Lessons learned from using Multiqueue and RPS

Holger Eitzenberger <holger.eitzenberger@sophos.com>

Outlook

- not so much about Netfilter, but about letting Network scale on multicore

Background

- multiqueue hardware (10 GB) becoming commodity, with 40 GB on the rise
- RPS / RFS / XPS integrated (in that order)
  - RPS: choose a different CPU at `netif_receive_skb()` time
  - RFS: same CPU than proxy chosen (local generated traffic only)
- but living with an old kernel (2.6.32) :(
- used irqbalance before

Integration

- backport: RFS / RFS (XPS later)
- backporting wasnt hard, because rather self-contained
- lets use it!

Integration

- irqbalance isnt́ RPS / XPS aware :(
- so lets check what others do:
  - Vyatta: Perl script
  - Ubuntu: RC script hacking
- so let's develop our own!

Performance

- Multiqueue: spreading the RX / TX queues across as much CPUs as possible
  - staircase effect in /proc/interrupts
- RPS: good performance increase up to 4 CPUs in CPU intensive work loads
- RPS: 2-CPU slightly worse compared to single CPU for in-dev and out-dev

irqd

- differentiate between hardware multiqueue and RPS
- starts by using 2 CPUs per RPS queue initially
- scales up to using 4 CPUs per RPS queue
- uses Netlink (RTNL link events most importantly)

Problems Encountered

- Several interfaces:
  - `/proc/interrupts`
  - `/proc/irq/$IRQ/smp_affinity`
  - `/proc/dev/net`
  - `/sys/class/net/eth0/queus/rx-0/rps_flow_cnt`
- `/proc/interrupts`: representation if different NICs is different when using Hypervisors (KVM, Xen, ...)
- both multiqueue and RPS / XPS fail kernel principle “choose sane defaults”

Conclusion / Wishlist

Kernel

- improve the kernel defaults!
  - but it is “Policy” in the end, so some of it has to remain in userland

`irqd`

- partitioning CPUs (e. g. dedicated CPUs for packet processing)
- make in-dev and out-dev use same CPU if CPU load permits
  - reduces RTT
  - then start from a single CPU
- github: `https://github.com/vaesoo/irqd.git`