P4 on the Edge

John Fastabend

Intel

Legal Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families: Go to: Learn About Intel® Processor Numbers

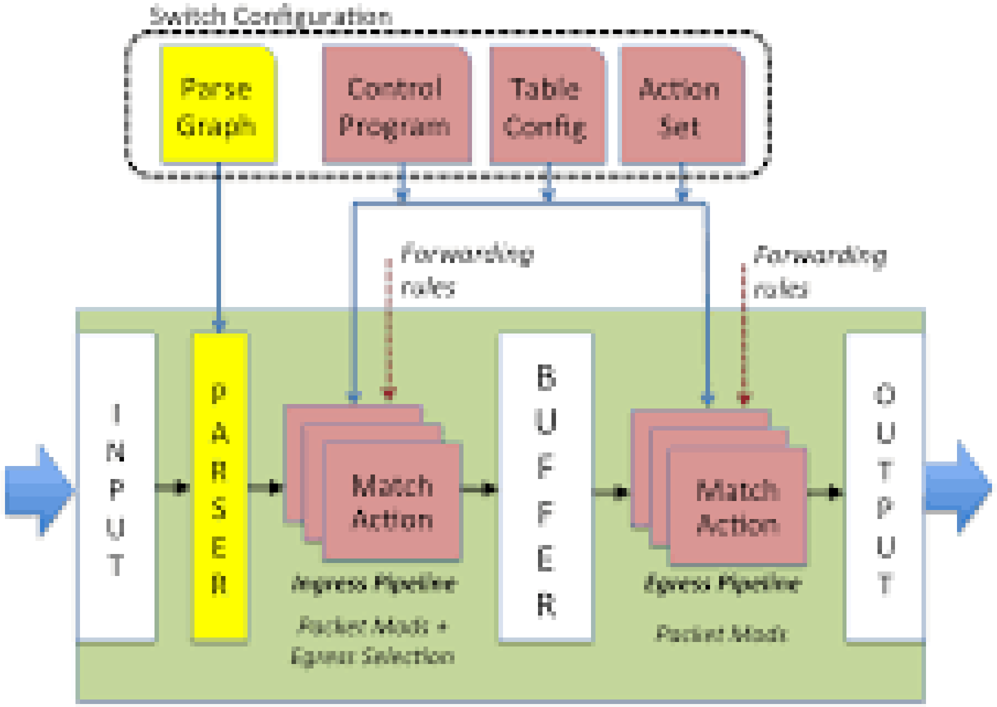
Intel, the Intel logo, Intel Atom, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Linux Datapath

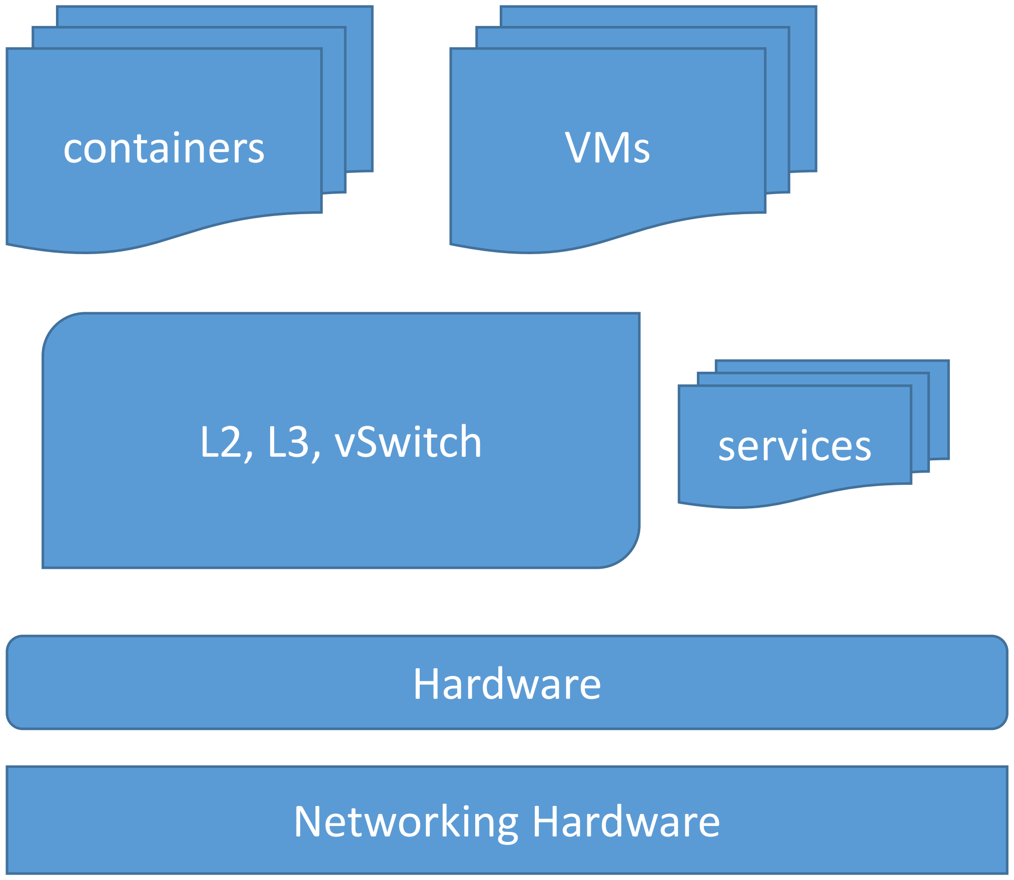
- Using P4 on the edge nodes
  - Is P4 a useful abstraction for an edge node?
- Tools:
  - eBPF : instruction set, “maps”, kernel helper routines
  - LLVM : “The LLVM Project is a collection of modular and reusable compiler and toolchain technologies.”
  - TC : traffic classifier – loads eBPF programs into ingress/egress path of networking stack.

Canonical P4 Model Architecture



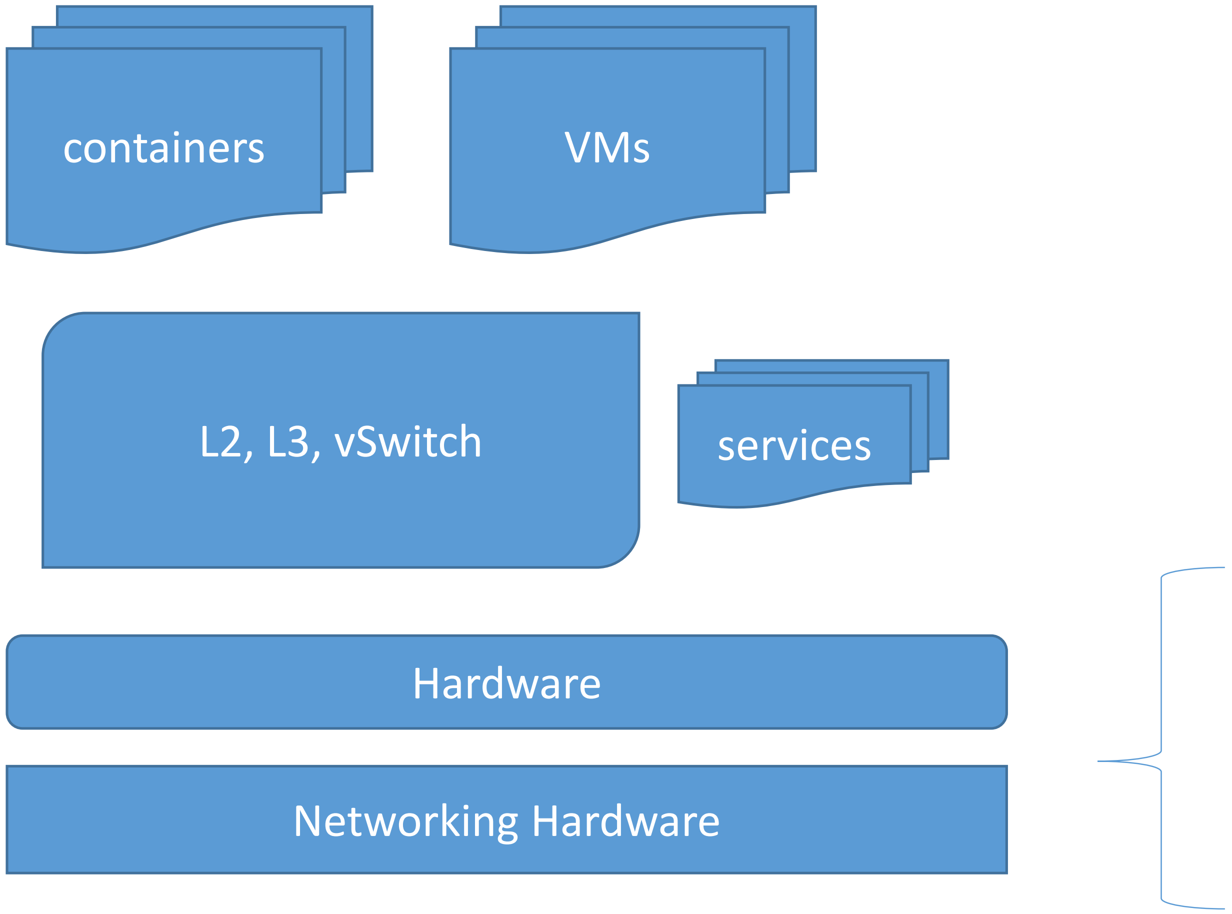
https://arxiv.org/pdf/1312.1719.pdf

Edge platform



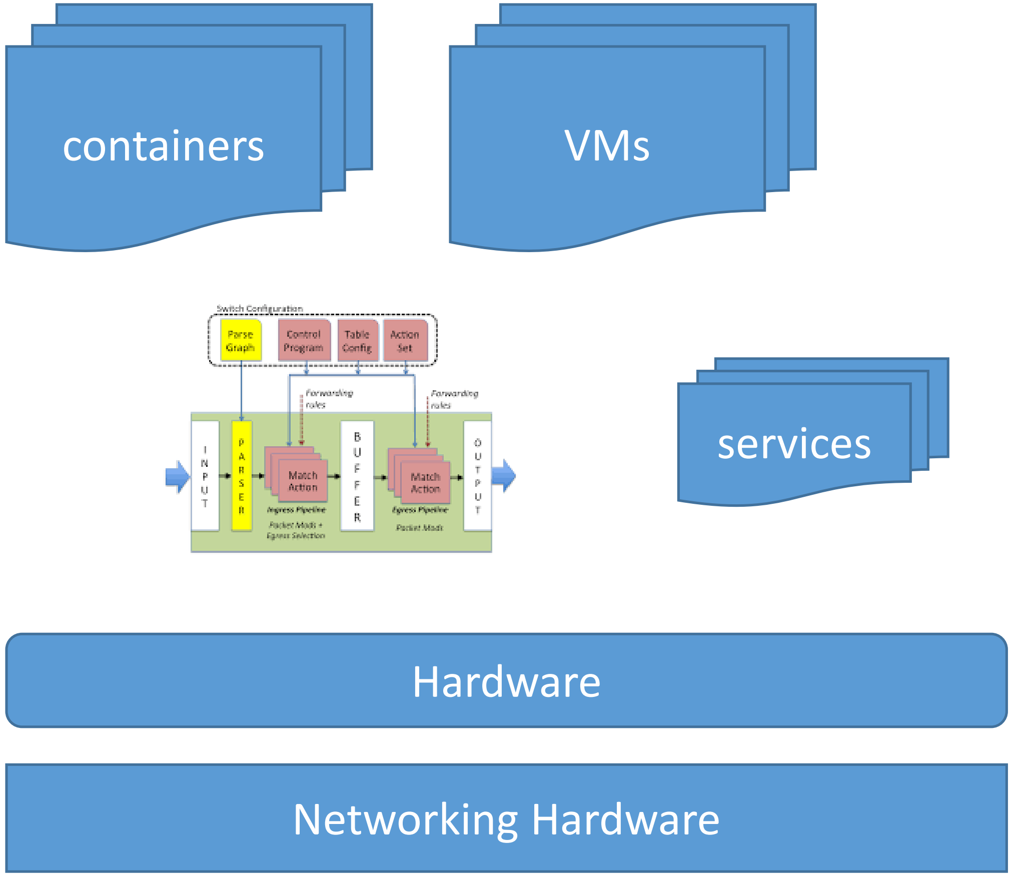
- Containers/VMs
  - virtual ports
  - create/migrate/destroy semantics
- Services (State!)
  - Connection Tracking
  - NAT
  - Load Balancing
- Forwarding Model
  - L2, L3, vSwitch, sockets
  - ACLs
- TEP
  - Port Based
  - Flow Based

Edge platform (hardware)



- Collection of CPUs
- Plus networking component (FPGA, NICs, NPU, etc)

Edge platform



containers

VMs

services

Hardware

Networking Hardware

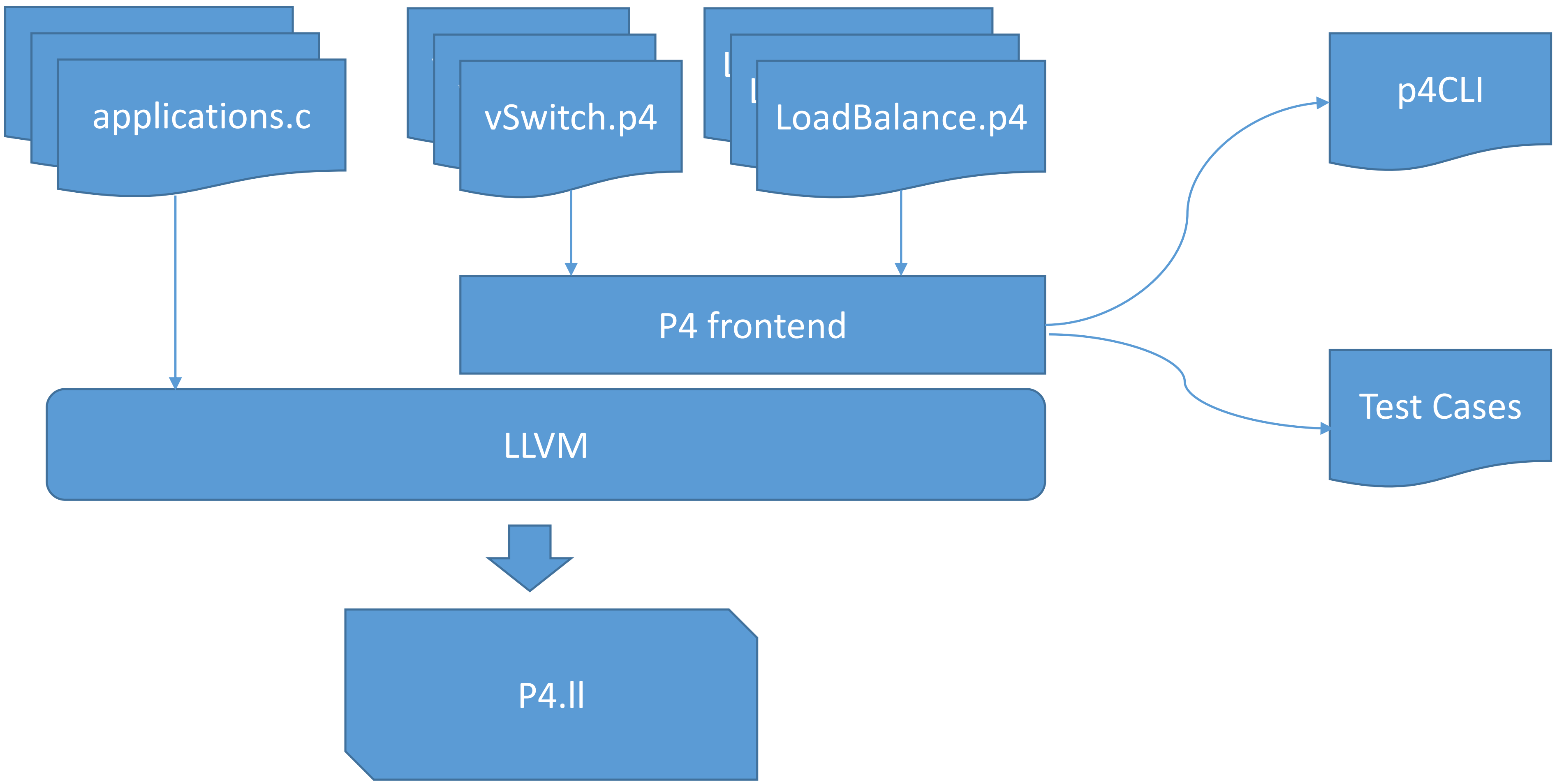
Integrate a P4 programming model with the flexibility of software.

P4 LLVM Development Environment

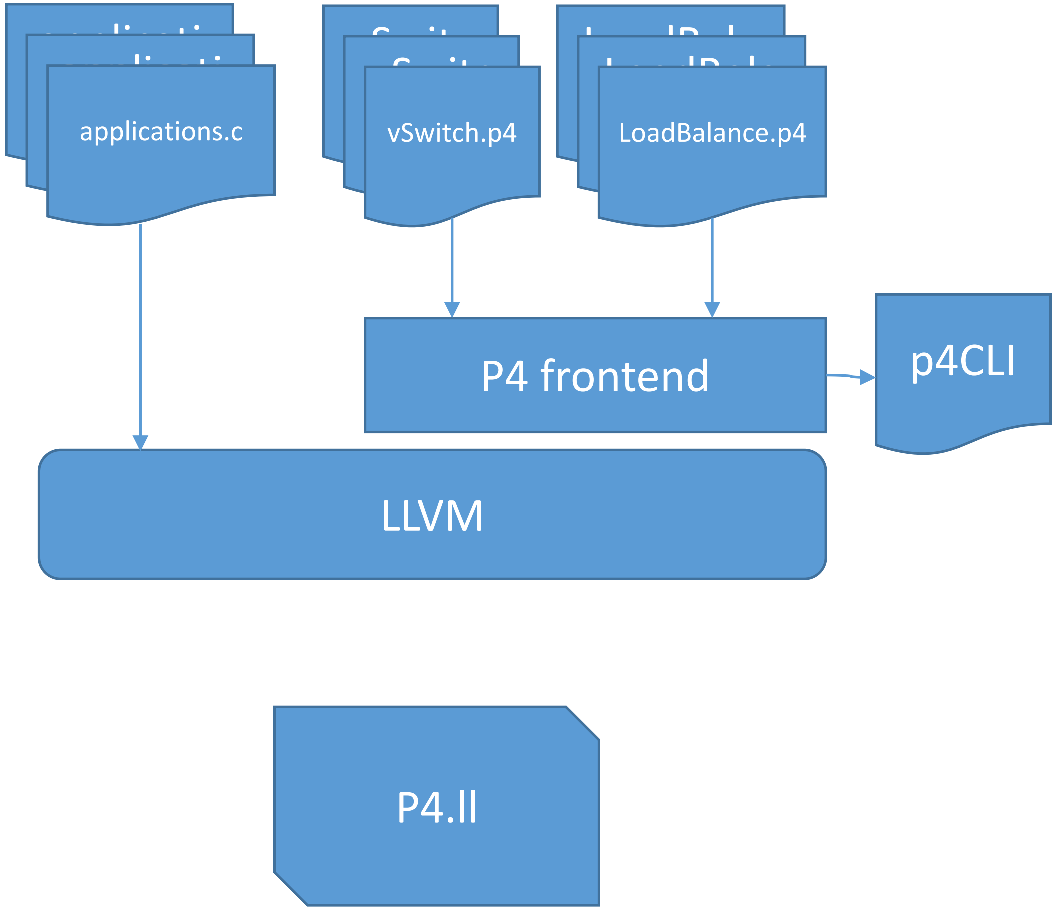
BPF

- Injects “**program**” into Linux kernel at ingress/egress hooks
- “**maps**” share data between programs and user space
- OS helper routines for basic operations, redirect, drop, set_field, etc.
- 10 64-bit registers, supporting arbitrary load/store
- Supported by a **LLVM** backend

LLVM_P4 Development Environment



LLVM_P4 Development Environment



- StdLib “Software SDK” implemented as LLVM IR
- P4 externs can augment stdlib using program files “application.c” written by the developer.
- Leverage OS helpers, data structures, etc. to provide auxiliary data, (e.g. actions, TCP windows size, cpu load, etc).
- Frontend generates eBPF calls for packet operations and uses maps for tables
- LLVM “opt” program to run passes over P4.ll

LLVM_P4 Development Environment

Supported Target: eBPF

```
# p4llvm switch1.p4
# opt –O2 switch1.ll –o switch1.ll
# llc –march=bpf –filetype=obj –o switch1.o switch1.ll
# tc qdisc add dev eth3 ingress
# tc filter add dev eth3 parent ffff: bpf obj switch1.o exp /tmp/p4cli
```

P4.ll
P4.ll
P4.o

StdLib, Services, State, and Functions

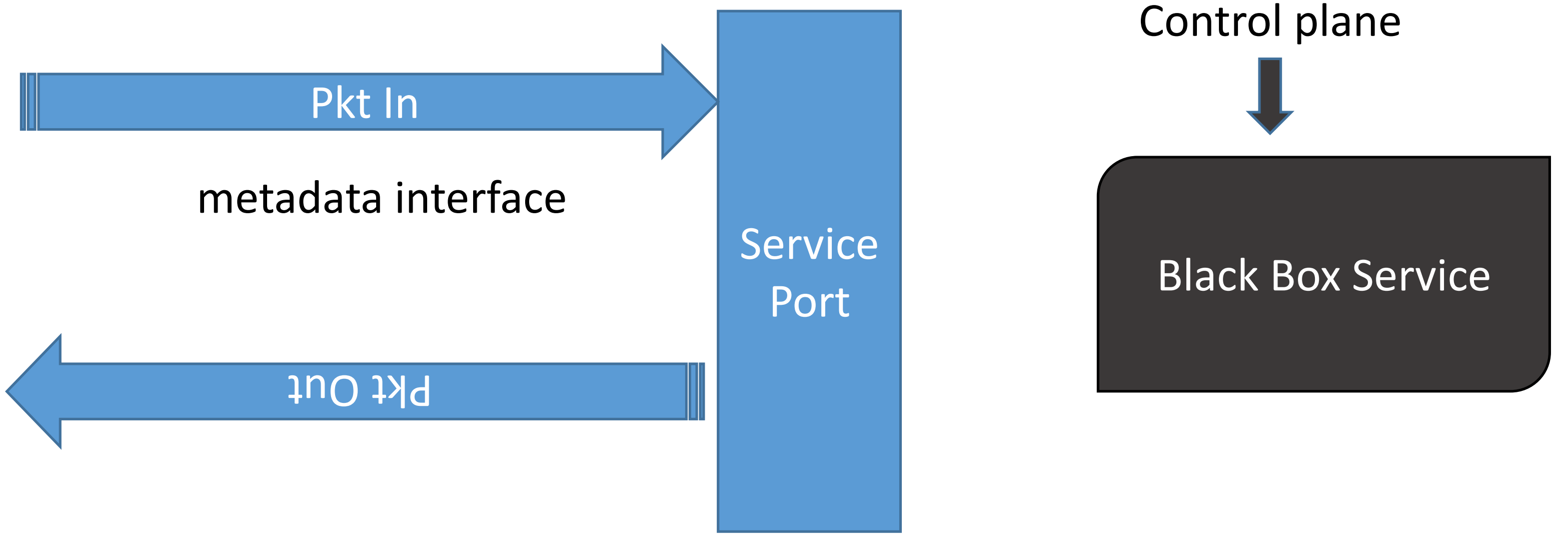
- Stdlib implements all the actions, metering, etc.
  - expected P4 spec features (drop, set_field, meter, counters, …)
- Applications.c (State!)
  - Connection Tracking
    - Not just TCP Flags. Sequence tracking, related flows, etc
  - Load Balancing
    - Operating System state e.g. socket/cpu affinity, cpu state, latency, etc.
  - Functions
    - Anything!

Abstractions for Applications.c

- Service as a parser method / as control flow method
- Service as an Action

Field_a	Field_b	Action
0xabcd	0x1234	do_extern()

- Service as a Port



Abstractions for Applications.c

- Service as an Action
  - Works well for items that map naturally to per packet operations
  - Easy to overload with lots of external dependencies
    - LB_action using OS state
- Service as a Port
  - Works well for “large” bricks for some definition of large
    - FPGA port
    - crypto port
    - connection tracker port
  - Anything that has a natural recirculation path
    - Remember recirculation may impact throughput/latency

P4 Dev Environment Loader

- The interesting problem that has been ignored so far…
  - CPU mapping to program (run to completion vs pipeline)
  - Resource mapping -- cache allocation, memory footprint, etc.
  - Hardware offloads
    - Leverage hardware capabilities
    - Push pipeline across multiple objects, software, NIC, FPGA, etc.

Future Work (even more experimental)

- Auto generate test pcap files
  - 100% test coverage at least on standard featuers
  - Formal methods
- Debugging backend
  - gdb for P4 development workbench
- P4 optimization opportunities
  - Vector operations
- P4 orchestrator
  - OVN(?)

Questions?