P4 Hands-on Tutorial

Rinku Shah[1], Assistant professor, IIIT Delhi
Neeraj Kumar Yadav[1], IIIT Delhi
Keshav Ghambir[2], Microsoft India
Harish S A[3], Research Scholar (PMRF), IIT Hyderabad

[1] IIIT Delhi
India

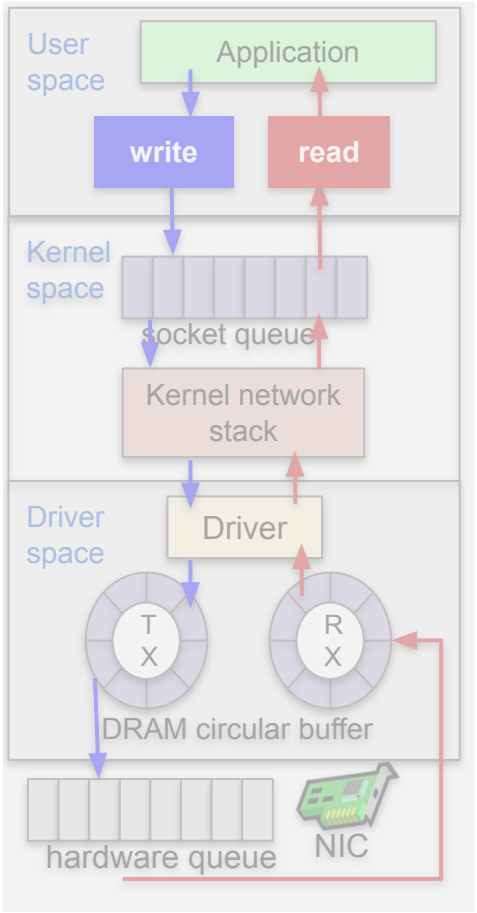
[2] Microsoft
India

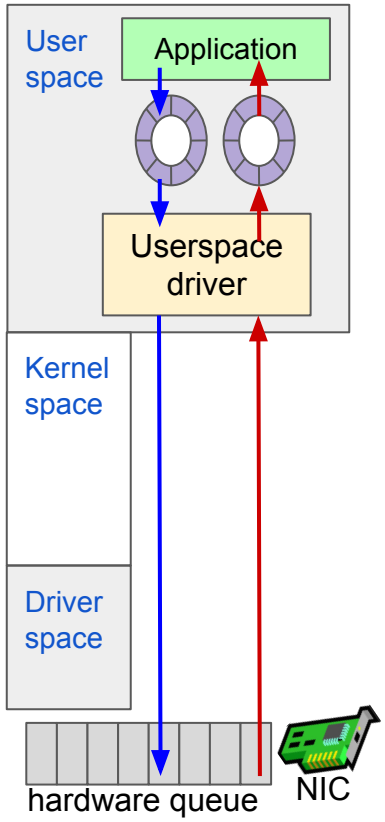
[3] IIT Hyderabad
India

Topic to be covered

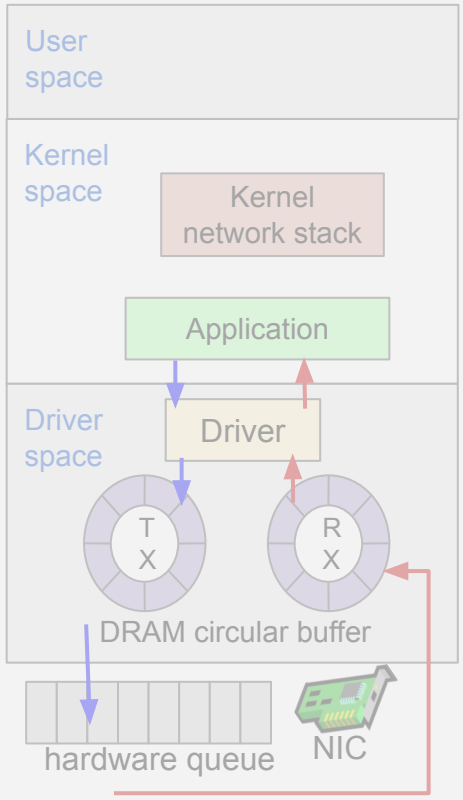
- Brief introduction on Linux Networking Stack
  - Flow of packets for transmit and receive path
  - Understanding overheads in the networking stack
- Introduction to Kernel Bypass method
- Introduction to DPDK
  - Theory
  - Demonstration of L2 Forwarding using DPDK
  - Compilation of P4 program to DPDK
- Using P4 with eBPF
  - Revision of eBPF
  - How to convert P4 program to eBPF
  - Packet filtering demo and exercise
  - L2 Forwarding Demonstration

Evolution of network packet processors

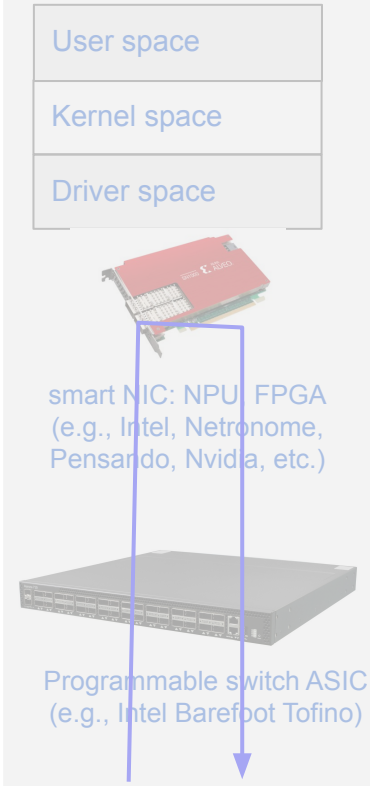




Kernel bypass (e.g. netmap, DPDK)

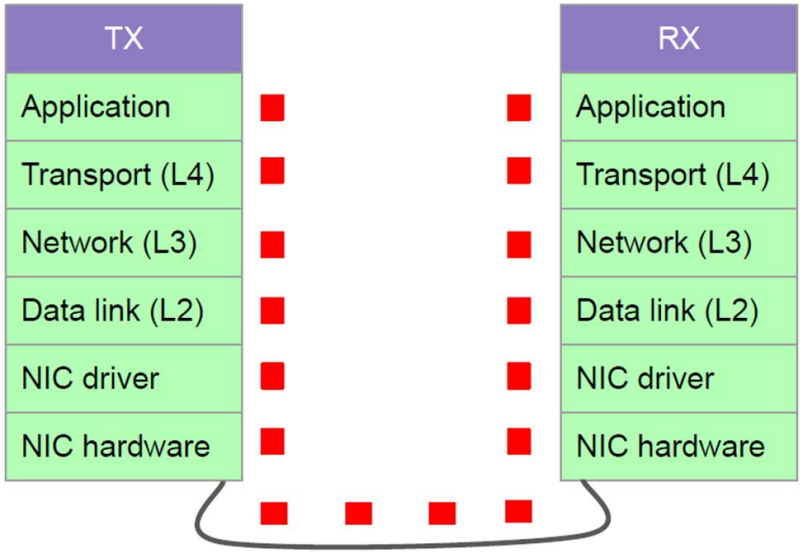


in-kernel compute (e.g. eBPF, XDP)



in-network compute (NIC/switch)

Flow of Packets Between 2 Machine

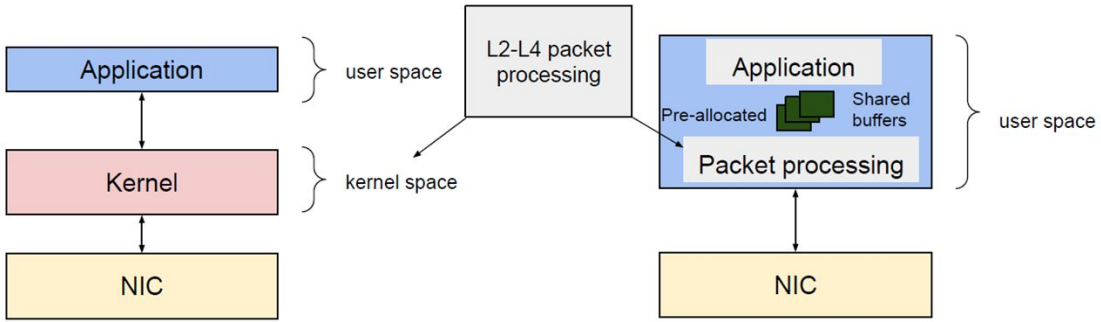


Overheads in Networking Stack

- System calls and context switch overheads
  - Hardware Interrupts + software Interrupts + read/write system calls
- Inefficient per packet processing
  - Heavy data structure allocation
  - DMA overheads
- Shared File Descriptor space
  - Sharing of file descriptor space amongst various threads
  - Sharing of VFS
- Lack of connection locality
  - Sharing of listening socket to accept connections
  - Cache miss and cache line sharing in multicore system

Kernel Bypass Methods

- Pushing L2 to L4 processing in User Space
- This helps in:
  - Removal of context switching
  - Packet copy overhead
  - Memory overheads



2 ways of Kernel Bypass Techniques

Interrupt Mode

- NIC notifies the application using hardware interrupt for a batch of packets

Poll Mode

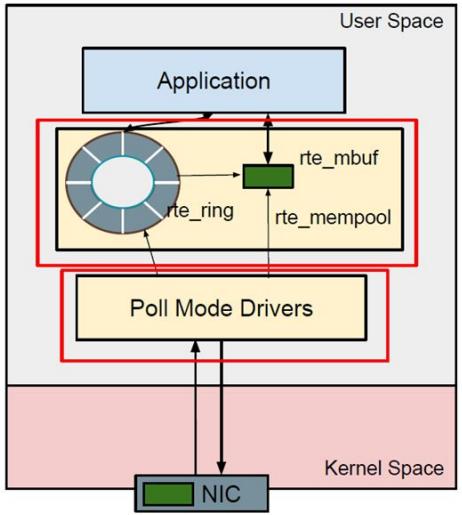
- CPU keeps checking for control bits for any packets that has been enqueued in NIC hardware queue.

Data Plane Development Kit (DPDK)

- DPDK was created by Intel in the year 2010 and made public in 2013
- Kernel bypass method based on poll mode
- Supported by wide variety of network card for fast packet processing

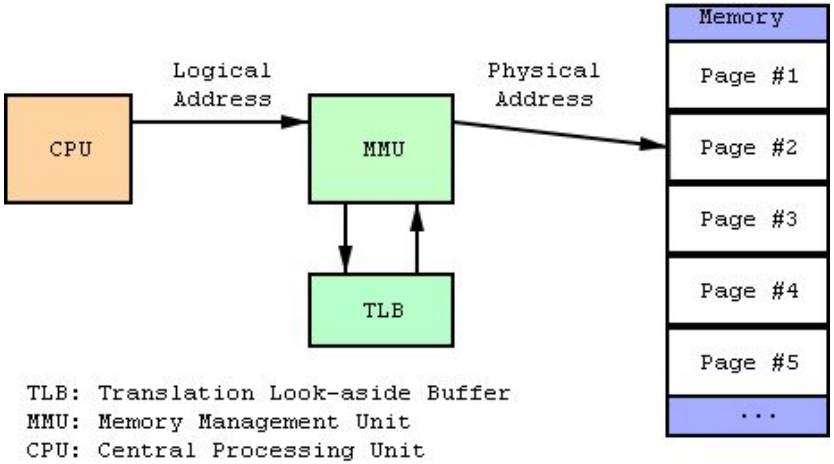
Components of DPDK

- Poll Mode Driver: Driver for constantly polling the NIC hardware queue for
- Rte_mempool: Pre-allocated memory which supports HUGE Pages
- Rte_mbuf: contains network packet buffer
- Rte_ring: TX and RX circular queues which holds pointers to Rte_mbuf



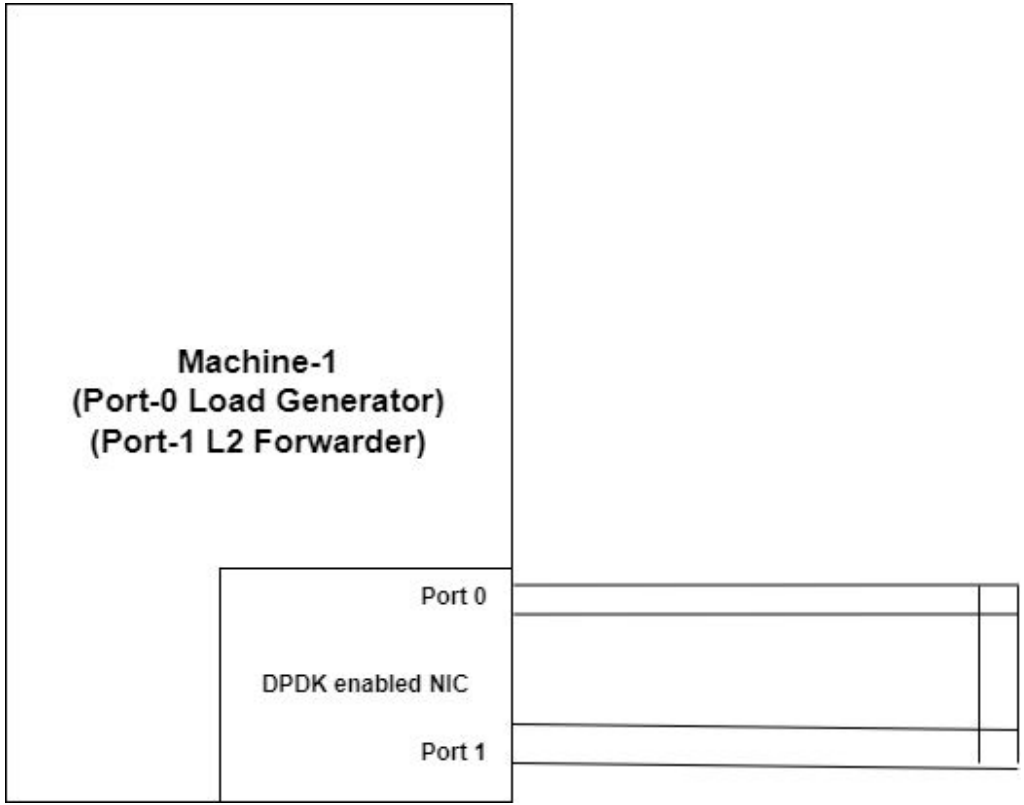
HugePages in DPDK

- HugePages are used to increase the performance of DPDK applications by reducing the number of TLB cache miss and page replacements
- Reduces the number of translations between virtual and physical memory



DEMO-1: Simple L2 Forwarding

Experimental Setup



Intel 810E NIC

- **100 Gb NIC**
- **Contains 2 Ports**

Steps for running L2 Forwarding

- Step-1: Setting up Huge Pages using the following command
  - sudo dpdk-hugepages.py --setup 4G --pagesize 1G
- Step-2: Attaching NIC to poll mode driver
  - Navigate to usertools/ directory inside the dpdk directory
  - sudo ./dpdk-devbind -b vfio-pci <pci-bus-address-1>
  - sudo ./dpdk-devbind -b vfio-pci <pci-bus-address-2>
- Step-3: Running the DPDK L2 Forward example
  - Navigate to examples/l2fwd folder
  - sudo make
  - Navigate to build/ folder in l2fwd
  - sudo ./l2fwd -l 0-3 -n 4 -- -q 8 -p f --portmap="(0,1)"
- Step-4: Run Packetgen using the following command
  - sudo ./app/pktgen -l 0-4 -n 3 -- -P -m "[1].0, [2].1"

Enabling P4 in DPDK

- We can use P4 language to write the code and compile it to DPDK
- Advantages of using P4
  - Much easier to code in P4 compared to C
  - P4 code is hardware independent
  - P4 code is cross compilable across various hardwares

```c
static void
l2fwd_simple_forward(struct rte_mbuf *m, unsigned portid)
{
        unsigned dst_port;
        int sent;
        struct rte_eth_dev_tx_buffer *buffer;

        dst_port = l2fwd_dst_ports[portid];

        if (mac_updating)
                l2fwd_mac_updating(m, dst_port);

        buffer = tx_buffer[dst_port];
        sent = rte_eth_tx_buffer(dst_port, 0, buffer, m);
        if (sent)
                port_statistics[dst_port].tx += sent;
}
```

```
control DemoIngress(inout headers hdr, inout metadata meta,
  bit<48> tmp;
  apply {
    tmp = hdr.ethernet.src_addr;
    hdr.ethernet.src_addr = hdr.ethernet.dst_addr;
    hdr.ethernet.dst_addr = tmp;
    ostd.egress_port = istd.ingress_port;
  }
}
```

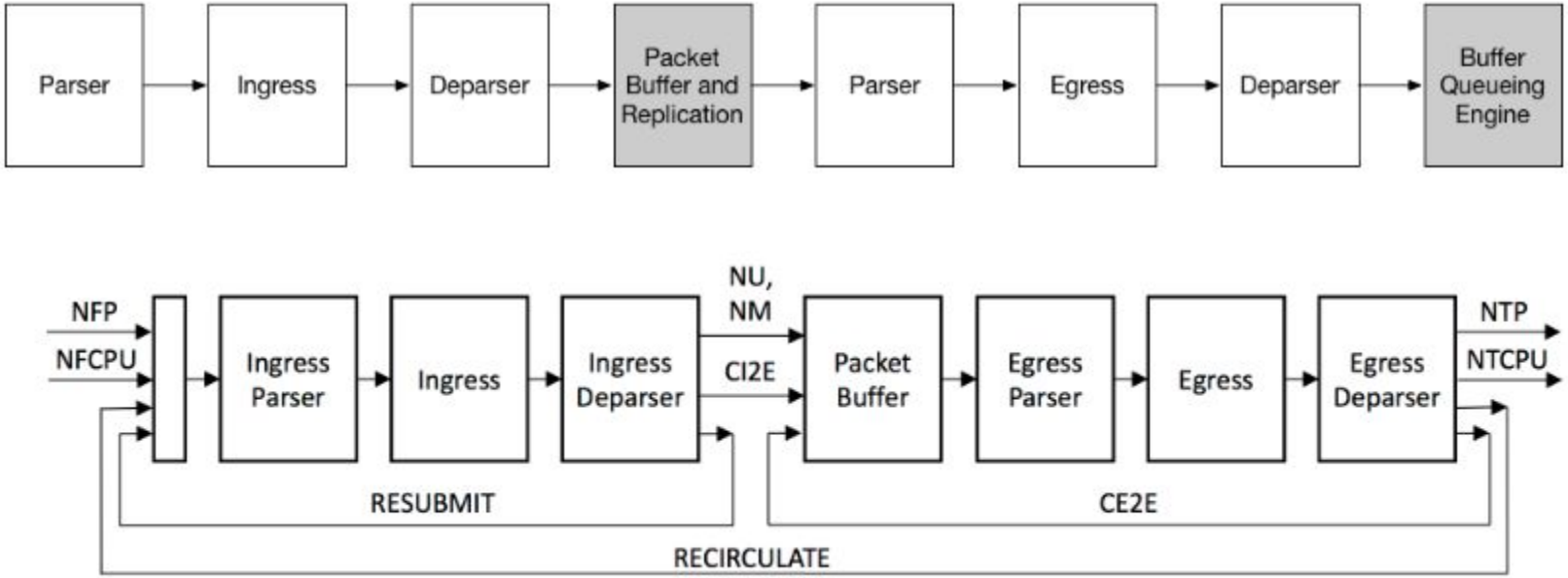
Hence we only need to code once!!!

Compiling P4 code DPDK

p4lang/**p4-dpdk-target**

P4 driver SW for P4 DPDK target.

PSA Architecture



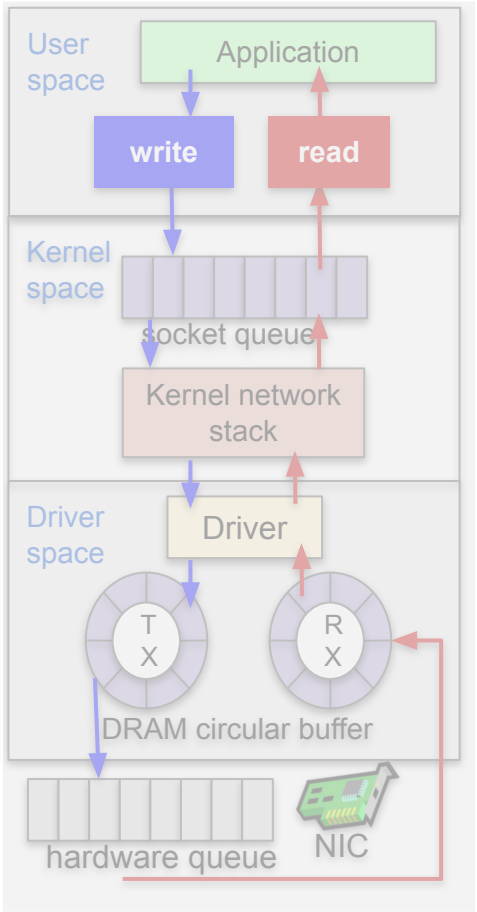
Exercise-1: Compiling P4 code to DPDK

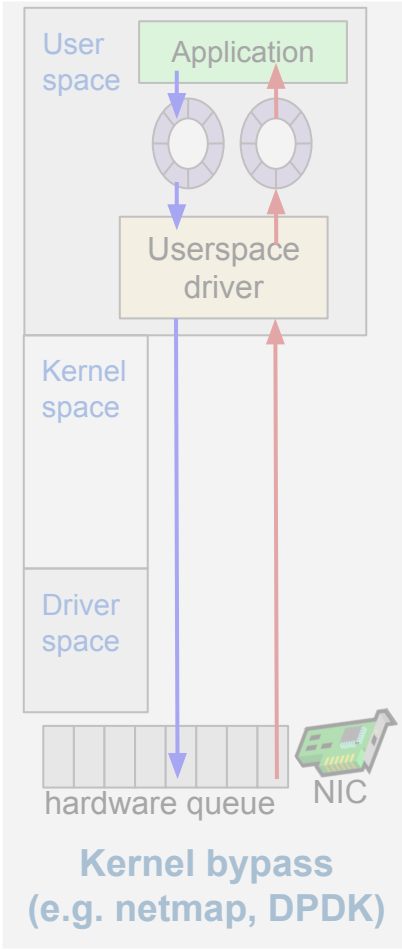
Steps for compilation of P4 code to DPDK

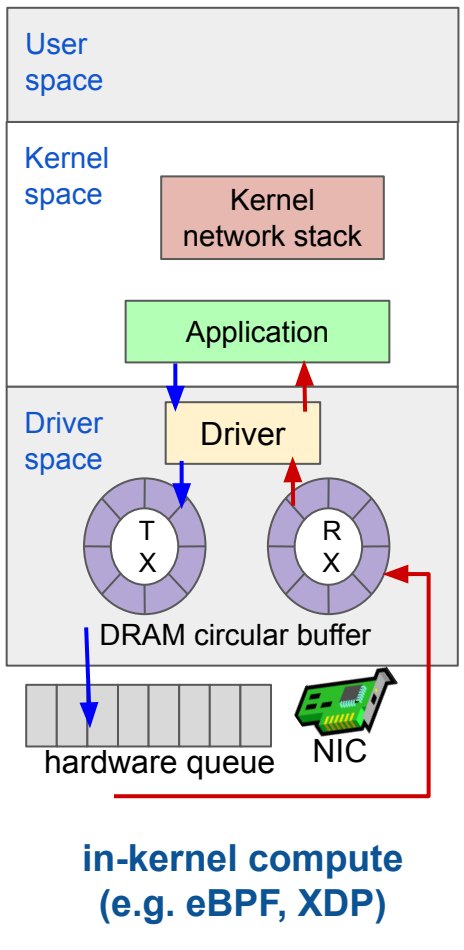
- Navigate to the repository where the P4 code is written in PSA arch.
- Run the following command
  - p4c-dpdk --arch psa l2fwd.p4 -o l2fwd.spec
  - This will be generate the .spec file for the PSA architecture which can be loaded to pipeline
- To run the .spec file generated we need to use the pipelines example in the dpdk folder
  - Navigate to example/pipelines in the dpdk root directory
  - sudo make
  - Navigate to the new build folder
  - sudo ./pipeline -l 0-3 -n 4 -- -s <path to l2fwd.spec file>
- Note: Before running the code we need to setup hugepages and bind NIC to the vfio-pci driver

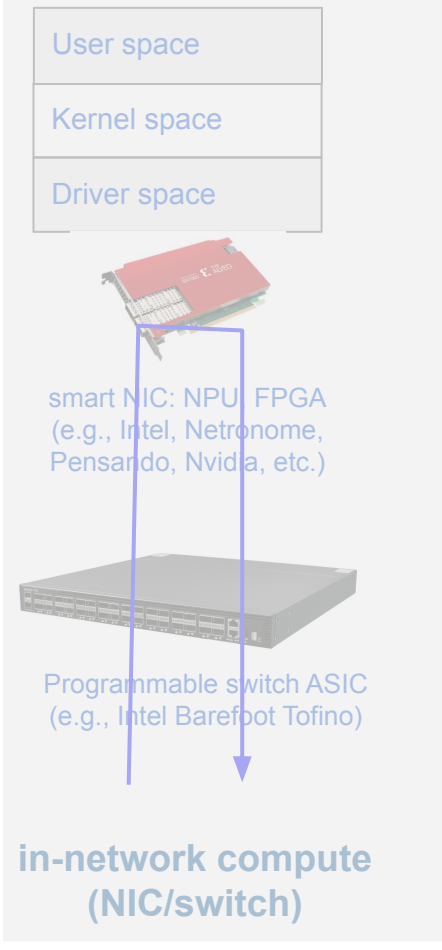
P4 + eBPF

Evolution of network packet processors



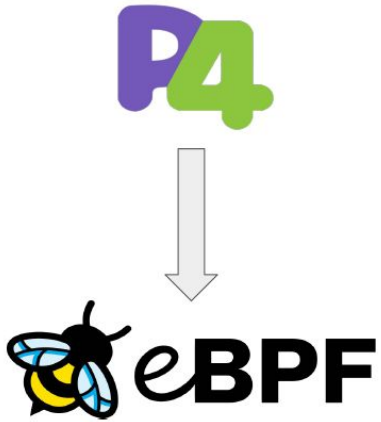




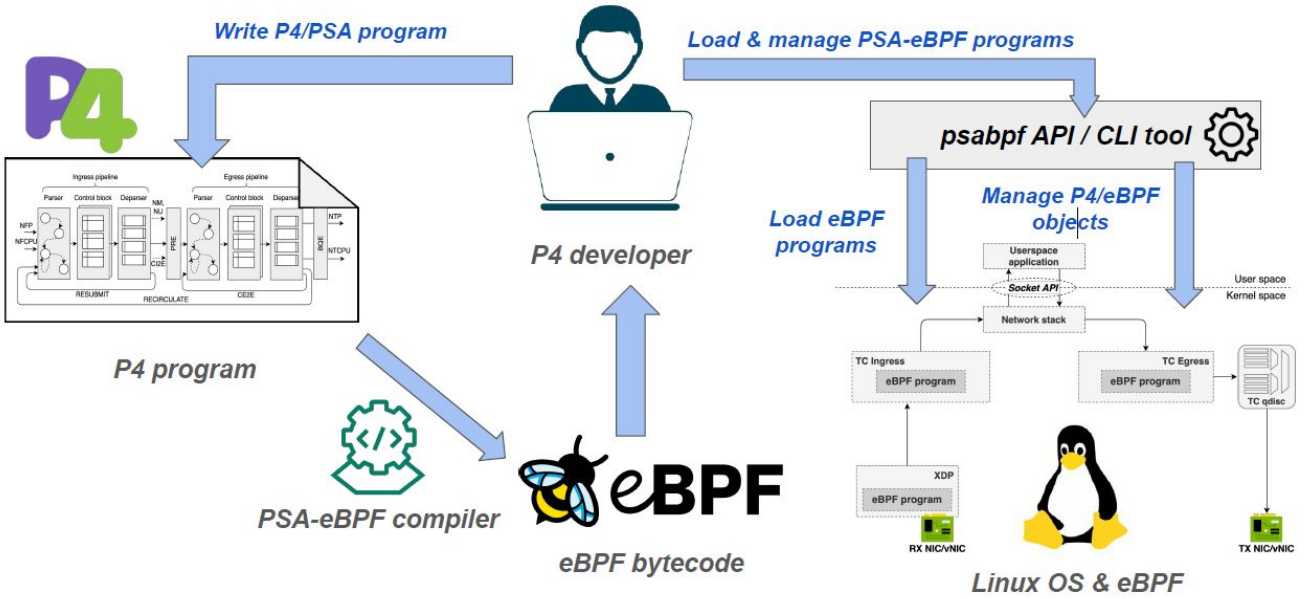


Compiling P4 program for eBPF target

- P4 compiler implements eBPF backend which can be used to compile P4 code to eBPF.
- The P4 supported architectures are Portable switch architecture and Portable NIC architecture.



Overview of PSA-eBPF compiler



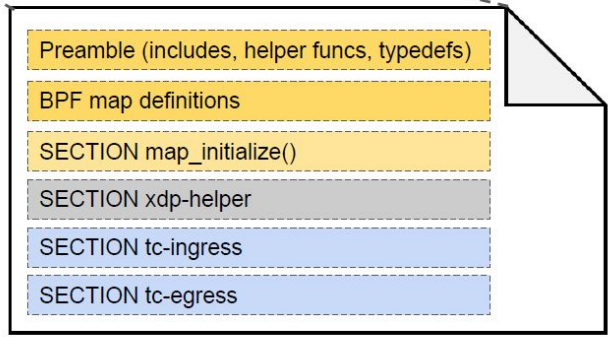
Compiling P4 program to eBPF bytecode

To compile:

```
$ p4c-ebpf -arch psa demo.p4 -o demo.c
$ P4C=$(path_to_p4c_repository)/backends/ebpf/runtime
$ clang -O2 -g -c -DPSA_PORT_RECIRCULATE=2 -I$P4C/usr/include/
-I$P4C/ -emit-llvm -DBTF -o demo.bc demo.c
$ llc -march=bpf -mcpu=generic -filetype=obj -o demo.o demo.bc
```

Two-step process automated via Makefile:

```
$ make -f ${P4C_REPO}/backends/ebpf/runtime/kernel.mk \
BPFOBJ=out.o P4FILE=program.p4 ARGS="-DPSA_PORT_RECIRCULATE=2" \
psa
```



Structure of C program generated by PSA-eBPF compiler

Exercise-2: Blocking ICMP packets

Steps to compile and load P4 code to eBPF

- Run the bash script to compile and run the environment
  - bash start.sh
- To compile the P4 code to eBPF run the following command inside the directory
  - bash compile.sh
- Load the generate .o from the above command output and load it to XDP hook of virtual network interface
  - sudo ip netns exec vnet0 ip link set veth0 xdpgeneric obj demo.o sec xdp_ingress/xdp-ingress
- This will load the .o eBPF code to veth0 and this can be tested using the following command
  - sudo ip netns exec vnet1 ping 10.0.0.1

Thank You