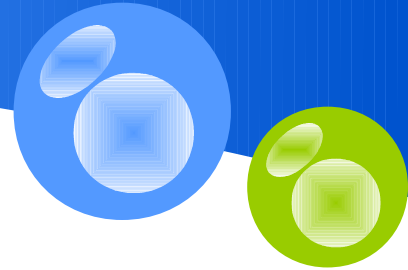




# Receive Packet Steering

富士通南大软件技术有限公司 魏勇军

# Receive Packet Steering(RPS)



**1. *RPS* 简介**

**2. Why RPS?**

**3. 内核实现**

# 1. RPS 简介

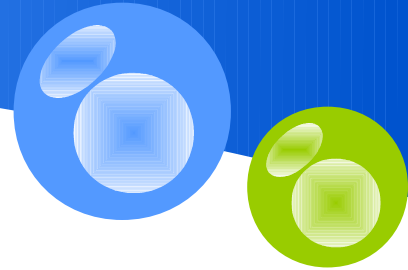


❖ Receive Packet Steering , 简称 RPS , 由来自 Google 的 Tom Herbert 贡献给 linux kernel 。

❖ 从 Linux kernel 2.6.35 开始进入 Main tree 。

❖ 用于解决多 CPU 下网络协议的软中断的负载均衡。

# Receive Packet Steering(RPS)



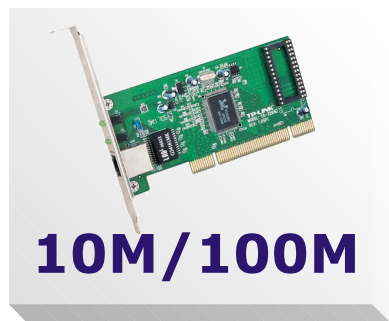
1. RPS 简介

2. Why RPS?

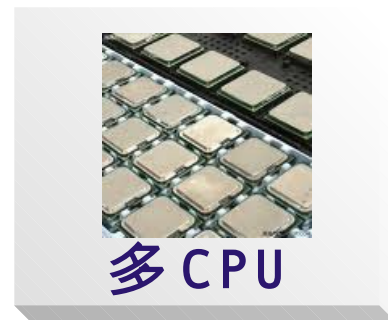
3. 内核实现

## 2. Why RPS?

### 1. 网卡速度的提升



速度难以提升



### 2. 单机 CPU 个数增加

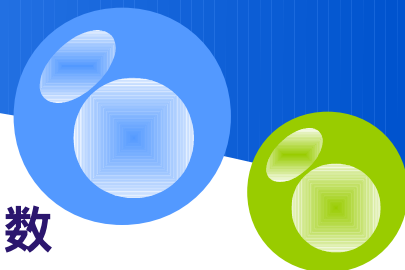
# 多核下数据包的处理

## ❖ 单队列，多cpu

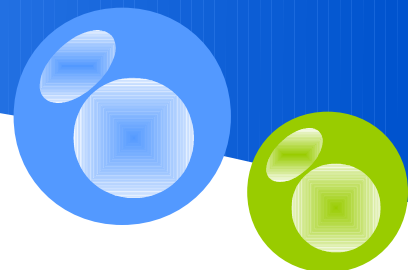
到达一个数据包，驱动将它中断传给cpu 0，再到一个数据包，驱动又将中断传给下一个cpu。这种方式的一个极大的缺陷就是，容易导致包乱序。

## ❖ 多队列、多中断号

网卡支持多个队列，每个队列又有一个独立的中断号。但网卡中的队列数往往是固定不变的，而这个数目如果与CPU个数相统一，那么就工作的很好，如果CPU个数跟它不一样呢？



# Receive Packet Steering(RPS)



**1. RPS 简介**

**2. Why RPS?**

**3. 内核实现**

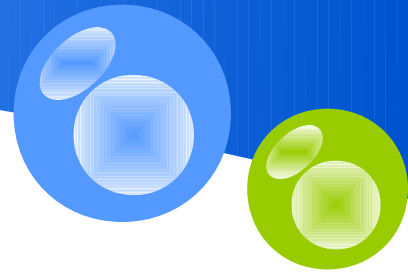
# 网卡驱动支持的两种模式

## ❖ NAPI

收到数据包后，调用 `__napi_schedule` 调度软中断，然后软中断处理函数中会调用注册的 `poll` 回调函数中调用 `netif_receive_skb` 将数据包发送到 3 层。

## ❖ 非 NAPI

中断收到数据包后调用 `netif_rx`，这个函数会将数据包保存到 `input_pkt_queue`，然后调度软中断，这里为了兼容 NAPI 的驱动，他的 `poll` 方法默认是 `process_backlog`，最终这个函数会从 `input_pkt_queue` 中取得数据包然后发送到 3 层。



# 网卡驱动支持的两种模式（续）

## ❖ 负载均衡？

不管是 NAPI 还是非 NAPI 的话都无法做到软中断的均衡。因为中断都是运行在硬件中断相应的 cpu 上。也就是说如果始终是 cpu0 相应网卡的硬件中断，那么始终都是 cpu0 在处理软中断，而此时 cpu1 就被浪费了，因为无法并行的执行多个软中断。

另外，即使中断能一会运行在 cpu0，一会运行在 cpu1 的话，也会出现数据包的乱序。

# Tom 的实现方案

hook 两个函数，在数据包被发送到 3 层之前做处理

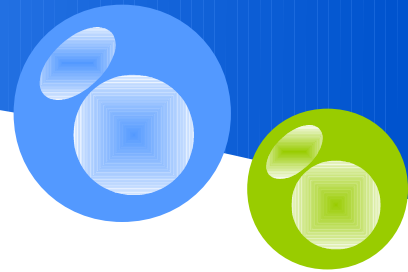
## ❖ netif\_rx

- 非 NAPI 的驱动

## ❖ netif\_receive\_skb

主要是针对 NAPI 的驱动。

# Tom 的实现方案（续 1）



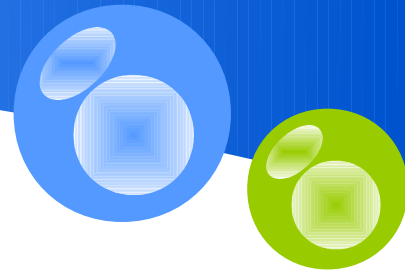
## 接收到数据包之后

### ❖ 计算每个数据包的 hash 值

使用数据包的源地址，目的地址，源端口号，目的端口号计算 hash 值，再映射到 cpu id 上。

### ❖ 将数据包添加到 cpu id 对应的 backlog 队列等待处理

# Tom 的实现方案（续 2）



## ❖ 对驱动程序的修改

如果计算 hash 值由 cpu 来做的话，每次需要读取数据包报头，这样会导致 cache miss。

好在很多网卡都支持计算这个 hash 值。我们只需要修改驱动去读取这个 hash 值。

# 性能影响

## ❖ **tg3 on 8 core Intel**

Without RPS: 90K tps at 34% CPU

With RPS: 285K tps at 70% CPU

## ❖ **e1000 on 8 core Intel**

Without RPS: 90K tps at 34% CPU

With RPS: 292K tps at 66% CPU

## ❖ **foredeth on 16 core AMD**

Without RPS: 117K tps at 10% CPU

With RPS: 327K tps at 29% CPU

## ❖ **bnx2x on 16 core AMD**

Single queue without RPS: 139K tps at 17% CPU

Single queue with RPS: 352K tps at 30% CPU

Multi queue (1 queues per CPU) 204K tps at 12% CPU



大概能够达到  
30%到50%的性能提高



**Thank You!**