Linux Multiqueue Networking

David S. Miller

Red Hat Inc.

Netfilter Workshop, ESIEA, Paris, 2008

Networking Packets as CPU Work

- Each packet generates some CPU work
- Both at protocol and application level
- Faster interfaces can generate more work
- Sophisticated networking can generate more work
- Firewalling lookups, route lookups, etc.

CPU Evolution

- Multiprocessor systems used to be uncommon
- This is changing... significantly
- Common systems will soon have 64 cpu threads, or more
- Lots of CPU processing resources
- The trouble is finding ways to use them

End to End Principle

- Basic premise: Communication protocol operations should be defined to occur at the end-points of a communications system, or as close as possible to the resource being controlled
- The network is dumb
- The end nodes are smart
- Computational complexity is pushed as far to the edges as possible

End-to-end applied to "system"

- The end-to-end principle extends all the way into your computer
- Each cpu in your system is an end node
- The "network" pushes work to your system
- Internally, work should be pushed down further to the execution resources
- With multi-threaded processors like Niagara, this is more important to get right than ever before

Network Device

- Devices must provide facilities to balance the load of incoming network work
- But they must do so without creating new problems (such as reordering)
- Packet reordering looks like packet loss to TCP
- Independant “work” should call upon independant cpus to process it
- PCI MSI-X interrupts
- Multiqueue network devices, with classification facilities

Software Solutions

- Not everyone will have multiqueue capable network devices or systems.
- Receive balancing is possible in software
- Older systems can have their usefulness extended by such facilities

NAPI, "New API"

- Basic abstraction for packet reception under Linux, providing software interrupt mitigation and load balancing.
- Device signals interrupt when receive packets are available.
- Driver disables chip interrupts, and NAPI context is queued up for packet processing
- Kernel generic networking processes NAPI context in software interrupt context, asking for packets from the driver one at a time
- When no more receive packets are pending, the NAPI context is dequeued and the driver re-enabled chip interrupts
- Under high load with a weak cpu, we may process thousands of packets per hardware interrupt

NAPI Limitations

- It was not ready for multi-queue network devices
- One NAPI context exists per network device
- It assumes a network device only generates one interrupt source which must be disabled during a poll
- Cannot handle cleanly the case of a one to many relationship between network devices and packet event sources

Fixing NAPI

- Implemented by Stephen Hemminger
- Remove NAPI context from network device structure
- Let device driver define NAPI instances however it likes in it's private data structures
- Simple devices will define only one NAPI context
- Multiqueue devices will define a NAPI context for each RX queue
- All NAPI instances for a device execute independantly, no shared locking, no mutual exclusion
- Independant cpus process independant NAPI contexts in parallel

Ok, what about non-multiqueue hardware?

- Remote softirq block I/O layer changes by Jens Axboe
- Provides a generic facility to execute software interrupt work on remote processors
- We can use this to simulate network devices with hardware multiqueue facilities
- netif_receive_skb() is changed to hash the flow of a packet and use this to choose a remote cpu
- We push the packet input processing to the selected cpu

Configurability

- The default distribution works well for most people.
- Some people have special needs or want to control how work is distributed on their system
- Configurability of hardware facilities is variable
- Some provide lots of control, some provide very little
- We can fully control the software implementation

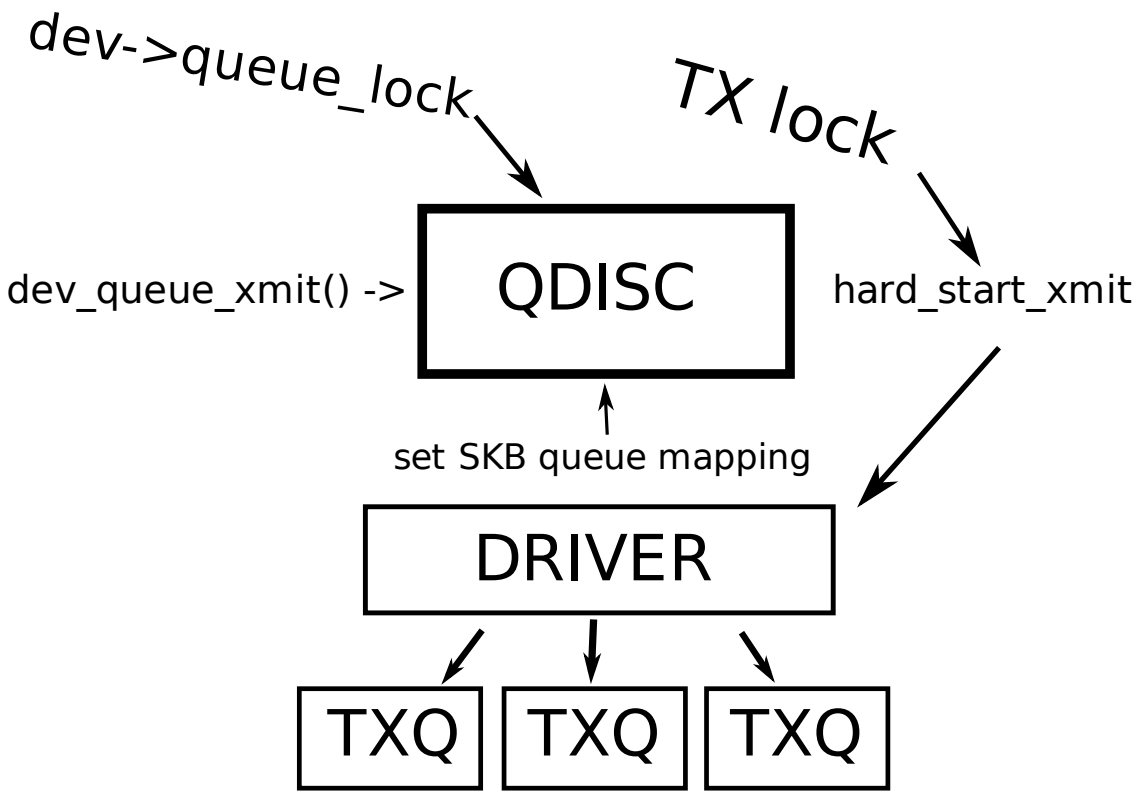
Impetus

- Writing NIU driver
- Peter W's multiqueue hacks
- TX multiqueue will be pervasive.
- Robert Olsson's pending paper on 10GB routing (found TX locking to be bottleneck in several situations)

Presentations

- Tokyo 2008
- Berlin 2008
- Seattle 2008
- Portland 2008
- Implementation plan changing constantly
- End result was different than all of these designs
- Full implementation will be in 2.6.27

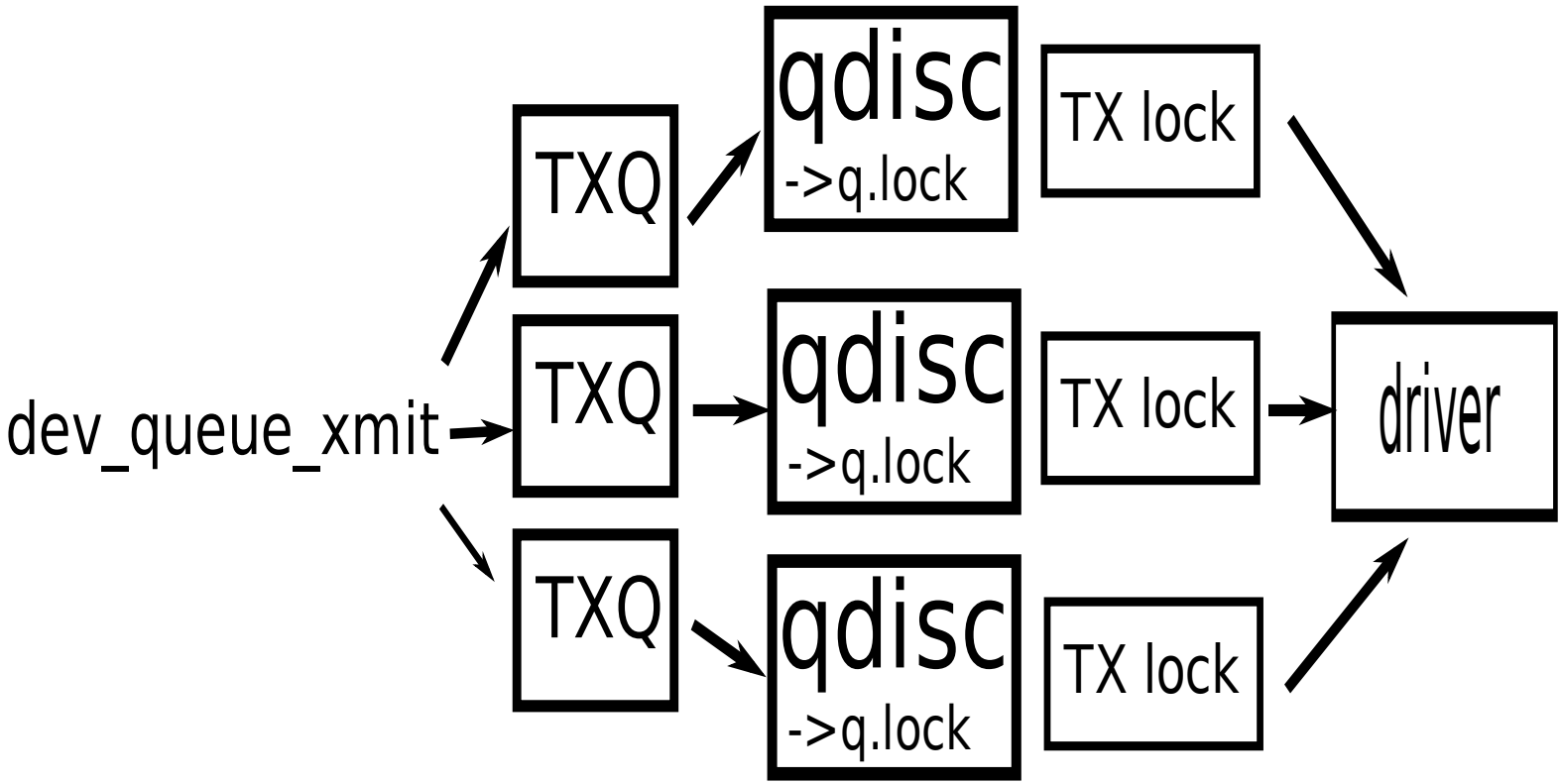
Picture of TX Engine



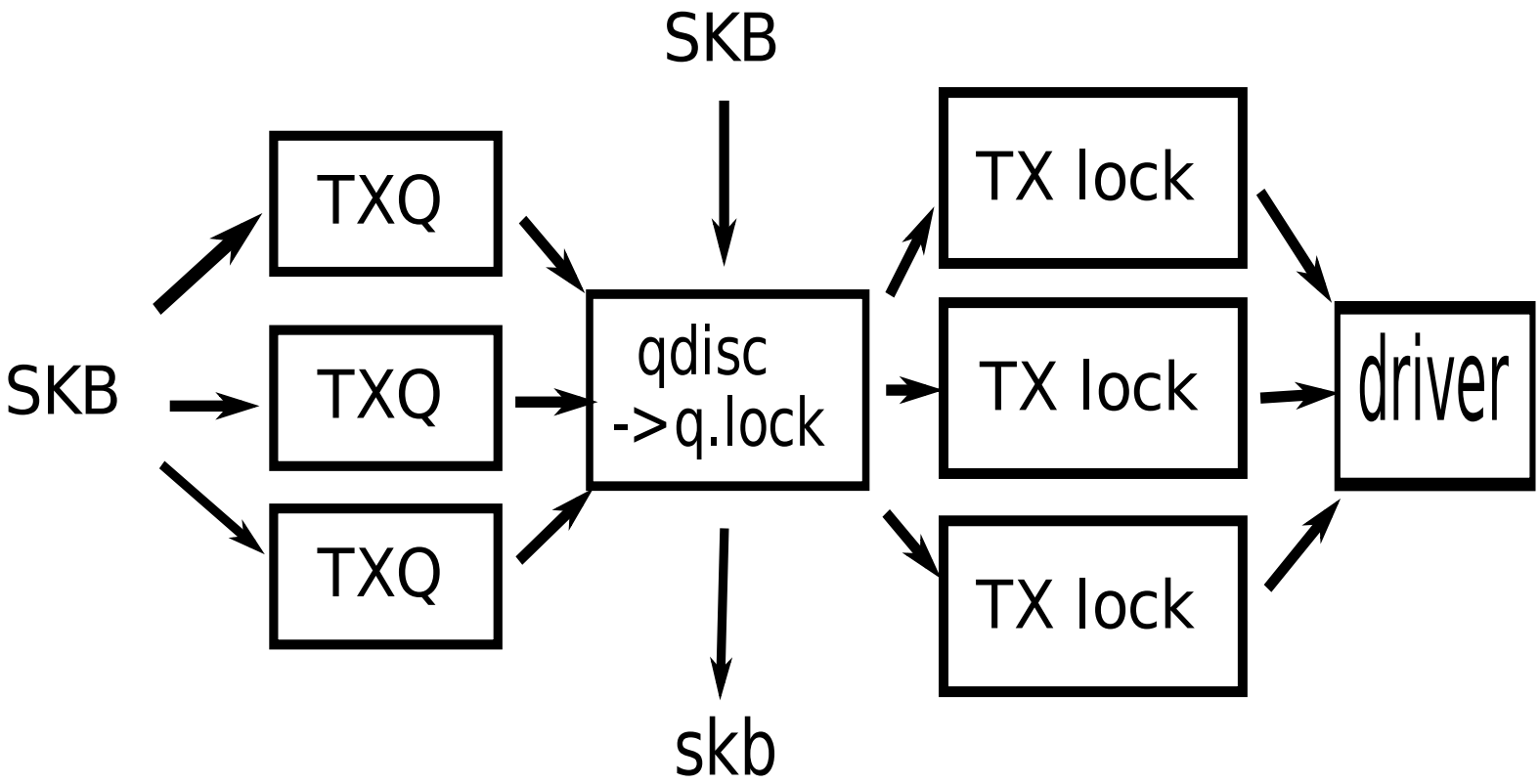
Details to Note

- Generic networking has no idea about multiple TX queues
- Parallelization impossible because of single queue and TX lock
- Only single root QDISC is possible, sharing is not allowed

Picture of Default Configuration



Picture with Non-trivial QDISC



The Network Device Queue

- Direction agnostic, can represent both RX and TX
- Holds:
  - Backpointer to struct net_device
  - Pointer to ROOT qdisc for this queue, and sleeping qdisc
  - TX lock for ->hard_start_xmit() synchronization
  - Flow control state bit (XOFF)

QDISC CHANGES

- Holds state bits for qdisc_run() scheduling
- Has backpointer to referring dev_queue
- Therefore, current configured QDISC root is always qdisc->dev_queue->qdisc_sleeping
- qdisc->q.lock on root is now used for tree synchronization
- qdisc->list of root qdisc replaces netdev->qdisc_list

TX Queue Selection

- Simple hash function over the packet FLOW information (source and destination address, source and destination port)
- Driver or driver layer is allowed to override this queue selection function if the queues of the device are used for something other than flow seperation or prioritization
- Example: Wireless
- In the future, this function will disappear.
- To be replaced with socket hash marking of transmit packets, and RX side hash recording of received packets for forwarding and bridging

Implementation and Semantic Issues

- Synchronization is difficult.
- People want to use queues for other tasks such as priority scheduling, will be supported by sch_multiq.c module from Intel in 2.6.28
- TX multiqueue interactions with the packet scheduler and it's semantics are still not entirely clear
- Most of the problems are about flow control