eXpress Data Path (XDP)

Programmable and high performance networking data path

Tom Herbert <therbert@fb.com>
Alexei Starovoitov<ast@fb.com>

What is XDP?

- A programmable, high performance, specialized application, packet processor in the Linux networking data path
- Bare metal packet processing at lowest point in the SW stack
- Use cases include
  - Pre-stack processing like filtering to do DOS mitigation
  - Forwarding and load balancing
  - Batching techniques such as in Generic Receive Offload
  - Flow sampling, monitoring
  - ULP processing (e.g. message delineation)

Properties

- XDP is **designed for high performance**. It uses known techniques and applies selective constraints to achieve performance goals
- XDP is also **designed for programmability**. New functionality can be implemented on the fly without needing kernel modification
- XDP is **not kernel bypass**. It is an integrated fast path in the kernel stack
- XDP does **not replace the TCP/IP stack**. It augments the stack and works in concert (*Don’t throw the baby out with the bathwater!*)
- XDP does **not require any specialized hardware**. It espouses the *less is more* principle for networking hardware

Relationship to kernel bypass

Think of it this way:

> *If the traditional kernel network stack is a freeway, kernel bypass is a proposal to build an infrastructure of high speed trains and XDP is a proposal for adding carpool lanes to the freeway.*

Performance techniques

- Lockless
- Batched I/O operations
- Busy polling
- Direct queue access
- Page recycling to avoid page allocation/free where possible
- Packet processing without meta data (skbuff) allocation
- Efficient table (flow state) lookup
- Packet steering
- Siloed processing, minimize cross CPU/NUMA node ops
- RX flow hash
- Common NIC offloads
- Judicious cache prefetch, DDIO

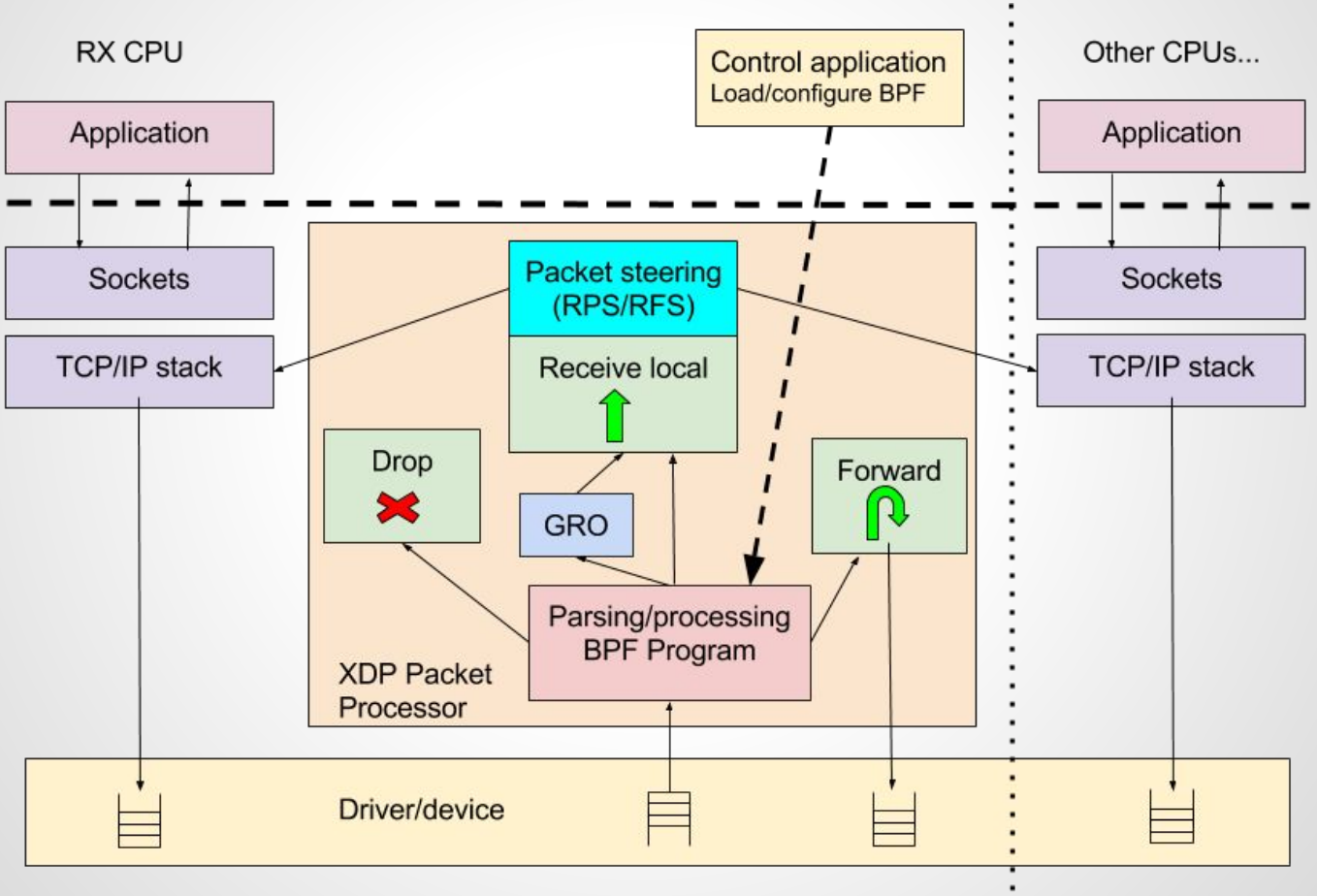
A few hardware requirements

- Multi-queue NICs
- Common **protocol-generic** offloads
  - TX/RX checksum offload
  - Receive Side Scaling (RSS)
  - Transport Segmentation Offload (TSO)
- LRO, aRFS, flow hash from device are “nice to have”s

XDP packet processor

- In kernel component that processes RX packets
- Process RX “packet-pages” directly out of driver
  - Functional interface
  - No early allocation of skbuff’s, no SW queues
- Nominally assign one CPU to each RX queue
  - No locking RX queue
  - CPU can be dedicated to busy poll or use interrupt model
- BPF programs performs processing
  - Parses packets
  - Performs table lookups, creates/manages stateful filters
  - Manipulates packet (e.g. for encap/decap)
  - Returns action

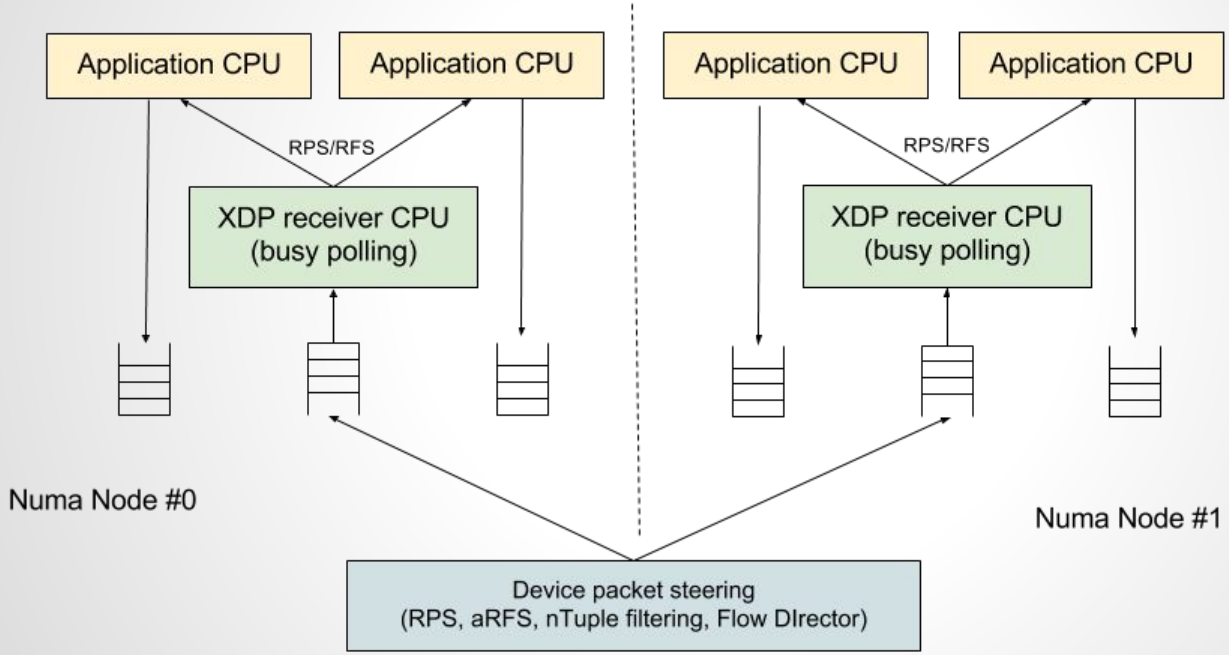
XDP packet processor



Basic XDP packet processor actions

- Forward
  - Possibly after packet modification (e.g. NAT, ILA router)
  - TX queue is exclusive to same CPU so no lock needed
- Drop
  - Just return error from the function
  - Driver recycles pages
- Normal receive
  - Allocate skbuff and receive into stack
  - Steer packet to another CPU for processing
  - Allows “raw” interfaces to userspace like AF_PACKET, netmap
- Generic Receive Offload
  - Coalesce packets of same connection
  - Perform receive of large packets

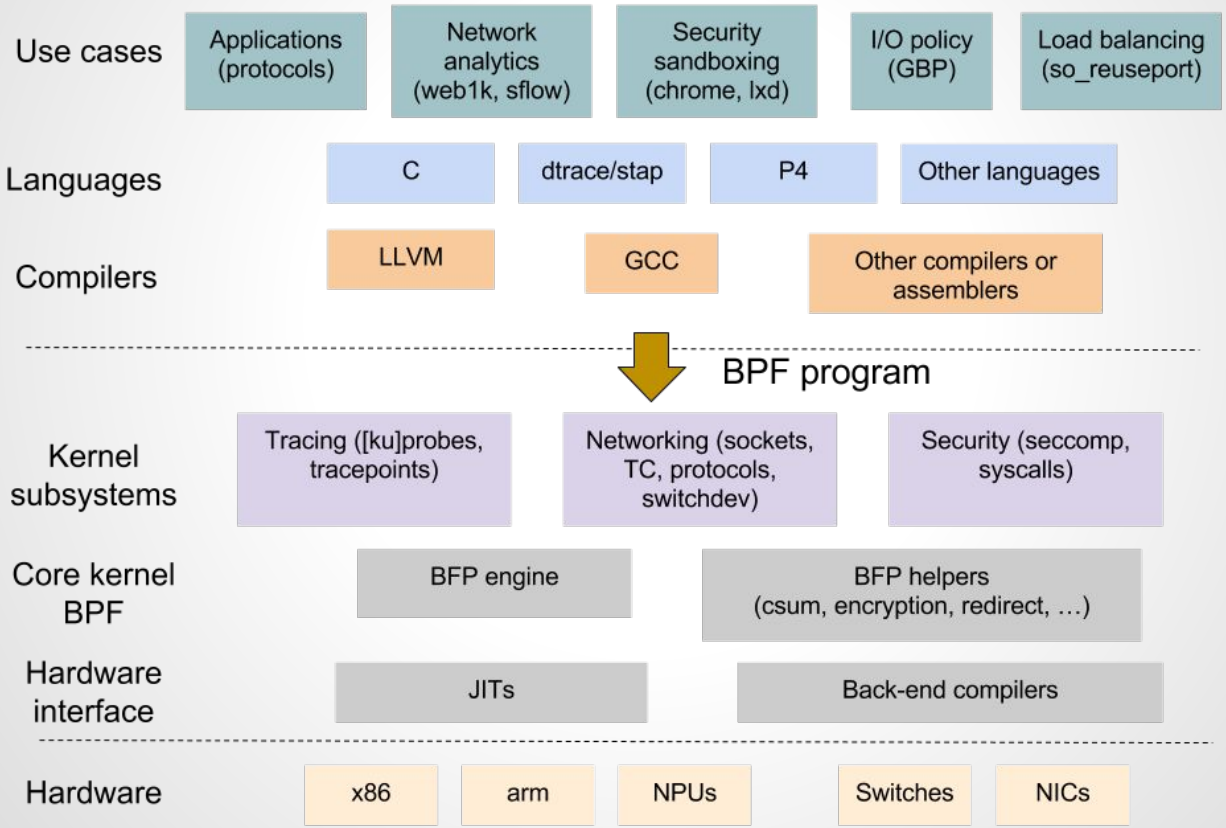
System configuration example



Programmability

- Packet inspection, action found by **BPF** programs
  - Flexible (loop free) protocol header parsing
  - Maybe stateful as a result of flow lookup
  - Simple packet field rewriting (encap/decap)
- Optimized lookup
  - Flow lookups (e.g. using flow hash provided by device)
  - Fixed length lookups (e.g. 64-bit identifier lookup for ILA)
  - Statefulness, make transient states (e.g. needed for GRO)
- Extensible model
  - Application processing (e.g. app. layer protocol GRO)
  - Leverage ongoing work offload BPF to HW
  - BPF programs can be portable to userspace or other OSes

BPF Architecture



Advantages of XDP over DPDK

- Allows option of busy polling or interrupt driven networking
- No need to allocate huge pages
- Dedicated CPUs are not required, user has many options on how to structure the work between CPUs
- No need to inject packets into the kernel from a third party userspace application
- No special hardware requirements
- No need to define a new security model for accessing networking HW
- No third party code/licensing required

Performance goals

- Packet drop (e.g. DOS mitigation)
  - 20M pps per CPU
- Forwarding (e.g ILA router)
  - 14M pps per CPU
- Generic Receive Offload
  - 100Gbps rate served by single CPU