OpenVSwitch and Trace Points: Getting Started in Open Source

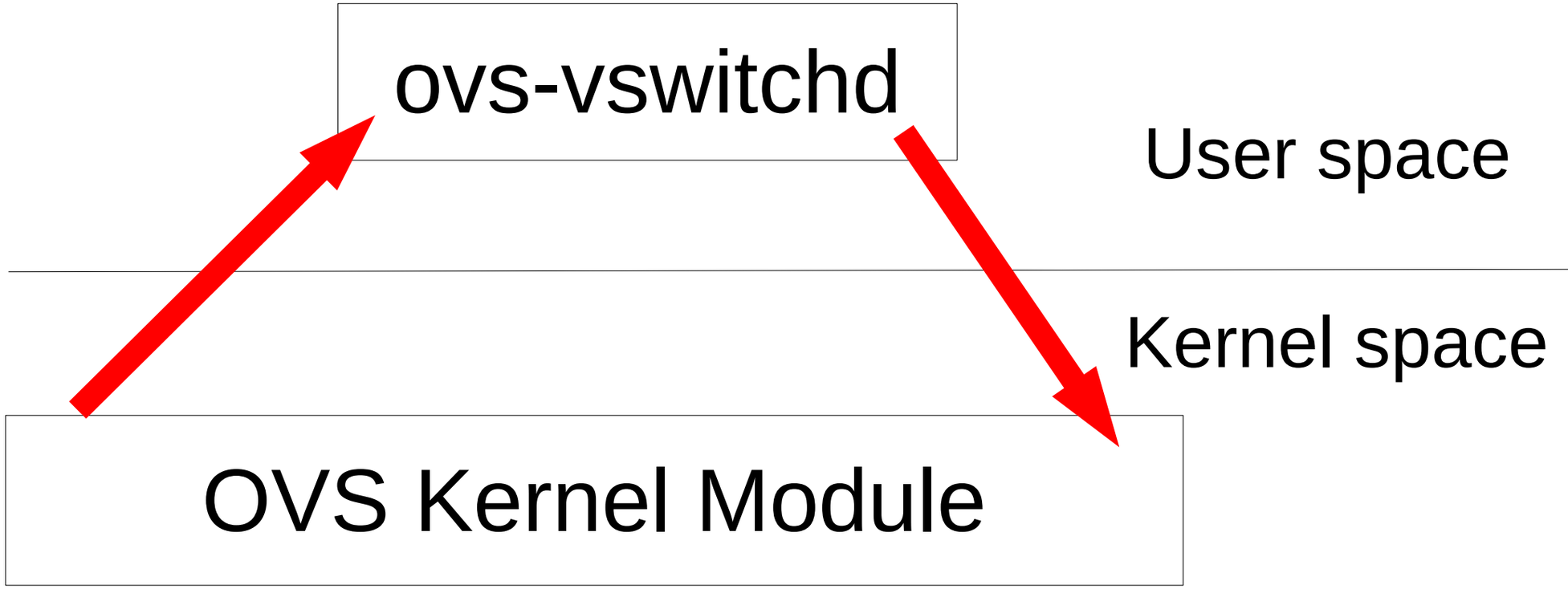
Mick Tarsel mjtarsel@us.ibm.com
LTC Networking Team, IBM
March 2016

Agenda

- Docker/OVS Project Setup
- Where to start
- Helpful tools & tips
- Trace events
- Tracing packets
- Closing thoughts

Project Overview

- OVS and Docker on GNU/Linux
- Sporadic latency of ARP packets in slow path



OVS Slow Path

Where to Start?

- Read, a lot.
- Learn git
  - git clone https://github.com/openvswitch/ovs
- Don't be overwhelmed
  - Stick with small chunks
- Learn about tools

Narrowing it Down

- Knowing your environment
- cscope
  - http://cscope.sourceforge.net/

```
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

cscope screenshot

Measuring Packet Latency?

- Tcpdump?
  - I want to ID a packet
- SystemTap

  https://sourceware.org/systemtap/
  - Events (upcall) and handlers (tag it)
  - Kprobes – dynamic tracing
- perf
  - Trace points – static event tracing
    - */Documentation/trace/tracepoints.txt*

Trace Points

- Record local variables & functions
- Fast tracing
- Requires some setup (not too simple)

Trace events

- http://lwn.net/Articles/379903/
  - “To solve this issue of automating the tracepoints, the TRACE_EVENT( ) macro was born.” - Rostedt
- Trace event must:
  - Create trace point
  - Create call back function
  - Record data
  - Parse data
- Output goes to /sys/kernel/debug/tracing/
- Usable from user space

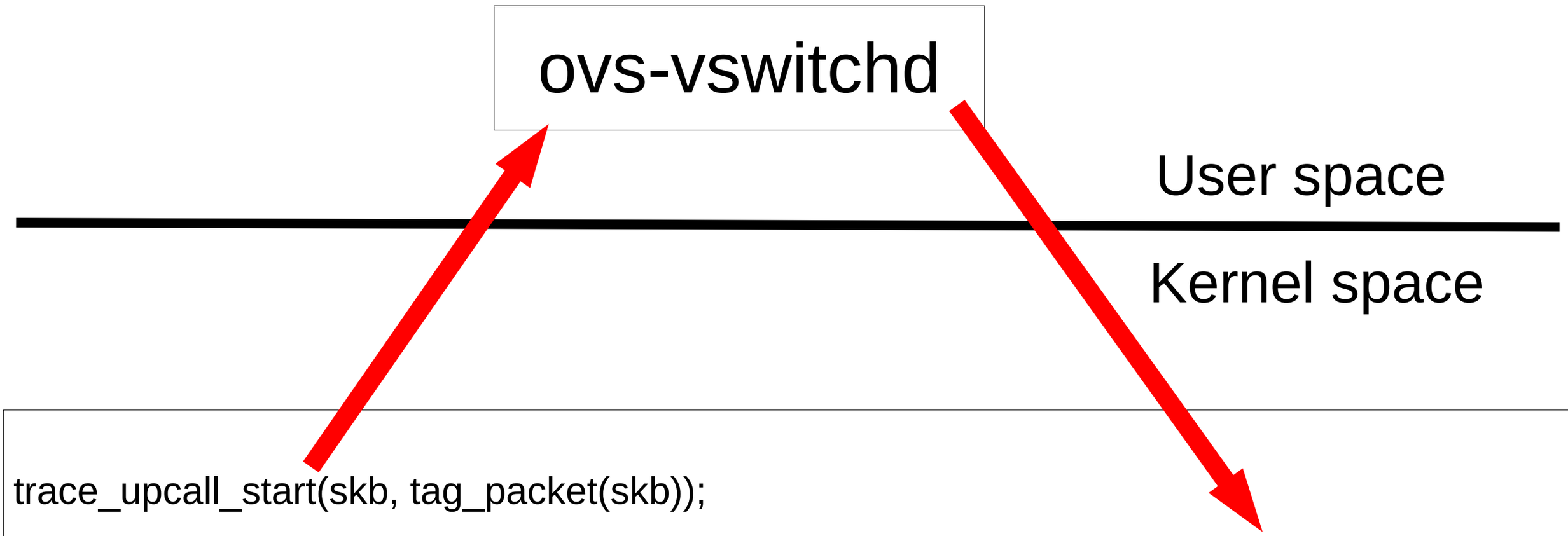
My OVS Trace Event

```
TRACE_EVENT(upcall_start,

        TP_PROTO(struct sk_buff *skb, int id),

        TP_ARGS(skb, id),

        TP_STRUCT__entry(
                __field(    const void *,      skbaddr      )
                __field(    u16,      protocol      )
                __field(    int,          id      )
        ),

        TP_fast_assign(
                __entry->skbaddr = skb;
                __entry->protocol = ntohs(skb->protocol);
                __entry->id = (const int) id;
        ),

        TP_printk("skbaddr=%p proto=0x%x id=%d\n",
                __entry->skbaddr, __entry->protocol, __entry->id)
);
```

Calling the Trace Event

```
int ovs_dp_upcall(struct datapath *dp, struct sk_buff *skb,
                  const struct sw_flow_key *key,
                  const struct dp_upcall_info *upcall_info)
{
        struct dp_stats_percpu *stats;
        int err;

        if (upcall_info->portid == 0) {
                err = -ENOTCONN;
                goto err;
        }

        if (trace_upcall_start_enabled())
                trace_upcall_start(skb, tag_packet(skb));

        if (!skb_is_gso(skb))
                err = queue_userspace_packet(dp, skb, key, upcall_info);
        else
                err = queue_gso_packets(dp, skb, key, upcall_info);
        if (err)
                goto err;

        return 0;
```

Following Packets



Output

perf record -e openvswitch:upcall_start -e openvswitch:upcall_end -a

```
 === ARP Packets ===
              id          Time in Nano Seconds
Max:       5           4721362.0
Min:       6           64742.0
Average:  883686.666667
Total ARP Packets: 6

              === IP Packets ===
              id          Time in Nano Seconds
Max:       3           95160.0
Min:       8           44850.0

Average:  69619.0
Total IP Packets: 4

              === All Captured Packets ===
              id          Time in Nano Seconds       Protocol
Max:       5           4721362.0                  0x806
Min:       8           44850.0             0x800

Average:  558059.6
Total Packets captured: 10
```

Results - ARP

Time:	COUNT	MIN	MAX	AVERAGE
RTT	1914	0	2001	1058
K-2-OVS	1914	0	1995	1013
OVS-2-K	1914	0	914	45

```
base-2 logarithmic histogram of round trip times
value |-------------------------------------------------- count
    0 |                                                   21
    1 |                                                   17
    2 |                                                   11
    4 |                                                    1
    8 |                                                    4
   16 |                                                   15
   32 |@                                                  39
   64 |@@@                                                87
  128 |@@@@@@                                            132
  256 |@@@@@@@@                                          184
  512 |@@@@@@@@@@@@@@                                    325
 1024 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 1078
 2048 |                                                    0
 4096 |                                                    0
```

Count = number of packets
Value = time in Jiffies

Results - IPv4

Time:	COUNT	MIN	MAX	AVERAGE
RTT	230	0	53	5
K-2-OVS	230	0	44	2
OVS-2-K	230	0	50	2

```
base-2 logarithmic histogram of round trip times
value |-------------------------------------------------- count
    0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@          102
    1 |@@@@@@@@@@@@@@@@@                           53
    2 |@@@@@@                                      20
    4 |@@@@                                        14
    8 |@@@                                         10
   16 |@@@@@@                                      18
   32 |@@@@                                        13
   64 |                                            0
  128 |                                            0
```

Count = number of packets
Value = time in Jiffies

Future Plans

- Extend packet tracing tools to other projects
- ovs-benchmark tool

Acknowledgments

- IBM LTC
- LTC Networking Team
  - Pradeep Satyanarayana
  - Dave Wilder

Closing thoughts

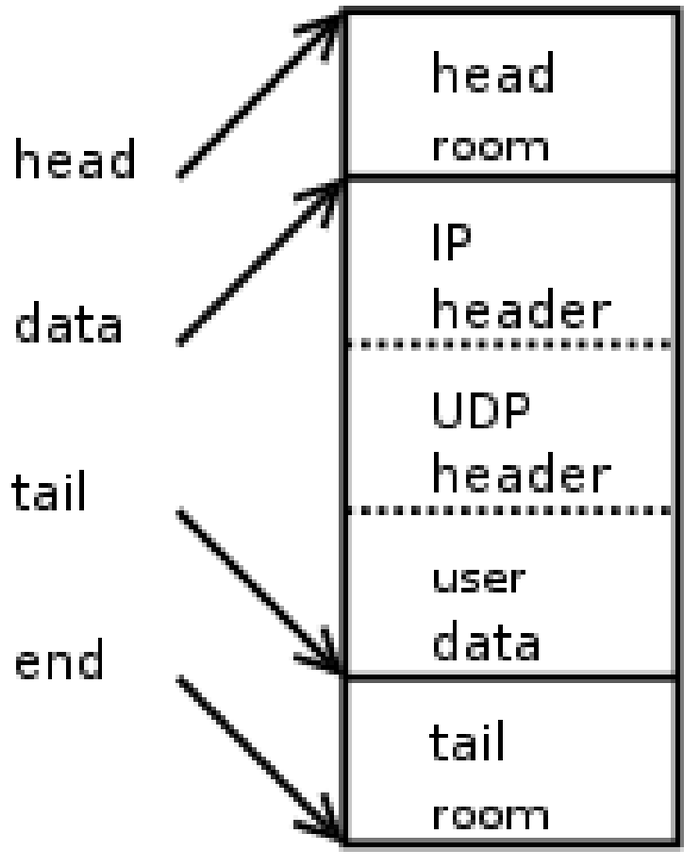
- These are things that helped me....
  - Stay organized with git
  - READ
  - Don't be intimidated
  - Subscribe to mailing lists and answer questions
  - Use tools

Questions?

Thank you!

Socket Buffer

- sk_buff, sk_buffer, skb
-



Source image: **http://vger.kernel.org/~davem/skb_data.html**

tag_packet()

```
static inline int tag_packet(struct sk_buff *skb)
{
        struct packet_tag pkt_id;
        static unsigned int id;
        unsigned char *temp_pkt;

        pkt_id.eyeCatch = TAGGED;
        pkt_id.reqid = ++id;

        if (sizeof(struct packet_tag) <= skb_tailroom(skb)) {
                temp_pkt = skb_put(skb, sizeof(pkt_id));
                memcpy(temp_pkt, &pkt_id, sizeof(pkt_id));
                pr_debug("%s:skb=%p reqid=%lu\n", __func__, skb, pkt_id.reqid);
                return id;
        }

        if (net_ratelimit())
                pr_debug("%s:Insufficient room to tag skb=%p\n", __func__, skb);

        return 0;
```

report_packet()

```
static inline int report_packet(struct sk_buff *skb)
{
        struct packet_tag *pkt_id;
        unsigned char *temp_pkt;

        temp_pkt = (skb->data)+(skb->len) - sizeof(struct packet_tag);
        pkt_id = (struct packet_tag *)temp_pkt;

        if (pkt_id->eyeCatch == TAGGED) {
                pr_debug("%s:skb=%p reqid=%lu\n", __func__, skb, pkt_id->reqid);
                return pkt_id->reqid;
        }

        return 0;
}
```

upcall_end trace point

```
/* define from upcall_start trace event a new event with the same proto,
 and args instead named upcall_end */
DEFINE_EVENT(upcall_start, upcall_end,
        TP_PROTO(struct sk_buff *skb, int id),
        TP_ARGS(skb, id)
);
```