% Networking
% Jiří Benc, Red Hat
% Advanced Operating Systems, MFF UK

Scope

- focused on Linux
- using Linux terminology
- the principles are general

Assumption

- knowledge of OSI model
- understanding of packet structure
- basic understanding of TCP/IP
- understanding of I/O (DMA, IRQ)

User Point of View

- network interfaces
    - usually having name and numeric ID
    - can be assigned IP addresses
    - can be administratively enabled/disabled
- apps operate with IP addresses
    - but can specify an interface
- system tables
    - routing tables
    - neighbor tables
    - ...

Basic Packet Processing

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor →
L2 → L3 → L4 →
socket lookup → socket queue → app wakeup →
app read → data copy → buffer release

. . .

app write → data copy → packet descriptor →
L4 → L3 → L2 →

enqueue → dequeue →
DMA descriptor → DMA →
tx trigger → NIC tx →
tx IRQ → IRQ handler → memory release

Driver Processing (rx)

NIC rx → DMA

DMA Ring Buffers

- separate tx and rx buffers
- configured by the driver
- contains data and metadata

Driver Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing

Interrupts

- IRQ handler in the driver
- bottom half scheduled
- packet fetched
- new DMA rx buffer allocated

Driver Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor

Packet Descriptor

- allocated by the driver
- sk_buff in Linux, mbuf in BSD, etc.
- packet metadata

Packet Descriptor

- buffer pointer
- data start
- data length
- header pointers
- incoming/outgoing interface
- L3 protocol

- queue priority
- packet mark
- reference count
- offload fields
    - vlan tag
    - hash
    - checksum
- ...

Packet Descriptor

- buffer pointer
- data start ⬅ allows pop/push
- data length
- header pointers
- incoming/outgoing interface
- L3 protocol
- queue priority
- packet mark
- reference count
- offload fields
    - vlan tag
    - hash
    - checksum
- ...

Kernel Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor →
L2

Entering the Network Stack

- driver calls helper functions for L2 processing
    - L3 protocol filled in
    - L2 header removed
- handed over to the core kernel

Kernel Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor →
L2

Common Handling

- taps on network interface (packet inspection)
- rx hooks (virtual interfaces)
- protocol-independent firewall

Kernel Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor →
L2 → L3 → L4

Protocol Layers

- L2 independent
- table of L3 handlers → L3 protocol handler
- L3 header processed and removed
- per-L3 table of L4 handlers → L4 protocol handler
- L4 header processed and removed

Kernel Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor →
L2 → L3

L3 – IP

- defragmentation
- routing decision
    - forwarding: skip to tx path
    - local delivery: continue up the stack
- IP firewall (various attachment points)

Kernel Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor →

L2 → L3 → L4 →
socket lookup → socket queue → app wakeup

L4 – TCP

- TCP state machine
- socket lookup
- socket enqueue (of the sk_buff)
- application woken up

Kernel Processing (rx)

NIC rx → DMA → rx IRQ →
IRQ handler → schedule processing →
packet descriptor →
L2 → L3 → L4 →
socket lookup → socket queue → app wakeup →
app read → data copy → buffer release

Application

- read() syscall
- packet copy
- sk_buff freed

Kernel Processing (tx)

app write → data copy → packet descriptor

Application

- write() syscall
- sk_buff allocation (for DMA)
- data copy

Kernel Processing (tx)

app write → data copy → packet descriptor →
L4 → L3

Protocol Layers

- TCP header pushed
- IP header pushed
    - IP firewall
    - routing decision
    - fragmentation (MTU, PMTU)

Kernel Processing (tx)

app write → data copy → packet descriptor →
L4 → L3 → L2

Protocol Layers

- L2 header pushed
    - neighbor cache, ARP lookup
- may need to wait for neighbor resolution
    - put to a wait list
    - resumed by incoming ARP reply
    - timer assigned for timeout
    - ICMP signalled back on error

Kernel Processing (tx)

app write → data copy → packet descriptor →
L4 → L3 → L2 →
enqueue → dequeue

Tx Queues

- packet classified and enqueued
- dequeued
    - based on queue discipline
    - sk_buff priority field
- passed to the driver

Driver Processing (tx)

app write → data copy → packet descriptor →
L4 → L3 → L2 →
enqueue → dequeue →
DMA descriptor → DMA →
tx trigger → NIC tx

Pushing to the NIC

- added to tx DMA ring buffer
- signalled to the NIC

Driver Processing (tx)

app write → data copy → packet descriptor →
L4 → L3 → L2 →
enqueue → dequeue →
DMA descriptor → DMA →
tx trigger → NIC tx →
tx IRQ → IRQ handler → memory release

Freeing Resources

- NIC signals transmit done
- buffer unmapped, sk_buff released
- counters incremented

Special Protocols

ICMP

- just another L4 protocol
- communicates back to IP
    - PMTU updates
    - route redirects etc.

ARP and ICMPv6

- neighbor discovery

Performance Matters!

Performance Problems

- packet length unknown in advance
    - DMA scatter-gather
    - complicates packet processing (fragmented data)
    - header pop may require realloc

Performance Problems

- header push requires realloc
    - reserve space (at the driver level)
    - still may get out of space

Performance Problems

- enqueueing before tx

    - bufferbloat – high latency, lattency jitter, failure of TPC congestion control
    - smaller buffers, better queueing disciplines

Performance Problems

- shared resources
    - flow caches, defrag buffers, etc.
        - remotely DoSable!
    - global limits
        - locally DoSable
    - per-group limits (cgroups)

Bottlenecks

- stack processing is too heavy
    - aggregation of packets
    - processing whole flows
- interrupts are slow
    - busy polling under load
- reading memory is slow
    - checksum offloading

Checksum Offloading

- for tx, checksum on copy from user
- FCS is always calculated by the NIC
- IP header checksum calculation is cheap
- L4 checksum
    - on rx, the NIC verifies the checksum
    - on tx, the NIC computes and fills in the checksum
- some protocols use CRC instead (SCTP)

Busy Polling under Load (NAPI)

- on rx, turn off IRQs
- fetch packets up to a limit
- repeat until there are no packets left
- turn on IRQ

Aggregation

Rx Aggregation (GRO)

- needs multiple rx queues in NIC
    - configurable filters
- on rx, packets for the same flow from a NAPI batch are combined into a super-packet
    - ⇒ GRO depends on NAPI
    - need to dissect the packets
    - passes the stack as a single packet
    - need to be able to reconstruct the original packets
    - split on tx (GSO)

Aggregation

Tx Aggregation (GSO)

- on tx, a packet is split into smaller packets
    - TCP segmentation for TCP super-packets
    - offloaded to NIC (TSO)
    - ⇒ TSO depends on checksum offloading
    - IP fragmentation for datagram protocols
    - done in software when needed

Virtual NICs

- a driver not backed up by a real hardware
- vlan interface
- tun/tap
- veth
- ...

Containers (Network Name Spaces)

- partitioning of the network stack
- TCP/IP:
    - isolated routing tables
    - ⇒ independent IP addresses
    - separate limits (subject to global limits)
- each network interface can be in a single name space only

Virtual Networks

- building blocks:
    - virtual interfaces
    - software bridges (even programmable)
    - containers (network name spaces)
    - VMs
    - tunnels

Virtual Networks

Offloading to Hardware

- packet classification and switching
- match/action tables
    - tc supporting match/action (and queues)
- SR-IOV switch

Other Bottlenecks

- data copy
    - zero copy
    - need to ensure security
    - tx: packet can be changed while in flight
    - rx: uninitialized data after packet end
    - resources problem: mem reclaim on rx
    - needs tx checksum offloading

Other Bottlenecks

- sk_buff allocation
    - a lot of mm tricks depending on use case
    - for some cases sk_buff may not be needed at all (L2 switching)

Other Bottlenecks

- too many features
    - generic OS
    - usually only a subset of features is needed
    - XDP and eBPF

Conclusion

- complex topic
- fast moving
- is there interest in a deep dive?

Contact: jbenc@redhat.com (mailto:jbenc@redhat.com)