Linux ftrace TCP Retransmit Tracing

06 Sep 2014

Why is my Linux system doing TCP retransmits? Lets see packet details and kernel state:

```
# ./tcpretrans
TIME     PID    LADDR:LPORT          -- RADDR:RPORT          STATE
05:16:44 3375   10.150.18.225:53874  R> 10.105.152.3:6001    ESTABLISHED
05:16:44 3375   10.150.18.225:53874  R> 10.105.152.3:6001    ESTABLISHED
05:16:54 4028   10.150.18.225:6002   R> 10.150.30.249:1710   ESTABLISHED
05:16:54 4028   10.150.18.225:6002   R> 10.150.30.249:1710   ESTABLISHED
05:16:54 4028   10.150.18.225:6002   R> 10.150.30.249:1710   ESTABLISHED
05:16:55 0      10.150.18.225:47115  R> 10.71.171.158:6001   ESTABLISHED
05:16:58 0      10.150.18.225:44388  R> 10.103.130.120:6001  ESTABLISHED
^C
Ending tracing...
```

Awesome!

tcpretrans is a script from my perf-tools collection, and uses ftrace to dynamically instrument the tcp_retransmit_skb() kernel function. One reason this is awesome is that the overhead is negligible: it only adds instrumentation to the retransmit path. It's also using existing Linux kernel features, ftrace and kprobes, and doesn't even need kernel debuginfo.

This does not trace every packet and then filter, such as if I had used any tcpdump/libpcap/kernel-packet-filter approach, which can become painful for high packet rates. (I think it would also be lazy to trace every packet when you can just trace the kernel retransmit code path.)

Using a tracer like ftrace means I can also dig out kernel state, which is invisible to on-the-wire tracers like tcpdump. I only included the kernel STATE column, but anything kernel-side could be included.

The -s option will include the kernel stack trace that led to the TCP retransmit:

```
# ./tcpretrans -s
TIME     PID    LADDR:LPORT          -- RADDR:RPORT          STATE
06:21:10 19516  10.144.107.151:22    R> 10.13.106.251:32167  ESTABLISHED
 => tcp_fastretrans_alert
 => tcp_ack
 => tcp_rcv_established
 => tcp_v4_do_rcv
 => tcp_v4_rcv
 => ip_local_deliver_finish
 => ip_local_deliver
 => ip_rcv_finish
 => ip_rcv
 => __netif_receive_skb
 => netif_receive_skb
 => handle_incoming_queue
 => xennet_poll
 => net_rx_action
 => __do_softirq
 => call_softirq
 => do_softirq
 => irq_exit
 => xen_evtchn_do_upcall
 => xen_do_hypervisor_callback
[...]
```

In this case, the retransmit was sent after receiving a packet (ip_rcv()), processing a TCP ACK (tcp_ack()), and then by tcp_fastretrans_alert(). This is a TCP fast retransmit. Ie, these:

```
# netstat -s | grep -i retr
    242 segments retransmited
    46 fast retransmits
    2 forward retransmits
    3 retransmits in slow start
```

Here are a couple of timer-based retransmits for comparison:

```
06:38:45 0      10.11.172.162:7102   R> 10.10.153.60:47538   ESTABLISHED
 => tcp_write_timer_handler
 => tcp_write_timer
 => call_timer_fn
 => run_timer_softirq
 => __do_softirq
 => irq_exit
 => xen_evtchn_do_upcall
 => xen_hvm_callback_vector
 => default_idle
 => arch_cpu_idle
 => cpu_startup_entry
 => start_secondary
06:38:45 0      10.11.172.162:7102   R> 10.10.153.60:47539   ESTABLISHED
 => tcp_write_timer_handler
 => tcp_write_timer
 => call_timer_fn
 => run_timer_softirq
 => __do_softirq
 => irq_exit
 => xen_evtchn_do_upcall
 => xen_hvm_callback_vector
 => default_idle
 => arch_cpu_idle
 => cpu_startup_entry
 => rest_init
 => start_kernel
 => x86_64_start_reservations
 => x86_64_start_kernel
```

These come from a callback, and tcp_write_timer_handler(). Timer-based retransmits are worse than fast retransmits, as they add timer-based latency to application requests. This is usually 1000 ms in Linux.

My tcpretrans ftrace-based tool is a hack, and may not work on some systems without alterations (it also doesn't support IPv6 yet). To dig out custom details like IP addresses as dotted quad strings, I really should be using a programmable tracer like SystemTap. However, I wanted to see if this was possible with just ftrace, as it would make it easier to use in my production environment (Netflix cloud).

It works like this:

1. Dynamically instrument tcp_retransmit_skb() using kprobes, and capture the %di register.
2. Assume the skb pointer is in register %di (to know for certain would require kernel debuginfo, which I don't normally have on production systems). On non-x86 systems, this may well be in another register, and this script will need editing.
3. Wait one second.
4. Read the kernel buffer of tcp_retransmit_skb() events, with skb pointers.
5. Read /proc/net/tcp, and cache socket details by skb.
6. Assume retransmits happen for long-lived sessions (> 1 second), and the session details would still have been in /proc/net/tcp when it was read.
7. Parse the kernel buffer of tcp_retransmit_skb() events and print retransmit events with session details from the /proc/net/tcp data we read earlier.
8. Goto 3.

On an earlier version, I read /proc/net/tcp synchronously when retransmits occurred, but on production systems with frequent retransmits and tens of thousands of open connections (making for a very large /proc/net/tcp), the CPU overhead of that approach was too high. The approach above only reads /proc/net/tcp once per second.

So, it was a gift to have socket details in /proc/net/tcp, and not need to dig them out using ftrace. That would be possible, but the script would become much more brittle without kernel debuginfo, as there would then be several assumptions about registers and struct offsets, rather than just skb being in %di. Without /proc/net/tcp, I'd only really attempt this *with* kernel debuginfo, where I could make it reasonbly reliable.

So far tcpretrans has proved quite useful to quickly get some details on TCP retransmits: not just source and destionation addresses and ports, but kernel state and stack traces.

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).

About this blog

Perf Methods
USE Method
TSA Method
Off-CPU Analysis
Active Bench.
Flame Graphs
Heat Maps
Frequency Trails
Colony Graphs
perf Examples
eBPF Tools
DTrace Tools
DTraceToolkit
DtkshDemos
Guessing Game
Specials
Books
Other Sites