Hi-speed fairness

Jens Låås, UU 2009. Version 2.

Using the same setup as for the 10Gb routing testing here at UU.
We tried to see how fairness via the sfq qdisc works under 10Gb load.

The sfq qdisc basically separates flows and uses round-robin dequeueing from the flows.

Number of CPUs are 4 unless otherwise noted.
The load is simulated internettraffic regarding packetsize distribution and number of flows. IP destination is varied.
Load is 9517.2 M bit/s and 1.3 M pps.
Since the load consists of UDP packets there are no TCP-sessions that may be affected by the sfq algorithm. The traffic is not normal in that respect.
In a real scenario we would espect the TCP-sessions to throttle down.
'tc' program from 2009 is needed to support the configuring the sfq classifier. This is included in bifrost 6.0.

We receive traffic on eth0 and route it to eth1.

sfq

Default setting: limit 128 ncpu 8
eth0 2962.5 M bit/s 416 k pps 0 bit/s 0 pps
eth1 3.0 k bit/s 0 pps 438.0 M bit/s 61 k pps

When we use sfq with all CPUs the system becomes almost livelocked. Hard to get results.

Next try with 4 cpus.

limit 128
eth0 4368.3 M bit/s 623 k pps 4.8 k bit/s 2 pps
eth1 2.3 k bit/s 1 pps 3543.3 M bit/s 524 k pps

Queue might be too short. Lets try with 1000 as queue length.
To increase sfq depth above 128 a small patch is needed.

limit 1000
eth0 4666.0 M bit/s 667 k pps 3.6 k bit/s 1 pps
eth1 2.6 k bit/s 0 pps 4627.3 M bit/s 662 k pps

Still bad with 8 cpus.
limit 1000 ncpu 8
eth0 2305.6 M bit/s 329 k pps 7.8 k bit/s 2 pps
eth1 6.8 k bit/s 1 pps 452.3 M bit/s 64 k pps

multiq + sfq

Next try is to try separate the flows (mainly the CPUs) from each other.

Tried first the multiq qdisc.
sch_multiq multiplexes the qdisc per tx queue of the device it is attached to. It does a round-robin dequeue which means the pkts are moved between CPUs.

We dont want the packets to move between ther CPUs so I pacthed sch_multiq to be per cpu instead. Dequeue is also per CPU.
So packets enqueued by CPU1 is always dequeued by CPU1.
Should be renamed pcpu or something like it...

Best result with a shorter queue length. Remember total number of packets in queue is Number of CPU times queue length: 8 x 64 = 512.
limit 64

eth0	6301.2 M bit/s	909 k pps	5.1 k bit/s	1 pps
eth1	4.2 k bit/s	0 pps	6248.1 M bit/s	902 k pps

A bit less with total queue of 1000.
limit 128

eth0	5691.7 M bit/s	818 k pps	3.1 k bit/s	1 pps
eth1	1.8 k bit/s	1 pps	5645.2 M bit/s	812 k pps

Still bad with all 8 CPUs.
limit 64 ncpu 8

eth0	4513.9 M bit/s	644 k pps	5.3 k bit/s	3 pps
eth1	2.1 k bit/s	0 pps	892.2 M bit/s	127 k pps

Ratelimit:

We used htb qdisc for ratelimiting. Rate is set to 8000mbit, so that we will never actually reach the limit.
We attach a HTB qdisc at root and one SFQ under the ratelimiting HTB.

htb + sfq

limit 128
eth0 4046.6 M bit/s 572 k pps 888 bit/s 0 pps
eth1 3.2 k bit/s 1 pps 1760.0 M bit/s 258 k pps

Best result with HTB + SFQ:
limit 256
eth0 4086.8 M bit/s 576 k pps 0 bit/s 0 pps
eth1 4.5 k bit/s 0 pps 1934.1 M bit/s 280 k pps

limit 1000
eth0 4198.0 M bit/s 590 k pps 0 bit/s 0 pps
eth1 4.3 k bit/s 1 pps 1821.5 M bit/s 258 k pps

htb + multiq + sfq

Lets try with the per-cpu qdisc. Using the patched multiq.

Best result. Still not very good though. Notice the imbalance between RX and TX.

limit 64

eth0	5162.5 M bit/s	728 k pps	968 bit/s	0 pps
eth1	3.0 k bit/s	1 pps	2729.3 M bit/s	386 k pps

limit 64 ncpu 8

eth0	4237.2 M bit/s	598 k pps	0 bit/s	0 pps
eth1	2.6 k bit/s	1 pps	637.0 M bit/s	90 k pps

limit 1000

eth0	5100.8 M bit/s	719 k pps	56 bit/s	0 pps
eth1	3.7 k bit/s	0 pps	2245.7 M bit/s	325 k pps

Factors:

A number of factors affect the test result.

Some of them are:

- Load. If we decrease the number of pps sent by pktgen we get higher throughput.
- Number of CPUs.
- multiq or not. IRQ/CPU affinity.
- Driver. card + select_queue algo.

Problems:

- If we use more than 4 CPU they will spend most of their time waiting for the spinlock when serializing at the qdisc. This will happen even if the qdisc is per CPU internally.
- We are close to the limit of what the hardware can do. Small changes may affect the result considerably. Each setup needs to be tuned individually for best performance...
- With the pcpu qdisc the sfq qdisc will work within sets of flows that are already partitioned by network-card. So we only get fairness within each set of "randomly" distributed flows.

Conclusion

- Defaults are not too bad. sfq without a pcpu infront performs quite good if you remember to limit the number of CPUs and increase the queue length.
- Never really got htb+sfq to perform in a high speed environment.