Linux Network Receive Stack

Monitoring and Tuning Deep Dive

MVHLUG Monthly Meeting April 2017

Patrick Ladd
Technical Account Manager
Red Hat
pladd@redhat.com

Slides available at http://people.redhat.com/pladd

#whatisTAM?

- Premium named-resource support
- Proactive and early access
- Regular calls and on-site engagements
- Customer advocate within Red Hat and upstream
- Multi-vendor support coordinator
- High-touch access to engineering
- Influence for software enhancements
- NOT Hands-on or consulting

Disclaimers

- This presentation is the result of some research I got into in the last year
- Information is distilled from several sources, including
  - https://access.redhat.com/articles/1391433
  - https://blog.packagecloud.io/
  - https://www.privateinternetaccess.com/blog/author/piaresearch/
  - https://www.kernel.org/doc/Documentation/networking/scaling.txt
  - Linux Kernel Development (2^{nd} Edition) – Robert Love
- I am not a kernel developer – my answer to some (many?) questions may be "I am not a kernel developer"

Strategy

How to approach this

Deep Dive Into the Kernel

"Use the source Luke"

How to Monitor

"What's Happening"

What to change

"Knobs"

“Under the Hood”

Overview

Path of a received packet

Protocol layers process and deliver to socket queues

‘skb’ structures passed up to network layer

ksoftirqd “bottom half”

Hardware Interrupt “top half”

Hardware Interrupt

DMA copy to Ring Buffer

Packet arrival at NIC

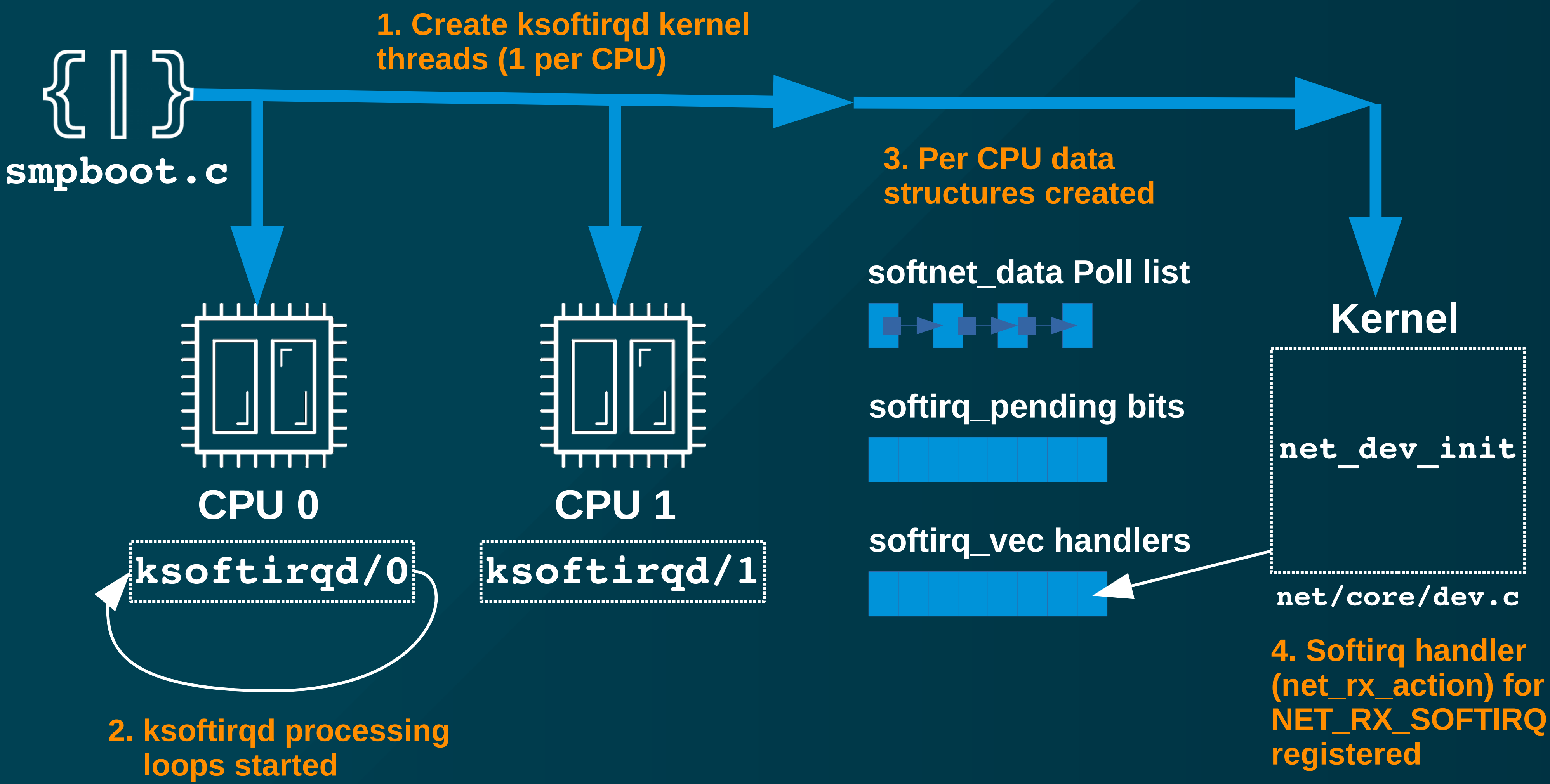
PCI Initialization

- PCI devices are identified by registers in PCI configuration space
- Device drivers are compiled with a list of PCI device IDs that they can control (`MODULE_DEVICE_TABLE`)
- The kernel uses these tables to determine which device drivers to load
  - Use '`lspci -nn`' to find your device
  - Find PCI vendor and device ID
  - Look in `/lib/modules/`uname -r`/`
    - `modules.pcimap` (RHEL6 and earlier)
    - `modules.alias` (RHEL7 and later)
  - `egrep -i {vid}.*{did} /lib/modules/`uname -r`/modules.alias`
- PCI probe functions of the device drivers are called to set up devices

PCI Probe Tasks (typical)

- Enable the device
- Request memory range & I/O ports
- Set DMA mask
- Register `ethtool` functions supported by driver
- Watchdog task setup
- `net_device_ops` structure setup
  - Function pointers for opening, sending data, setting MAC, etc.
- `net_device` struct creation

softirq Subsystem Initialization



Network Device Initialization

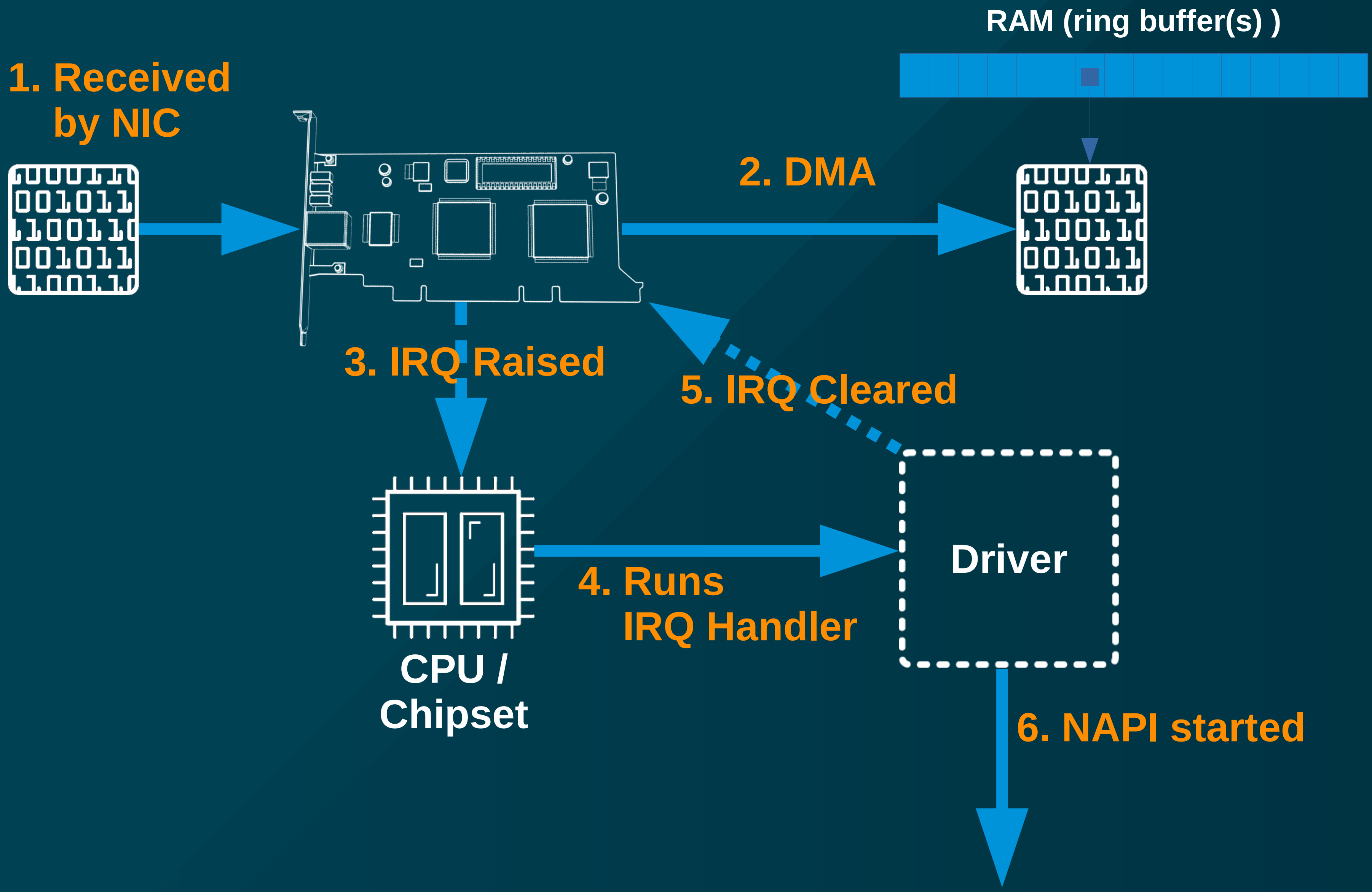
- `net_device_ops` Data Structure
  - Function pointers to driver implementation of function

```
static const struct net_device_ops igb_netdev_ops = {
  .ndo_open               = igb_open,
  .ndo_stop               = igb_close,
  .ndo_start_xmit         = igb_xmit_frame,
  .ndo_get_stats64        = igb_get_stats64,
  .ndo_set_rx_mode        = igb_set_rx_mode,
  .ndo_set_mac_address    = igb_set_mac,
  .ndo_change_mtu         = igb_change_mtu,
  .ndo_do_ioctl           = igb_ioctl,
```

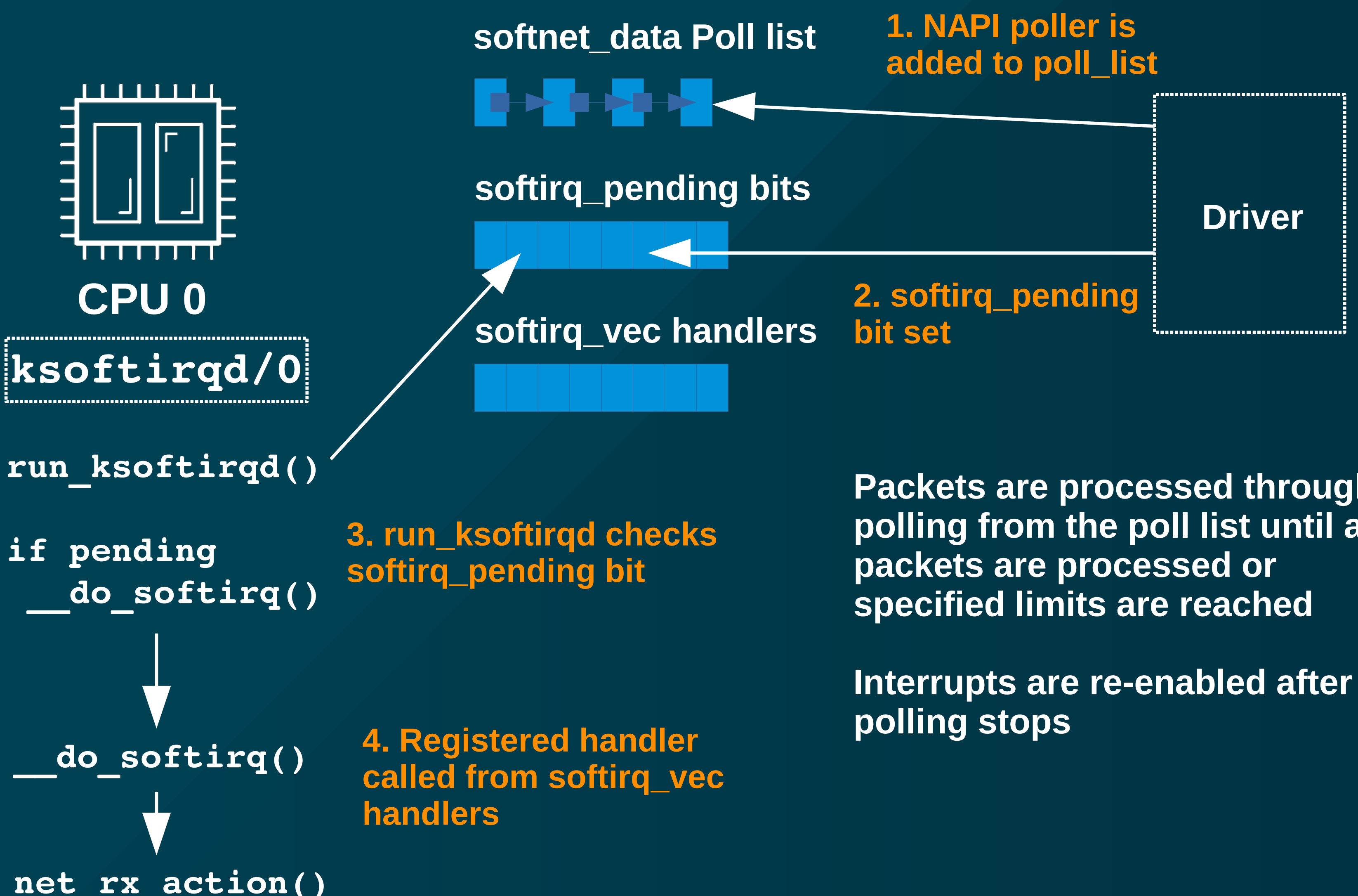
- `ethtool_ops` Data Structure

```
static const struct ethtool_ops igb_ethtool_ops = {
  .get_settings          = igb_get_settings,
  .set_settings          = igb_set_settings,
  .get_drvinfo           = igb_get_drvinfo,
  .get_regs_len          = igb_get_regs_len,
  .get_regs              = igb_get_regs,
```

NIC Data Processing “Top Half”



NAPI (New API) Processing



Packets are processed through polling from the poll list until all packets are processed or specified limits are reached

Interrupts are re-enabled after polling stops

NAPI Advantages

- Reduced interrupt load
  - Without NAPI: 1 interrupt per packet → high CPU load
  - With NAPI: polling during high packet arrival times
- No work to drop packets if kernel is too busy
  - Ring buffer overwrite by NIC
- Device drivers have been re-written to support and enable NAPI by default

Multiqueue / RSS (Receive Side Scaling)

- NIC with Multiple Send/Receive Queues
  - Explore with "`ethtool -l {ifname}`"
  - Modify with "`ethtool -L {ifname} {parm} {value}`"
  - Each has it's own interrupt
    - Used to distribute queue among multiple CPUs
    - Examine `/proc/interrupts` for details
    - Manual steering or dynamic
    - Some systems run `irqbalance` daemon
  - Distribution
    - Typically a fixed hash function of header data (IP addr & port are common)
    - Some NICs support programmable hashes "n-tuple" (`ethtool –config-ntuple`)

Sample RSS ethtool output

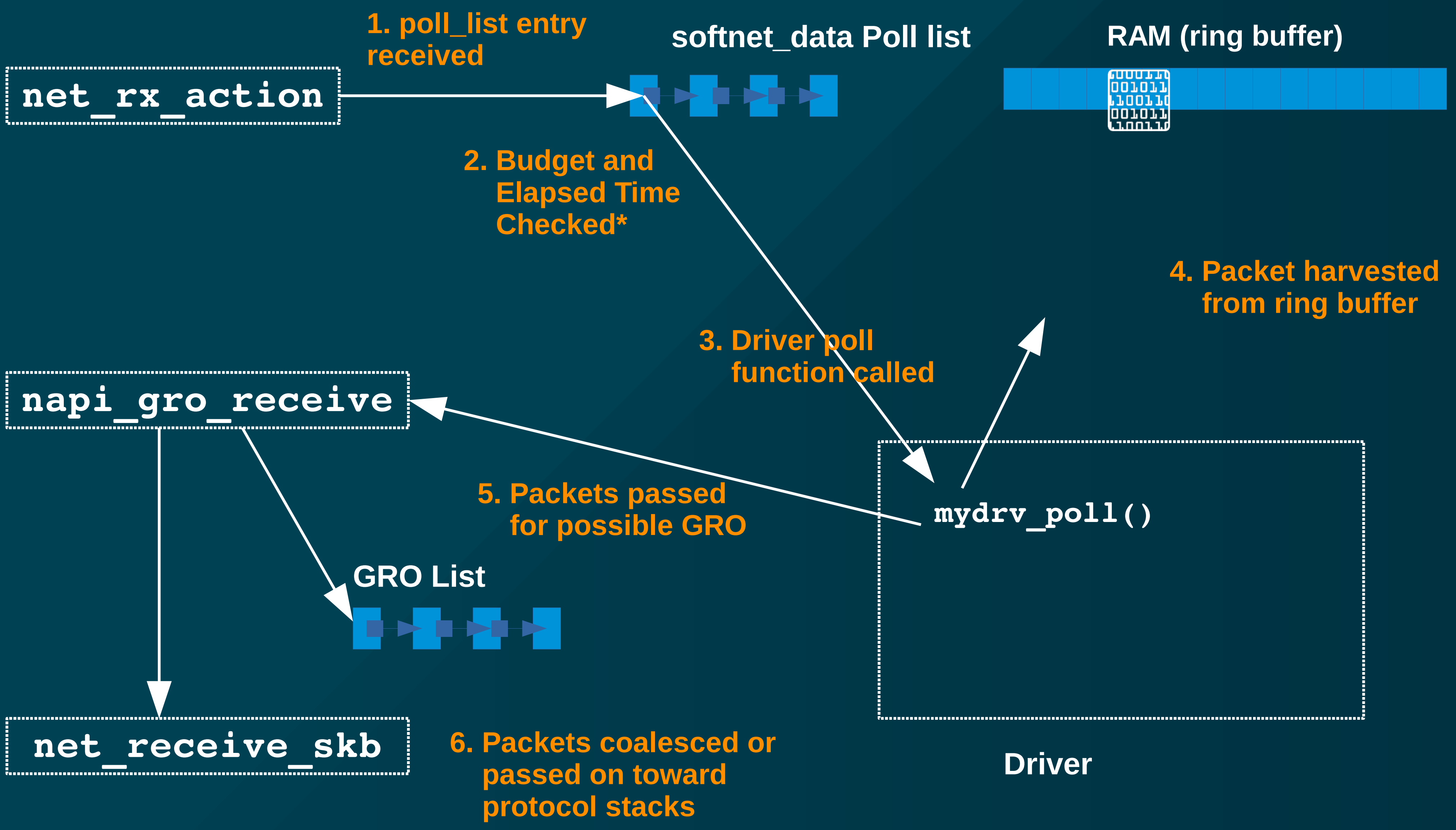
```
# ethtool -l eth0
Channel parameters for eth0:
Pre-set maximums:
RX:    0
TX:    0
Other:     0
Combined: 8
Current hardware settings:
RX:    0
TX:    0
Other:     0
Combined: 4


# ethtool -l eth0
Channel parameters for eth0:
Cannot get device channel parameters
: Operation not supported
```

Multiqueue / RSS (Receive Side Scaling)

- Recommendations:
  - Enable for latency concerns or when interrupt bottlenecks form
  - Lowest latency:
    - 1 queue per CPU or max supported by NIC
  - Best efficiency:
    - Smallest number with no overflows due to CPU saturation
- Aggressive techniques:
  - Lock IRQ & userspace process to CPU
  - Custom n-tuple setups (i.e. “all TCP/80 to CPU1)

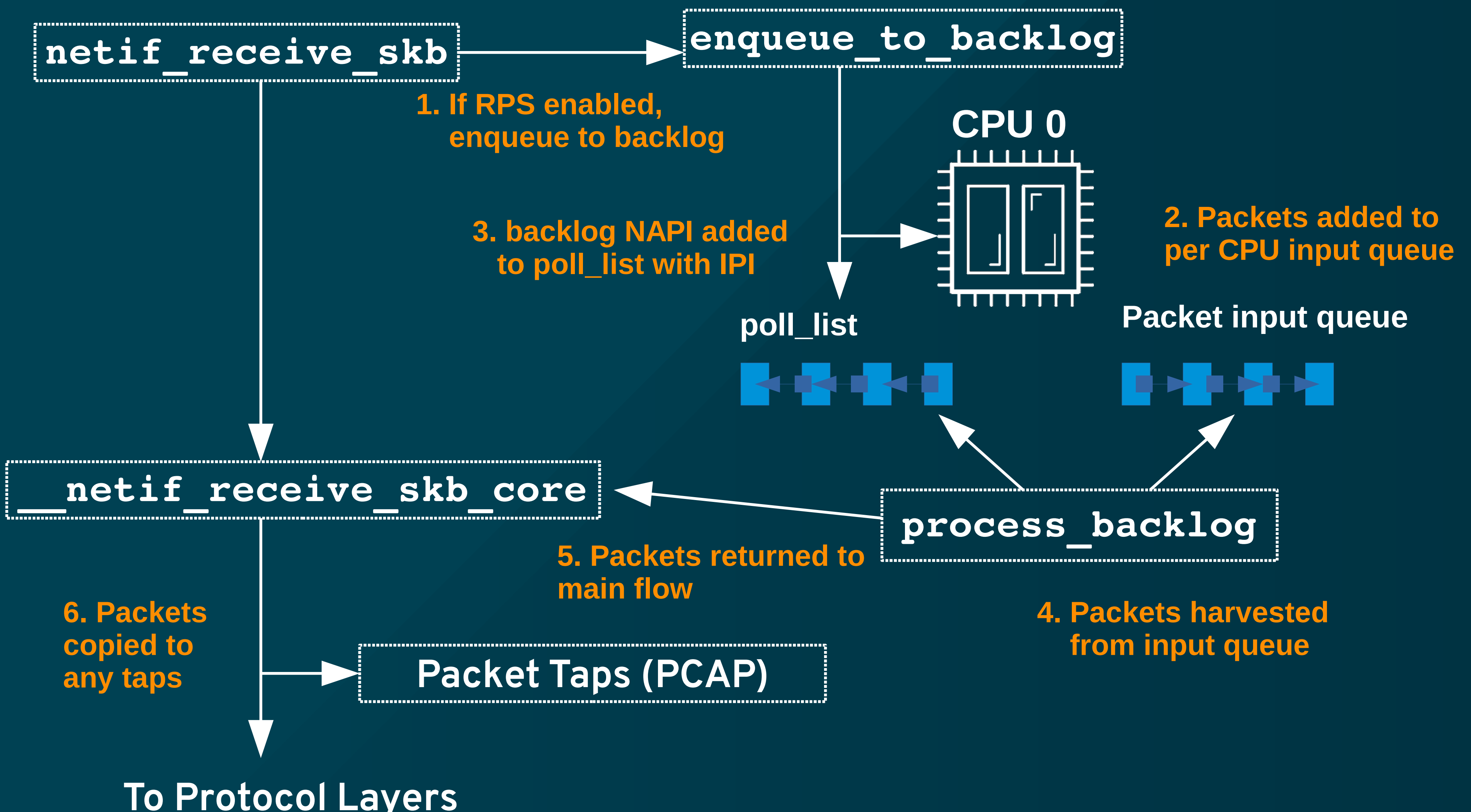
Network Data Processing “Bottom Half”



NAPI Exit

- Exits when:
  - No more NAPI poll structures to process
  - netdev_budget Exceeded
    - Each driver hardcoded budget for one NAPI structure of 64
    - Default is 300
    - → Approximately 5 driver poll calls
  - softirq Time Window Exceeded
    - 2 “jiffies”
- If no structures remain, re-enable IRQ interrupt

Network Data Processing (Continued)



Monitoring

Monitoring

- **`ethtool -S {ifname}`** – Direct NIC level Statistics
  - Hard to use – no standards, variation between drivers or even different releases of same driver
  - May have to resort to reading the driver source or NIC datasheet to determine true meaning
- **`/sys/class/net/{ifname}/statistics/`** – Kernel Statistics
  - Slightly higher level
  - Still some ambiguity in what values are incremented when
  - May need to read source to get exact meanings
- **`/proc/net/dev`** – Kernel Device Statistics
  - Subset of statistics from above for all interfaces
  - Same caveats as above

Monitoring

- Monitoring SoftIRQs
  - `watch -n1 grep RX /proc/softirqs`
- Packets dropped by the kernel: `dropwatch`

```
# dropwatch -l kas
Initalizing kallsyms db
dropwatch> start
Enabling monitoring…
Kernel monitoring activated.
Issue Ctrl-C to stop monitoring
1 drops at skb_queue_purge+18 (0xffffffff8151a968)
41 drops at __brk_limit+1e6c5938 (0xffffffffa0a1d938)
1 drops at skb_release_data+eb (0xffffffff8151a80b)
2 drops at nf_hook_slow+f3 (0xffffffff8155d083)
```

Finding the Bottleneck

- Drops at NIC level:
  - ethtool -S {ifname}

```
rx_errors: 0
tx_errors: 0
rx_dropped: 0
tx_dropped: 0
rx_length_errors: 0
rx_over_errors: 3295
rx_crc_errors: 0
rx_frame_errors: 0
rx_fifo_errors: 3295
rx_missed_errors: 3295
```

Finding the Bottleneck

- IRQs out of balance
  - egrep “CPU0|{ifname}” /proc/interrupts

```
           CPU0    CPU1    CPU2    CPU3    CPU4      CPU5
105: 1430000       0       0       0       0         0  IR-PCI-MSI-edge   eth2-rx-0
106: 1200000       0       0       0       0         0  IR-PCI-MSI-edge   eth2-rx-1
107: 1399999       0       0       0       0         0  IR-PCI-MSI-edge   eth2-rx-2
108: 1350000       0       0       0       0         0  IR-PCI-MSI-edge   eth2-rx-3
109:   80000       0       0       0       0         0  IR-PCI-MSI-edge   eth2-tx
```

- Check irqbalance service or manual IRQ settings

Finding the Bottleneck

- Insufficient `netdev_budget` for traffic
  - `cat /proc/net/softnet_stat`

    ```
    0073d76b 00000000 [illegible] 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    000000d2 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    0000015c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
    ```
  - Rows are by CPU
    - 1st column: number of frames received by the interrupt handler
    - 2nd column: number of frames dropped due to netdev_max_backlog being exceeded
    - 3rd column: number of times ksoftirqd ran out of netdev_budget or CPU time when there was still work to be done
- Overall system load – overloaded CPU not spending enough time processing SoftIRQs

General Tuning

Tuned

- Profile driven adaptive tuning daemon
  - Install

```
# yum install tuned
# systemctl enable tuned
# systemctl start tuned
```

  - Examine profiles (or look in `/etc/tune-profiles`)

```
# tuned-adm list
Available profiles:
- throughput-performance
- default
- desktop-powersave
- enterprise-storage
...
```

  - Activate a profile

```
# tuned-adm profile throughput-performance
Switching to profile 'throughput-performance'
...
```

Numad

- Intelligently move processes and memory among NUMA domains
  - Activate
    ```
    # systemctl enable numad
    # systemctl start numad
    ```
  - For more information
    ```
    # man numad
    ```

Hardware Tuning

HowTo: Persist `ethtool` settings

- For all techniques: https://access.redhat.com/solutions/2127401
- RHEL 5,6,7 without NetworkManager

  In `/etc/sysconfig/network-scripts/ifcfg`

  `ETHTOOL_OPTS=“-G ${ifname} {parm} {value}”`
- RHEL 6,7 with NetworkManager
  - Network manager dispatcher script (https://access.redhat.com/solutions/2841131)
  - In `/etc/NetworkManager/dispatcher.d/`

```
#!/bin/bash
if [ "$1" = "eth0" ] && [ "$2" = "up" ]; then
    ethtool -K "$1" rx off gro off lro off
fi
```

- ifup-local or udev rules

HowTo: Persist Kernel Tunables

- https://access.redhat.com/solutions/2587
- Runtime:
  - `sysctl  -w {parm}={value}`
  - `echo {value} > /proc/sys/{parmtree…}/{parm}`
- Persistent
  - RHEL7:
    - Add `{myname}.conf` file in `/etc/sysctl.d/`
  - Prior to RHEL7:
    - Insert or update parameter in `/etc/sysctl.conf`

Adapter Buffer Sizes

- Customize the size of RX ring buffer(s)
  - "`ethtool –g {ifname}`" to View
    - `# ethtool -g eth3`
      ```
      Ring parameters for eth3:
      Pre-set maximums:
      RX:       8192
      RX Mini:      0
      RX Jumbo:     0
      TX:       8192
      Current hardware settings:
      RX:       1024
      RX Mini:      0
      RX Jumbo:     0
      TX:       512
      ```
  - "`ethtool –G {ifname} [rx N] [rx-mini N] [rx-jumbo N] [tx N]`" to Alter

Backlog Queue (2nd column of softnet_stat)

- Increase the `netdev_max_backlog`
  - May need increase for multiple 1GB adapters or single 10GB
  - Double, if rate decreases, double and test again. Repeat until optimum size found.
  - `sysctl net.netdev_max_backlog`
    `netdev_max_backlog=1000`
  - `sysctl -w net.core.netdev_max_backlog=2000`

SoftIRQ time (3rd column of softnet_stat)

- Increase the `netdev_budget`
  - Seldom needed on 1GB adapters, 10GB and above may need
  - `sysctl net.core.netdev_budget`
    `net.core.netdev_budget=300`
  - `sysctl -w net.core.netdev_budget=600`

Interrupt Coalesce (IC)

- Modern NICs support collecting packets together before issuing interrupt
  - “`ethtool –c {ifname}`” to View

```
# ethtool -c eth3
Coalesce parameters for eth3:
Adaptive RX: on  TX: off
stats-block-usecs: 0
sample-interval: 0
pkt-rate-low: 400000
pkt-rate-high: 450000

rx-usecs: 16
rx-frames: 44
rx-usecs-irq: 0
rx-frames-irq: 0
```

  - “`ethtool –G {ifname} {parm} {value}`” to Alter

Adapter Offloading

- NIC Hardware Assist processing some protocol features
  - GRO: Generic Receive Offload
  - LRO: Large Receive Offload
  - TSO: TCP Segmentation Offload
  - RX check-summing = Processing of receive data integrity
  - "`ethtool –k {ifname}`" to View

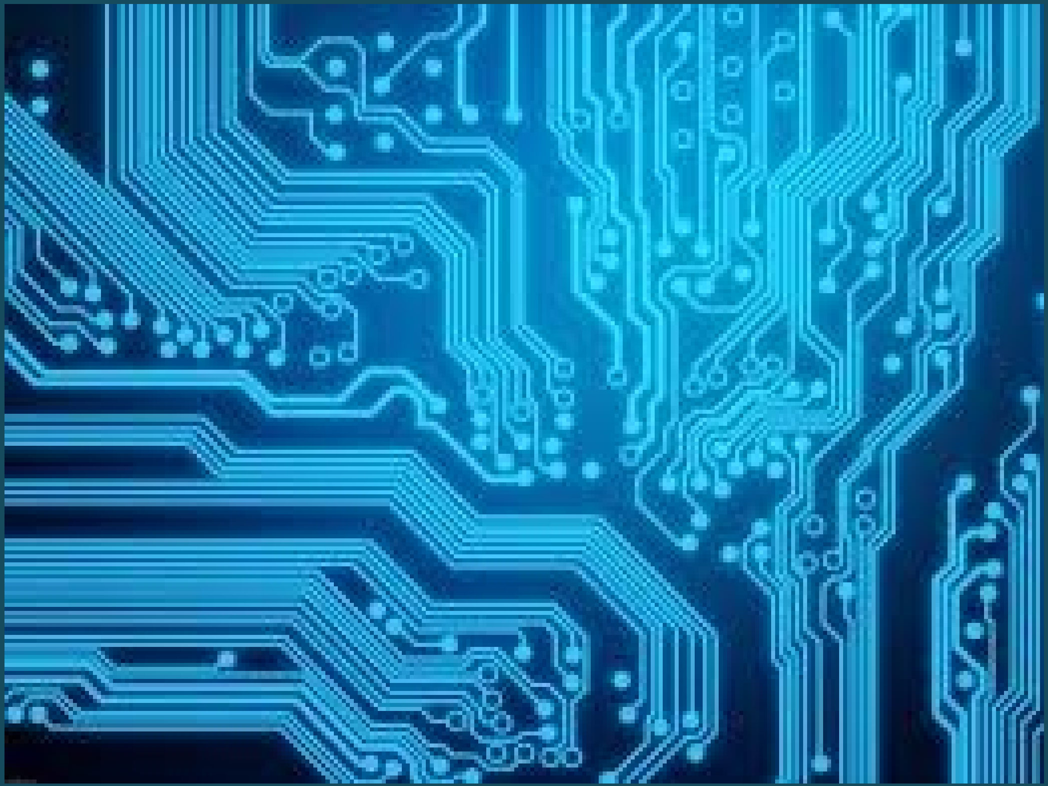
```
Features for eth0:
rx-checksumming: on
tx-checksumming: on
scatter-gather: on
tcp-segmentation-offload: on
udp-fragmentation-offload: off
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: on
rx-vlan-offload: on
tx-vlan-offload: on
ntuple-filters: off
receive-hashing: on
```

  - "`ethtool –K {ifname} {parm} {value}`" to Alter

Module Parameters

- Other special settings for your NIC hardware
  - Identify driver with “`lsmod`”
  - “`modinfo {driver_module}`” to View

Software Tuning



GRO (Generic Receive Offload)

- Combine “similar” packets into larger packets
  - Implemented in software
  - LRO has some issues – information loss
  - GRO is more restrictive
- See stack location on “Network Processing Bottom Half” slide

RPS (Receive Packet Steering)

- "RSS in software"
  - Routes packets to particular CPUs based on hash
- Advantages over RSS
  - Usable with any NIC
  - Easier to add custom filters
  - Does not increase HW interrupt rate
- Configuration:
  - Bitmap in `/sys/class/net/{ifname}/queues/rx-{n}/rps_cpus`

RPS (Receive Packet Steering)

- Recommendations:
  - Set rps_cpus to CPUs in same NUMA domain as interrupting CPU
  - May be redundant if RSS is enabled
    - If much larger number of hardware CPUs than queues, RSS for CPUs in same NUMA domain
  - If packet flows are non-uniform, CPU load imbalance could be a problem
    - Investigate flow limits if this occurs

RFS (Receive Flow Steering)

- https://access.redhat.com/solutions/62885
- Steer packets to CPU processing application is running on
- Increase CPU cache hit rate by improving locality of reference
  - Configure
    - `/sys/class/net/{ifname}/queues/rps_cpus`
    - `/sys/class/net/{ifname}/queues/rps_flow_count`
  - `sysctl -w net.core.rps_sock_flow_entries=32768`

Slides available at http://people.redhat.com/pladd

THANK YOU

Patrick Ladd
Technical Account Manager
Red Hat
pladd@redhat.com