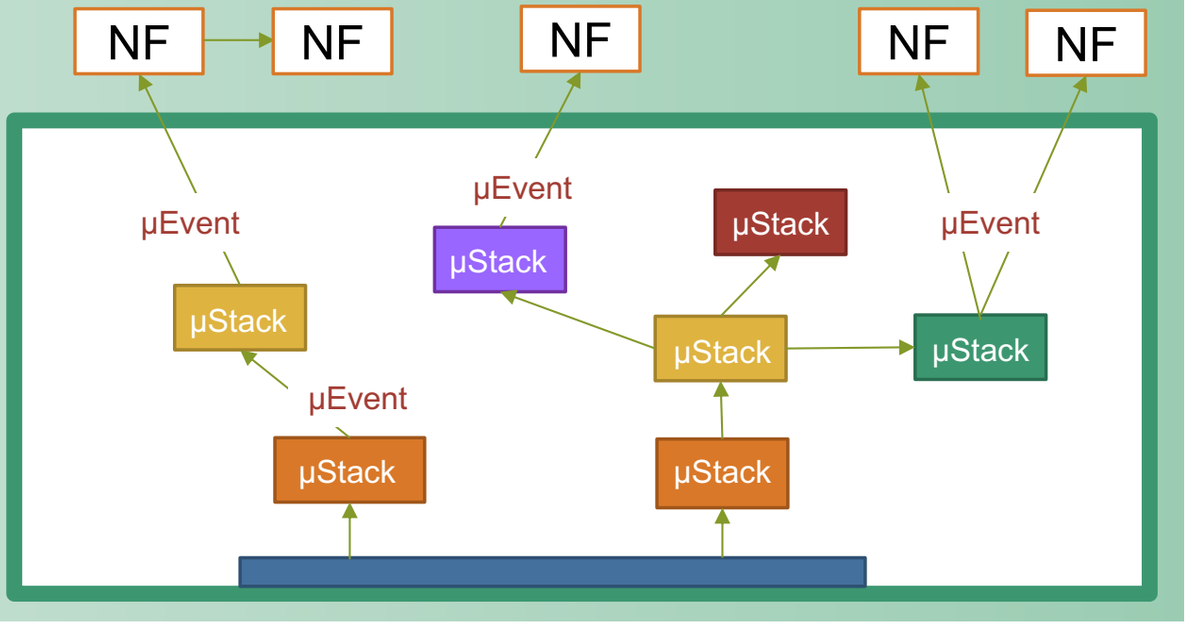
⇒

- ✓ Consolidate Stack Processing
- ✓ Customizable Stack Modules
- ✓ Unified Event Interface

Microboxes

= µStack + µEvent

= stack snapshot + parallel stacks + parallel events + event hierarchy + publish/subscribe interface

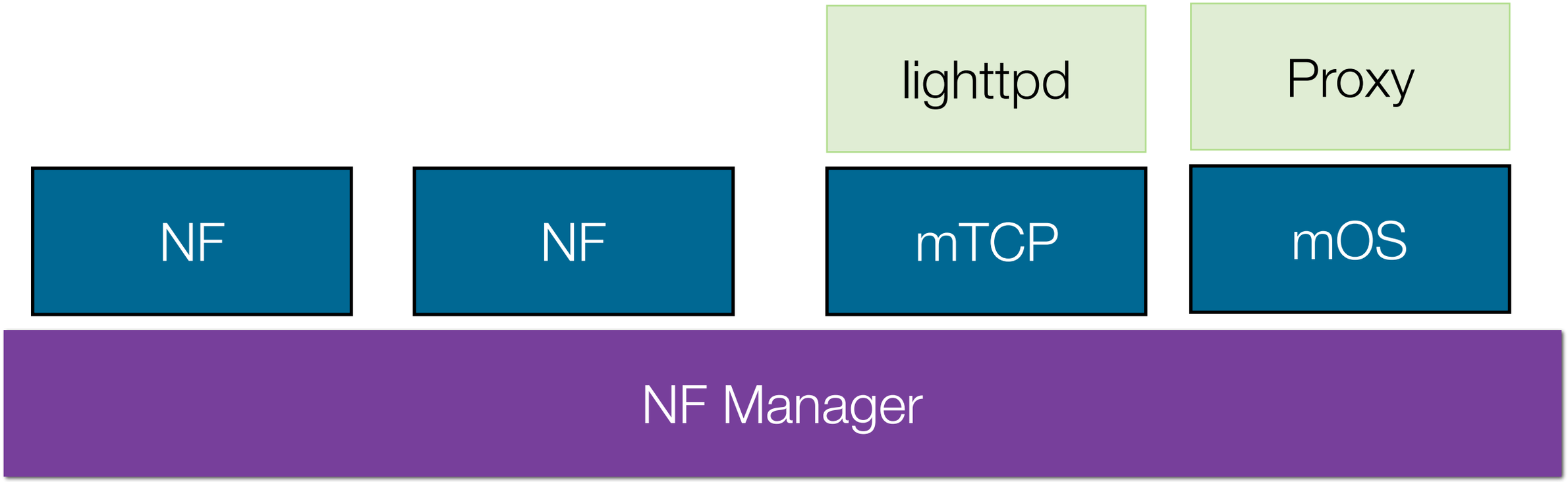


TCP + OpenNetVM

We have ported mOS/mTCP to run on OpenNetVM

Allows deployment of mixed NFs and endpoints

Allows several different mTCP endpoints on same host



TCP + OpenNetVM

Mixed NFs + endpoints blurs the line of the application and the network

- NF services could expose APIs to work with endpoints