Open Talk

API 网关与高性能服务最佳实践
HANGZHO 2019

性能优化：接收数据包

杨鹏 / 又拍云系统开发高级工程师

主办方：

主题

- 基于 linux 内核，机器是如何接受数据包的？
- 如何针对性的进行监控和调优？

基本原则

- 在网络栈的各个层次监控 丢包率
- 最优 sysctl 配置？
- 明确的可监控指标来验证实际的效果

概述

1. 驱动加载和初始化
2. **数据包到达网卡控制器(NIC)**
3. 数据包被复制到内核空间（ DMA -> ring buffer ）
4. **产生硬件中断，通知系统数据可读**
5. 驱动调用 NAPI 激活 poll 循环（如果该循环处于休眠状态）
6. ksoftirpd 调用驱动注册的 poll 函数，读取 ring buffer 中的数据包
7. ring buffer 对应的内存区域被解除映射（ memory region unmapped ）
8. **数据包被封装为 skb 结构体，准备传递到上层协议栈**
9. 如果开启网卡多队列，数据帧会被负载均衡到多个 CPU 进行处理
10. 数据帧经由队列，递交上层协议栈
11. 协议栈处理（ IP -> UDP/TCP ）
12. 数据被填充到套接字的 receive buffer

网卡驱动

网卡驱动

- 网卡初始化和启动
- 监控和调优

网卡驱动 – 初始化

- 注册结构体 net_device_ops
- 注册 ethtool 的一系列操作
- 从 NIC 读取 MAC 地址

网卡驱动 – 初始化

```
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,

  /* ... */
```

网卡驱动 – 启动

1. 分配 rx / tx 队列内存
2. 开启 NAPI
3. 注册中断处理函数
4. 开启中断

NAPI

1. 驱动开启 NAPI，但是一开始处于未激活状态
2. 数据包到达并被 NIC 通过 DMA 拷贝到内存
3. NIC 产生中断，触发驱动注册的中断处理函数
4. **驱动通过 softirq 激活 NAPI 子系统**；这时会有专门的内核线程开始收割数据帧
5. 驱动**关闭 NIC 中断**，防止 NAPI 子系统在收割数据帧时被再次中断
6. 数据帧处理结束之后，NAPI 子系统挂起，并重新激活 NIC 中断信号
7. 重复执行上述第二步

网卡驱动 – 监控

- ethtool -S
- sysfs
- /proc/net/dev

网卡驱动 – 监控

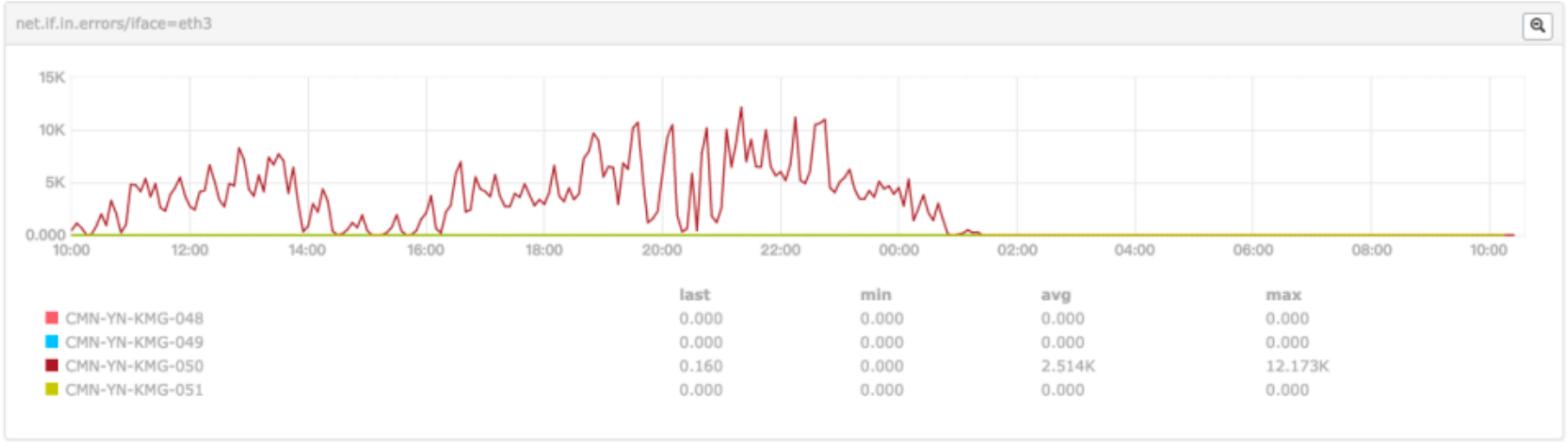
```
$ sudo ethtool -S eth0
NIC statistics:
     rx_packets: 597028087
     tx_packets: 5924278060
     rx_bytes: 112643393747
     tx_bytes: 990080156714
     rx_broadcast: 96
     tx_broadcast: 116
     rx_multicast: 20294528
     ....
```

```
$ cat /sys/class/net/eth0/statistics/rx_dropped
2
```

```
$ cat /proc/net/dev
Inter-|   Receive                                                |  Transmit
 face |bytes    packets errs drop fifo frame compressed multicast|bytes    packets
  eth0: 110346752214 597737500    0    2    0     0          0  20963860 990024805
    lo: 428349463836 1579868535    0    0    0     0          0         0 42834946
```

线上问题

```
eth3      Link encap:Ethernet  HWaddr 70:E2:84:0F:1E:CA
          UP BROADCAST RUNNING SLAVE MULTICAST  MTU:1500  Metric:1
          RX packets:31204251478 errors:307999764 dropped:0 overruns:0 frame:307999764
          TX packets:56397738064 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23243177916252 (21.1 TiB)  TX bytes:70755997153911 (64.3 TiB)
```



线上问题

- frame 错误是因为网卡在收到数据包进行 CRC 校验失败
- 一般是网线质量差导致的，或者是机器网卡和交换机端口其中一个存在硬件质量问题

线上问题


net.if.total.errors/iface=bond0
10K
8K
6K
4K
2K
0.000
Jan 07 12:00
Jan 07 16:00
Jan 07 20:00
Jan 08 00:00
Jan 08 04:00
Jan 08 08:00
Jan 08 12:00
Jan 08 16:00
Jan 08 20:00
Jan 09 00:00
Jan 09 04:00
Jan 09 08:00
Jan 09 12:00
Jan 09 16:00
Jan 10 04:00
Jan 10 08:00
CMN-YN-KMG-050
last
min
avg
max
0.000
0.000
1.208K
9.416K
全选
反选
刷新
输入字符过滤, 回车刷新...
CMN-YN-KMG-050
2019-01-09 18:20:00
0.000
CMN-YN-KMG-048
CMN-YN-KMG-049
CMN-YN-KMG-050
CMN-YN-KMG-051

网卡驱动 - 调优

- 检查当前使用的队列数目
- 调整队列数量
- 调整队列大小
- 调整队列权重
- 调整哈希字段
- ntuple filtering 控制特定数据帧到指定队列

```
$ sudo 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
```

网卡驱动 - 调优

例子

- web server 绑定到了 CPU2
- 某个网卡接收队列的中断信号，被指定到 CPU2 处理
- 目的端口为 80 的 tcp 数据帧被 *过滤* 到 CPU2
- 这样的话，从数据帧到达网卡开始，一直到应用层，80 端口的所有流量就都被 CPU2 处理了
- 缓存命中率和网络延迟都会有改善

内核（网络部分）

软中断

- 内核延时任务机制
- 内核线程 ksoftirqd
- 监控 /proc/softirqs

软中断 - 内核线程

```
static struct smp_hotplug_thread softirq_threads = {
  .store              = &ksoftirqd,
  .thread_should_run  = ksoftirqd_should_run,
  .thread_fn          = run_ksoftirqd,
  .thread_comm        = "ksoftirqd/%u",
};

static __init int spawn_ksoftirqd(void)
{
  register_cpu_notifier(&cpu_nfb);

  BUG_ON(smpboot_register_percpu_thread(&softirq_threads));

  return 0;
}
early_initcall(spawn_ksoftirqd);
```

https://github.com/torvalds/linux/blob/master/kernel/softirq.c

软中断 - 监控

```
$ cat /proc/softirqs
                    CPU0       CPU1       CPU2       CPU3
          HI:           0          0          0          0
       TIMER: 2831512516 1337085411 1103326083 1423923272
      NET_TX:   15774435     779806     733217     749512
      NET_RX: 1671622615 1257853535 2088429526 2674732223
       BLOCK: 1800253852    1466177    1791366     634534
BLOCK_IOPOLL:           0          0          0          0
     TASKLET:          25          0          0          0
       SCHED: 2642378225 1711756029  629040543  682215771
     HRTIMER:    2547911    2046898    1558136    1521176
         RCU: 2056528783 4231862865 3545088730  844379888
```

内核 - 初始化

net_dev_init：基于当前 cpu 数目创建各自的 softnet_data 结构体

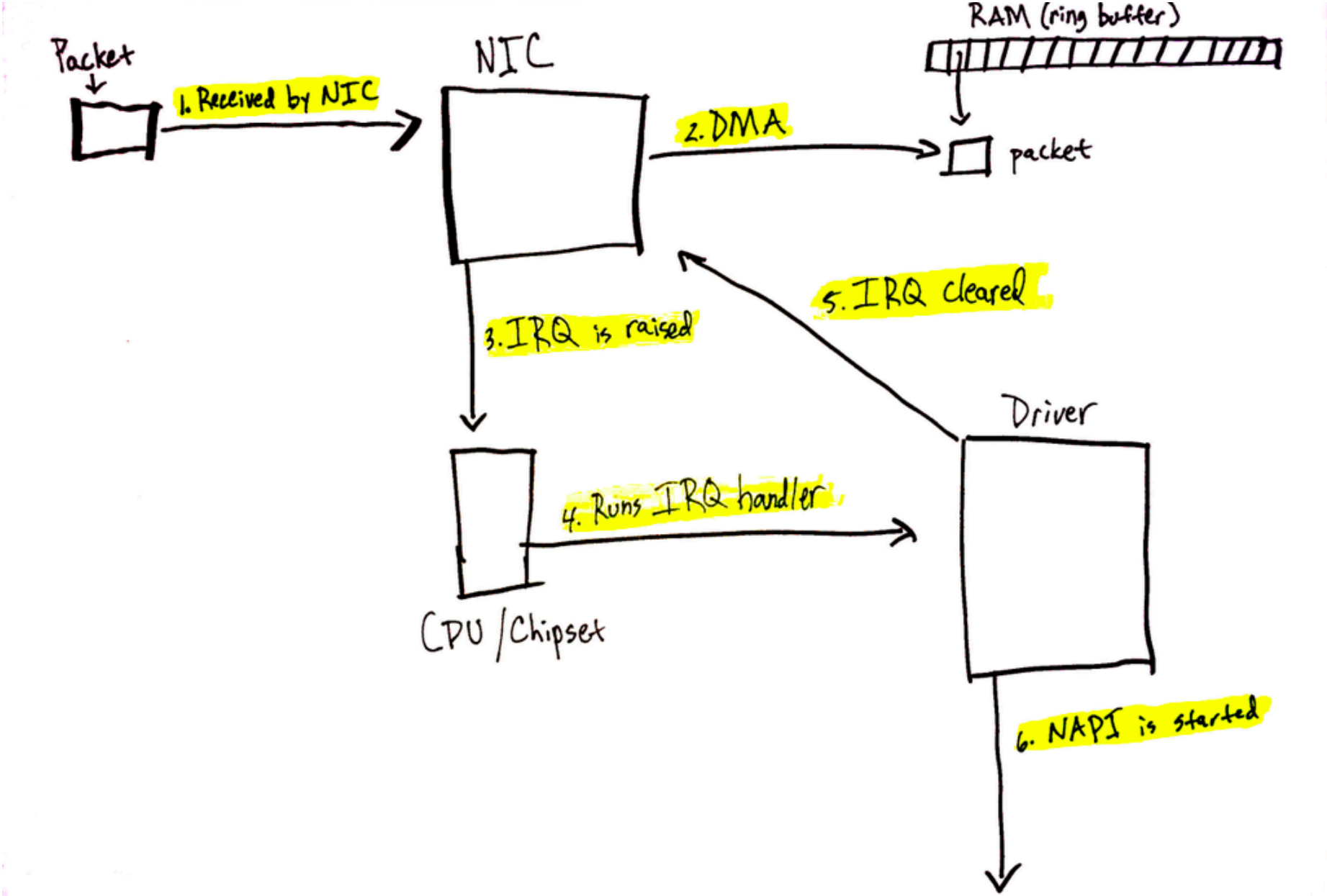
- NAPI 结构体列表
- backlog 队列
- 处理权重 - processing weight
- RPS 配置

- 注册软中断处理函数

```
static int __init net_dev_init(void)
{
  /* ... */

  open_softirq(NET_TX_SOFTIRQ, net_tx_action);
  open_softirq(NET_RX_SOFTIRQ, net_rx_action);

 /* ... */
}
```

内核 - 数据接收

- 中断处理
- 激活 NAPI


Packet
1. Received by NIC
NIC
RAM (ring buffer)
2. DMA
packet
3. IRQ is raised
5. IRQ cleared
Driver
4. Runs IRQ handler
CPU / Chipset
6. NAPI is started

内核 - 数据接收 - 中断处理

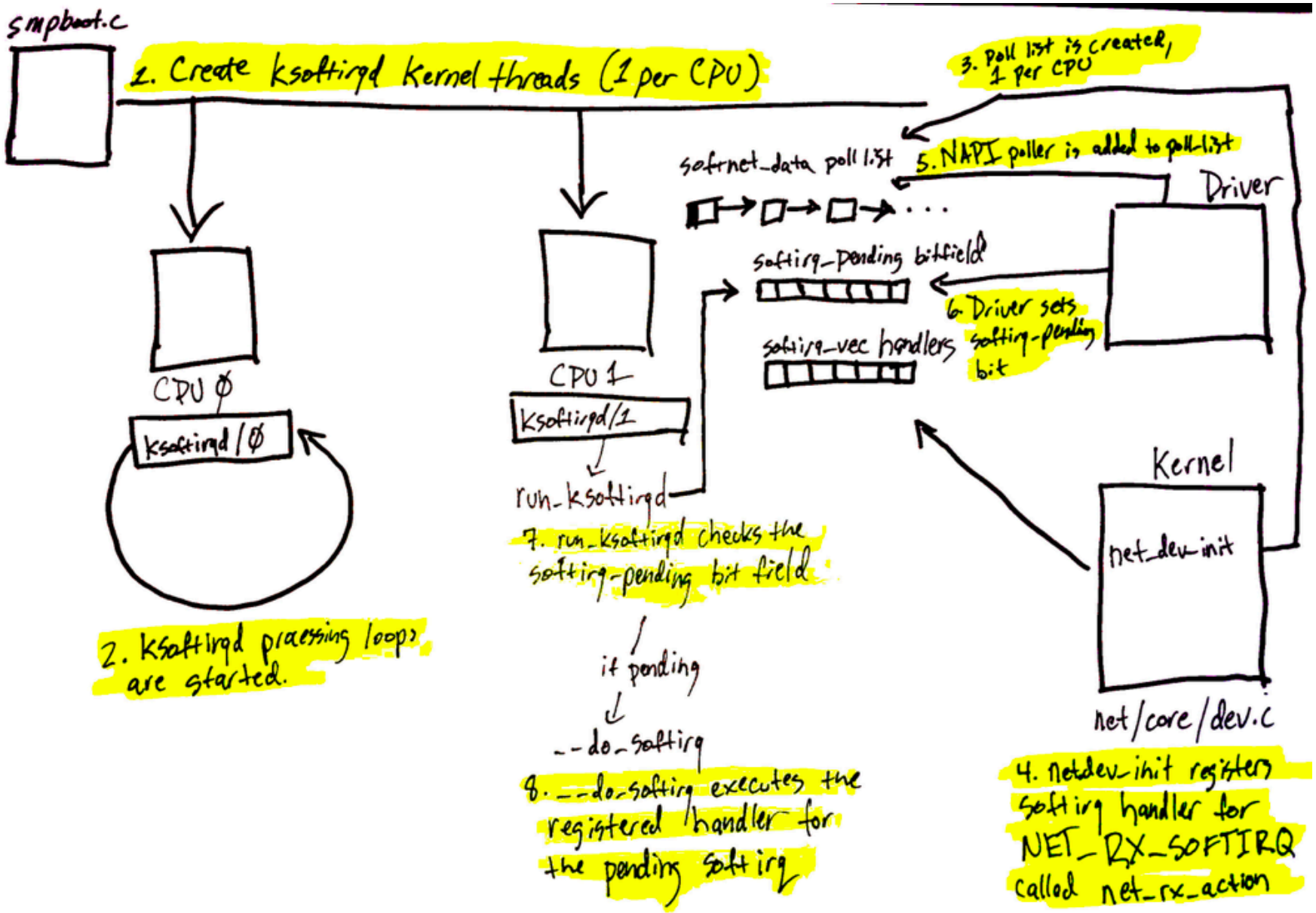
1. 更新硬件寄存器
2. 激活 NAPI 处理循环

```
static irqreturn_t igb_msix_ring(int irq, void *data)
{
  struct igb_q_vector *q_vector = data;

  /* Write the ITR value calculated from the previous interrupt. */
  igb_write_itr(q_vector);

  napi_schedule(&q_vector->napi);

  return IRQ_HANDLED;
}
```

NAPI 激活

1. 注册 callback 到当前 CPU 的 poll list
2. 触发 NET_RX_SOFTIRQ 软中断

```
/* Called with irq disabled */
static inline void ____napi_schedule(struct softnet_data *sd,
                                     struct napi_struct *napi)
{
  list_add_tail(&napi->poll_list, &sd->poll_list);
  __raise_softirq_irqoff(NET_RX_SOFTIRQ);
}
```


smpboot.c
1. Create ksoftirqd kernel threads (1 per CPU)
3. Poll list is created, 1 per CPU
softnet_data poll list
5. NAPI poller is added to poll_list
Driver
softirq_pending bitfield
6. Driver sets softirq-pending bit
softirq_vec handlers
CPU 0
ksoftirqd/0
CPU 1
ksoftirqd/1
run_ksoftirqd
7. run_ksoftirqd checks the softirq_pending bit field
if pending
__do_softirq
8. __do_softirq executes the registered handler for the pending softirq
2. Ksoftirqd processing loops are started.
Kernel
net_dev_init
net/core/dev.c
4. netdev_init registers softirq handler for NET_RX_SOFTIRQ called net_rx_action

数据接收 - 监控

- 因为 NAPI 和中断合并的存在，这里的数据并不可靠

```
$ cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:         46          0          0          0 IR-IO-APIC-edge      timer
  1:          3          0          0          0 IR-IO-APIC-edge      i8042
 30: 3361234770          0          0          0 IR-IO-APIC-fasteoi   aacraid
 64:          0          0          0          0 DMAR_MSI-edge      dmar0
 65:          1          0          0          0 IR-PCI-MSI-edge      eth0
 66:  863649703          0          0          0 IR-PCI-MSI-edge      eth0-TxRx-0
 67:  986285573          0          0          0 IR-PCI-MSI-edge      eth0-TxRx-1
 68:         45          0          0          0 IR-PCI-MSI-edge      eth0-TxRx-2
 69:        394          0          0          0 IR-PCI-MSI-edge      eth0-TxRx-3
NMI:    9729927    4008190    3068645    3375402  Non-maskable interrupts
LOC: 2913290785 1585321306 1495872829 1803524526  Local timer interrupts
```

数据接收 - 调优

- 中断合并

  自适应中断合并
- 中断亲和性

自适应中断合并

流量低峰期降低延时，流量高峰期提升吞吐

- rx-usecs：数据帧到达后，延迟多长时间产生中断信号，单位微妙
- rx-frames：触发中断前积累数据帧的最大个数
- rx-usecs-irq：如果有中断处理正在执行，当前中断延迟多久送达 CPU
- rx-frames-irq：如果有中断处理正在执行，最多积累多少个数据帧

中断亲和性

```
$ sudo bash -c 'echo 1 > /proc/irq/8/smp_affinity'
```

- /proc/interrupts

irqbalance (http://irqbalance.github.io/irqbalance/)

- https://github.com/Irqbalance/irqbalance/blob/master/activate.c#L67

内核 - 数据处理

遍历当前 CPU 的 NAPI 结构体列表

- budget 和 时长限制
- poll 轮询权重

限制条件

- 整体限制

```
while (!list_empty(&sd->poll_list)) {
  struct napi_struct *n;
  int work, weight;

  /* If softirq window is exhausted then punt.
   * Allow this to run for 2 jiffies since which will allow
   * an average latency of 1.5/HZ.
   */
  if (unlikely(budget <= 0 || time_after_eq(jiffies, time_limit)))
    goto softnet_break;
```

- 单次限制

```
weight = n->weight;

work = 0;
if (test_bit(NAPI_STATE_SCHED, &n->state)) {
        work = n->poll(n, weight);
        trace_napi_poll(n);
}

WARN_ON_ONCE(work > weight);

budget -= work;
```

内核 - 数据处理 - 结束循环

```
softnet_break:
  sd->time_squeeze++;
  __raise_softirq_irqoff(NET_RX_SOFTIRQ);
  goto out;
```

- time_squeeze：被迫中断的次数

poll 循环

1. 分配新的 buffer 加入到 ring buffer（接收队列 - RX queue）
2. 从接收队列中取走一个 buffer，加入到 skb 结构体
3. 检查该 buffer 是不是 End of Packet，如果不是则继续取下一个 buffer 追加到 skb
4. 校验整个数据帧的布局和头部是否完整
5. 更新统计数据：已处理的字节数（skb->len）
6. 设置 skb 结构体的 哈希值，校验和，时间戳，协议簇 等，这些值都是硬件提供的；如果硬件报告了校验和失败，这里会更新对应的统计数据 csum_error，继续交给上层协议处理；协议字段通过一个单独的函数 eth_type_trans 计算处理，并保存在 skb 结构体里
7. 构造完毕的 skb 结构体通过 napi_gro_receive 向协议栈上层传递
8. 更新统计数据：已处理的包数量
9. 重复上述步骤，处理下一个数据帧

数据处理 - 监控

- 每一行对应一个 CPU
- 数值之间空格分割，输出格式是 十六进制是 十六进制

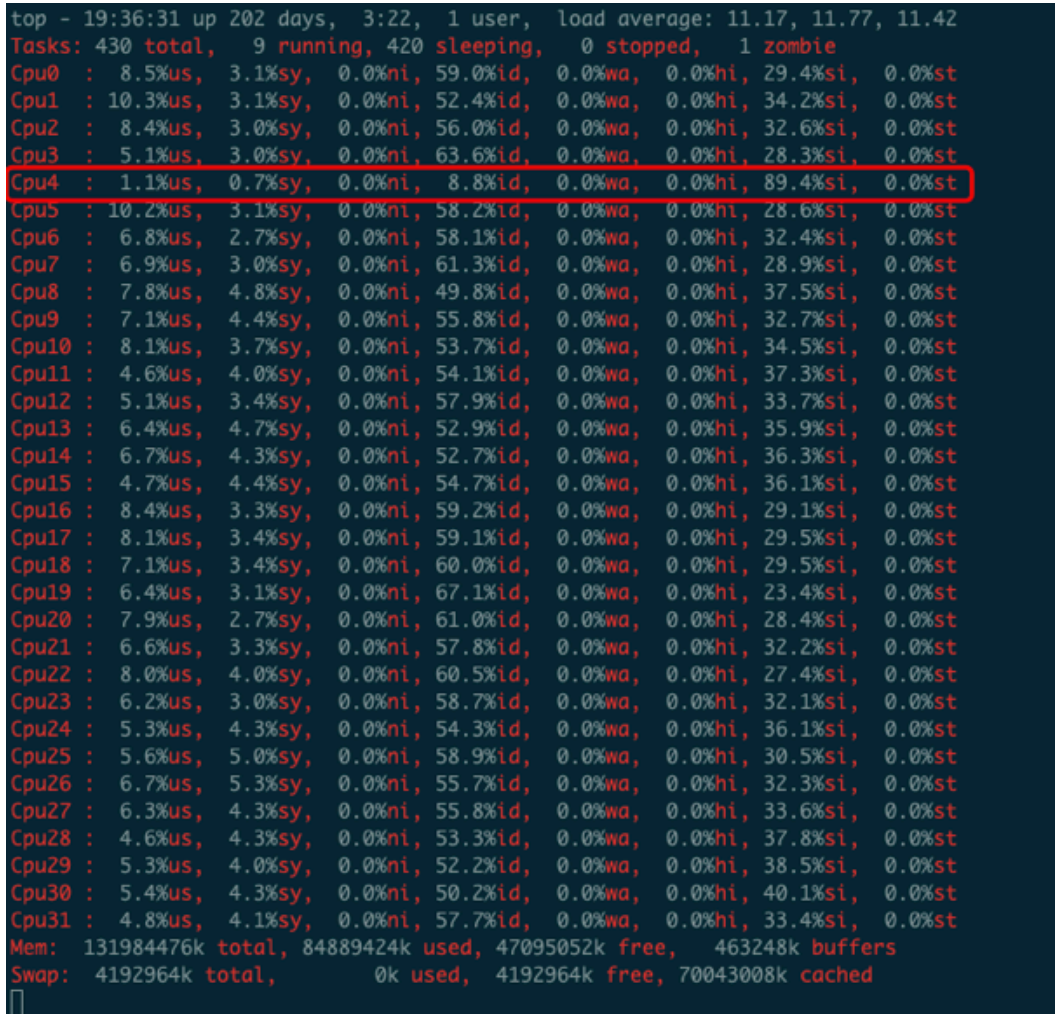
```
$ cat /proc/net/softnet_stat
6dcad223 00000000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
6f0e1565 00000000 00000002 00000000 00000000 00000000 00000000 00000000 00000000
660774ec 00000000 00000003 00000000 00000000 00000000 00000000 00000000 00000000
61c99331 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
6794b1b3 00000000 00000005 00000000 00000000 00000000 00000000 00000000 00000000
6488cb92 00000000 00000001 00000000 00000000 00000000 00000000 00000000 00000000
```

数据处理 - 监控

```
seq_printf(seq,
     "%08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x\n",
     sd->processed, sd->dropped, sd->time_squeeze, 0,
     0, 0, 0, 0, /* was fastroute */
     sd->cpu_collision, sd->received_rps, flow_limit_count);
```

* sd->processed　　处理的包数量（多网卡 bond 模式可能多于实际的收包数量）
* sd->dropped　　丢包数量，因为处理队列满了
* sd->time_squeeze　　软中断处理 net_rx_action 被迫打断的次数
* sd->cpu_collision　　发送数据时获取设备锁冲突，比如多个 CPU 同时发送数据
* sd->received_rps　　当前 CPU 被唤醒的次数（通过处理器间中断）
* sd->flow_limit_count 触发 flow limit 的次数

线上问题


top - 19:36:31 up 202 days, 3:22, 1 user, load average: 11.17, 11.77, 11.42
Tasks: 430 total,   9 running, 420 sleeping,   0 stopped,   1 zombie
Cpu0 : 8.5%us, 3.1%sy, 0.0%ni, 59.0%id, 0.0%wa, 0.0%hi, 29.4%si, 0.0%st
Cpu1 : 10.3%us, 3.1%sy, 0.0%ni, 52.4%id, 0.0%wa, 0.0%hi, 34.2%si, 0.0%st
Cpu2 : 8.4%us, 3.0%sy, 0.0%ni, 56.0%id, 0.0%wa, 0.0%hi, 32.6%si, 0.0%st
Cpu3 : 5.1%us, 3.0%sy, 0.0%ni, 63.6%id, 0.0%wa, 0.0%hi, 28.3%si, 0.0%st
Cpu4 : 1.1%us, 0.7%sy, 0.0%ni, 8.8%id, 0.0%wa, 0.0%hi, 89.4%si, 0.0%st
Cpu5 : 10.2%us, 3.1%sy, 0.0%ni, 58.2%id, 0.0%wa, 0.0%hi, 28.6%si, 0.0%st
Cpu6 : 6.8%us, 2.7%sy, 0.0%ni, 58.1%id, 0.0%wa, 0.0%hi, 32.4%si, 0.0%st
Cpu7 : 6.9%us, 3.0%sy, 0.0%ni, 61.3%id, 0.0%wa, 0.0%hi, 28.9%si, 0.0%st
Cpu8 : 7.8%us, 4.8%sy, 0.0%ni, 49.8%id, 0.0%wa, 0.0%hi, 37.5%si, 0.0%st
Cpu9 : 7.1%us, 4.4%sy, 0.0%ni, 55.8%id, 0.0%wa, 0.0%hi, 32.7%si, 0.0%st
Cpu10 : 8.1%us, 3.7%sy, 0.0%ni, 53.7%id, 0.0%wa, 0.0%hi, 34.5%si, 0.0%st
Cpu11 : 4.6%us, 4.0%sy, 0.0%ni, 54.1%id, 0.0%wa, 0.0%hi, 37.3%si, 0.0%st
Cpu12 : 5.1%us, 3.4%sy, 0.0%ni, 57.9%id, 0.0%wa, 0.0%hi, 33.7%si, 0.0%st
Cpu13 : 6.4%us, 4.7%sy, 0.0%ni, 52.9%id, 0.0%wa, 0.0%hi, 35.9%si, 0.0%st
Cpu14 : 6.7%us, 4.3%sy, 0.0%ni, 52.7%id, 0.0%wa, 0.0%hi, 36.3%si, 0.0%st
Cpu15 : 4.7%us, 4.4%sy, 0.0%ni, 54.7%id, 0.0%wa, 0.0%hi, 36.1%si, 0.0%st
Cpu16 : 8.4%us, 3.3%sy, 0.0%ni, 59.2%id, 0.0%wa, 0.0%hi, 29.1%si, 0.0%st
Cpu17 : 8.1%us, 3.4%sy, 0.0%ni, 59.1%id, 0.0%wa, 0.0%hi, 29.5%si, 0.0%st
Cpu18 : 7.1%us, 3.4%sy, 0.0%ni, 60.0%id, 0.0%wa, 0.0%hi, 29.5%si, 0.0%st
Cpu19 : 6.4%us, 3.1%sy, 0.0%ni, 67.1%id, 0.0%wa, 0.0%hi, 23.4%si, 0.0%st
Cpu20 : 7.9%us, 2.7%sy, 0.0%ni, 61.0%id, 0.0%wa, 0.0%hi, 28.4%si, 0.0%st
Cpu21 : 6.6%us, 3.3%sy, 0.0%ni, 57.8%id, 0.0%wa, 0.0%hi, 32.2%si, 0.0%st
Cpu22 : 8.0%us, 4.0%sy, 0.0%ni, 60.5%id, 0.0%wa, 0.0%hi, 27.4%si, 0.0%st
Cpu23 : 6.2%us, 3.0%sy, 0.0%ni, 58.7%id, 0.0%wa, 0.0%hi, 32.1%si, 0.0%st
Cpu24 : 5.3%us, 4.3%sy, 0.0%ni, 54.3%id, 0.0%wa, 0.0%hi, 36.1%si, 0.0%st
Cpu25 : 5.6%us, 5.0%sy, 0.0%ni, 58.9%id, 0.0%wa, 0.0%hi, 30.5%si, 0.0%st
Cpu26 : 6.7%us, 5.3%sy, 0.0%ni, 55.7%id, 0.0%wa, 0.0%hi, 32.3%si, 0.0%st
Cpu27 : 6.3%us, 4.3%sy, 0.0%ni, 55.8%id, 0.0%wa, 0.0%hi, 33.6%si, 0.0%st
Cpu28 : 4.6%us, 4.3%sy, 0.0%ni, 53.3%id, 0.0%wa, 0.0%hi, 37.8%si, 0.0%st
Cpu29 : 5.3%us, 4.0%sy, 0.0%ni, 52.2%id, 0.0%wa, 0.0%hi, 38.5%si, 0.0%st
Cpu30 : 5.4%us, 4.3%sy, 0.0%ni, 50.2%id, 0.0%wa, 0.0%hi, 40.1%si, 0.0%st
Cpu31 : 4.8%us, 4.1%sy, 0.0%ni, 57.7%id, 0.0%wa, 0.0%hi, 33.4%si, 0.0%st
Mem: 131984476k total, 84889424k used, 47095052k free,   463248k buffers
Swap: 4192964k total,   0k used, 4192964k free, 70043008k cached

线上问题

```
[root@CMN-AH-HNN-003 ~]# date;cat /proc/net/softnet_stat
Wed Jan 16 20:26:06 CST 2019
278520c5 00000000 0056872d 00000000 00000000 00000000 00000000 00000000 00000000 7fb53f96 00000000
4800ddf8 00000000 005a29ef 00000000 00000000 00000000 00000000 00000000 00000000 d01d5401 00000000
8c049a5a 00000000 0060cc33 00000000 00000000 00000000 00000000 00000000 00000000 c6cabfa2 00000000
5f834cf9 00000000 006d44da 00000000 00000000 00000000 00000000 00000000 00000000 ac5ea4c6 00000000
a2add74b 00000000 16a3007d 00000000 00000000 00000000 00000000 00000000 00000000 0fc2961a 00000000
4db356a2 00000000 00534117 00000000 00000000 00000000 00000000 00000000 00000000 ba7f9c0e 00000000
752332ba 00000000 00546988 00000000 00000000 00000000 00000000 00000000 00000000 b68cccc2 00000000
6f44f8cd 00000000 0054cb05 00000000 00000000 00000000 00000000 00000000 00000000 b1a65102 00000000
3a11d721 00000000 00929ec9 00000000 00000000 00000000 00000000 00000000 00000000 fdadad4e 00000000
74b1f972 00b782b7 00880ace 00000000 00000000 00000000 00000000 00000000 00000000 c8744446 00000000
0fc80861 00000000 00867d3c 00000000 00000000 00000000 00000000 00000000 00000000 b41c8de9 00000000
62c9fc4f 00000000 00843b8f 00000000 00000000 00000000 00000000 00000000 00000000 af244440 00000000
42728f83 00000000 0019ee9f 00000000 00000000 00000000 00000000 00000000 00000000 9c0e9db2 00000000
6d0b05b5 00000000 00197d13 00000000 00000000 00000000 00000000 00000000 00000000 86361e46 00000000
bca3f84b 00000000 00197448 00000000 00000000 00000000 00000000 00000000 00000000 76f8ee5e 00000000
07b4ea5e 00000000 001accf2 00000000 00000000 00000000 00000000 00000000 00000000 6400c3fe 00000000
0bf23c70 00000000 000d710e 00000000 00000000 00000000 00000000 00000000 00000000 f275a0c1 00000000
6ba37a45 00000000 000eab63 00000000 00000000 00000000 00000000 00000000 00000000 29eb9c9d 00000000
52cc6787 00000000 000ee5f4 00000000 00000000 00000000 00000000 00000000 00000000 2591cae8 00000000
1bc2906e 00000000 000fc31f 00000000 00000000 00000000 00000000 00000000 00000000 1f23e7c0 00000000
3aa491a6 00000000 000ea875 00000000 00000000 00000000 00000000 00000000 00000000 d3655a00 00000000
3e03c476 002c6481 000f4170 00000000 00000000 00000000 00000000 00000000 00000000 1becc768 00000000
f7db9bc9 00000000 000f34d6 00000000 00000000 00000000 00000000 00000000 00000000 13c6f8b9 00000000
d6937c7c 00000000 000f697b 00000000 00000000 00000000 00000000 00000000 00000000 0d731206 00000000
44bc9f21 00000000 001e5d9d 00000000 00000000 00000000 00000000 00000000 00000000 dd3c7f4b 00000000
d2da9a27 00000000 001d346a 00000000 00000000 00000000 00000000 00000000 00000000 f19f13e7 00000000
ac0e3fa4 00000000 001d8364 00000000 00000000 00000000 00000000 00000000 00000000 ece7788b 00000000
9fe609c5 00000000 001d3475 00000000 00000000 00000000 00000000 00000000 00000000 eb84ac30 00000000
05bdf1e3 00000000 001d922e 00000000 00000000 00000000 00000000 00000000 00000000 f427f2d1 00000000
e81687e5 00000000 001d4ba2 00000000 00000000 00000000 00000000 00000000 00000000 f0794f5c 00000000
cff90be0 00000000 001dd67e 00000000 00000000 00000000 00000000 00000000 00000000 ec8e44ac 00000000
b4d49cad 00000000 002019e4 00000000 00000000 00000000 00000000 00000000 00000000 e47068a8 00000000
```

线上问题

cpu4 核心存在性能衰退，来不及处理网络数据

线上问题

```
pid = os.getpid()
affinity.set_process_affinity_mask(pid, 16L)

res = 0
start = time.time()

for idx in xrange(10000000):
    res = res + idx

end = time.time()
print 'time: %f' % (end - start)
```

其他

- GRO
- RSS / RPS
  - backlog 队列 - per CPU
- RFS / aRFS
- 协议层
  - IP -> UDP / TCP

THANKS !