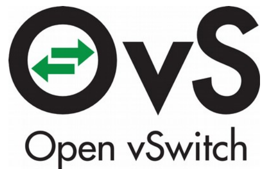
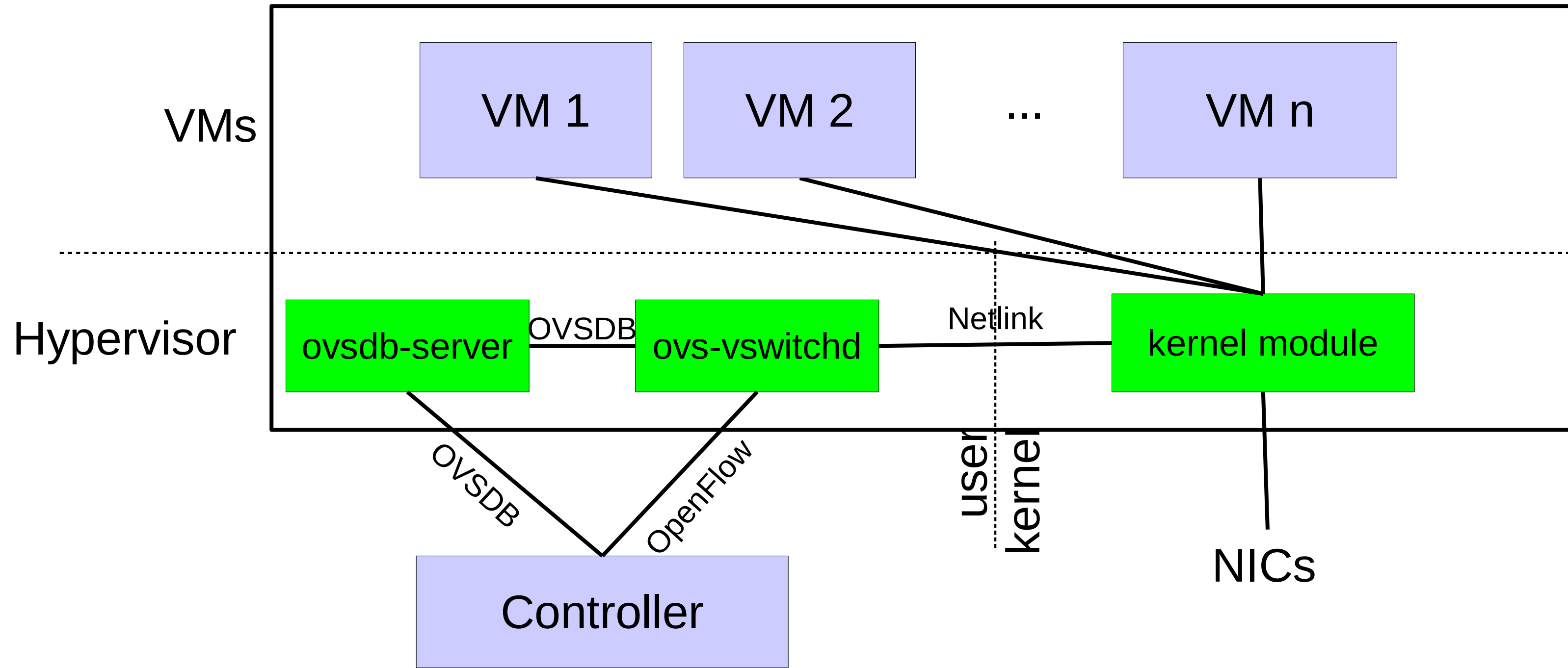
P4 and Open vSwitch

Ben Pfaff
blp@nicira.com
Open vSwitch Commiter



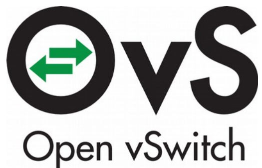
Open vSwitch Architecture



Open vSwitch

Where is Open vSwitch Used?

- Broad support:
  - Linux, FreeBSD, NetBSD, Windows, ESX
  - KVM, Xen, Docker, VirtualBox, Hyper-V, …
  - OpenStack, CloudStack, OpenNebula, (OVN!), …
- Widely used:
  - Most popular OpenStack networking backend
  - Default network stack in XenServer
  - 1,440 hits in Google Scholar
  - Thousands of subscribers to OVS mailing lists



The Big Picture

- Most releases of OVS add support for new fields or protocols.
- Every new field or protocol requires changes throughout OVS.
- Every change to OVS requires building, distributing, and installing a new version of OVS.
- Every field needs coordination with controller authors (+ONF).
- (Sometimes reasonable people disagree about a field, too!)
- It would be great to avoid all of this!

OvS
Open vSwitch

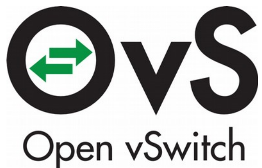
The Big Vision

Requirements

- Reconfigure protocols and fields without recompiling
- Maintain or boost performance
- Maintain OVS backward compatibility
- Maintain (and extend) OpenFlow compatibility

Nice-to-Have

- Minimize dependencies.
- Support both OVS software datapaths (kernel and user/DPDK).
- Avoid making OVS modal.
- Define “legacy” features in terms of new interface
- Avoid fragmenting P4 spec.



The Easy Part

- Define OVS fields and protocols with P4 **header_type**, **header**, **parser**s (~300 lines of P4 for everything in OVS).
- Map these fields and protocols to OpenFlow “OXM” matches
  - by naming (e.g. name your eth_dst field OXM_8000_03)
  - with an external mapping table
  - with special comments in the P4 source code
  - by defining OXM matches as metadata and storing into them

Open vSwitch

The Harder Part: OVS Linux datapath

- How do we make **openvswitch.ko** extensible?
- Probably not acceptable to support P4 directly in kernel.
- Kernel already has an extensibility mechanism, eBPF:
  - 64-bit hardware-like virtual machine.
  - Extended form of BPF (used e.g. for tcpdump).
  - Safe for untrusted user code via verifier.
  - JIT for high performance on popular archs (e.g. x86, ARM).
- eBPF is a suitable compiler target for P4! (Size could be an issue: 1415/4096.)
- (User/DPDK datapath can use any approach we like.)

OvS
Open vSwitch

Unsolved Conceptual Issues in P4-OVS Binding

- Weird fields
  - Transforming P4 to OpenFlow is not too hard
  - Transforming back from OpenFlow to P4 might need help
  - e.g. OVS treats CFI bit in VLANs as “present” bit
- Inserting and removing fields
  - VLAN and other encapsulation push/pop
  - MPLS requires reparse after pop

OvS
Open vSwitch

The Prototype

- P4 syntax lexer (complete language)
- P4 syntax parser (**header_type**, **header**, **parser**)
- ovs.p4: P4 for OVS supported protocols and fields (minus IPv6)
- Compiler that accepts P4 and emits eBPF (just what ovs.p4 needs)
- eBPF interpreter for OVS userspace
  - (would be replaced by JIT for production use)
- Replacement flow parsing routine that invokes eBPF
- Total: 5,500 new lines of code written over about 1 week

OvS
Open vSwitch

The Worst Part of the Prototype: Mapping P4 to OpenFlow via metadata

OVS flow definition

```
struct flow {
    ...
    …
    uint8_t eth_src[6];
    uint8_t eth_dst[6];
    ovs_be16 eth_type;
    …
```

P4 metadata definition

```
header_type flow_t {
    fields {
        …
        eth_src : 48;
        eth_dst : 48;
        eth_type : 16;
        …
```

P4 parser mapping

```
header_type flow_t flow;
parser eth {
    extract(l2_eth);
    set_metadata(flow.eth_src,
                 l2_eth.eth_src);
    set_metadata(flow.eth_dst,
                 l2_eth.eth_dst);
    ...
```

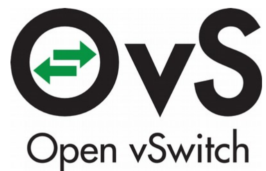
Open vSwitch

Future Work

- Most important: solve unresolved questions in P4-OVS and P4-OpenFlow bindings
- P4 to eBPF compiler refinement (needs optimizer; use LLVM?)
- Lots of code to crank out.

Open vSwitch

Questions?



Source code:
https://github.com/blp/ovs-reviews/releases/tag/p4-workshop

P4-to-eBPF Example

P4

```
header_type l2_eth_t {
    fields {
        eth_dst : 48;
        eth_src : 48;
    }
}
header l2_eth_t l2_eth;
parser l2_eth {
    extract(l2_eth);
    set_metadata(flow.dl_dst, latest.eth_dst);
    set_metadata(flow.dl_src, latest.eth_src);
    return select(current(0, 16)) {
        0x8100: l2_vlan;
        default: l2_ethertype;
    }
}
```

BPF

```
# set_metadata(flow.dl_dst)
# r5 = l2_eth.eth_dst
  1: ld #0, r5
  3: ldd [r5], r5
  4: rshd #0x10, r5
  5: lshd #0x10, r5
  6: ld #0x10068, r6
  8: ldd [r6], r7
  9: ld #0xffff, r8
 11: andd r7, r8
 12: ord r5, r7
 13: std r7, [r6]
...
```

```
# r5 = current(0,16)
  27: ld #0, r5
  29: ldh 0xc[r5], r5
  30: if (r5 != #0x8100) jmp 32
  31: jmp 33
  32: jmp 744
```

Open vSwitch