

## Re: network performance, a low-level measurement

**Source:** <http://unix.derkeiler.com/Mailing-Lists/FreeBSD/current/2005-01/1087.html>

---

**From:** Robert Watson ([rwatson\\_at\\_FreeBSD.org](mailto:rwatson_at_FreeBSD.org))

**Date:** 01/31/05

Date: Mon, 31 Jan 2005 13:01:51 +0000 (GMT)

To: Poul-Henning Kamp <[phk@phk.freebsd.dk](mailto:phk@phk.freebsd.dk)>

(The upshot is at the bottom, if you don't care about the diagnosis, skip to the last paragraph)

On Mon, 31 Jan 2005, Poul-Henning Kamp wrote:

>  
> >Why are we entering `sis_start()`  
> >here if `net.isr.enable=0` and the system is otherwise idle?  
>  
> `sis_start` is called to return the ICMP ECHO reply packet obviously :-)

On an unmodified system, here's the normal (approximate) series of events to get the ping processed:

- Low level interrupt handler fires
- Device driver `ithread` is scheduled
- Device driver `ithread` preempts
- Device driver `ithread` runs
- Device driver does inbound packet processing
- Device driver pulls packet out of hardware (so to speak)
- Device driver passes packet to link layer (`ether_input`)
- Link layer demuxes packet to `netinet netisr q`
- Link layer schedules `netisr`
- Link layer returns
- Device driver does outbound packet processing
- Device driver `ithread` finishes
- Context switch to `netisr`
- `Netisr` runs `ip_input()`
- `ip_input()` converts ICMP echo request to ICMP echo response
- `ip_output()` enqueues ICMP echo response to device driver `ifnet` queue
- `ip_output()` calls `if_start`
- `if_start` calls device driver start
- Device driver queues packet to hardware
- Device driver starts hardware

freebsd-current: Re: network performance, a low-level measurement

The primary difference between this sequence and your sequence, modulo `s/ithread/taskqueue/` is that the `if_start` call happens before the task queue (`ithread`) finishes its run. This can happen in one of three cases: (1) you have `net.isr.enable` set to 1, so the link layer direct dispatches to the network layer, invoking `ip_input()` directly and resulting in the eventual call to `if_start()`, (2), you have an SMP system and the `netisr` runs on the second CPU in parallel and manages to send before the first processor has unwound, or (3) the `netisr` preempts the task queue due to having a lower priority as an `ithread`, causing it to run to completion before returning to the task queue thread to complete its run.

Given that it's a Soekris, and hence not SMP, I'm assuming that (3) is the answer, since you're not using (1). So you're getting an extra two context switches in there as a result of what is effectively priority inversion, as the hardware driver is running at a priority that gives precedence to the `netisr` (`ithread`). This is probably undesirable, since `net.isr.enable=1` is a much cheaper way to accomplish the same thing (avoiding the context switches, etc).

Presumably the preemption occurs because `SWI_TQ/SWI_TQ_FAST` are greater than `SWI_NET`. Generally, we give hardware `ithreads` precedence over the `netisr`. Do we need a new SWI priority for task queue threads supporting device drivers — i.e., acting in the role of an `ithread`?

Robert N M Watson

---

freebsd-current@freebsd.org mailing list

<http://lists.freebsd.org/mailman/listinfo/freebsd-current>

To unsubscribe, send any mail to "freebsd-current-unsubscribe@freebsd.org"