Linux Network Stack

To Hell and back
Sysadmin #8

Adrien Mahieux - Performance Engineer (nanosecond hunter)
gh: github.com/Saruspete
tw: @Saruspete
em: adrien.mahieux@gmail.com

Summary

A lot of text...

- What’s available atm ?
- Definitions
- How does it really work ?
- Monitoring
- Configuration
- What I’d like to have...

And a big graph !

- Receive Flow
- (Transmit Flow)

Disclaimer : this is not a presentation

Timeline....

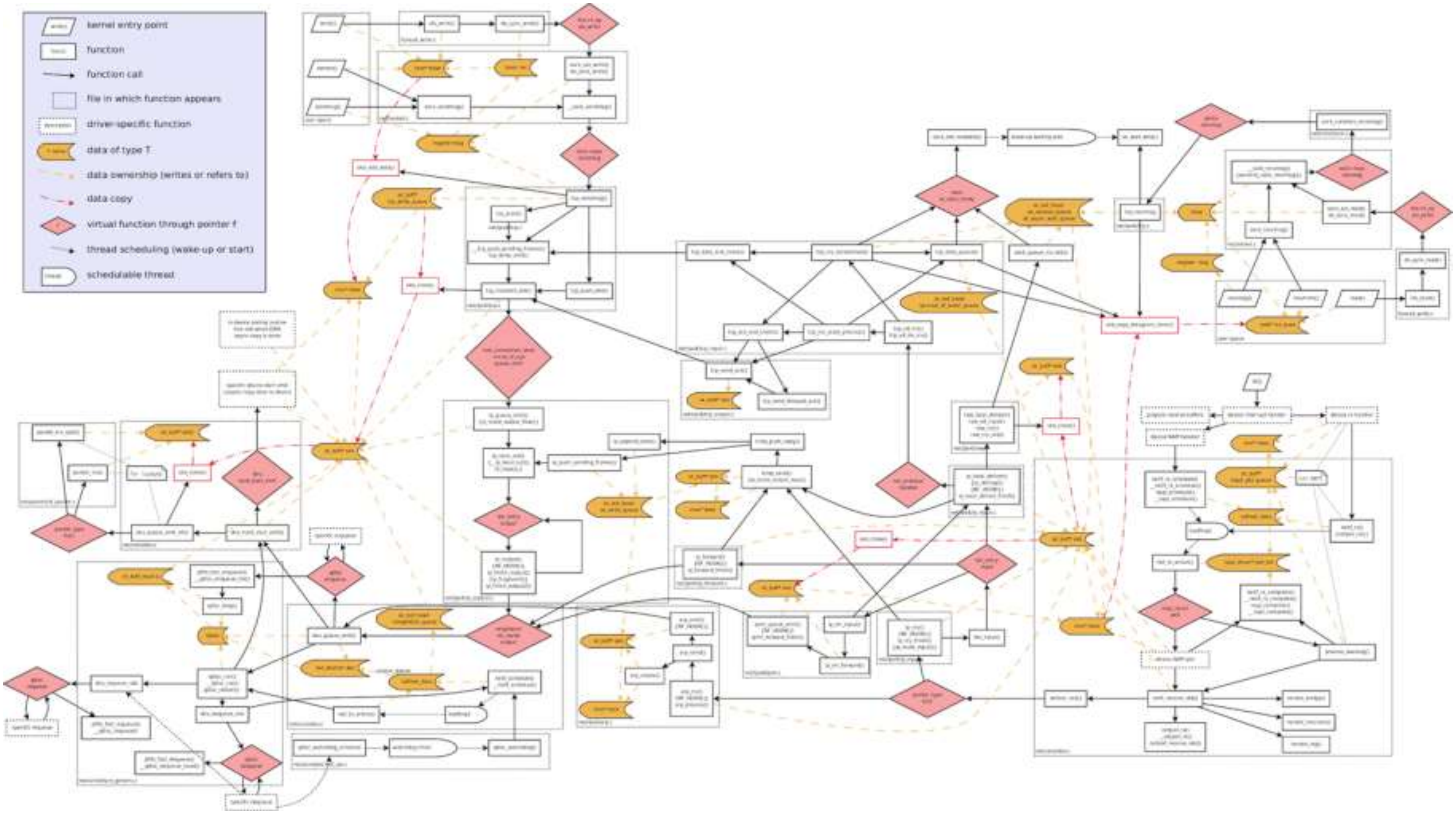
- Renchap proposed this talk
- Nice, should be interesting
- Should be easy
    - Indexed source with opengrok
    - Should be lots of comments...
    - I “know” globally how it works
    - (turns out, nope...)
- Let’s do it very complete !
    - 1 version for sysadmin, 1 for Kernel dev.
- Hey hardware interrupts... vectors ?
- Drivers ? NAPI ? DCA ? XDP ?
- WTF SO MUCH FUNC POINTERS
- Wait, sysadmindays are tomorrow ?!

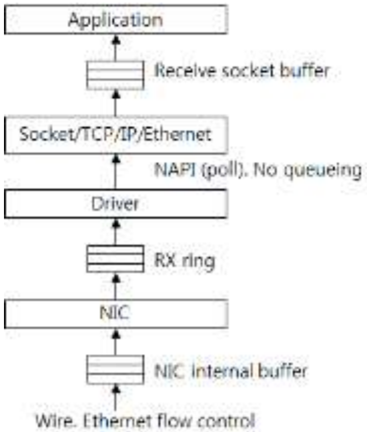
What’s available atm ?

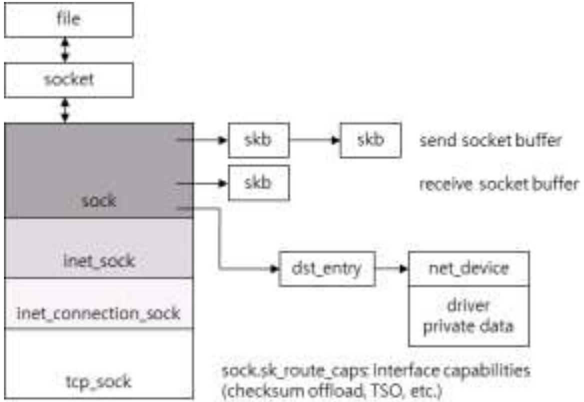
- Lots of tutorials
- Lots of LKML discussions
- Lots of (copy/pasted) code

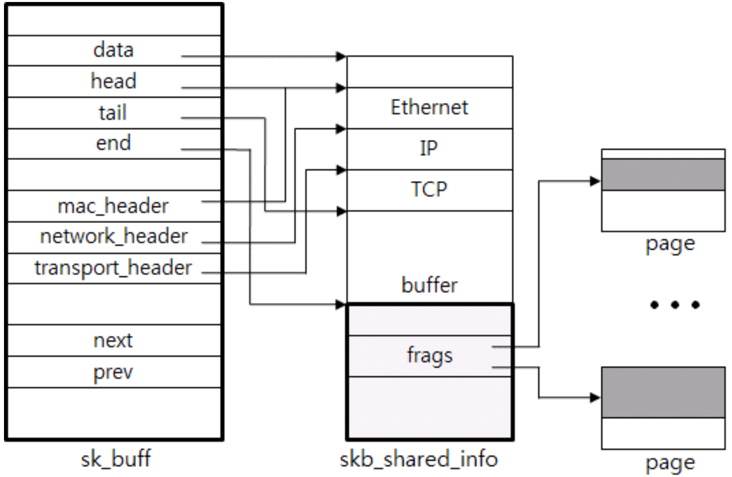
- Applications using black magic
- Robust and tested protocols

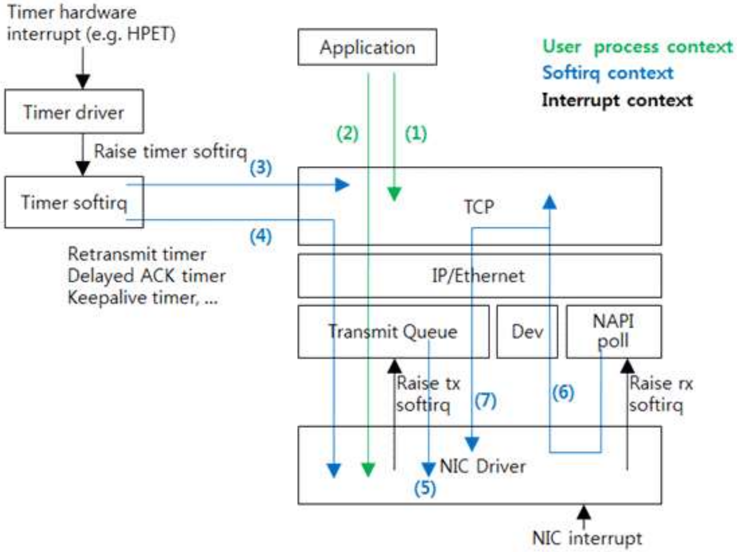
- New tools to explore
- Clear and detailed graphs…

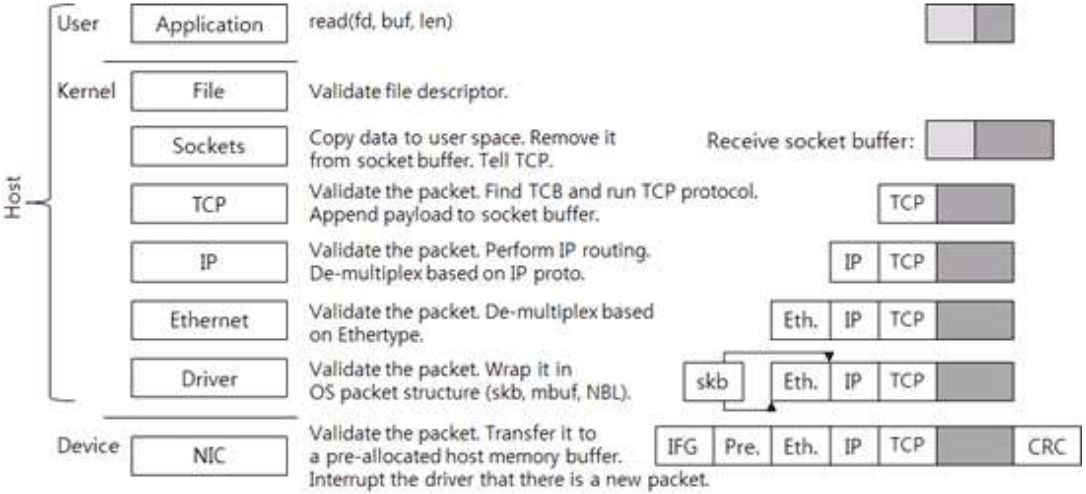

kernel entry point
function
function call
file in which function appears
driver-specific function
data of type T
data ownership (writes or refers to)
data copy
virtual function through pointer f
thread scheduling (wake-up or start)
schedulable thread


Application
Receive socket buffer
Socket/TCP/IP/Ethernet
NAPI (poll). No queueing
Driver
RX ring
NIC
NIC internal buffer
Wire. Ethernet flow control


file
socket
sock
inet_sock
inet_connection_sock
tcp_sock
skb
skb
send socket buffer
skb
receive socket buffer
dst_entry
net_device
driver
private data
sock.sk_route_caps: interface capabilities
(checksum offload, TSO, etc.)


data
head
tail
end
mac_header
network_header
transport_header
next
prev
sk_buff
Ethernet
IP
TCP
buffer
frags
skb_shared_info
page
page


Timer hardware
interrupt (e.g. HPET)
Timer driver
Raise timer softirq
Timer softirq
Retransmit timer
Delayed ACK timer
Keepalive timer, ...
Application
User process context
Softirq context
Interrupt context
(1)
(2)
(3)
(4)
(5)
(6)
(7)
TCP
IP/Ethernet
Transmit Queue
Dev
NAPI
poll
Raise tx
softirq
Raise rx
softirq
NIC Driver
NIC interrupt


Host
User
Application
read(fd, buf, len)
Kernel
File
Validate file descriptor.
Sockets
Copy data to user space. Remove it
from socket buffer. Tell TCP.
Receive socket buffer:
TCP
Validate the packet. Find TCB and run TCP protocol.
Append payload to socket buffer.
TCP
IP
Validate the packet. Perform IP routing.
De-multiplex based on IP proto.
IP TCP
Ethernet
Validate the packet. De-multiplex based
on Ethertype.
Eth. IP TCP
Driver
Validate the packet. Wrap it in
OS packet structure (skb, mbuf, NBL).
skb
Eth. IP TCP
Device
NIC
Validate the packet. Transfer it to
a pre-allocated host memory buffer.
Interrupt the driver that there is a new packet.
IFG Pre. Eth. IP TCP CRC

ftrace shows a nice stack

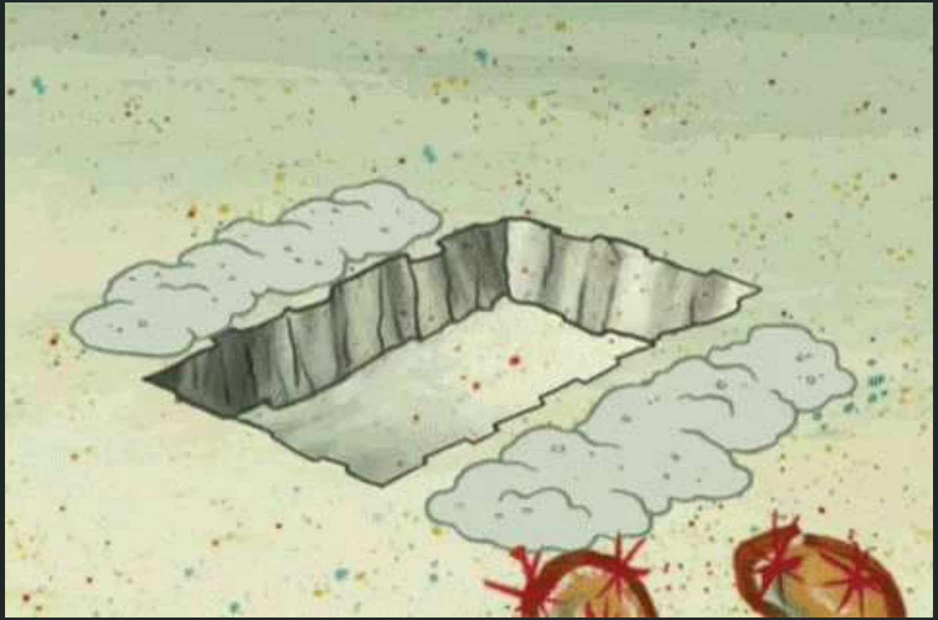
```
net_rx_action() {
  ixgbe_poll() {
    ixgbe_clean_tx_irq();
    ixgbe_clean_rx_irq() {
      ixgbe_fetch_rx_buffer() {
        ... // allocate buffer for packet
      } // returns the buffer containing packet data
      ... // housekeeping
      napi_gro_receive() {
        // generic receive offload
        dev_gro_receive() {
          inet_gro_receive() {
            udp4_gro_receive() {
              udp_gro_receive();
            }
          }
        }
      napi_skb_finish() {
        netif_receive_skb_internal() {
          __netif_receive_skb() {
            __netif_receive_skb_core() {
              ...
              ip_rcv() {
                ...
                ip_rcv_finish() {
                . . .
ip_local_deliver() {
  ip_local_deliver_finish() {
    raw_local_deliver();
    udp_rcv() {
      __udp4_lib_rcv() {
        __udp4_lib_mcast_deliver() {
          ...
          // clone skb & deliver
          flush_stack() {
            udp_queue_rcv_skb() {
              ... // data preparation
              // deliver UDP packet
              // check if buffer is full
              __udp_queue_rcv_skb() {
                // deliver to socket queue
                // check for delivery error
                sock_queue_rcv_skb() {
                  ...
                  _raw_spin_lock_irqsave();
                  // enqueue packet to socket buffer list
                  _raw_spin_unlock_irqrestore();
                  // wake up listeners
                  sock_def_readable() {
                    __wake_up_sync_key() {
      _raw_spin_lock_irqsave();
      __wake_up_common() {
        ep_poll_callback() {
          ...
          _raw_spin_unlock_irqrestore();
        }
      }
      _raw_spin_unlock_irqrestore();
    }
```

Plenty of comments !

```
 * struct net_device - The DEVICE structure.
 *
 * Actually, this whole structure is a big mistake.  It mixes I/O
 * data with strictly "high-level" data, and it has to know about
 * almost every data structure used in the INET module.

/* Accept zero addresses only to limited broadcast;
 * I even do not know to fix it or not. Waiting for complains :-)

/* An explanation is required here, I think.
 * Packet length and doff are validated by header prediction,

* Really tricky (and requiring careful tuning) part of algorithm
* is hidden in functions tcp_time_to_recover() and
tcp_xmit_retransmit_queue().

/* skb reference here is a bit tricky to get right, since
 * shifting can eat and free both this skb and the next,
 * so not even _safe variant of the loop is enough.

/* Here begins the tricky part :
 * We are called from release_sock() with :

/* This is TIME_WAIT assassination, in two flavors.
 * Oh well... nobody has a sufficient solution to this
 * protocol bug yet.

BUG(); /* "Please do not press this button again." */
```

```
/* The socket is already corked while preparing it. */
/* ... which is an evident application bug. --ANK */

/* Ugly, but we have no choice with this interface.
 * Duplicate old header, fix ihl, length etc.

* Parse and mangle SNMP message according to mapping.
* (And this is the fucking 'basic' method).


/* 2. Fixups made earlier cannot be right.
 *             If we do not estimate RTO correctly without them,
 *             all the algo is pure shit and should be replaced
 *             with correct one. It is exactly, which we pretend to do.

/* OK, ACK is valid, create big socket and
 * feed this segment to it. It will repeat all
 * the tests. THIS SEGMENT MUST MOVE SOCKET TO
 * ESTABLISHED STATE. If it will be dropped after
 * socket is created, wait for troubles.

 * packets force peer to delay ACKs and calculation is correct too.
 * The algorithm is adaptive and, provided we follow specs, it
 * NEVER underestimate RTT. BUT! If peer tries to make some clever
 * tricks sort of "quick acks" for time long enough to decrease RTT
 * to low value, and then abruptly stops to do it and starts to delay
 * ACKs, wait for troubles.
```

Let’s do it myself !



Definitions

- Intel’ “**ixgbe**” 10G driver + **MSI-x** mode + Kernel **4.14.68**
    - Well-tested in production
    - Fairly generic and available on many systems
    - 10G PCIe is now the default in high-workload servers
    - 10G drivers have multiple queues
    - MSIx is the current standard (nice for VMs)
    - Kernel 4.14 is the current LTS at time of analysis

Definitions (keep it handy)

NIC	Network Interface Controller
INTx	Legacy Interrupt
MSI	Message Signaled Interrupt
MSI-x	MSI extended
VF	Virtual Function
Coalescing	Delaying events to group them in one shot
RingBuffer	pre-defined area of memory for data
DMA	Direct Memory Access.
DCA	Direct Cache Access
XDP	Xtreme Data Path
BPF	Berkeley Packet Filter
RSS	Receive Side Scaling (multiqueue)
RPS	Receive Packet Steering (CPU dist)
RFS	Receive Flow Steering.
aRFS	Accelerated RFS.
XPS	Transmit Packet Steering
LRO	Large Receive Offload
GRO	Generic Receive Offload
GSO	Generic Segmentation Offload

Optimization	Hardware	Software
Distribute load of 1 NIC across multiple CPUs	RSS (multiple queues)	RPS (only 1 queue)
CPU locality between consumer app and net processing	aRFS	RFS
Select queue to transmit packets	XPS	
Combining data of “similar enough” packets	LRO	GRO
Push data in CPU cache	DCA	
Processing packets before reaching higher protocols	XDP + BPF	

How does it really work ?

It’s a stack of multiple layers

SOFTWARE	L7: socket, write
SYSCALL	libc: vfs write, sendmsg, sendmmsg
TCP/UDP	L4+: Session, congestion, state, membership, timeouts...
ROUTING	L3: IPv4/6 IP routing
DRIVER	L2: MAC addr, offloading, checksum...
DEVICE	L1: hardware, interrupts, statistics, duplex, ethtool...

Let’s get physical

Speed and duplex negotiation : The speed and duplex used by the NIC for transfers is determined by autonegotiation (required for 1000Base-T).
If only one device can do autoneg, it’ll determine and match the speed of the other, but assume half-duplex.

Autoneg is based on pulses to detect the presence of another device. Called Normal Link Pulses (NLP) they are generated every 16ms and required to be sent if no packet is being transmitted. If no packet and no NLP is being received for 50-150ms, a link failure is considered as detected.

HardIRQ / Interrupts

Level-triggered VS edge-triggered

- Edge-triggered Interrupts are generated one time upon event arrival by an edge (rising or falling electrical voltage). The event needs to happen again to generate a new interrupt.
  This is used to reflect an occurring event. In /proc/interrupts, this is shown as “IO-APIC-edge” or “PCI-MSI-edge”.
  Remember the time you configured your SoundBlaster to “IRQ 5” and “DMA 1” ? If multiple cards were on the same IRQ line, their signals collided and were lost.

- Level-sensitive keep generating interrupts until acknowledged in the programmable interrupt controller.
  This is mostly used to reflect a state. In /proc/interrupts, this is shown as “IO-APIC-level” or “IO-APIC-fasteoi” (end of interrupt).

Standard / fixed IRQs

- 0 - Timer
- 1 - PS2 Keyboard (i8042)
- 2 - cascade
- 3,4 - Serial
- 5 - (Modem)
- 6 - Floppy
- 7 - (Parallel port for printer)
- 8 - RealTime Clock
- 9 - (ACPI / sound card)
- 10 - (sound / network card)
- 11 - (Video Card)
- 12 - (PS2 Mouse)
- 13 - Math Co-Processor
- 14 - EIDE disk chain 1
- 15 - EIDE disk chain 2

SoftIRQ / Ksoftirqd

- In Realtime kernels, the hard IRQ is also managed by a kernel thread. This to allow priority management.
- While processing critical code path, interrupt handling can be disabled. But NMI and SMI cannot be disabled.
- Softirq process jobs in a specified order (kernel/softirq.c :: softirq_to_name[]).
- Most interrupts are defined per CPU. You'll find "ksoftirqd/X" where X is the CPU-ID

Netfilter Hooks

```
 static inline int nf_hook(... )  {
             struct nf_hook_entries *hook_head;

#ifdef HAVE_JUMP_LABEL
             if (__builtin_constant_p(pf) &&
                 __builtin_constant_p(hook) &&
                 !static_key_false(&nf_hooks_needed[pf][hook]))
                          return 1;
#endif

             rcu_read_lock();
             hook_head = rcu_dereference(net-
>nf.hooks[pf][hook]);
             if (hook_head) {
                          struct nf_hook_state state;

                          nf_hook_state_init(&state, hook, pf,
indev, outdev,
                                                         sk, net,
okfn);

                          ret = nf_hook_slow(skb, &state,
hook_head, 0);
             }
             rcu_read_unlock();

             return ret;
}
```

Hook overhead is considered low, as no processing is done if no rule is loaded.
Even lower overhead if using JUMP_LABELS.

CC_HAVE_ASM_GOTO && CONFIG_JUMP_LABEL
⇒ # define HAVE_JUMP_LABEL

The recurring pattern for function & callback using NF_HOOK
:
`funcname ⇒ NF_HOOK ⇒ funcname_finish`

Monitoring

ethtool

ethtool manages the NIC internals: PHY, firmware, counters, indirection table, ntuple table, ring-buffer, WoL...
It's the driver duty to provide these features through "`struct ethtool_opts`"

ethtool -i $iface : details on the iface (kmod, firmware, version...)

ethtool -c $iface : Interrupt coalescing values

ethtool -k $iface : Features (pkt-rate, GRO, TSO...)

ethtool -g $iface : Show the ring-buffer sizes

ethtool -n $iface : Show ntuple configuration table

ethtool -S $iface : Show NIC internal statistics (driver dependant)

ethtool -S | --statistics

ethtool -S $iface : shows custom interface statistics. No standard,

```
rx_packets: 71938676
tx_packets: 69504119
rx_bytes: 42215356407
tx_bytes: 52355860539
rx_broadcast: 675957
tx_broadcast: 346481
rx_multicast: 1245202
tx_multicast: 2847
multicast: 1245202
collisions: 0
rx_crc_errors: 0
rx_no_buffer_count: 0
rx_missed_errors: 0
tx_aborted_errors: 0
tx_carrier_errors: 0
tx_window_errors: 0
tx_abort_late_coll: 0
tx_deferred_ok: 0
tx_single_coll_ok: 0
tx_multi_coll_ok: 0
```

```
tx_timeout_count: 0
rx_long_length_errors: 0
rx_short_length_errors: 0
rx_align_errors: 0
tx_tcp_seg_good: 4266569
tx_tcp_seg_failed: 0
rx_flow_control_xon: 0
rx_flow_control_xoff: 0
tx_flow_control_xon: 0
tx_flow_control_xoff: 0
rx_long_byte_count: 42215356407
tx_dma_out_of_sync: 0
tx_smbus: 12468
rx_smbus: 479
dropped_smbus: 0
os2bmc_rx_by_bmc: 336860
os2bmc_tx_by_bmc: 12468
os2bmc_tx_by_host: 336860
os2bmc_rx_by_host: 12468
tx_hwtstamp_timeouts: 0
tx_hwtstamp_skipped: 0
rx_hwtstamp_cleared: 0
rx_errors: 0
tx_errors: 0
tx_dropped: 0
rx_length_errors: 0
rx_over_errors: 0
rx_frame_errors: 0
rx_fifo_errors: 0
tx_fifo_errors: 0
tx_heartbeat_errors: 0
```

```
tx_queue_0_packets: 15742031
tx_queue_0_bytes: 13149125228
tx_queue_0_restart: 0
tx_queue_1_packets: 14280094
tx_queue_1_bytes: 6326351422
tx_queue_1_restart: 0
tx_queue_2_packets: 33093890
tx_queue_2_bytes: 29099316424
tx_queue_2_restart: 0
tx_queue_3_packets: 6375636
tx_queue_3_bytes: 3191645669
tx_queue_3_restart: 0
rx_queue_0_packets: 32746660
rx_queue_0_bytes: 32200636085
rx_queue_0_drops: 0
rx_queue_0_csum_err: 0
rx_queue_0_alloc_failed: 0
rx_queue_1_packets: 12761541
rx_queue_1_bytes: 2017306804
rx_queue_1_drops: 0
rx_queue_1_csum_err: 0
rx_queue_1_alloc_failed: 0
rx_queue_2_packets: 17801690
rx_queue_2_bytes: 4863010445
rx_queue_2_drops: 0
rx_queue_2_csum_err: 0
rx_queue_2_alloc_failed: 0
rx_queue_3_packets: 8640774
rx_queue_3_bytes: 2566051868
rx_queue_3_drops: 0
rx_queue_3_csum_err: 0
rx_queue_3_alloc_failed: 0
```

ethtool -x | --show-rxfh-indir

```
ethtool -x enp5s0
RX flow hash indirection table for enp5s0 with 4 RX ring(s):
    0:      0     0     0     0     0     0     0     0
    8:      0     0     0     0     0     0     0     0
   16:      0     0     0     0     0     0     0     0
   24:      0     0     0     0     0     0     0     0
   32:      1     1     1     1     1     1     1     1
   40:      1     1     1     1     1     1     1     1
   48:      1     1     1     1     1     1     1     1
   56:      1     1     1     1     1     1     1     1
   64:      2     2     2     2     2     2     2     2
   72:      2     2     2     2     2     2     2     2
   80:      2     2     2     2     2     2     2     2
   88:      2     2     2     2     2     2     2     2
   96:      3     3     3     3     3     3     3     3
  104:      3     3     3     3     3     3     3     3
  112:      3     3     3     3     3     3     3     3
  120:      3     3     3     3     3     3     3     3
RSS hash key:
```

procfs

/proc/net contains a lot of statistics on many parts of the network… but most of them don't have description

- netlink
- netstat Protocols stats
- ptype Registered transports
- snmp Multiple proto details
- softnet_stat NAPI polling

/proc/interrupts : show CPU / interruptions matrix table

/proc/irq/$irq/smp_affinity : see & set CPU masks for interruption processing

sysfs

/sys/class/net/$iface/ : Details on the interface status (duplex, speed, mtu..)

/sys/class/net/$iface/statistics/ :

/sys/class/net/$iface/queues/[rt]x-[0-9]/ : RPS configuration and display

Configuration

ethtool

All parameters showing something, also allow to set the same category of parameters with an uppercase letter.

ethtool bonus - Built-in firewall

```
ethtool -N eth4 flow-type udp4 src-ip 10.10.10.100 m 255.255.255.0 action -1

ethtool -n eth4
4 RX rings available
Total 1 rules

Filter: 8189
    Rule Type: UDP over IPv4
    Src IP addr: 0.0.0.100 mask: 255.255.255.0
    Dest IP addr: 0.0.0.0 mask: 255.255.255.255
    TOS: 0x0 mask: 0xff
    Src port: 0 mask: 0xffff
    Dest port: 0 mask: 0xffff
    VLAN EtherType: 0x0 mask: 0xffff
    VLAN: 0x0 mask: 0xffff
    User-defined: 0x0 mask: 0xffffffffffffffff
    Action: Drop
```

netlink

Special socket type for communicating with network stack

Used by iproute2 (ip), and any tool that want to configure it

Also used to monitor events (ip monitor, nltrace)

sysctls

Tweaks most values of the network stack

- Polling budget
- TCP features & timeouts
- Buffer sizes
- Forwarding
- Reverse Path filtering
- Routing Table cache size

sysfs

/sys/class/net/$iface/

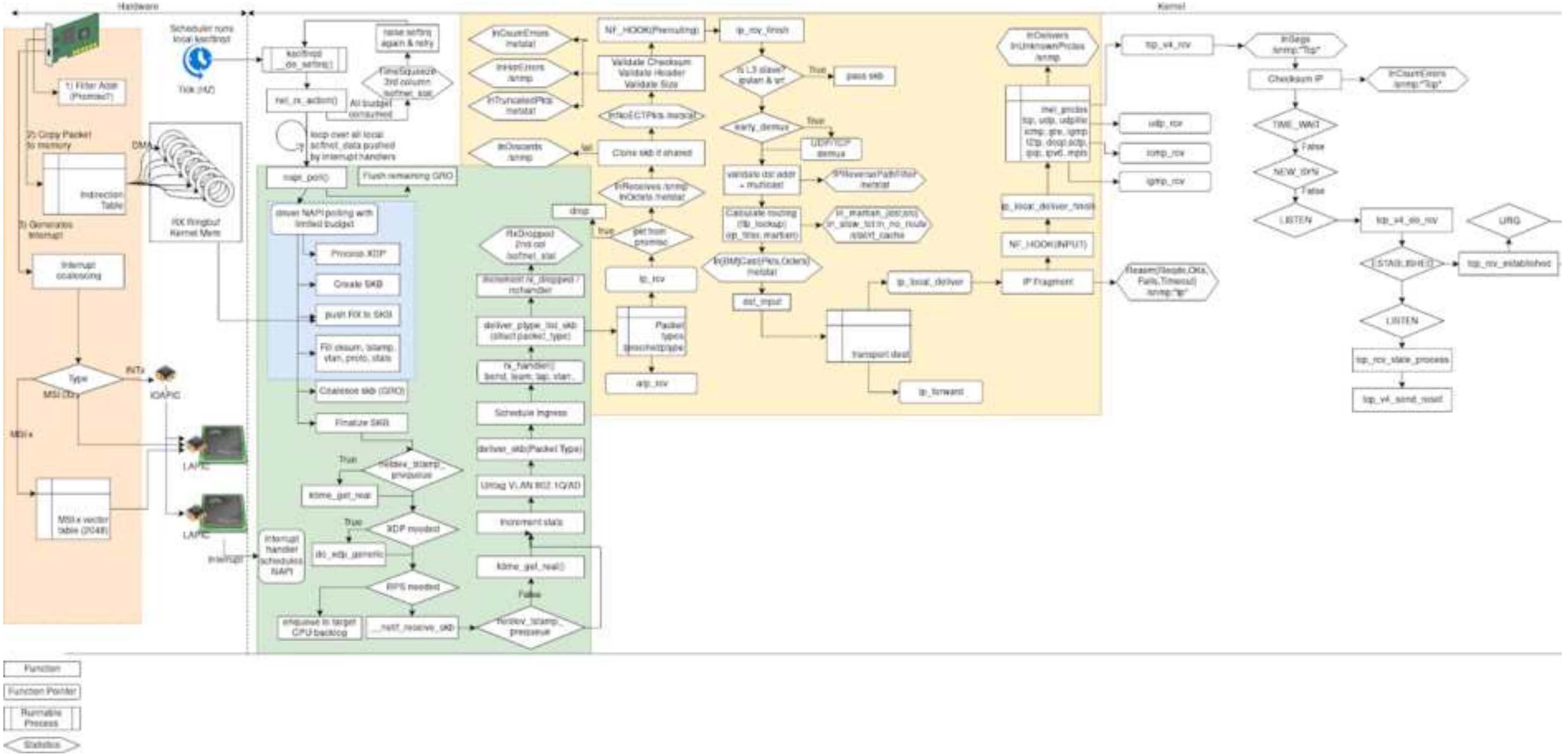
/statistics : file-based access to standard counters

/queues/tx-* : show and configure XPS + rate limits

/queues/rx-* : show and configure RPS

Under the hood

Global schema... in progress



Global schema... please contribute

That's a lot of work ! I need your help...

- Complete the schema to map the most used protocols
- Add more details to the documentation
- Make the schema more interactive to have a live view from the OS
- Reference other schemas and map their details

Documents and analysis : https://github.com/saruspete/LinuxNetworking

Source Code browser : https://dev.mahieux.net/kernelgrok/

Schema made with: https://draw.io

Thank you

Bibliography

https://blog.packagecloud.io/eng/2016/06/22/monitoring-tuning-linux-networking-stack-receiving-data/
https://blog.packagecloud.io/eng/2017/02/06/monitoring-tuning-linux-networking-stack-sending-data/
https://blog.packagecloud.io/eng/2016/10/11/monitoring-tuning-linux-networking-stack-receiving-data-illustrated/
https://blog.cloudflare.com/how-to-receive-a-million-packets/
https://blog.cloudflare.com/how-to-drop-10-million-packets/
https://blog.cloudflare.com/the-story-of-one-latency-spike/
https://wiki.linuxfoundation.org/networking/kernel_flow
https://www.lmax.com/blog/staff-blogs/2016/05/06/navigating-linux-kernel-network-stack-receive-path/
http://www.jauu.net/talks/data/runtime-analyse-of-linux-network-stack.pdf
https://www.xilinx.com/Attachment/Xilinx_Answer_58495_PCIe_Interrupt_Debugging_Guide.pdf
https://www.cubrid.org/blog/understanding-tcp-ip-network-stack
https://events.static.linuxfound.org/sites/events/files/slides/Chaiken_ELCE2016.pdf