Network Stacks for the Virtualized Mobile Packet Core

Ashwin Kumar*, Priyanka Naik+, Sahil Patki*, Pranav Chaudhary*, Mythili Vutukuru*

Indian Institute of Technology, Bombay* IBM Research, India+
Network Function Virtualization (NFV)

Why NFV?
- Cost effective
- Easy to scale
- Easy to roll out updates

NFV is great

BUT

Harder to ensure Service Level Objectives (SLOs) in the NFV case

NFs have to make use of abstractions provided by already existing software

Linux kernel network stack
one such existing software!!
Why Kernel Bypass?

Linux kernel was not built for being on the data path of 40 Gbps and 100 Gbps network cards.

Difficult to fix the kernel, so bypass it.

The era of Kernel bypass … **DPDK and netmap**
Novel Network Stacks

mTCP[1]

IX[2]

User space networking stack

Application thread

Network thread

Core X

Core local data structures

Batched I/O to amortize cost of context switch

Data plane Operating System

- Borrows the core local data structure design from mTCP
- The network stack runs on the same thread as the application as a data plane OS
- Run to completion instead of batching

Good breadth of various user space network stack designs!!


TAS[3]

User space Networking stack

CORE X

CORE Y

Application thread

Network thread

- Deviates from co-locating the network stack and the application
- Prefetching
5G Use Case

- Telco embracing NFV, fast!!
- Network functions in the 5G mobile packet core being standardized as VNFs[1]

Mobile packet core connects mobile devices and base stations to external networks

Access Mobility Function (AMF) \(\rightarrow\) each signalling message goes through this NF, and hence current user state maintained here

Authentication Service Function (AUSF) \(\rightarrow\) responsible for user authentication and security setup during user registration

Authentication Service Function (AUSF)

Control plane performs subscriber registration, authentication, and other signalling messages

We will be focussing on the user registration procedure to figure out effective network stack designs for AMF and AUSF

Complex 5G NFs

● Novel research network stacks have mostly looked at I/O intensive network functions
● But, the 5G use case is different
  ○ Control plane consists of CPU intensive network functions which terminate TCP endpoints
  ○ **AUSF takes more than 300,000 CPU cycles to service some requests as opposed to some load balancers[^1]** which take ~500 cycles to service a request.
  ○ **AMF maintains and manages user state**
● Our goal is to look at effective network stack designs in the CPU intensive NFs on the control plane

[^1]: Stateless Datacenter Load-balancing with Beamer, NSDI ‘18
We focus on the user registration procedure in the 5G packet core

We benchmark the user registration procedure to judge network stack performance
Dummy AUSF with short connections

- AUSF just responds with dummy data without doing any processing
- Emulates previously looked at I/O intensive NFs

Novel network stacks like IX outperform the kernel network stack by a factor of 5.

TAS lags behind on short connection workloads.
  - Lot of Prefetching!!

![Graph](image.png)

**X axis → Simultaneous users sending registration requests**

**Y axis → Throughput**
AUSF with short connections

- Connection between AMF and AUSF terminated after each request/response cycle

- Throughput of the Linux kernel stack is only 23% lower than IX at saturation.

- Major overhead in the kernel stack of managing connection state

---

**Graph Details**

- **X axis → Simultaneous users**
- **Y axis → Throughput**

**Legend**

- Kernel
- mTCP
- TAS
- IX
AUSF with persistent connections

- Single connection between AMF and AUSF to service each request/response cycle

- TAS does a lot better in this case.
  - Prefetching makes sense here

- Kernel does better because of lesser state management overheads.

X axis → Simultaneous users sending registration requests

Y axis → Throughput
Key Takeaways

- Run to Completion policy works best for AUSF
  - IX doing a little better than the other network stacks
- **Linux networking stack not a bad option**
  - CPU intensive applications don’t benefit enough from the optimizations done in the data path of the networking stack.
  - Alternate network stacks hard to set up
  - Protocols like SCTP not supported by these novel network stacks
  - Stacks other than the kernel use lightweight versions of the TCP/IP networking stack
- **NFs on the 5G control plane just not CPU intensive**
  - AMF maintains and manages user state. Why is it relevant?
User State Aware Steering

NFs like load balancer maintain state at the granularity of TCP flows

AMF maintains and manages subscriber (user) context

Still locking required because user state information not encapsulated in transport protocol header

Shared user state

NIC

Packet redirected based on TCP/IP headers

No Locking needed

Network stack

Application

State for core W

State for core X

State for core Y

CORE W

CORE X

CORE Y

NIC

Packet redirected based on TCP/IP headers

CORE W

CORE X

CORE Y

AMF Network Function

Network stack
What if we had user state aware steering?

Emulated user state aware steering to enable a lock free AMF

- Lock free design outperforms the locking design by ~67%
- Worth building UE aware steering into the network stack
Summary

- Network stack design pivotal in NF’s performance
- Best performing network stack outperform the kernel network stack by at least a factor of 5 on I/O intensive workloads
- The gap between the kernel and kernel bypass stacks drop significantly (just 23%) when we move to more CPU intensive workloads.
- Some NFs in the 5G packet core would significantly benefit if application state aware packet steering is incorporated into the network stack.
Thank You! Questions?

You can also reach out to me at ashkumar@cse.iitb.ac.in