Queues in the Linux kernel

An overview

Mathieu Goutelle

Mathieu.Goutelle@ens-lyon.fr

INRIA-RESO (France)

INRIA

DataTAG

Goal of the presentation

- When some High Performance tests are performed (SR, RHJ, AA), we observe that TCP encounters some loss during the tests...
- but there is no loss measured on the line...
- So my goal here is to investigate and clarify the influence of all the queues in the Linux kernel, from the socket layer to the network device driver ;
- To know where and when losses happen in the kernel.

INRIA

DataTAG

Summary (source side)

- The process do a `write()` on a socket. The data is copied from the process space into the **send socket buffer** [queue #1] ;
- The data goes through TCP/IP and the packets are put (by reference) into the **queuing discipline** attached to the NIC [queue #2] ;
- Finally, the transmission procedure of the driver is called. Generally, the driver implements a **ring buffer** (`tx_ring`) shared with the NIC [queue #3].

INRIA

DataTAG

Summary (reception side) (1)

- The driver implements a **ring buffer** (`rx_ring`) in which the NIC puts the incoming paquets. To have a ring buffer rather than a single packet buffer allows to deal with bursts [queue #1] ;
- The interrupt handler of the driver takes the packet from the `rx_ring`, puts it (by reference) in the **backlog queue** [queue #2] and schedules a `softirq` (a kind of kernel thread) ;

INRIA

DataTAG

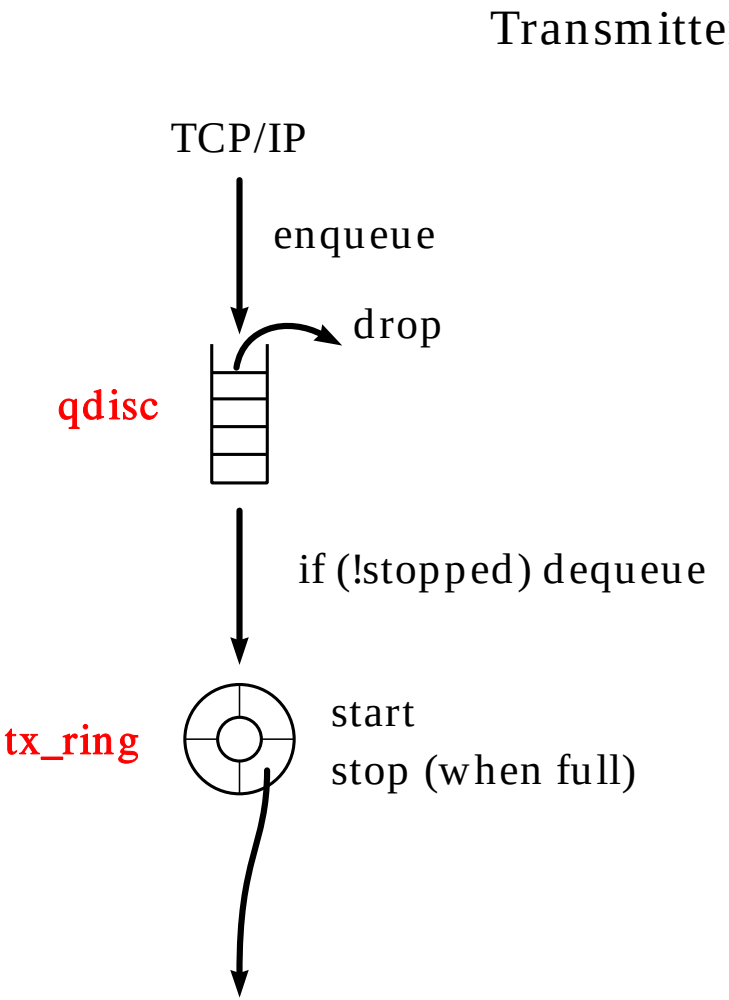
Summary (reception side) (2)

- When the interrupt handler returns, the `softirq` (previously scheduled) triggers and executes the TCP/IP stack, then puts the data (always by reference) into the **receive socket buffer** [queue #3] and awakes the sleeping over this queue processes (*e.g.* with a blocking `read()` on the socket) ;
- The process copies the data from the kernel space into the reception buffer (the one specified in the parameters of `recv()`).

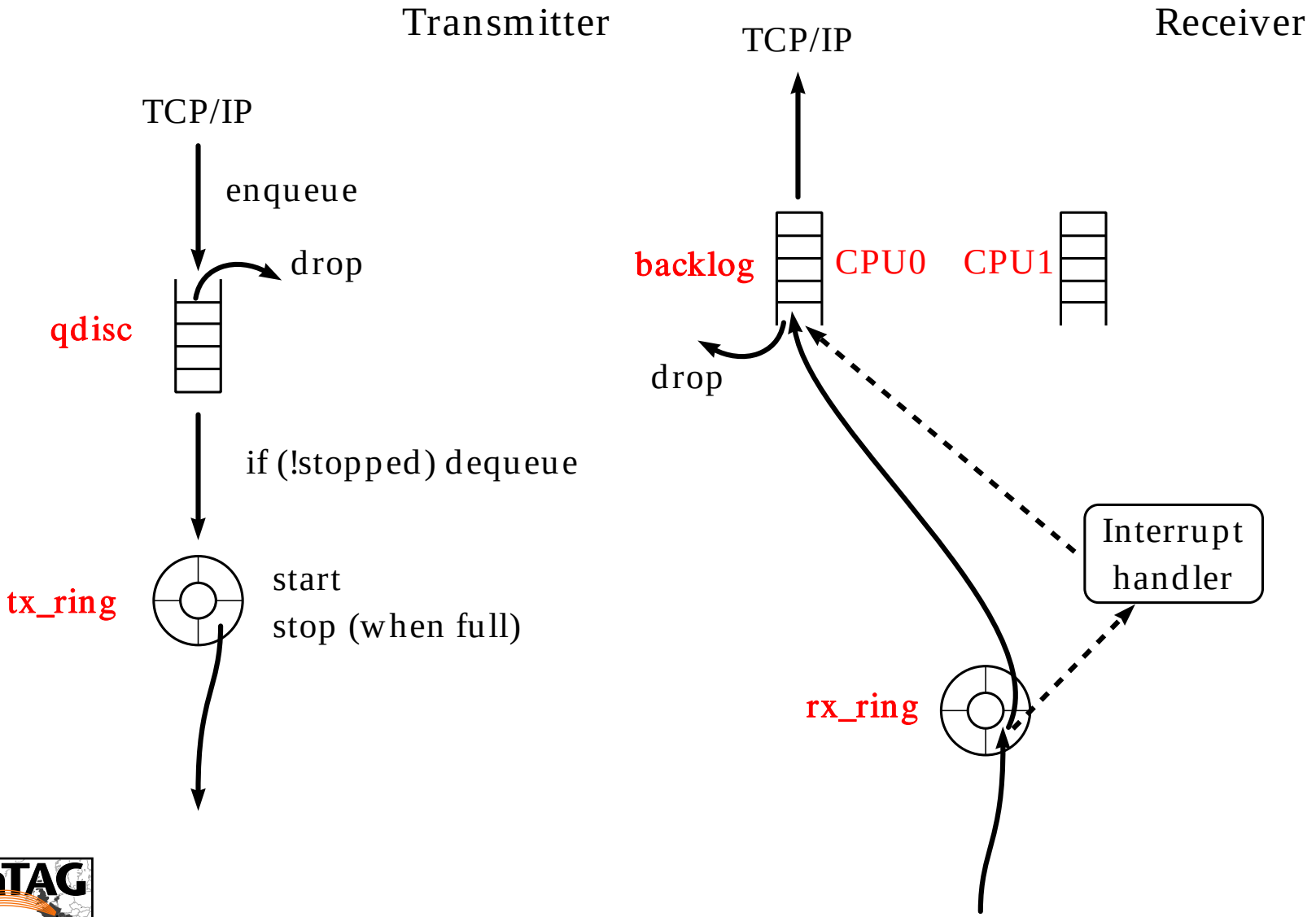
INRIA

DataTAG

Drops ?


Transmitter
TCP/IP
enqueue
drop
qdisc
if (!stopped) dequeue
tx_ring
start
stop (when full)

Drops ?


Transmitter
Receiver
TCP/IP
TCP/IP
enqueue
drop
qdisc
backlog
CPU0
CPU1
drop
if (!stopped) dequeue
Interrupt handler
tx_ring
start
stop (when full)
rx_ring

Typical queue lengths (1)

- The **socket buffers** can be set by the application (`set_sockopt()`) (default value: 65 535 B, `/proc/sys/net/core/[rw]mem_default`);
- The default **queuing discipline** is a FIFO queue. Default length is 100 packets (`ether_setup(): dev->queue_len`, `drivers/net/net_init.c`);

INRIA

DataTAG

Typical queue lengths (2)

- The **tx_ring** and **rx_ring** are driver dependent (e.g. the `e1000` driver set these lengths to 80 packets) ;
- The **backlog** is a 300-packet queue (`/proc/sys/net/core/netdev_max_backlog`). When it is full, it waits for being totally empty to allow again an `enqueue()` (`netif_rx()`, `net/core/dev.c`).

INRIA

DataTAG

Looking for drop ?

During tests, you can check the drops in the machines:

- For the **backlog**, stats are available in `/proc/net/softnet_stats`: one line per CPU, the first two column are packets and drops counts;
- For the **qdisc**, stats are **not** available. So you have to replace the **qdisc** with a FIFO queue of the same length (package iproute).

```
$ tc qdisc add dev eth0 root pfifo limit 100
$ tc -s -d qdisc show dev eth0
$ tc qdisc del dev eth0 root
```

INRIA

DataTAG

Acknowledgments

- All this stuff is only relevant to 2.4 linux kernel.
- I want to thank Éric Lemoine (RESO/SUNLabs Europe) for his help and his detailled explanations about all this mechanisms.

INRIA

DataTAG

Questions