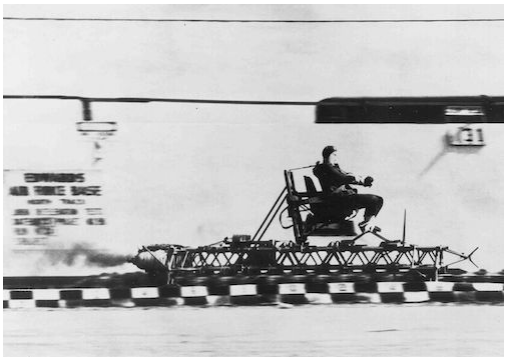
Kernel Line Tracing: Linux perf Rides the Rocket

11 Sep 2014

WHAT DOES IT MEAN?? Ubuntu Trusty was dropping packets in our cloud instance, and leaving us with the mysterious system message:



Riding the rocket

```
[85290.555808] xen_netfront: xennet:
 skb rides the rocket: 19 slots
```

This led us to ride the rocket of advanced Linux kernel tracing...

In this post, I'll demonstrate some fairly unknown features (practically secrets) of Linux kernel tracing using perf_events, which is part of the Linux kernel source (tools/perf).

Mysterious system messages are better than no system messages, because we have something at least to search for. The message comes from the following function (this is Linux 3.13.6), which transmits a packet from a Xen guest (this is on AWS EC2):

```
static int xennet_start_xmit(struct sk_buff *skb, struct net_device *dev)
{
[...]
        slots = DIV_ROUND_UP(offset + len, PAGE_SIZE) +
                xennet_count_skb_frag_slots(skb);
        if (unlikely(slots > MAX_SKB_FRAGS + 1)) {
                net_alert_ratelimited(
                        "xennet: skb rides the rocket: %d slots\n", slots);
                goto drop;
        }
```

Yes, goto drop, which bumps a counter and frees the packet. Client problem now! The client will, after a performance-problem-inducing timeout, retransmit the packet. Let's hope the retransmitted packet doesn't ride the rocket as well (... that could lead to exponential latency.)

We know the value of `slots`, since it's part of the system message (19). We know what MAX_SKB_FRAGS is from the kernel source: it's 16, and is related to limiting the number of fragments or pages that can be sent to a device ring buffer. We don't know `offset`, `len`, or the return of `xennet_count_skb_frag_slots(skb)`.

I'm a little familiar with this codepath already, and have an idea of the real size of these skbs (eg, by using "perf-stat-hist net:net_dev_xmit len 10" from perf-tools; see the example), and what the offset might be. I'd like to check them directly here, but I'll start by instrumenting the return of xennet_count_skb_frag_slots() using perf_events:

```
# perf probe 'xennet_count_skb_frag_slots%return ret=$retval'
Return probe must be on the head of a real function.
  Error: Failed to add events. (-22)
# grep xennet_count_skb_frag_slots /proc/kallsyms
#
```

... Really compiler, can't you inline someone else?

So this symbol doesn't exist. For most tracers, this is a dead end.

Having used a kernel dynamic tracer for a decade, I've developed all kinds of tricks and hacks to work around this: maybe there's a child of xennet_count_skb_frag_slots() function I can trace; maybe I can duplicate the logic from xennet_start_xmit(), where I can observe skb. I began trying such workarounds, but in this case it was becomming onerous. At this point I'd usually start considering editing the kernel and inserting instrumentation just to see this, which involves a compile, test, and deploy cycle.

But on Linux, **with kernel debuginfo**, I can go a lot further directly. Let's switch to the entry to that function, and use "-nv" to show what perf probe would have done without doing it:

```
# perf probe -nv xennet_count_skb_frag_slots
probe-definition(0): xennet_count_skb_frag_slots
symbol:xennet_count_skb_frag_slots file:(null) line:0 offset:0 return:0 lazy:(null)
0 arguments
Looking at the vmlinux_path (6 entries long)
symsrc__init: cannot get elf header.
Using /lib/modules/3.13.11.6/build/vmlinux for symbols
found inline addr: 0xffffffff8152dae8
Probe point found: xennet_start_xmit+88
find 1 probe_trace_events.
Opening /sys/kernel/debug//tracing/kprobe_events write=1
Added new event:
Writing event: p:probe/xennet_count_skb_frag_slots xennet_start_xmit+88
  probe:xennet_count_skb_frag_slots (on xennet_count_skb_frag_slots)

You can now use it in all perf tools, such as:

    perf record -e probe:xennet_count_skb_frag_slots -aR sleep 1
```

Wow, so the beginning of this inlined function can indeed be instrumented, since Linux can trace kernel instructions (or line numbers). Note the offset "+88".

Lines inside this function (despite it being inlined) can also be traced. The perf report command will list candidates with -L:

```
# perf probe -L xennet_count_skb_frag_slots
      0  static int xennet_count_skb_frag_slots(struct sk_buff *skb)
         {
      2          int i, frags = skb_shinfo(skb)->nr_frags;
      3          int pages = 0;

      5          for (i = 0; i < frags; i++) {
                         skb_frag_t *frag = skb_shinfo(skb)->frags + i;
      7                  unsigned long size = skb_frag_size(frag);
                         unsigned long offset = frag->page_offset;

                         /* Skip unused frames from start of page */
     11                  offset &= ~PAGE_MASK;

     13                  pages += PFN_UP(offset + size);
                  }

                  return pages;
```

This is pretty amazing. perf shows line numbers of those that can be instrumented directly, and those that can't in blue.

Some local variables can also be inspected. Let's look at what is available at line 11:

```
# perf probe -V xennet_count_skb_frag_slots:11
Available variables at xennet_count_skb_frag_slots:11
        @<xennet_start_xmit+155>
                int     pages
                long unsigned int       size
```

The `offset` variable is missing, but I have others to work with, which is pretty useful. I can also include external symbols; add an "--externs" to the end of that one-liner for the list.

But don't you hate kernel debuginfo?

Yes, it's rainbows and ponies with kernel debuginfo, but it's also over 100 Mbytes. At Netflix, we create and destroy cloud instances frequently (auto-scaling and code deployments), and it's important to keep the instance size small to reduce creation time, and keep network traffic down.

I've come up with a simple workaround: create one small test instance with kernel debuginfo for each kernel version used, and use it for reference. Let's say I wanted to trace those local variables on line 11 of xennet_count_skb_frag_slots(). On my reference instance:

```
# perf probe -nv 'xennet_count_skb_frag_slots:11 pages size' 2>&1 | grep Writing
Writing event: p:probe/xennet_count_skb_frag_slots xennet_start_xmit+155 pages=%si:s32 size=%di:u
```

Now I copy-n-paste, with a mouse, the highlighted details for use in perf probe (or my kprobe from perf-tools, if it's a real function entry) on the system without debuginfo. Eg:

```
# perf probe 'xennet_count_skb_frag_slots:11 pages size'
Failed to find path of kernel module.
Failed to open debuginfo file.
  Error: Failed to add events. (-2)
# perf probe 'xennet_start_xmit+155 pages=%si:s32 size=%di:u64'
Failed to find path of kernel module.
Added new event:
  probe:xennet_start_xmit (on xennet_start_xmit+155 with pages=%si:s32 size=%di:u64)

You can now use it in all perf tools, such as:

    perf record -e probe:xennet_start_xmit -aR sleep 1
```

Awesome!

I began by showing that this system really doesn't have kernel debuginfo. The second try, with the register details from the reference system, worked. Note that this approach will only create a valid probe if the kernel versions are identical. If you try this on a different kernel, it may appear to work, but provide invalid results.

SystemTap can automate the use of reference systems, although when I tried I had issues with firewalling and port forwarding due to our environment. As for other ways: I met Masami Hiramatsu at LinuxCon North America, and he came up with a way to build a simple text database of functions and variables – stripping debuginfo down to just what I needed. I'll blog about that when I get a chance.

These reference systems are also useful for testing tracing invocations, before using them in production.

Some output

Enabling this probe:

```
# perf record -e probe:xennet_start_xmit -aR sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 1.466 MB perf.data (~64033 samples) ]
# perf script
[...]
  sshd 92592 [009] 87585.288990: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=0 size=280
  sshd 92592 [009] 87585.295461: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=0 size=3058
  sshd 92592 [009] 87585.295472: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=0 size=4f8
  sshd 92592 [009] 87585.296417: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=0 size=538
  sshd 92592 [009] 87585.296426: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=0 size=1c8
  sshd 92592 [009] 87585.304101: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=0 size=29e0
  sshd 92592 [009] 87585.304102: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=3 size=bdc
  sshd 92592 [009] 87585.304111: probe:xennet_start_xmit: (ffffffff8152e6bb) pages=0 size=4d4
[...]
```

Great. The sizes are larger than the expected MTU, because of TCP send offload (TSO). Warning: any network packet tracing can cost significant overheads, especially for 10 GbE speeds and beyond, so be careful on the network path and look for other solutions first.

What about the rocket?

It's a driver bug with TSO. A very large skb can span too many pages (more than 16) to be put in the driver ring buffer. One workaround is "sudo ethtool -K eth0 tso off", for your interface. There's plenty of articles about this on the Internet, and they are easy to find thanks to our mysterious message. :-)

Conclusion

I don't always need Linux kernel line number tracing, but sometimes it is very handy. Local variables at given line numbers can also be inspected. This is useful for both performance analysis and debugging, such as the analysis of our "skb rides the rocket" issue.

To use this feature without kernel debuginfo on all cloud instances, I used a reference system approach. This reference system also serves as places to test specific tracing, before using it in production.

As with all kernel tracing: be careful, as there have been panics and freezes in the past, and know what you are doing before use. For more about perf, see my perf_events page and the perf wiki.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).