RSS/RPS + locking qdisc

August 6, 2020

I recently learned this fun fact: With RSS or RPS enabled [1] and a lock-based qdisc on a VM's tap device (e.g., fq_codel) a UDP packet storm targeted at the VM can severely impact the entire server.

The point of RSS/RPS is to distribute the packet processing load across all hardware threads (CPUs) in a server / host. However, when those packets are forwarded to a single device that has a lock-based qdisc (e.g., virtual machines and a tap device or a container and veth based device) that distributed processing causes heavy spinlock contention resulting in ksoftirqd spinning on all CPUs trying to handle the packet load.

As an example, my server has 96 cpus and 1 million udp packets per second targeted at the VM is enough to push all of the ksoftirqd threads to near 100%:

```
  PID %CPU COMMAND               P
   58 99.9 ksoftirqd/9           9
  128 99.9 ksoftirqd/23         23
  218 99.9 ksoftirqd/41         41
  278 99.9 ksoftirqd/53         53
  318 99.9 ksoftirqd/61         61
  328 99.9 ksoftirqd/63         63
  358 99.9 ksoftirqd/69         69
  388 99.9 ksoftirqd/75         75
  408 99.9 ksoftirqd/79         79
  438 99.9 ksoftirqd/85         85
 7411 99.9 CPU 7/KVM            64
   28 99.9 ksoftirqd/3           3
   38 99.9 ksoftirqd/5           5
   48 99.9 ksoftirqd/7           7
   68 99.9 ksoftirqd/11         11
   78 99.9 ksoftirqd/13         13
   88 99.9 ksoftirqd/15         15
   ...
```

`perf top` shows the spinlock contention:

```
    96.79%  [kernel]          [k] queued_spin_lock_slowpath
     0.40%  [kernel]          [k] _raw_spin_lock
     0.23%  [kernel]          [k] __netif_receive_skb_core
     0.23%  [kernel]          [k] __dev_queue_xmit
     0.20%  [kernel]          [k] __qdisc_run
```

With the callchain leading to

```
    94.25%  [kernel.vmlinux]    [k] queued_spin_lock_slowpath
            |
             --94.25%--queued_spin_lock_slowpath
                       |
                        --93.83%--__dev_queue_xmit
                                  do_execute_actions
                                  ovs_execute_actions
                                  ovs_dp_process_packet
                                  ovs_vport_receive
```

A little code analysis shows this is the qdisc lock in __dev_xmit_skb.

The overloaded ksoftirqd threads means it takes longer to process packets resulting in budget limits getting hit and packet drops at ingress. The packet drops can cause ssh sessions to stall or drop or cause disruptions in protocols like LACP.

Changing the qdisc on the device to a lockless one (e.g., noqueue) dramatically lowers the ksoftirqd load. `perf top` still shows the hot spot as a spinlock:

```
    25.62%  [kernel]          [k] queued_spin_lock_slowpath
     6.87%  [kernel]          [k] tasklet_action_common.isra.21
     3.28%  [kernel]          [k] _raw_spin_lock
     3.15%  [kernel]          [k] tun_net_xmit
```

but this time it is the lock for the tun ring:

```
    25.10%  [kernel.vmlinux]    [k] queued_spin_lock_slowpath
            |
             --25.05%--queued_spin_lock_slowpath
                       |
                        --24.93%--tun_net_xmit
                                  dev_hard_start_xmit
                                  __dev_queue_xmit
                                  do_execute_actions
                                  ovs_execute_actions
                                  ovs_dp_process_packet
                                  ovs_vport_receive
```

which is a much lighter lock in the sense of the amount of work done with the lock held.

systemd commit e6c253e363dee, released in systemd 217, changed the

default qdisc from pfifo_fast (kernel default) to fq_codel (/usr/lib/sysctl.d/50-default.conf for Ubuntu). As of v5.8 kernel fq_codel still has a lock to enqueue packets, so systems using fq_codel with RSS/RPS are hitting this lock contention which affects overall system performance. pfifo_fast is lockless as of v4.16 so for newer kernels the kernel's default is best.

But, it begs the question why have a qdisc for a VM tap device (or a container's veth device) at all? To the VM a host is just part of the network. You would not want a top-of-rack switch to buffer packets for the server, so why have the host buffer packets for a VM? (The "Tx" path for a tap device represents packets going *to* the VM.)

You can change the default via:

```
sysctl -w net.core.default_qdisc=noqueue
```

or add that to a sysctl file (e.g., /etc/sysctl.d/90-local.conf). sysctl changes affect new devices only.

Alternatively, the default can be changed for selected devices via a udev rule:

```
cat > /etc/udev/rules.d/90-tap.rules <<EOF
ACTION=="add|change", SUBSYSTEM=="net", KERNEL=="tap*", PROGRAM="/sbin/tc
qdisc add dev $env{INTERFACE} root handle 1000: noqueue"
```

Running `sudo udevadm trigger` *should* update existing devices. Check using `tc qdisc sh dev <NAME>`:

```
$ tc qdisc sh dev tapext4798884
qdisc noqueue 1000: root refcnt 2
qdisc ingress ffff: parent ffff:fff1 ----------------
```

[1] https://www.kernel.org/doc/Documentation/networking/scaling.txt