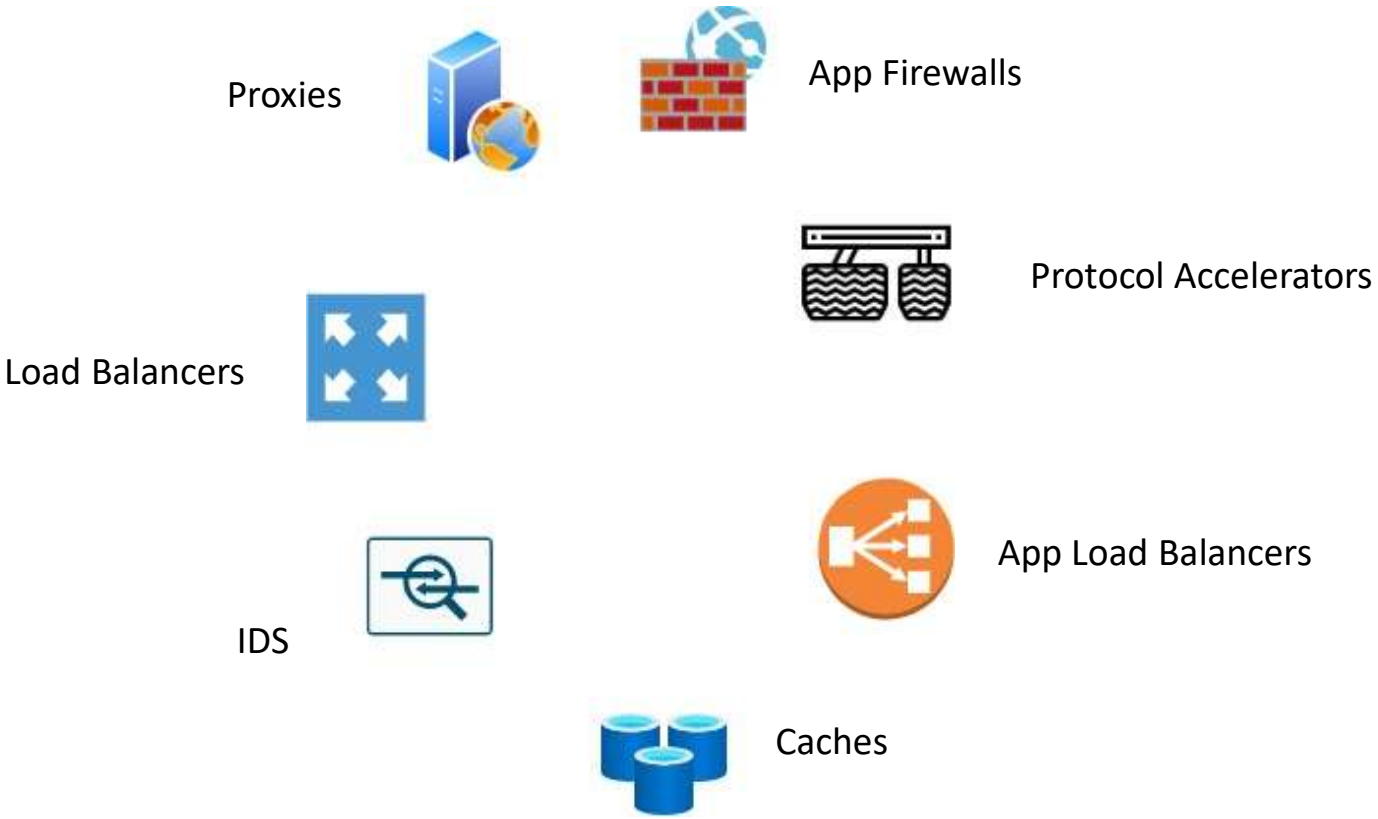
A Userspace Transport Stack Doesn't Have to Mean Losing Linux Processing

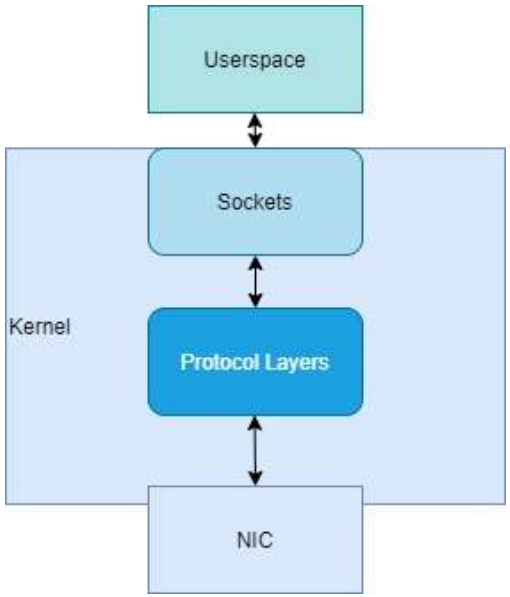
Marcelo Abranches (University of Colorado)
Eric Keller (University of Colorado)

IEEE NFV-SDN 2020

L4-L7 NFV and Applications



Kernel Networking

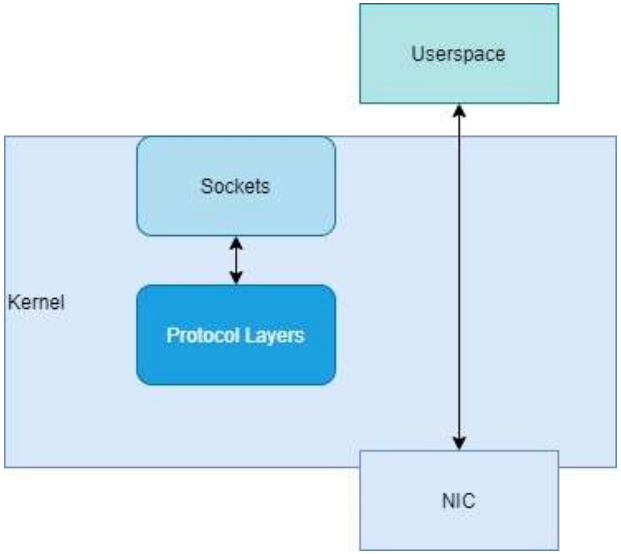


Linux Kernel Networking

- Solid Implementation
- Support to a variety of Protocols and Network Devices
- Well defined APIs
- Efficient Resource Consumption
- Efficient sharing of resources

- Heavy Weight
- May introduce unnecessary overheads
- Difficult to customize
- May slowdown innovation

Kernel-Bypass Networking



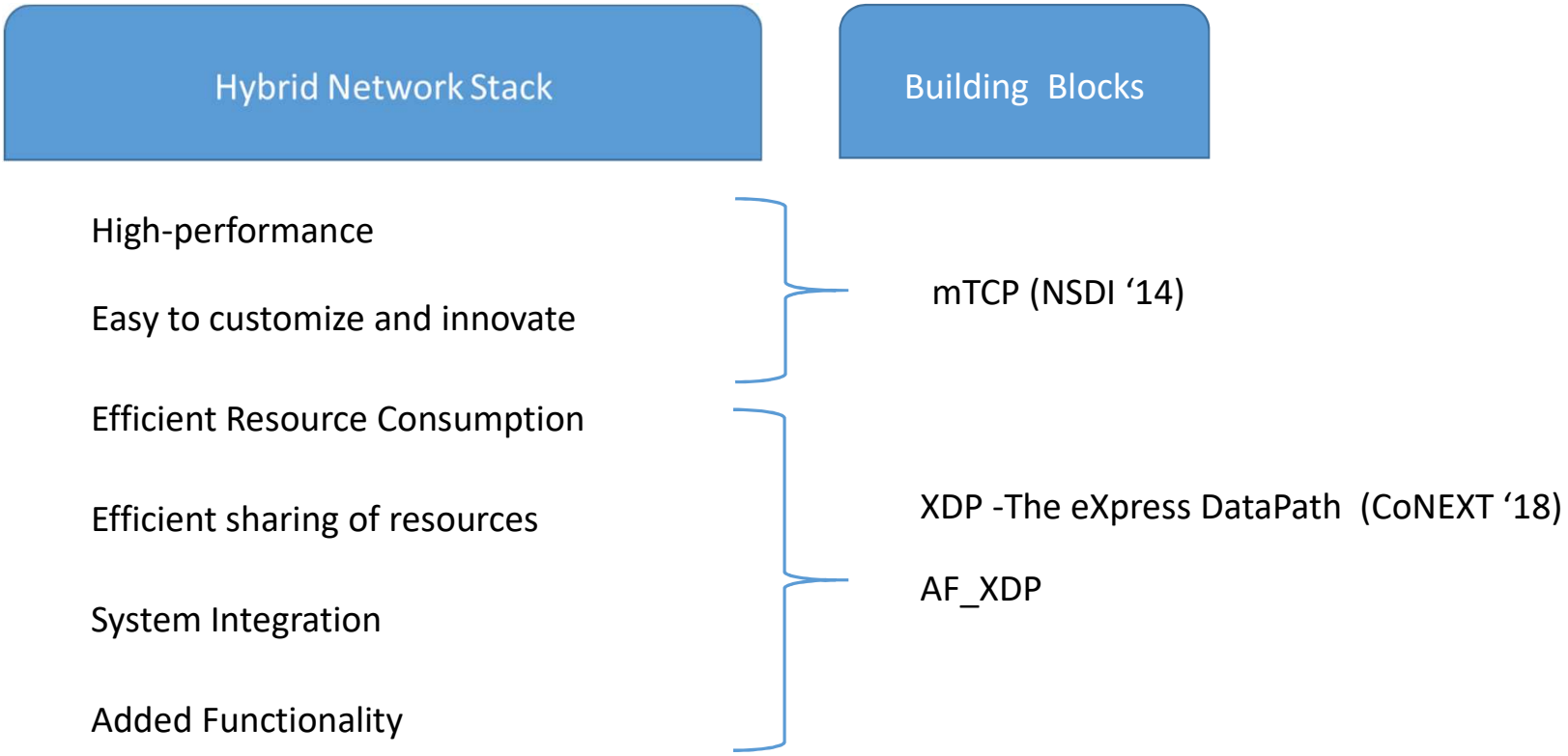
Kernel-Bypass Networking (e.g., DPDK)

- High-performance
- Easy to customize and innovate

- Inefficient resource consumption (relies on busy polling)
- Energy consumption disproportion
- Poor system integration
- Difficult to share resources
- All kernel security and isolation features are also bypassed

What if we leverage the good features provided by Linux Kernel to Enhance high-performance userspace L4-L7 Network Functions and applications?

We propose a hybrid network stack



mTCP: High-Performance Userspace TCP Stack

mTCP

Userspace TCP stack → Easy to customize/innovate → Basis of L4-7 NFV Frameworks → mOS (NSDI'17), Microboxes (SIGCOMM'18)

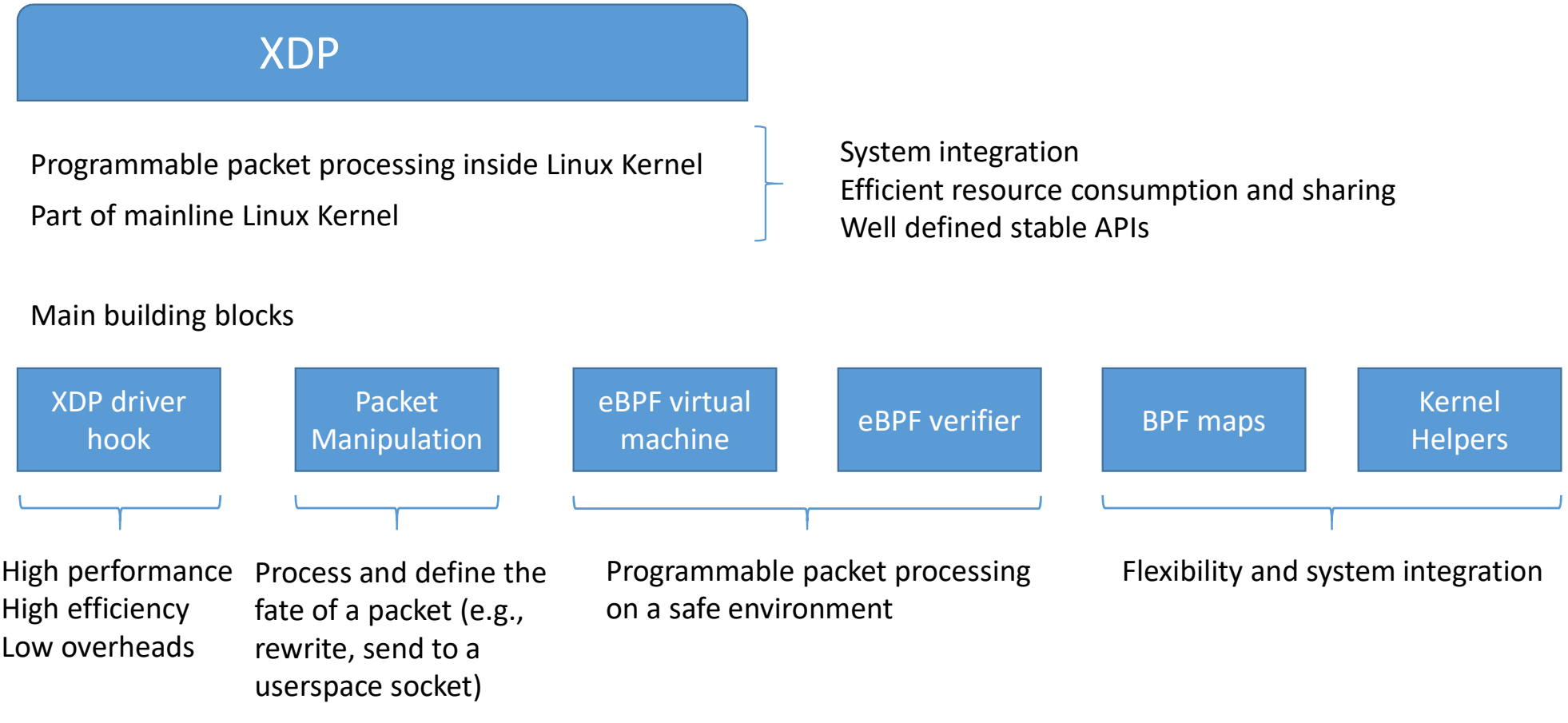
Built on top of kernel bypass technologies (e.g., DPDK)

Optimized to run on multi-core systems

Lock-free, per-core cache friendly data-structures

→ High-Performance, Highly Scalable → Up to 25 times faster than Linux TCP for small messages (see NSDI'14 paper)

The eXpress DataPath



High-Performance Socket

AF_XDP

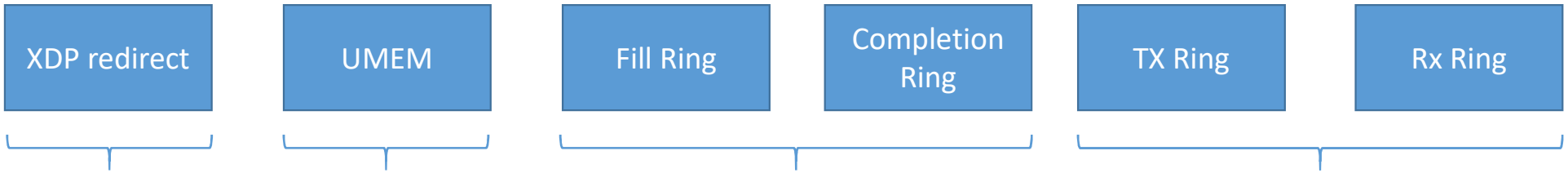
Part of mainline Linux Kernel

Allows raw packets to be sent to userspace

Packets can be preprocessed at XDP layer

Flexible kernel/userspace packet processing

Main components



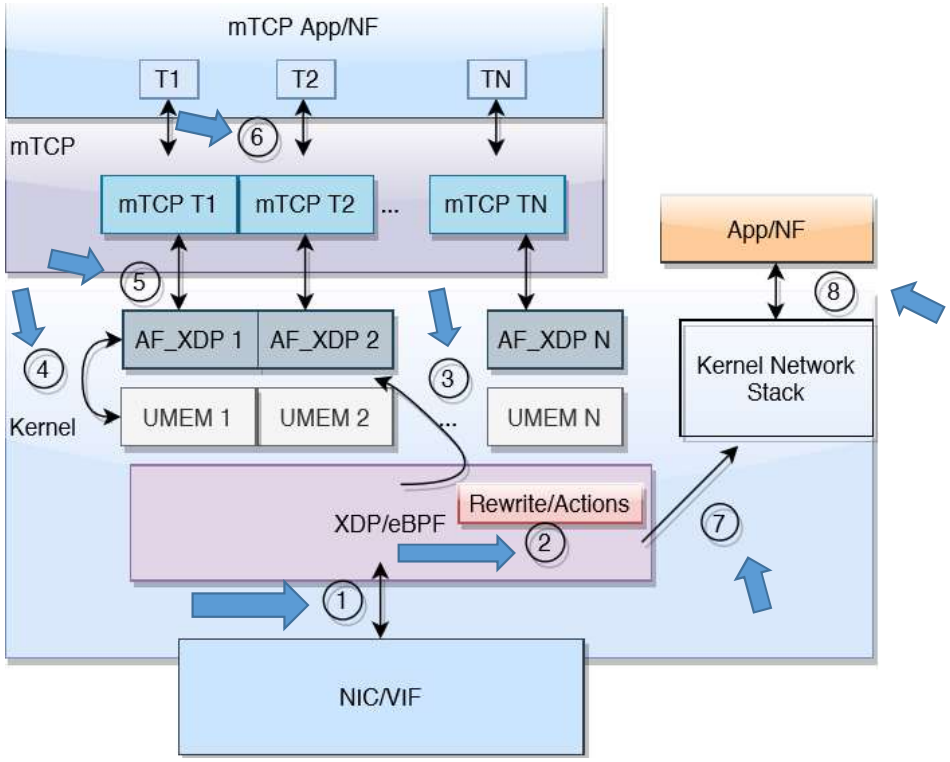
Send the packet to AF_XDP socket

Packet buffer

Transfer UMEM ownership between Kernel and userspace

Allows userspace to send and receive packets

Putting all together



Life of a packet in **mTCP/AF_XDP**…

1) The packet arrives
2) eBPF code is executed
3) Packet is sent to AF_XDP socket
4) UMEM area ownership is transferred
5) mTCP thread sends/receives packets
6) mTCP app/NF thread produce/consume data
7) Packets can be sent to Kernel
8) Kernel based apps/NFs can produce/consume data/packets

Evaluation

In our evaluation we answer the following questions

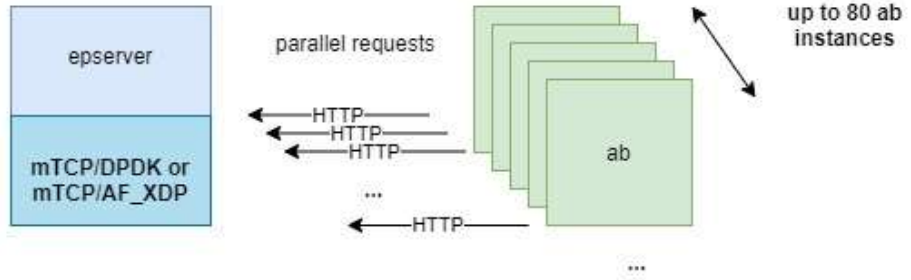
- Can our approach have good performance?
- Have a better resource consumption profile (CPU) comparing with mTCP/DPDK?
- Add new functionalities to mTCP?

Evaluation Setup

2 cloudlab Wisconsin deployments (mTCP/DPDK and mTCP/AF_XDP)

Server

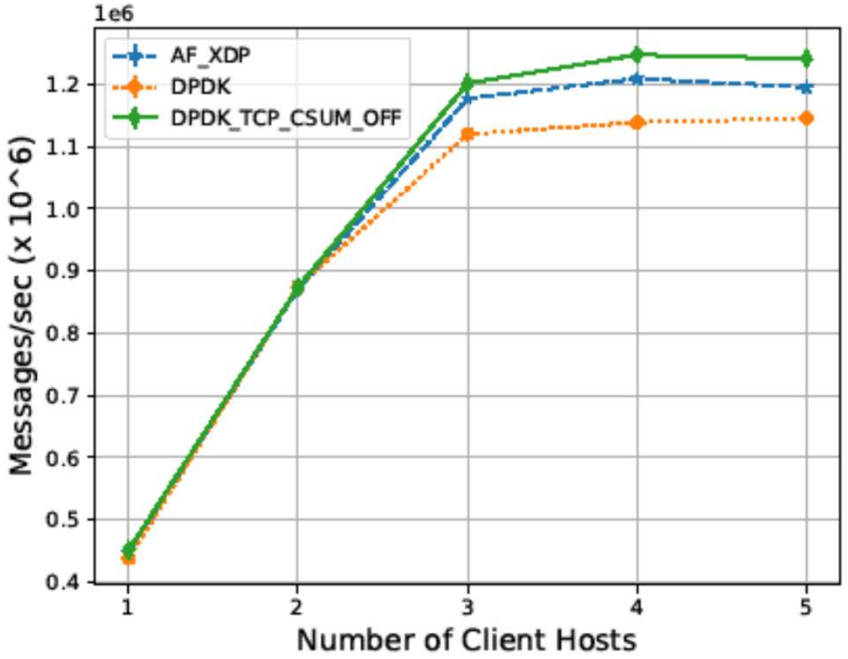
1 Server (c220g5)
HTTP server (mTCP's epserver)
Kernel 5.3.0-61-generic
Up to 8 cores
10 Gbps NIC (Intel i40e driver)

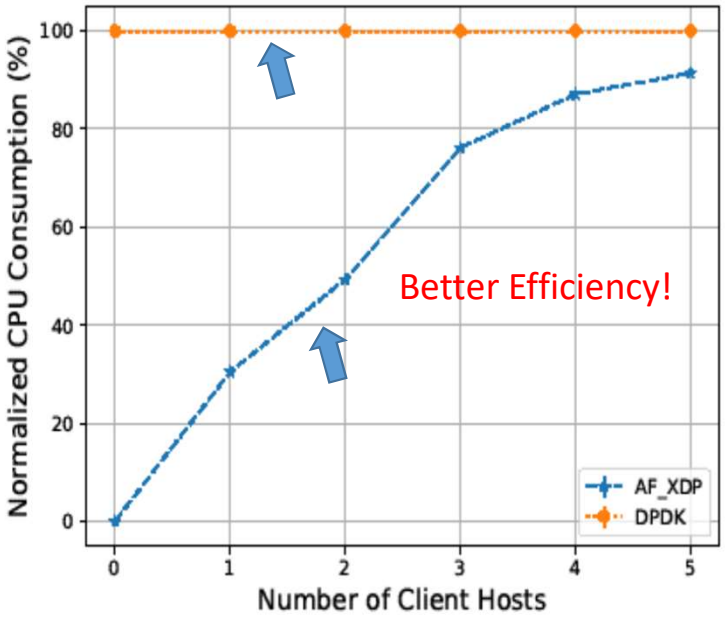


Clients

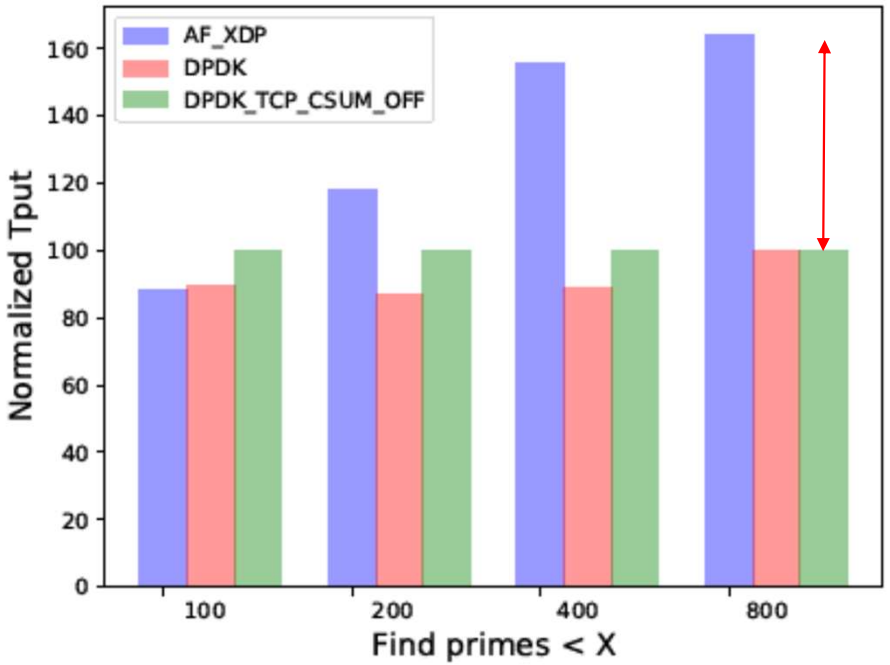
5 clients (c220g1)
Kernel 5.3.0-61-generic
16 ab instances (50 parallel HTTP connections each)
1 million downloads of a 64B file (each instance)
Up to 4000 parallel connections

CPU Efficiency





CPU intensive workload



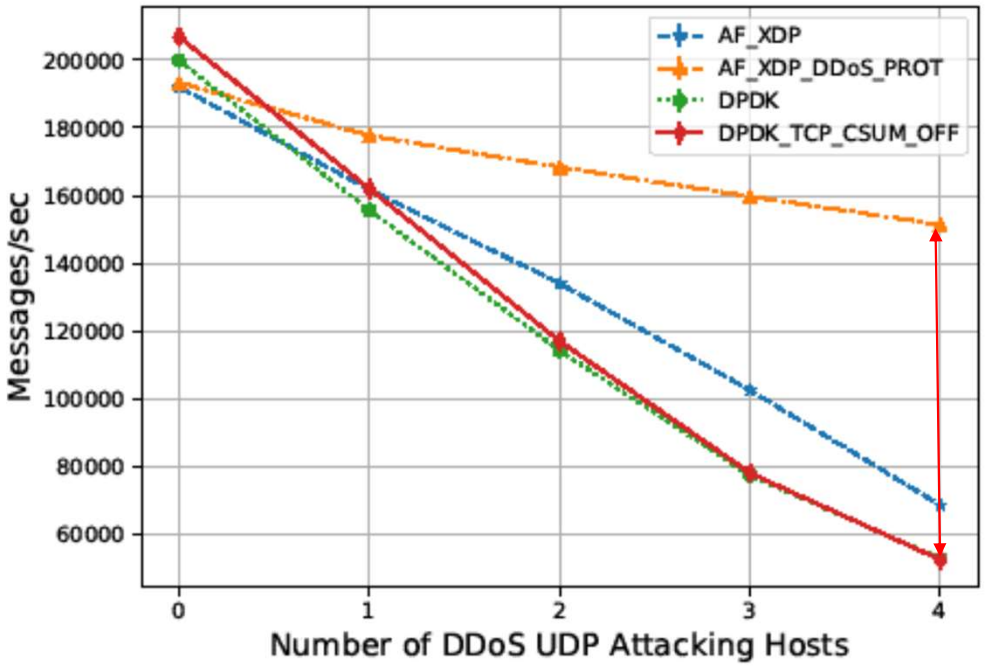
64% more throughput!

DDoS Protection

4 of the 5 clients generate malicious UDP Traffic

1 client generates benign HTTP requests to the server

Sever runs on one core



2.87x more tput

Conclusion

We enabled the power of eBPF and Linux system integration to enhance a high-performance userspace TCP stack

Our solution enables a better CPU consumption profile while maintaining high performance on the userspace stack

mTCP/AF_XDP enables better performance for CPU intensive TCP applications running on userspace

We showed the XDP layer cooperating with userspace to protect a TCP application from DDoS attack

Now that we have full and integrated programmability on both packet processing and transport layer, what new solutions and use cases can we build on top of it?

Our code is available at https://github.com/mcabranches/mtcp

Thank You!