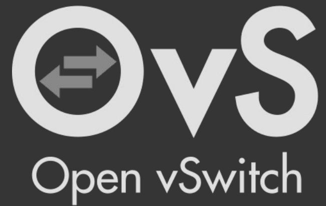


Debugging OVS using static tracepoints

OVS/OVN Conference 2021

Eelco Chaudron
Principal Software Engineer

Red Hat

What is a USDT probe?

- USDT => User Statically Define Trace probe/point
- A specific event in the code identified by the Developer
- Well defined by name, so scripts can continue to use them
- Variables are not optimized out by the compiler
- Easy to develop and maintain external tools for troubleshooting

- USDT probe is defined using a MACRO, i.e. DTRACE_PROBE2()

```c
#include <sys/sdt.h>
#include <sys/time.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    struct timeval tv;

    for (int x; x < 1024; x++) {
        gettimeofday(&tv, NULL);
        DTRACE_PROBE2(test_app, test_probe, tv.tv_sec, x);
        sleep(1);
    }
    return 0;
}
```


Red Hat

- MACRO inserts a NOP in the code, and additional information in the *.note.stapsdt* ELF section

```
int main(int argc, char **argv)
{
  ...
    for (int x=0; x < 1024; x++) {
  ...
      gettimeofday(&tv, NULL);
  ...
  40115a:       e8 d1 fe ff ff          callq  401030 <gettimeofday@plt>
        DTRACE_PROBE2(test_app, test_probe, tv.tv_sec, x);
  40115f:       48 8b 45 e0             mov    -0x20(%rbp),%rax
  401163:       90                      nop
        sleep(1);
  401164:       bf 01 00 00 00          mov    $0x1,%edi
  401169:       e8 d2 fe ff ff          callq  401040 <sleep@plt>
  ...
```


Red Hat

- MACRO inserts a NOP in the code, and additional information in the *.note.stapsdt* ELF section

```
$ tplist.py -v -l ./test_dtrace
  b'test_app':b'test_probe' [sema 0x0]
    location #1 b'./test_dtrace.o' 0x401163
      argument #1 8 signed   bytes @ b'ax'
      argument #2 4 signed   bytes @ *(b'bp' - 4)

$ readelf -n ./test_dtrace
...
Displaying notes found in: .note.stapsdt
  Owner                Data size    Description
  stapsdt              0x00000040   NT_STAPSDT (SystemTap probe descriptors)
    Provider: test_app
    Name: test_probe
    Location: 0x0000000000401163, Base: 0x0000000000402010, Semaphore: 0x0000000...
    Arguments: -8@%rax -4@-4(%rbp)
```


Red Hat

- Userspace implementation
    - Ring buffer to store events [LTTng]
- Linux Kernel side implementations
    - uProbe
    - uProbe + eBPF

Red Hat

```
(gdb) disas /m main
Dump of assembler code for function main:
6    {
   ...
7      struct timeval tv;
8
9      for (int x=0; x < 1024; x++) {
   ...
10           gettimeofday(&tv, NULL);
   0x000000000040114e <+24>:    lea    -0x20(%rbp),%rax
   0x0000000000401152 <+28>:    mov    $0x0,%esi
   0x0000000000401157 <+33>:    mov    %rax,%rdi
   0x000000000040115a <+36>:    callq  0x401030 <gettimeofday@plt>

11           DTRACE_PROBE2(test_app, test_probe, tv.tv_sec, x);
   0x000000000040115f <+41>:    mov    -0x20(%rbp),%rax
   0x0000000000401163 <+45>:    int3

12           sleep(1);
   0x0000000000401164 <+46>:    mov    $0x1,%edi
   0x0000000000401169 <+51>:    callq  0x401040 <sleep@plt>

13     }
14     return 0;
   0x000000000040117b <+69>:    mov    $0x0,%eax

15    }
```


Red Hat

- ftrace (kernel debug fs)
- trace-cmd (ftrace fontend)
- perf
- SystemTap
- DTrace
- bpftrace
- BCC (BPF Compiler Collection)

Red Hat

- OVS Instrumented with the following DTrace probes:

```
diff --git a/lib/netdev-dpdk.c b/lib/netdev-dpdk.c
index 9d8096668..2a75ba2fa 100644
--- a/lib/netdev-dpdk.c
+++ b/lib/netdev-dpdk.c
@@ -2630,6 +2631,7 @@ __netdev_dpdk_vhost_send(struct netdev *netdev, int qid,

      n_packets_to_free = cnt;

+       DTRACE_PROBE2(__netdev_dpdk_vhost_send, enqueu, cnt, cur_pkts);
        do {
               int vhost_qid = qid * VIRTIO_QNUM + VIRTIO_RXQ;
               unsigned int tx_pkts;
        @@ -2646,6 +2648,7 @@ __netdev_dpdk_vhost_send(struct netdev *netdev, int qid,
                      ,* and no retries have already occurred.
                      ,*/
                      atomic_read_relaxed(&dev->vhost_tx_retries_max, &max_retries);
+              DTRACE_PROBE2(__netdev_dpdk_vhost_send, retry_start, tx_pkts, cnt);
               }
               } else {
               /* No packets sent - do not retry.*/
```

- Same for LTTng:

```
+      tracepoint(netdev_dpdk,  __netdev_dpdk_vhost_send__enqueu, cnt, cur_pkts);
```

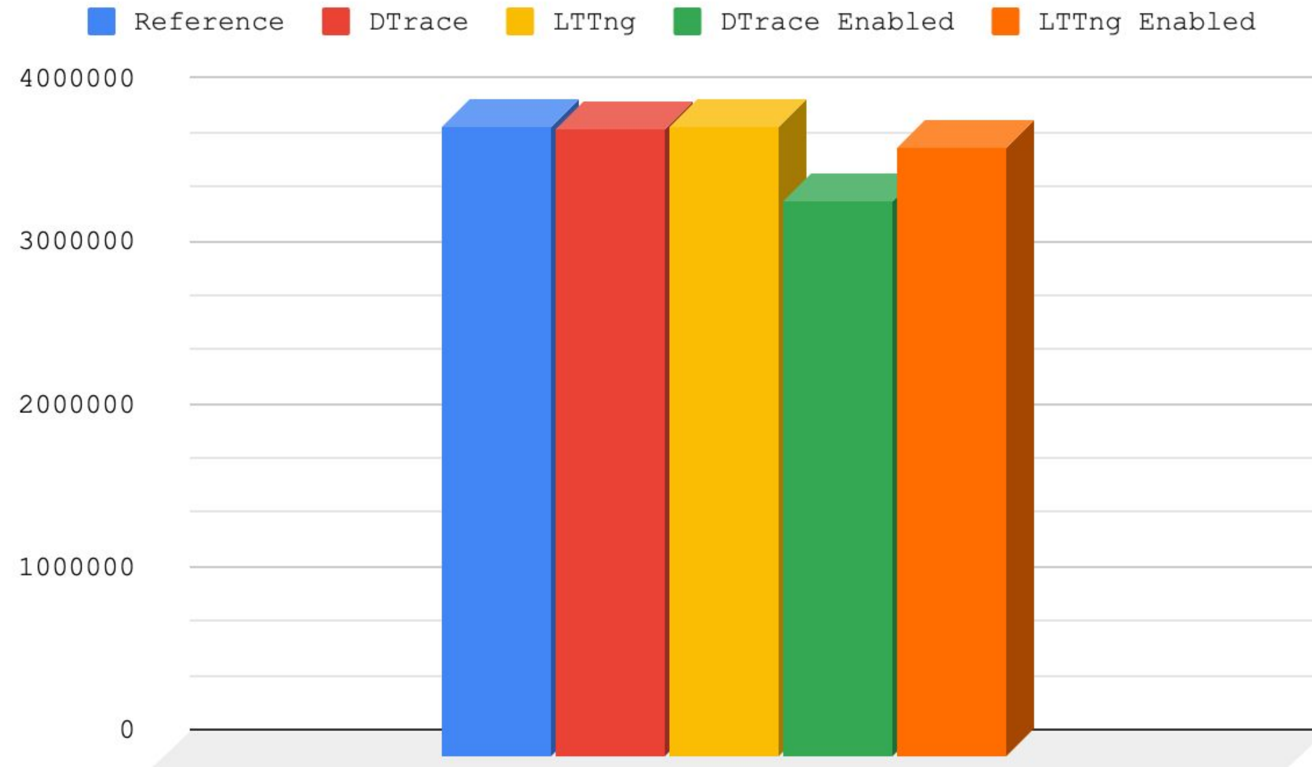

Red Hat

- Running the OVS_Perf test suite[1]
- Ran both wire-speed and zero packet loss tests
- Using 64 byte packets with 100 IPv4 streams
- CPU, Intel E5-2690 v4 @ 2.60GHz
- NIC was an Intel XL710 running at 40G
- OVS Configuration:

```
ovs-vsctl set Open_vSwitch . other_config:dpdk-lcore-mask=0x10004
ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x8002
ovs-vsctl set Open_vSwitch . other_config:dpdk-init=true
ovs-vsctl add-br ovs_pvp_br0 -- \
  set bridge ovs_pvp_br0 datapath_type=netdev
ovs-vsctl add-port ovs_pvp_br0 dpdk0 -- \
  set Interface dpdk0 type=dpdk -- \
  set Interface dpdk0 options:dpdk-devargs=0000:05:00.1 -- \
  set interface dpdk0 options:n_rxq=2
ovs-vsctl add-port ovs_pvp_br0 vhost0 -- \
  set Interface vhost0 type=dpdkvhostuserclient -- \
  set Interface vhost0 options:vhost-server-path='/tmp/vhost-sock0' -- \
  set Interface vhost0 options:n_rxq=2
```


[1] https://github.com/chaudron/ovs_perf

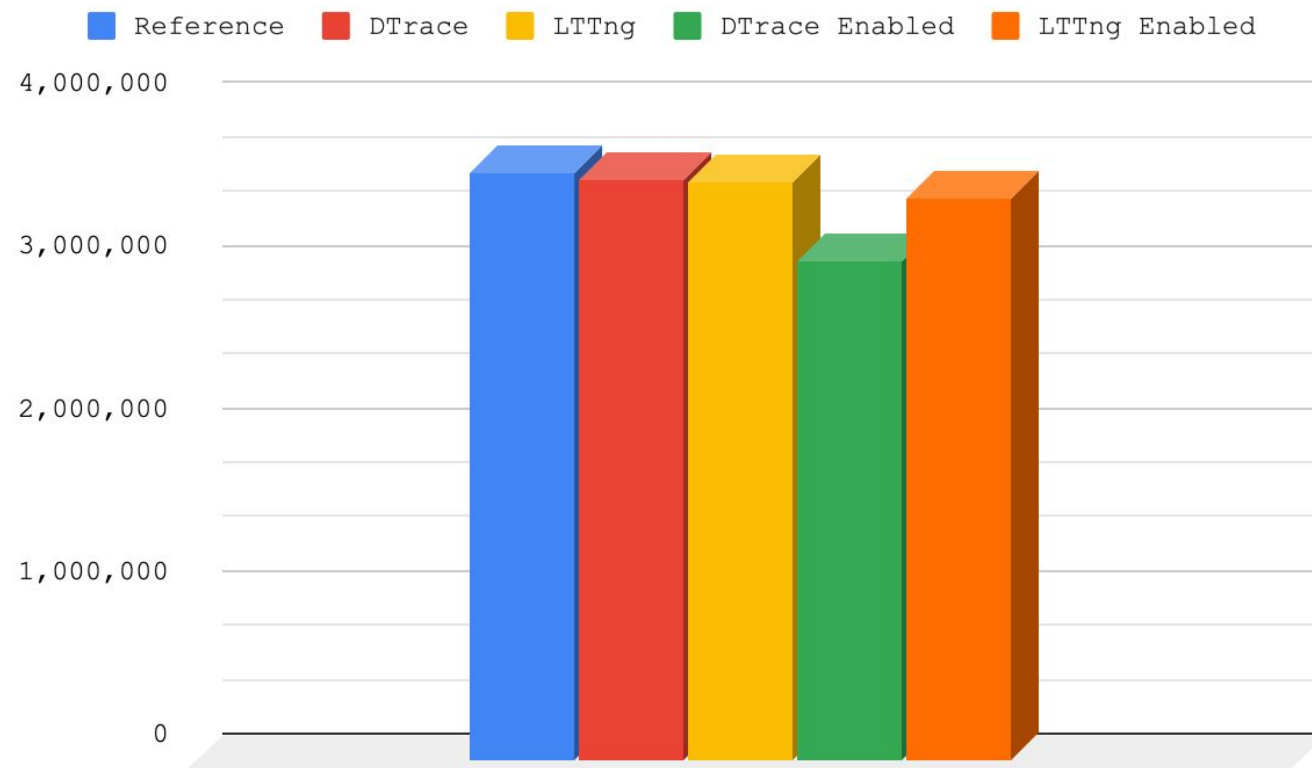
Red Hat



	Reference	DTrace	LTTng	DTrace Enabled	LTTng Enabled
Average	3,866,872	3,862,641	3,874,459	3,408,726	3,744,774
RS Deviation	0.27%	0.33%	0.08%	0.45%	0.15%
Delta to Ref	-	0.11%	-0.20%	11.85%	3.16%

Red Hat

Performance impact
Zero-loss results



	Reference	DTrace	LTTng	DTrace Enabled	LTTng Enabled
Average	3,619,026	3,571,429	3,559,557	3,065,603	3,452,307
RS Deviation	0.74%	0.00%	0.75%	1.38%	0.00%
Delta to Ref	-	1.32%	1.64%	15.29%	4.61%

Red Hat

- Re-use the SystemTap DTRACE_PROBEx() MACROs
  - This to avoids the additional complexity of integrating LTTng
- Add OVS specific wrapper to support variable arguments

```
#define OVS_USDT_PROBE(provider, name, ...) \
...
```

- These MACRO's allow all kind of tools to be used as the frontend:
  - BPF Compiler Collection (BCC)
  - bpftrace
  - DTrace for Linux
  - Perf
  - SystemTap

Red Hat

Simple example [1/3]

- Add two probes to the ovs-vswitchd main loop:

```
diff --git a/vswitchd/ovs-vswitchd.c b/vswitchd/ovs-vswitchd.c
@@ -113,10 +114,11 @@ main(int argc, char *argv[])
     while (!exiting) {
+        OVS_USDT_PROBE(main, run_start);
         memory_run();
...
         if (exiting) {
             poll_immediate_wake();
         }
+        OVS_USDT_PROBE(main, poll_block);
         poll_block();
         if (should_service_stop()) {
             exiting = true;
         }
     }
```

- With these two probes we can measure things like:
  - Number of bridge runs (per second)
  - Time each bridge run will take
  - Delay between bridge runs

Red Hat

Simple example [2/3]

- The following bpftrace snippet will show a bridge run histogram:

```
usdt::main:poll_block
{
        @pb_start[tid] = nsecs;
        if (@rs_start[tid] != 0) {
        $delta = nsecs - @rs_start[tid];
        printf("- [%d@%s] bridge run loop time  : %u:%2.2u:%2.2u.%9.9u\n",
                tid, comm,
                $delta / 3600 / 1000000000,
                $delta / 60   / 1000000000 % 60,
                $delta / 1000000000 % 60,
                $delta % 1000000000);
        @bridge_run_time = lhist($delta / 1000, 0, 1000000, 1000);
        }
}

usdt::main:run_start
{
        @rs_start[tid] = nsecs;
        if (@pb_start[tid] != 0) {
        $delta = nsecs - @pb_start[tid];
        printf("- [%d@%s] poll_block() wait time: %u:%2.2u:%2.2u.%9.9u\n",
                tid, comm,
                $delta / 3600 / 1000000000,
                $delta / 60   / 1000000000 % 60,
                $delta / 1000000000 % 60,
                $delta % 1000000000);
        @poll_block_wait_time = lhist($delta / 1000, 0, 30000000, 30000);
        }
}
```


Red Hat

Script in action:

```
# ./bridge_loop.bt -p `pidof ovs-vswitchd`
Attaching 4 probes...
--------------------------------------------------------------
Tracing ovs-vswitchd's main() loop... Hit Ctrl-C to end.
--------------------------------------------------------------
- [411887@ovs-vswitchd] bridge run loop time  : 0:00:00.000090274
- [411887@ovs-vswitchd] poll_block() wait time: 0:00:00.500756124
- [411887@ovs-vswitchd] bridge run loop time  : 0:00:00.000064352
...
^C
--------------------------------------------------------------
Showing run time histograms in micro seconds:
--------------------------------------------------------------
@bridge_run_time:
[0, 1000)          20201 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[1000, 2000)           2 |                                                    |
[2000, 3000)           0 |                                                    |
[3000, 4000)           2 |                                                    |
[4000, 5000)           0 |                                                    |
[5000, 6000)           0 |                                                    |
[6000, 7000)           6 |                                                    |
[7000, 8000)           0 |                                                    |
[8000, 9000)           0 |                                                    |
[9000, 10000)          0 |                                                    |
[10000, 11000)         0 |                                                    |
[11000, 12000)         0 |                                                    |
[12000, 13000)         0 |                                                    |
[13000, 14000)         1 |                                                    |
```

```
@poll_block_wait_time:
[0, 30000)          20061 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[30000, 60000)          3 |                                                    |
[60000, 90000)          3 |                                                    |
[90000, 120000)        18 |                                                    |
[120000, 150000)        3 |                                                    |
[150000, 180000)        2 |                                                    |
[180000, 210000)        1 |                                                    |
[210000, 240000)        1 |                                                    |
[240000, 270000)        0 |                                                    |
[270000, 300000)        0 |                                                    |
[300000, 330000)        0 |                                                    |
[330000, 360000)        1 |                                                    |
[360000, 390000)        2 |                                                    |
[390000, 420000)       11 |                                                    |
[420000, 450000)        1 |                                                    |
[450000, 480000)        1 |                                                    |
[480000, 510000)      102 |                                                    |
[510000, 540000)        1 |                                                    |
```

- The following probe allows monitoring all NetLink upcalls:

```
@@ -1602,6 +1603,12 @@ dpif_recv(struct dpif *dpif, uint32_t handler_id, struct
dpif_upcall *upcall,
      if (dpif->dpif_class->recv) {
      error = dpif->dpif_class->recv(dpif, handler_id, upcall, buf);
      if (!error) {
+           OVS_USDT_PROBE(dpif_recv, recv_upcall, dpif->full_name,
+                             upcall->type,
+                             dp_packet_data(&upcall->packet),
+                             dp_packet_size(&upcall->packet),
+                             upcall->key, upcall->key_len);
+
            dpif_print_packet(dpif, upcall);
      } else if (error != EAGAIN) {
```

- A script can be attached to capture the events and even save the packet content in a PCAP file.

Red Hat

- A Python script was developed using the BCC tools
- Here is an example output:

```
$ ./upcall_monitor.py --flow-key-decode=nlraw --packet-decode=decode --pcap packets.pcap
TIME               CPU  COMM            PID    DPIF_NAME          TYPE PKT_LEN  FLOW_KEY_LEN
685799.566914183   21   handler15       411906 system@ovs-system  0    60       132
  Flow key size 132 bytes, size captured 64 bytes.
      nla_len 8, nla_type OVS_KEY_ATTR_RECIRC_ID[20], data: 00 00 00 00
      nla_len 8, nla_type OVS_KEY_ATTR_DP_HASH[19], data: 00 00 00 00
      nla_len 8, nla_type OVS_KEY_ATTR_PRIORITY[2], data: 00 00 00 00
      nla_len 8, nla_type OVS_KEY_ATTR_IN_PORT[3], data: 02 00 00 00
      nla_len 8, nla_type OVS_KEY_ATTR_SKB_MARK[15], data: 00 00 00 00
      nla_len 8, nla_type OVS_KEY_ATTR_CT_STATE[22], data: 00 00 00 00
      nla_len 6, nla_type OVS_KEY_ATTR_CT_ZONE[23], data: 00 00
      nla_len 8, nla_type OVS_KEY_ATTR_CT_MARK[24], data: 00 00 00 00
1: Receive dp_port 2, packet size 60 bytes, size captured 60 bytes.
      ###[ Ethernet ]###
      dst    = 00:00:02:00:00:00
      src    = 00:00:01:00:00:00
      type   = IPv4
```

```
###[ IP ]###
version   = 4
ihl    = 5
tos    = 0x0
len    = 46
id     = 1
flags  =
frag   = 0
ttl    = 64
proto  = udp
chksum = 0x77bf
src    = 1.0.0.0
dst    = 2.0.0.0
\options   \
###[ UDP ]###
      sport  = domain
      dport  = domain
      len    = 8
      chksum = 0x0
###[ Padding ]###
      load   = '*+,-./0123456789:;'
```


Red Hat

- Even more complex scripts can be created
- The BPF Compiler Collection (BCC) suite can be used to use mixed tracepoints
- The upcall cost script is an example, i.e., it combines:
  - OVS USDT probes (:recv_upcal/:op_flow_put/:op_flow_execute)
  - Kernel Tracepoint for the OVS ovs_dp_upcall probe

    See the following kernel commit for details:

    https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=c4ab7b56be0f6
  - Kernel kprobe on an OVS kmod specific function
- upcall_cost.py collects information like:
  - Total events hit (missed)
  - Batches size of upcalls
  - Upcalls per thread
  - Upcall from kernel till time OVS receives it
  - OVS Execute till Kernel receives it
  - Upcall overhead (total time – lookup)
  - Total Upcall time

- Script can store retrieved information and re-use it
- The output on below uses a previously collected trace set:

```
$  ./upcall_cost.py -r pvp_test.cost
- Reading events from "pvp_test_2.cost"...
[openvswitch__dp_upcall] swapper/14            0 [014]  2681636.100836451: ovs-system         1   60       0
[openvswitch__dp_upcall] swapper/0             0 [000]  2681636.100836516: ovs-system         1   60       0
[openvswitch__dp_upcall] swapper/27            0 [027]  2681636.100836587: ovs-system         1   60       0
...
...
[..s_packet_cmd_execute] handler10       3171019 [021]  2681636.112639849
[..s_packet_cmd_execute] handler10       3171019 [021]  2681636.112643060
[..s_packet_cmd_execute] handler10       3171019 [021]  2681636.112646129
[..s_packet_cmd_execute] handler10       3171019 [021]  2681636.112651455
- Analyzing results (8720 events)...

=> Events received per type (usable/total) [missed events]:
  dpif_recv__recv_upcall            :     1744/        1744 [       0]
  Ktrace__ovs_packet_cmd_execute    :     1744/        1744 [       0]
  Netlink_opperate__op_flow_execute :     1744/        1744 [       0]
  Netlink_opperate__op_flow_put     :     1744/        1744 [       0]
  openvswitch__dp_upcall            :     1744/        1744 [       0]

- Analyzing 1744 event sets...
```


Red Hat

Upcall cost [3/5]

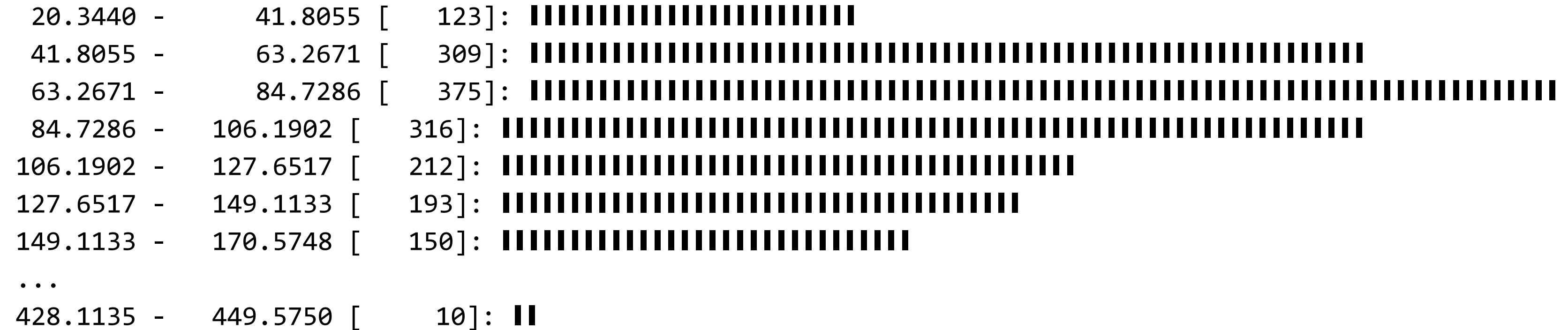
```
=> Upcalls handled per thread:
  handler15                          :             34
  handler12                          :             71
  handler10                          :            780
  handler11                          :             38
  handler3                           :             37
  handler24                          :            141
  handler16                          :             72
  handler25                          :             72
  handler13                          :             70
  handler28                          :            110
  handler14                          :             71
  handler27                          :             71
  handler30                          :             71
  handler17                          :             36
  handler26                          :             70

=> Histogram of upcalls per batch:
# NumSamples = 38; Min = 6; Max = 64
# each ▌ represents a count of 5
      1 [         0]:                            33 [         0]:
      2 [         0]:                            34 [         1]:
      3 [         0]:                            35 [         0]:
    ...
     30 [         0]:                            62 [         0]:
     31 [         0]:                            63 [         0]:
     32 [         0]:                            64 [        22]: ▌▌▌▌
```


Red Hat

Upcall cost [4/5]

```
=> Kernel upcall action to vswitchd receive (microseconds):
# NumSamples = 1744; Min = 14.27; Max = 7033.98
# Mean = 2143.184408; Variance = 4306646.487499; SD = 2075.246127; Median 1431.211000
# each ▌ represents a count of 6
   14.2650 -   365.2508 [   271]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  365.2508 -   716.2367 [   496]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  716.2367 -  1067.2225 [    63]: ▌▌▌▌▌▌▌▌▌▌
 1067.2225 -  1418.2084 [    34]: ▌▌▌▌▌
 ...
 5630.0386 -  5981.0244 [    31]: ▌▌▌▌▌
 5981.0244 -  6332.0103 [    33]: ▌▌▌▌▌
 6332.0103 -  6682.9961 [    64]: ▌▌▌▌▌▌▌▌▌▌
 6682.9961 -  7033.9820 [    34]: ▌▌▌▌▌
```



```
=> vsiwtchd execute to kernel receive (microseconds):
# NumSamples = 1744; Min = 20.34; Max = 449.57
# Mean = 97.846040; Variance = 2666.500578; SD = 51.638170; Median 88.206500
# each ▌ represents a count of 5
   20.3440 -      41.8055 [   123]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
   41.8055 -      63.2671 [   309]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
   63.2671 -      84.7286 [   375]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
   84.7286 -   106.1902 [   316]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  106.1902 -   127.6517 [   212]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  127.6517 -   149.1133 [   193]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  149.1133 -   170.5748 [   150]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  ...
  428.1135 -   449.5750 [    10]: ▌▌
```


Red Hat

Upcall cost
[5/5]

```
=> Upcall overhead (total time minus lookup) (microseconds):
# NumSamples = 1744; Min = 36.76; Max = 7119.30
# Mean = 2242.328563; Variance = 4217546.355050; SD = 2053.666564; Median 1490.968500
# each ▌ represents a count of 7
   36.7580 -    390.8850 [   111]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  390.8850 -    745.0120 [   577]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  745.0120 -   1099.1390 [   142]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
 1099.1390 -   1453.2660 [    26]: ▌▌▌
 1453.2660 -   1807.3930 [   168]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
 1807.3930 -   2161.5200 [    93]: ▌▌▌▌▌▌▌▌▌▌▌▌▌
 ...
 6411.0440 -   6765.1710 [    64]: ▌▌▌▌▌▌▌▌▌
 6765.1710 -   7119.2980 [    34]: ▌▌▌▌

=> Kernel upcall to kernel packet execute (microseconds):
# NumSamples = 1744; Min = 76.95; Max = 7310.12
# Mean = 2683.598093; Variance = 3907302.746489; SD = 1976.689846; Median 1831.959000
# each ▌ represents a count of 5
   76.9480 -    438.6067 [     6]: ▌
  438.6067 -    800.2654 [   134]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
  800.2654 -   1161.9241 [   275]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
 1161.9241 -   1523.5828 [   396]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
 1523.5828 -   1885.2415 [    84]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
 1885.2415 -   2246.9002 [   115]: ▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌▌
 ...
 6225.1459 -   6586.8046 [    64]: ▌▌▌▌▌▌▌▌▌▌▌▌
 6586.8046 -   6948.4633 [    50]: ▌▌▌▌▌▌▌▌▌▌
 6948.4633 -   7310.1220 [    48]: ▌▌▌▌▌▌▌▌▌
```


Red Hat

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

Red Hat