Scaling in the Linux Networking Stack Introduction

Linux网络栈的性能缩放

本文翻译自：https://www.kernel.org/doc/html/latest/networking/scaling.html

This document describes a set of complementary techniques in the Linux networking stack to increase parallelism and improve performance for multi-processor systems.

The following technologies are described:

RSS: Receive Side Scaling
RPS: Receive Packet Steering
RFS: Receive Flow Steering
Accelerated Receive Flow Steering
XPS: Transmit Packet Steering

本文档介绍了Linux网络堆栈中的一组辅助性技术，这些技术用于提高多处理器系统并行度和性能。

本文描述了以下技术：

- RSS：Receive Side Scaling 接收方缩放
- RPS：Receive Packet Steering 数据包接收导向
- RFS：Receive Flow Steering 接收流导向
- Accelerated Receive Flow Steering 加速接收流导向
- XPS：Transmit Packet Steering 数据包传输导向

RSS: Receive Side Scaling

接收方缩放

Contemporary NICs support multiple receive and transmit descriptor queues (multi-queue). On reception, a NIC can send different packets to different queues to distribute processing among CPUs. The NIC distributes packets by applying a filter to each packet that assigns it to one of a small numberof logical flows. Packets for each flow are steered to a separate receivequeue, which in turn can be processed by separate CPUs. This mechanism is generally known as receive-side scaling(RSS). The goal of RSS and
the other scaling techniques is to increase performance uniformly.Multi-queue distribution can also be used for traffic prioritization, but that is not the focus of these techniques.

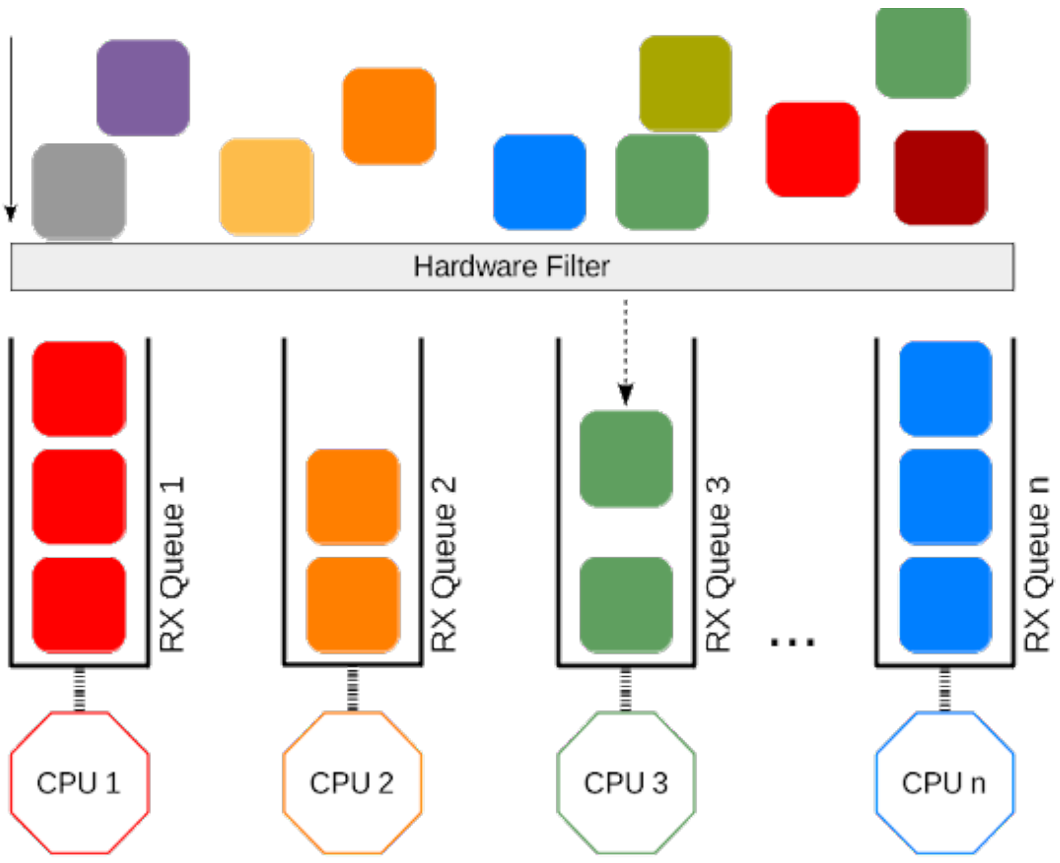
译者：*在一台运行着大量交易的多处理器系统上，交易数据都通过一张网卡传输，如果这些网络包都按照顺序单线程处理，所有网络中断都会集中在一个处理器上，网络包的处理会成为性能瓶颈。*

当代的网卡支持多个接收和发送队列（多队列）。接收时，NIC可以将不同的数据包发送到不同的队列，把包的处理分配到不同的CPU。网卡通过过滤器来分配每个数据包，过滤器将数据包分配给若干个逻辑流。每个流的数据包都被引导到一个单独的接收队列，队列可以由不同的CPU处理。这种机制通常称为"接收方缩放"（RSS）。RSS和其他缩放技术的目标都是为了提高性能。多队列分配也可以用于流量优先级划分，但这并不是这些技术的关注点。

The filter used in RSS is typically a hash function over the network and/or transport layer headers-- for example, a 4-tuple hash over IP addresses and TCP ports of a packet. The most common hardware
implementation of RSS uses a 128-entry indirection table where each entry stores a queue number. The receive queue for a packet is determined by masking out the low order seven bits of the computed hash for the packet (usually a Toeplitz hash), taking this number as a key into the indirection table and reading the corresponding value.

RSS中使用的过滤器通常是针对网络和/或传输层头部数据建立的哈希函数，例如，数据包的IP地址4元组和TCP端口的哈希值。RSS的最常见硬件实现使用128项间接表，其中每个条目都存储一个队列号。通过掩码计算出的数据包哈希值的低阶七位（通常是Toeplitz Hash），将该数字作为间接表中的键就可以确定数据包的接收队列。

注：图片引用自https://garycplin.blogspot.com/2017/06/linux-network-scaling-receives-packets.html



Some advanced NICs allow steering packets to queues based on programmable filters. For example, webserver bound TCP port 80 packets can be directed to their own receive queue. Such n-tuple filters can be configured from ethtool (--config-ntuple).

一些高级NIC允许通过可编程过滤器将数据包引导到相应的队列。例如，可以将Web服务器绑定的TCP端口80数据包定向到其自己的接收队列。可以通过ethtool（–config-ntuple）配置此类“ n元组”过滤器。

RSS Configuration

RSS配置

The driver for a multi-queue capable NIC typically provides a kernel module parameter for specifying the number of hardware queues to configure. In the bnx2x driver, for instance, this parameter is called
num_queues. A typical RSS configuration would be to have one receive queue for each CPU if the device supports enough queues, or otherwise at least one for each memory domain, where a memory domain is a set of CPUs that share a particular memory level (L1, L2, NUMA node, etc.).

具有多队列功能的网卡驱动程序通常提供一个内核模块参数，用于指定要配置的硬件队列的数量。例如，在bnx2x驱动程序中，此参数称为num_queues。如果设备支持足够的队列，则典型的RSS配置将是每个CPU有一个接收队列，否则，每个内存域至少有一个接收队列，其中一个内存域是一组共享特定内存级别（L1，L2，NUMA节点等）的CPU。

The indirection table of an RSS device, which resolves a queue by masked hash, is usually programmed by the driver at initialization. The default mapping is to distribute the queues evenly in the table, but the
indirection table can be retrieved and modified at runtime using ethtool commands (--show-rxfh-indir and --set-rxfh-indir). Modifying the indirection table could be done to give different queues different
relative weights.

RSS设备的间接表通过掩码散列解析队列，通常由驱动程序在初始化时进行编程。默认映射是在表中均匀分配队列，但是可以使用ethtool命令（–show-rxfh-indir和–set-rxfh-indir）在运行时检索和修改间接表。通过修改间接表可以赋予不同的队列不同的相对权重。

RSS IRQ Configuration

RSS IRQ配置

Each receive queue has a separate IRQ associated with it. The NIC triggers this to notify a CPU when new packets arrive on the given queue. The signaling path for PCIe devices uses message signaled interrupts (MSI-X), that can route each interrupt to a particular CPU. The active mapping of queues to IRQs can be determined from /proc/interrupts. By default, an IRQ may be handled on any CPU. Because a non-negligible part of packet processing takes place in receive interrupt handling, it is advantageous
to spread receive interrupts between CPUs. To manually adjust the IRQ
affinity of each interrupt see Documentation/IRQ-affinity.txt. Some systems
will be running irqbalance, a daemon that dynamically optimizes IRQ
assignments and as a result may override any manual settings.

每个接收队列都有一个与之关联的单独的IRQ。当新数据包到达给定队列时，NIC会触发此IRQ通知CPU。PCIe设备的信令路径使用消息信号中断（MSI-X），可以将每个中断路由到特定的CPU。队列到IRQ的活动映射可以从/ proc / interrupts文件中确定。默认情况下，一个IRQ可以在任何CPU上处理。因为数据包处理有不可忽视的部分必须发生在接收中断处理中，所以在CPU之间分散接收中断是对性能有利的。如果要手动调整每个中断的IRQ亲和力，请参见Documentation / IRQ-affinity.txt。某些系统会运行irqbalance，它是一个动态优化IRQ分配的守护程序，可能会覆盖任何手动设置。

Suggested Configuration

建议配置

RSS should be enabled when latency is a concern or whenever receive interrupt processing forms a bottleneck. Spreading load between CPUs decreases queue length. For low latency networking, the optimal setting is to allocate as many queues as there are CPUs in the system (or the NIC maximum, if lower). The most efficient high-rate configuration is likely the one with the smallest number of receive queues where no
receive queue overflows due to a saturated CPU, because in default mode with interrupt coalescing enabled, the aggregate number of interrupts (and thus work) grows with each additional queue.

当担忧延迟或接收中断处理成为瓶颈时，应启用RSS。在CPU之间分散负载会减少队列长度。对于低延迟网络，最佳设置是分配与系统中CPU数量一样多的队列（或网卡最大支持的队列数，如果更低）。最高效的高速率配置可能是给处理器接收队列数量最少的配置，这样不会因某个CPU饱和而导致接收队列溢出，因为在默认模式下启用了中断合并的情况下，总的中断数量（因此工作）会随着队列数而增长。

Per-cpu load can be observed using the mpstat utility, but note that on processors with hyperthreading (HT), each hyperthread is represented as a separate CPU. For interrupt handling, HT has shown no benefit in initial tests, so limit the number of queues to the number of CPU cores in the system.

使用mpstat实用程序可以观察到每个CPU的负载，但是请注意，在具有超线程（HT）的处理器上，每个超线程都表示为一个单独的CPU。对于中断处理，HT在初始测试中没有显示任何好处，因此将队列数限制为系统中CPU内核数。

RPS: Receive Packet Steering

RPS：接收数据包指导

Receive Packet Steering (RPS) is logically a software implementation of RSS. Being in software, it is necessarily called later in the datapath. Whereas RSS selects the queue and hence CPU that will run the hardware interrupt handler, RPS selects the CPU to perform protocol processing above the interrupt handler. This is accomplished by placing the packet on the desired CPU backlog queue and waking up the CPU for processing.

RPS has some advantages over RSS:

1) it can be used with any NIC,

2) software filters can easily be added to hash over new protocols,

3) it does not increase hardware device interrupt rate (although it does introduce inter-processor interrupts (IPIs)).

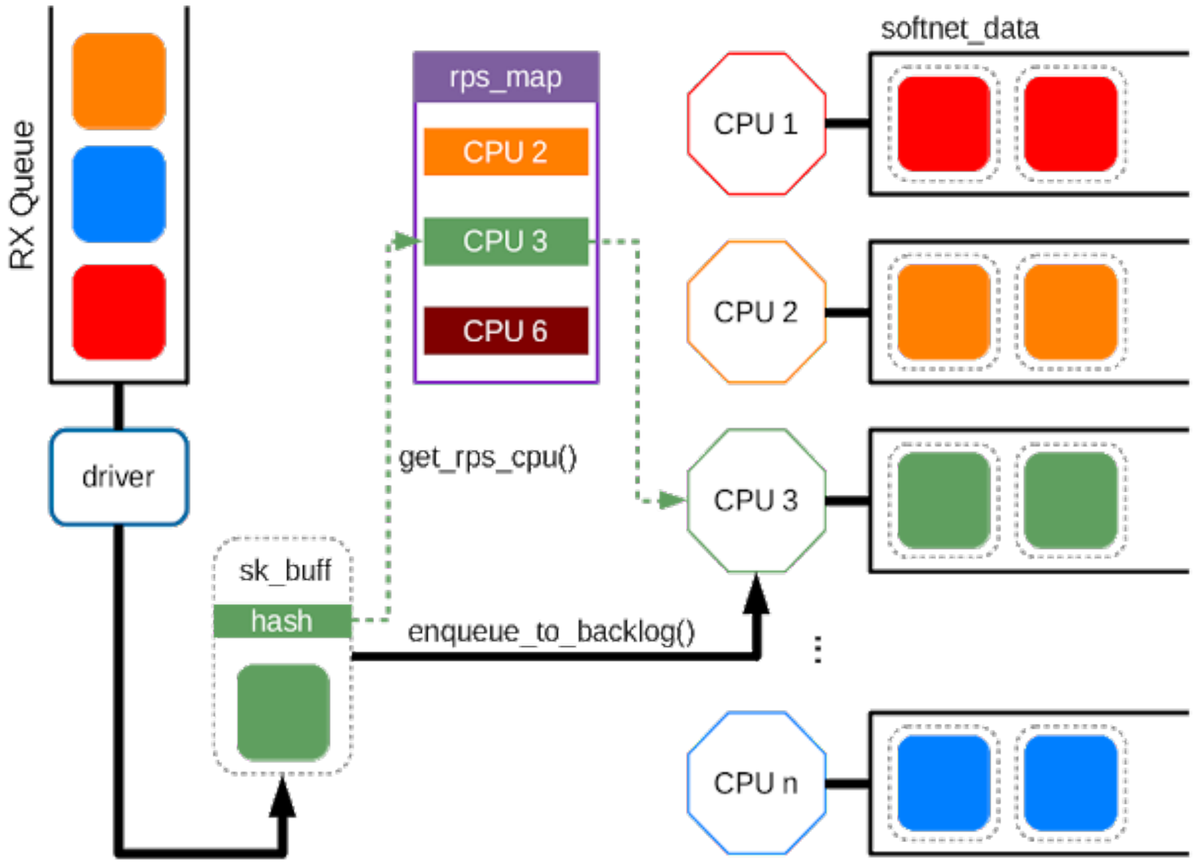
从逻辑上说，接收数据包导向（RPS）是RSS的软件实现。在软件中，必须稍后在数据路径中调用它。RSS选择队列，从而选择了运行硬件中断处理程序的CPU；而RPS在中断处理程序之上选择CPU执行协议处理。这是通过将数据包放置在所需的CPU积压队列中并唤醒CPU进行处理来实现的。

与RSS相比，RPS具有一些优点：

1. 它可以与任何网卡一起使用
2. 可以轻松地将软件过滤器添加到新协议的哈希中
3. 它不会增加硬件设备的中断率（尽管确实会引入处理器间中断（IPI））

RPS is called during bottom half of the receive interrupt handler, when a driver sends a packet up the network stack with netif_rx() or netif_receive_skb(). These call the get_rps_cpu() function, which
selects the queue that should process a packet.

当驱动程序使用`netif_rx()`或`netif_receive_skb()`在网络堆栈上发送数据包时，将在接收中断处理程序的下半部分调用RPS 。这些调用get_rps_cpu()函数，该函数选择应处理数据包的队列。



The first step in determining the target CPU for RPS is to calculate a flow hash over the packet addresses or ports (2-tuple or 4-tuple hash depending on the protocol). This serves as a consistent hash of the associated flow of the packet. The hash is either provided by hardware or will be computed in the stack. Capable hardware can pass the hash in the receive descriptor for the packet; this would usually be the same hash used for RSS (e.g. computed Toeplitz hash). The hash is saved in skb->hash and can be used elsewhere in the stack as a hash of the packet flow.

确定RPS的目标CPU的第一步是根据数据包的地址或端口的计算出流向的哈希值（取决于协议使用2元组或4元组哈希）。这个哈希值会持续用做数据包流的持续哈希。哈希由硬件提供或将在堆栈中计算得到。有能力的硬件可以通过数据包的接收描述符中传递哈希值；这通常与用于RSS的哈希相同（例如，计算的Toeplitz哈希）。哈希保存在skb-> hash中，并且可以在堆栈的其他地方用作数据包流的哈希。

Each receive hardware queue has an associated list of CPUs to which RPS may enqueue packets for processing. For each received packet, an index into the list is computed from the flow hash modulo the size
of the list. The indexed CPU is the target for processing the packet, and the packet is queued to the tail of that CPU backlog queue. Atthe end of the bottom half routine, IPIs are sent to any CPUs for which packets have been queued to their backlog queue. The IPI wakes backlog processing on the remote CPU, and any queued packets are then processed up the networking stack.

每个接收硬件队列都有一个相关的CPU列表，RPS可以将要处理的数据包排队到这些CPU列表中。对于每个接收到的数据包，根据以列表大小对流散列求模来计算列表的索引。带索引的CPU是处理数据包的目标，数据包排队到该CPU的待办事项队列的末尾。在下半部分例程的末尾，将IPI发送到已将数据包排队到其积压队列的任何CPU。IPI唤醒远程CPU上的积压处理，然后将所有排队的数据包在网络堆栈上处理。

RPS Configuration

RPS配置

RPS requires a kernel compiled with the CONFIG_RPS kconfig symbol (on by default for SMP). Even when compiled in, RPS remains disabled until explicitly configured. The list of CPUs to which RPS may forward traffic can be configured for each receive queue using a sysfs file entry:

RPS需要使用CONFIG_RPS kconfig符号编译的内核（SMP默认情况下处于启用状态）。即使在编译到内核，RPS仍保持禁用状态，直到显式配置RPS。可以使用sysfs文件条目为每个接收队列配置RPS可能会将流量转发到的CPU列表：

`/sys/class/net/<dev>/queues/rx-<n>/rps_cpus`

This file implements a bitmap of CPUs. RPS is disabled when it is zero (the default), in which case packets are processed on the interrupting CPU. Documentation/IRQ-affinity.txt explains how CPUs are assigned to
the bitmap.

该文件实现CPU的位图。RPS为零（默认值）时将被禁用，在这种情况下，数据包将在中断的CPU上处理。Documentation / IRQ-affinity.txt解释了如何将CPU分配给位图。

Suggested Configuration

建议配置

For a single queue device, a typical RPS configuration would be to set the rps_cpus to the CPUs in the same memory domain of the interrupting CPU. If NUMA locality is not an issue, this could also be all CPUs in
the system. At high interrupt rate, it might be wise to exclude the interrupting CPU from the map since that already performs much work.

对于单个队列设备，典型的RPS配置是将rps_cpus设置为与中断CPU处于同一存储域中的CPU。如果不考虑NUMA的本地效应，则也可以是系统中的所有CPU。在高中断率的情况下，将中断的CPU从映射中排除是明智的选择，因为该中断已经执行了大量工作。

For a multi-queue system, if RSS is configured so that a hardware receive queue is mapped to each CPU, then RPS is probably redundant and unnecessary. If there are fewer hardware queues than CPUs, then
RPS might be beneficial if the rps_cpus for each queue are the ones that share the same memory domain as the interrupting CPU for that queue.

对于多队列系统，如果配置RSS以便将硬件接收队列映射到每个CPU，则RPS可能是多余的，并且不必要。如果硬件队列少于CPU，那么如果每个队列的rps_cpus与中断该队列的CPU共享相同的内存域的RPS可能会有所帮助。

RPS Flow Limit

RPS流量限制

RPS scales kernel receive processing across CPUs without introducing reordering. The trade-off to sending all packets from the same flow to the same CPU is CPU load imbalance if flows vary in packet rate. In the extreme case a single flow dominates traffic. Especially on common server workloads with many concurrent connections, such behavior indicates a problem such as a misconfiguration or spoofed source Denial of Service attack.

RPS扩展了内核多CPU接收处理的过程，而无需引入重新排序。如果数据包速率不同，则将所有数据包从相同流发送到同一CPU的代价是CPU负载不平衡。在极端情况下，单个流会主导流量。尤其是在具有许多并发连接的普通服务器工作负载上，此类行为表示诸如配置错误或欺骗性的源“拒绝服务”攻击之类的问题。

Flow Limit is an optional RPS feature that prioritizes small flows during CPU contention by dropping packets from large flows slightly ahead of those from small flows. It is active only when an RPS or RFS destination CPU approaches saturation. Once a CPU's input packet queue exceeds half the maximum queue length (as set by sysctl net.core.netdev_max_backlog), the kernel starts a per-flow packet count over the last 256 packets. If a flow exceeds a set ratio (by default, half) of these packets when a new packet arrives, then the new packet is dropped. Packets from other flows are still only dropped once the input packet queue reaches netdev_max_backlog. No packets are dropped when the input packet queue length is below the threshold, so flow limit does not sever connections outright: even large flows maintain connectivity.

流限制是一项可选的RPS功能，该功能在CPU争用期间会先丢弃大流量中的数据包，来优先处理小流量。仅当RPS或RFS目标CPU达到饱和时，此功能才激活。一旦CPU的输入数据包队列超过最大队列长度的一半（通过sysctl net.core.netdev_max_backlog设置），内核就会开始对最后256个数据包进行逐流数据包计数。如果在新数据包到达时，某个流超过数据包的这个设置比率（默认情况下为一半），则将丢弃新数据包。一旦输入数据包队列到达netdev_max_backlog，才会丢弃来自其他流的数据包。当输入数据包队列长度低于阈值时，不会丢弃任何数据包，因此流量限制不会完全切断连接：即使是大流量也可以保持连接性。

Interface

接口

Flow limit is compiled in by default (CONFIG_NET_FLOW_LIMIT), but not turned on. It is implemented for each CPU independently (to avoid lock and cache contention) and toggled per CPU by setting the relevant bit in sysctl net.core.flow_limit_cpu_bitmap. It exposes the same CPU bitmap interface as rps_cpus (see above) when called from procfs:

流量限制默认情况下已编译（CONFIG_NET_FLOW_LIMIT），但未打开。它是为每个CPU独立实现的（以避免锁定和缓存争用），并通过在sysctl net.core.flow_limit_cpu_bitmap中设置相关位来对每个CPU进行开关。通过procfs调用时，它暴露了与rps_cpus相同的CPU位图接口（请参见上文）

`/proc/sys/net/core/flow_limit_cpu_bitmap`

Per-flow rate is calculated by hashing each packet into a hashtable bucket and incrementing a per-bucket counter. The hash function is the same that selects a CPU in RPS, but as the number of buckets can
be much larger than the number of CPUs, flow limit has finer-grained identification of large flows and fewer false positives. The default table has 4096 buckets. This value can be modified through sysctl

通过将每个数据包散列到哈希表桶中并递增每个桶计数器，可以计算出每个流速率。散列函数与在RPS中选择CPU的功能相同，但是由于存储桶的数量可能远远大于CPU的数量，因此流量限制可以更好地识别大流量，并减少误报。默认表有4096个存储桶。可以通过sysctl修改此值：

`net.core.flow_limit_table_len`

The value is only consulted when a new table is allocated. Modifyingit does not update active tables.

仅在分配新表时才查询该值。对其进行修改不会更新活动表。

Suggested Configuration

建议配置

Flow limit is useful on systems with many concurrent connections, where a single connection taking up 50% of a CPU indicates a problem. In such environments, enable the feature on all CPUs that handle network rx interrupts (as set in /proc/irq/N/smp_affinity).

流量限制在具有多个并发连接的系统上非常有用，其中单个连接占用50%的CPU表示有问题。在这种环境中，请在处理网络rx中断的所有CPU上启用该功能（在/proc/irq/N/smp_affinity中设置）。

The feature depends on the input packet queue length to exceed he flow limit threshold (50%) + the flow history length (256). Setting net.core.netdev_max_backlog to either 1000 or 10000 performed well in experiments.

该功能取决于输入数据包队列长度是否超过流量限制阈值（50%）+流历史记录长度（256）。在实验中，将net.core.netdev_max_backlog设置为1000或10000效果很好。

RFS: Receive Flow Steering

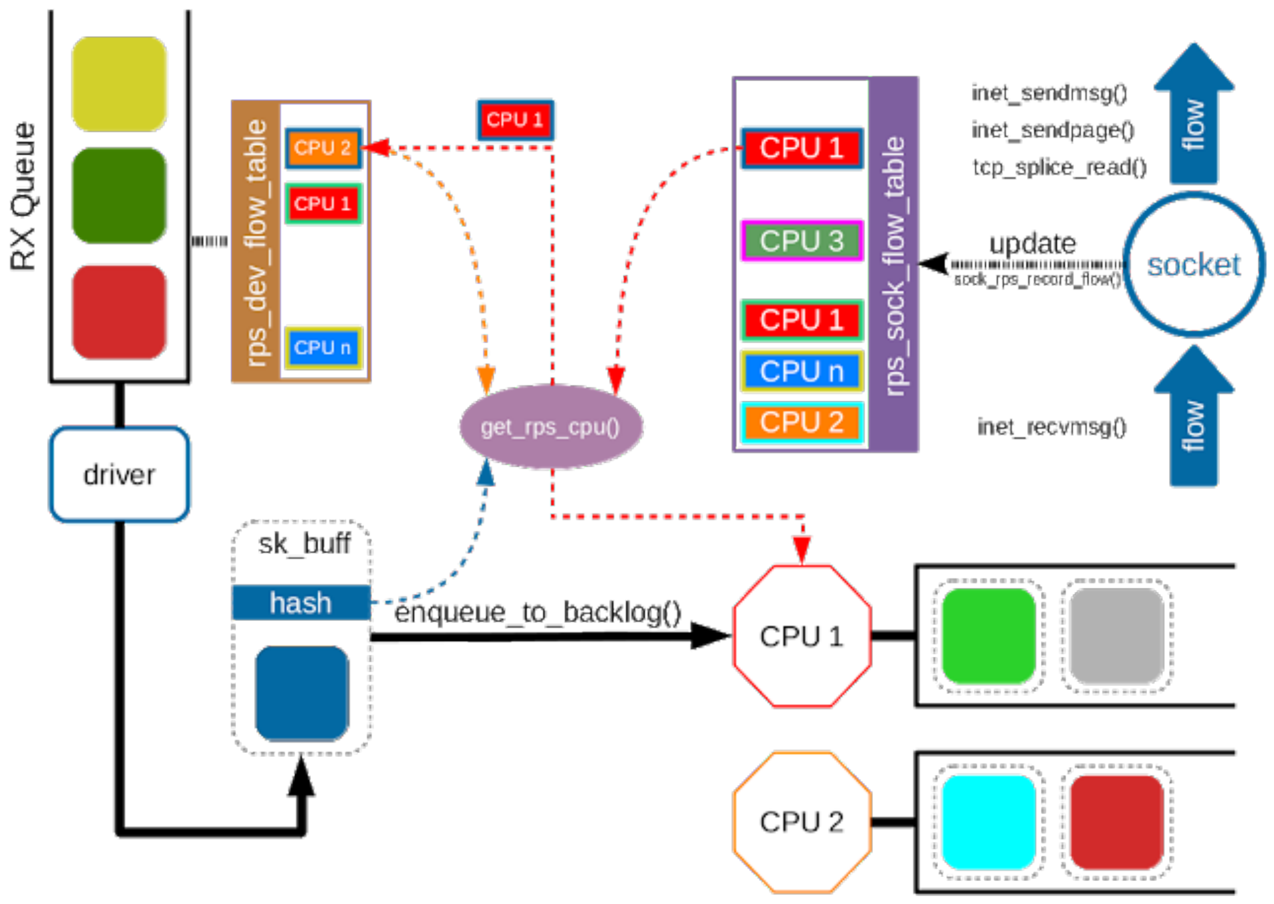
RFS：接收流控制

While RPS steers packets solely based on hash, and thus generally provides good load distribution, it does not take into account application locality. This is accomplished by Receive Flow Steering
(RFS). The goal of RFS is to increase datacache hit rate by steering kernel processing of packets to the CPU where the application thread consuming the packet is running. RFS relies on the same RPS mechanisms
to enqueue packets onto the backlog of another CPU and to wake up that CPU.

RPS仅基于散列来引导数据包，从而通常可以提供良好的负载分配，但它并未考虑应用程序的局部性。这是通过接收流控制（RFS）来完成的。RFS的目标是通过将数据包在内核中的处理引导到使用该数据包的应用程序线程对应的CPU来提高数据缓存的命中率。RFS依靠RPS相同的机制将数据包排队到另一个CPU的待办事项中，并唤醒该CPU。

In RFS, packets are not forwarded directly by the value of their hash, but the hash is used as index into a flow lookup table. This table maps flows to the CPUs where those flows are being processed. The flow hash
(see RPS section above) is used to calculate the index into this table. The CPU recorded in each entry is the one which last processed the flow. If an entry does not hold a valid CPU, then packets mapped to that entry
are steered using plain RPS. Multiple table entries may point to the same CPU. Indeed, with many flows and few CPUs, it is very likely that a single application thread handles flows with many different flow hashes.

在RFS中，数据包不通过其哈希值直接转发，而是将哈希用作流查找表的索引。该表将流映射到正在处理这些流的CPU。流哈希（请参阅上面的RPS部分）用于计算此表中的索引。每个条目中记录的CPU是最后处理该流程的CPU。如果某个条目没有有效的CPU，则直接使用RPS引导映射到该条目的数据包。多个表条目可能指向同一CPU。确实，在具有许多流和少量CPU的情况下，单个应用程序线程很有可能使用许多不同的流散列来处理流。



rps_sock_flow_table is a global flow table that contains the *desired* CPU for flows: the CPU that is currently processing the flow in userspace. Each table value is a CPU index that is updated during calls to recvmsg
and sendmsg (specifically, inet_recvmsg(), inet_sendmsg(), inet_sendpage() and tcp_splice_read()).

rps_sock_flow_table是一个全局流表，其中包含流*所需的* CPU：当前正在用户空间中处理流的CPU。每个表值都是一个CPU索引，在调用recvmsg和sendmsg（特别是inet_recvmsg()，inet_sendmsg()，inet_sendpage()和tcp_splice_read()）期间将对其进行更新。

When the scheduler moves a thread to a new CPU while it has outstanding receive packets on the old CPU, packets may arrive out of order. To avoid this, RFS uses a second flow table to track outstanding packets
for each flow: rps_dev_flow_table is a table specific to each hardware receive queue of each device. Each table value stores a CPU index and a counter. The CPU index represents the *current* CPU onto which packets for this flow are enqueued for further kernel processing. Ideally, kernel and userspace processing occur on the same CPU, and hence the CPU index in both tables is identical. This is likely false if the scheduler has recently migrated a userspace thread while the kernel still has packets enqueued for kernel processing on the old CPU.

当调度程序将线程移动到新的CPU时，旧的CPU上仍有未完成的接收数据包时，数据包可能会乱序到达。为了避免这种情况，RFS使用第二个流表来跟踪每个流的未完成数据包：rps_dev_flow_table表指定了每个设备的每个硬件接收队列。每个表值都存储一个CPU索引和一个计数器。CPU索引表示将*当前*流的数据包排入队列以进行进一步内核处理的*当前* CPU。理想情况下，内核和用户空间处理发生在同一CPU上，因此两个表中的CPU索引是相同的。如果调度程序最近迁移了一个用户空间线程，而内核仍有在旧CPU上排队等待处理的数据包，则可能是错误的。

The counter in rps_dev_flow_table values records the length of the current CPU's backlog when a packet in this flow was last enqueued. Each backlog queue has a head counter that is incremented on dequeue. A tail counter is computed as head counter + queue length. In other words, the counter in rps_dev_flow[i] records the last element in flow i that has been enqueued onto the currently designated CPU for flow i (of course, entry i is actually selected by hash and multiple flows may hash to the same entry i).

rps_dev_flow_table值中的计数器记录该流中的最后一个数据包入队时当前CPU待办事项的长度。每个积压队列都有一个头部计数器，该计数器在出队时增加。尾部计数器计算为头部计数器+队列长度。换句话说，rps_dev_flow [i]中的计数器记录流i的最后一个元素，该元素已排队到流i的当前指定CPU上（当然，条目i实际上是由哈希选择的，多个流可能会哈希到同一条目i）。

And now the trick for avoiding out of order packets: when selecting the CPU for packet processing (from get_rps_cpu()) the rps_sock_flow table and the rps_dev_flow table of the queue that the packet was received on are compared. If the desired CPU for the flow (found in the rps_sock_flow table) matches the current CPU (found in the rps_dev_flow table), the packet is enqueued onto that CPU's backlog. If they differ, the current CPU is updated to match the desired CPU if one of the following is true:

- The current CPU's queue head counter >= the recorded tail counter value in rps_dev_flow[i]
- The current CPU is unset (>= nr_cpu_ids)
- The current CPU is offline

现在，避免出现乱序数据包的技巧是：在选择CPU进行数据包处理时（从get_rps_cpu()），比较接收数据包的队列的rps_sock_flow表和rps_dev_flow表。如果流的所需CPU（在rps_sock_flow表中找到）与当前CPU（在rps_dev_flow表中找到）匹配，则将数据包排队到该CPU的待办事项列表中。如果它们不同，则在满足以下条件之一的情况下，将更新当前CPU以匹配所需的CPU：

> - 当前CPU的队列头计数器> = rps_dev_flow [i]中记录的尾计数器值
> - 当前的CPU未设置（> = nr_cpu_ids）
> - 当前CPU**离线**

After this check, the packet is sent to the (possibly updated) current CPU. These rules aim to ensure that a flow only moves to a new CPU when there are no packets outstanding on the old CPU, as the outstanding packets could arrive later than those about to be processed on the new CPU.

进行此检查后，数据包将发送到（可能已更新）当前的CPU。这些规则旨在确保仅当旧CPU上没有未完成的数据包时，流才移至新CPU，因为未完成的数据包可能比新CPU上要处理的数据包晚到达。

RFS Configuration

RFS配置

RFS is only available if the kconfig symbol CONFIG_RPS is enabled (on by default for SMP). The functionality remains disabled until explicitly configured. The number of entries in the global flow table is set through:

仅当启用了kconfig符号CONFIG_RPS时，RFS才可用（默认情况下，SMP处于启用状态）。除非明确配置，否则该功能将保持禁用状态。全局流表中的条目数是通过以下方式设置的：

`/proc/sys/net/core/rps_sock_flow_entries`

The number of entries in the per-queue flow table are set through:

通过以下方式设置每个队列流表中的条目数：

`/sys/class/net/<dev>/queues/rx-<n>/rps_flow_cnt`

Suggested Configuration

建议配置

Both of these need to be set before RFS is enabled for a receive queue. Values for both are rounded up to the nearest power of two. The suggested flow count depends on the expected number of active connections at any given time, which may be significantly less than the number of open connections. We have found that a value of 32768 for rps_sock_flow_entries works fairly well on a moderately loaded server.

在为接收队列启用RFS之前，都需要同时设置这两项。两者的值均四舍五入到最接近的2的幂。建议的流量计数取决于任何给定时间的活动连接的预期数量，该数量可能大大少于打开的连接的数量。我们发现，在中等负载的服务器上，rps_sock_flow_entries的值为32768效果很好。

For a single queue device, the rps_flow_cnt value for the single queue would normally be configured to the same value as rps_sock_flow_entries. For a multi-queue device, the rps_flow_cnt for each queue might be
configured as rps_sock_flow_entries / N, where N is the number of queues. So for instance, if rps_sock_flow_entries is set to 32768 and there are 16 configured receive queues, rps_flow_cnt for each queue might be configured as 2048.

对于单个队列设备，通常将单个队列的rps_flow_cnt值配置为与rps_sock_flow_entries相同的值。对于多队列设备，每个队列的rps_flow_cnt可能配置为rps_sock_flow_entries / N，其中N是队列数。因此，例如，如果rps_sock_flow_entries设置为32768，并且配置了16个接收队列，则每个队列的rps_flow_cnt可以配置为2048。

Accelerated RFS

加速的RFS

Accelerated RFS is to RFS what RSS is to RPS: a hardware-accelerated load balancing mechanism that uses soft state to steer flows based on where the application thread consuming the packets of each flow is running. Accelerated RFS should perform better than RFS since packets are sent directly to a CPU local to the thread consuming the data. The target CPU will either be the same CPU where the application runs, or at least a CPU which is local to the application thread's CPU in the cache hierarchy.

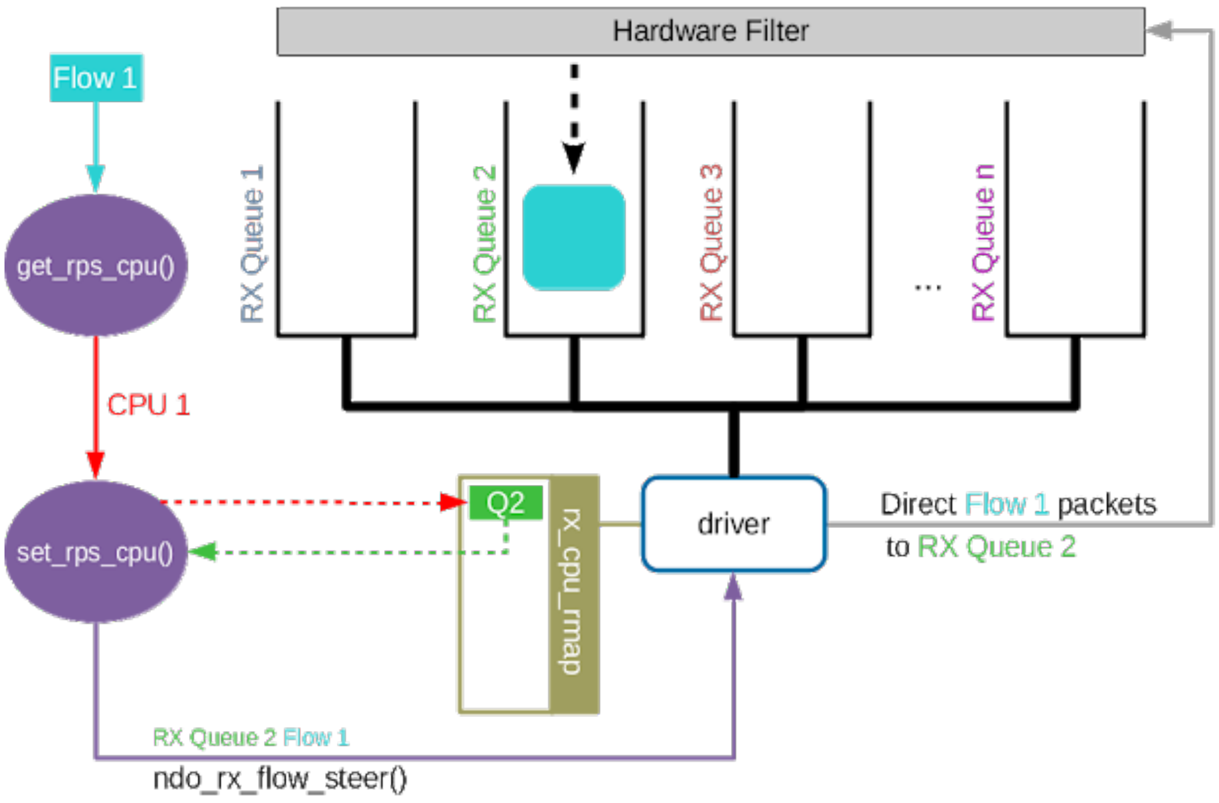
加速的RFS相对于RFS而言，相当于RSS相对于RPS：硬件加速的负载平衡机制，该机制使用软状态根据消费数据包的流对应应用程序线程在何处运行来引导流。加速的RFS应该比RFS更好，因为数据包直接发送到使用数据的线程的本地CPU。目标CPU将是运行应用程序的同一CPU，或者至少是应用程序线程的所在CPU对应的同一高速缓存层次结构中本地的CPU。

To enable accelerated RFS, the networking stack calls the ndo_rx_flow_steer driver function to communicate the desired hardware queue for packets matching a particular flow. The network stack automatically calls this function every time a flow entry in rps_dev_flow_table is updated. The driver in turn uses a device specific method to program the NIC to steer the packets.

为了启用加速的RFS，网络堆栈调用ndo_rx_flow_steer驱动程序函数来和所需的硬件队列通讯，以查找与特定流匹配的数据包。每当rps_dev_flow_table中的流条目更新时，网络堆栈都会自动调用此函数。驱动程序进而使用设备特定的方法对网卡进行编程以引导数据包。

The hardware queue for a flow is derived from the CPU recorded in rps_dev_flow_table. The stack consults a CPU to hardware queue map which is maintained by the NIC driver. This is an auto-generated reverse map of the IRQ affinity table shown by /proc/interrupts. Drivers can use functions in the cpu_rmap (CPU affinity reverse map) kernel library to populate the map. For each CPU, the corresponding queue in the map is set to be one whose processing CPU is closest in cache locality.

流的硬件队列是从rps_dev_flow_table中记录的CPU派生的。网卡驱动程序维护了CPU到硬件队列的映射，网络堆栈需要查询他。这是从/proc/interrupts显示的IRQ亲和力表自动生成的反向映射。驱动程序可以使用cpu_rmap（“ CPU关联性反向映射”）内核库中的函数来填充映射。对于每个CPU，映射中的相应队列设置为处理CPU在缓存位置最接近的队列。



Accelerated RFS Configuration

加速的RFS配置

Accelerated RFS is only available if the kernel is compiled with CONFIG_RFS_ACCEL and support is provided by the NIC device and driver. It also requires that ntuple filtering is enabled via ethtool. The map
of CPU to queues is automatically deduced from the IRQ affinities configured for each receive queue by the driver, so no additional configuration should be necessary.

仅当内核使用CONFIG_RFS_ACCEL编译，且网卡备和驱动程序支持时，加速RFS才可用。它还需要通过ethtool启用ntuple过滤。驱动程序根据每个接收队列配置的IRQ亲和力自动推断出CPU到队列的映射，因此不需要其他配置。

Suggested Configuration

建议配置

This technique should be enabled whenever one wants to use RFS and the NIC supports hardware acceleration.

只要使用RFS并且网卡支持硬件加速，就应该启用此技术。

XPS: Transmit Packet Steering

XPS：传输数据包导向

Transmit Packet Steering is a mechanism for intelligently selecting which transmit queue to use when transmitting a packet on a multi-queue device. This can be accomplished by recording two kinds of maps, either a mapping of CPU to hardware queue(s) or a mapping of receive queue(s) to hardware transmit queue(s).

传输数据包导向是一种机制，用于在多队列设备上传输数据包时智能地选择要使用的传输队列。这可以通过记录两种映射来完成，即CPU到硬件队列的映射或接收队列到硬件发送队列的映射。

1. XPS using CPUs map

The goal of this mapping is usually to assign queues exclusively to a subset of CPUs, where the transmit completions for these queues are processed on a CPU within this set. This choice provides two benefits. First, contention on the device queue lock is significantly reduced since fewer CPUs contend for the same queue(contention can be eliminated completely if each CPU has its own transmit queue). Secondly, cache miss rate on transmit completion is reduced, in particular for data cache lines that hold the sk_buff structures.

1. 使用CPU的映射的XPS

该映射的目标通常是将队列专门分配给CPU的一个子集，其中这些队列的传输完成在此子集的一个CPU上进行处理。这种选择有两个好处：首先，由于较少的CPU争用同一队列，因此大大减少了设备队列锁定上的争用（如果每个CPU都有自己的传输队列，则可以完全消除争用）。其次，减少了传输完成时的缓存未命中率，尤其是对于持有sk_buff结构的数据缓存行而言。

2. XPS using receive queues map

This mapping is used to pick transmit queue based on the receive queue(s) map configuration set by the administrator. A set of receive queues can be mapped to a set of transmit queues (many:many), although the common use case is a 1:1 mapping. This will enable sending packets on the same queue associations for transmit and receive. This is useful for busy polling multi-threaded workloads where there are challenges in associating a given CPU to a given application thread. The application threads are not pinned to CPUs and each thread handles packets received on a single queue. The receive queue number is cached in the socket for the connection. In this model, sending the packets on the same transmit queue corresponding to the associated receive queue has benefits in keeping the CPU overhead low. Transmit completion work is locked into the same queue-association that a given application is polling on. This avoids the overhead of triggering an interrupt on another CPU. When the application cleans up the packets during the busy poll, transmit completion may be processed along with it in the same thread context and so result in reduced latency.

2. 使用接收队列映射XPS

此映射用于根据管理员设置的接收队列映射配置来选择发送队列。一组接收队列可以映射到一组发送队列（多对多），尽管通常的使用情况是1：1映射。发送和接收关联到了相同队列，这将在这队列上发送数据包。这对于繁忙的轮询多线程工作负载很有用，因为在将给定CPU与给定应用程序线程相关联方面存在挑战。应用程序线程未固定到CPU，并且每个线程处理在单个队列中接收到的数据包。接收队列号缓存在用于连接的套接字中。在此模型中，在与关联的接收队列相对应的同一发送队列上发送数据包有利于保持较低的CPU开销。发送完成工作被锁定在给定应用程序正在轮询的队列关联中。这样可以避免在另一个CPU上触发中断的开销。当应用程序在忙碌的轮询期间清理数据包时，可以在同一线程上下文中连同它一起处理传输完成，因此可以减少延迟。

XPS is configured per transmit queue by setting a bitmap of CPUs/receive-queues that may use that queue to transmit. The reverse mapping, from CPUs to transmit queues or from receive-queues to transmit queues, is computed and maintained for each network device. When transmitting the first packet in a flow, the function get_xps_queue() is called to select a queue. This function uses the ID of the receive queue
for the socket connection for a match in the receive queue-to-transmit queue lookup table. Alternatively, this function can also use the ID of the running CPU as a key into the CPU-to-queue lookup table. If the
ID matches a single queue, that is used for transmission. If multiple queues match, one is selected by using the flow hash to compute an index into the set. When selecting the transmit queue based on receive queue(s) map, the transmit device is not validated against the receive device as it requires expensive lookup operation in the datapath.

通过设置CPU /接收队列的位图，可以为每个传输队列配置XPS。为每个网络设备计算并维护从CPU到传输队列或从接收队列到传输队列的反向映射。在流中传输第一个数据包时，将调用函数get_xps_queue（）来选择队列。此函数将套接字连接的接收队列的ID用于接收队列到发送队列查找表中的匹配项。或者，此功能也可以使用正在运行的CPU的ID作为CPU->队列查找表的键。如果ID与单个队列匹配，则用于传输。如果有多个队列匹配，则通过使用流哈希值选择一个队列来计算集合中的索引。当基于接收队列映射选择发送队列时，传输设备不会被去验证接受设备，因为这需要在数据路径上做昂贵的查找操作。

The queue chosen for transmitting a particular flow is saved in the corresponding socket structure for the flow (e.g. a TCP connection). This transmit queue is used for subsequent packets sent on the flow to prevent out of order (ooo) packets. The choice also amortizes the cost of calling get_xps_queues() over all packets in the flow. To avoid ooo packets, the queue for a flow can subsequently only be changed if skb->ooo_okay is set for a packet in the flow. This flag indicates that there are no outstanding packets in the flow, so the transmit queue can change without the risk of generating out of order packets. The transport layer is responsible for setting ooo_okay appropriately. TCP, for instance, sets the flag when all data for a connection has been acknowledged.

为传输特定流而选择的队列被保存在该流的相应套接字结构中（例如TCP连接）。此传输队列用于流中发送的后续数据包，以防止数据包乱序（ooo）。该选择还分摊了流中所有数据包调用get_xps_queues()的成本。为了避免出现ooo数据包，仅当为流中的数据包设置了skb-> ooo_okay时，才可以更改该流的队列。此标志指出流中没有未完成的数据包，因此可以更改传输队列而不会产生乱序数据包的风险。传输层负责适当地设置ooo_okay。例如，TCP会在确认连接的所有数据后设置标志。

XPS Configuration

XPS配置

XPS is only available if the kconfig symbol CONFIG_XPS is enabled (on by default for SMP). The functionality remains disabled until explicitly configured. To enable XPS, the bitmap of CPUs/receive-queues that may use a transmit queue is configured using the sysfs file entry:

仅当启用了kconfig符号CONFIG_XPS时，XPS才可用（默认情况下，SMP处于启用状态）。除非明确配置，否则该功能将保持禁用状态。要启用XPS，可使用sysfs文件条目来配置使用传输队列的CPU /接收队列的位图：

For selection based on CPUs map:

如果选择基于CPU映射：

`/sys/class/net/<dev>/queues/tx-<n>/xps_cpus`

如果选择基于接收队列映射：
`/sys/class/net/<dev>/queues/tx-<n>/xps_rxqs`

Suggested Configuration

建议配置

For a network device with a single transmission queue, XPS configuration has no effect, since there is no choice in this case. In a multi-queue system, XPS is preferably configured so that each CPU maps onto one queue. If there are as many queues as there are CPUs in the system, then each queue can also map onto one CPU, resulting in exclusive pairings that experience no contention. If there are fewer queues than CPUs, then the best CPUs to share a given queue are probably those that share the cache with the CPU that processes transmit completions for that queue (transmit interrupts).

对于具有单个传输队列的网络设备，XPS配置无效，因为在这种情况下没有选择。在多队列系统中，最好配置XPS，以便每个CPU映射到一个队列。如果队列与系统中的CPU数量一样多，那么每个队列也可以映射到一个CPU，从而导致互不竞争的互斥配对。如果队列少于CPU，则共享给定队列的最佳CPU可能是与处理该队列的传输完成（传输中断）的处理器共享高速缓存的CPU。

For transmit queue selection based on receive queue(s), XPS has to be explicitly configured mapping receive-queue(s) to transmit queue(s). If the user configuration for receive-queue map does not apply, then the transmit queue is selected based on the CPUs map.

对于基于接收队列的发送队列选择，必须明确配置XPS，将接收队列映射到发送队列。如果接收队列映射的用户配置不适用，则根据CPU映射选择传输队列。

Per TX Queue rate limitation:

每个TX队列速率限制

These are rate-limitation mechanisms implemented by HW, where currently a max-rate attribute is supported, by setting a Mbps value to

这些是硬件实施的速率限制机制，当前通过将Mbps值设置为来支持max-rate属性：

`/sys/class/net/<dev>/queues/tx-<n>/tx_maxrate`

A value of zero means disabled, and this is the default.

零值表示禁用，并且这是默认值。

Further Information

更多的信息

RPS and RFS were introduced in kernel 2.6.35. XPS was incorporated into 2.6.38. Original patches were submitted by Tom Herbert
(therbert@google.com)

RPS和RFS在内核2.6.35中引入。XPS被合并到2.6.38中。原补丁由汤姆·赫伯特提交。

Accelerated RFS was introduced in 2.6.35. Original patches were submitted by Ben Hutchings (bwh@kernel.org)

Authors:

Tom Herbert (therbert@google.com)
Willem de Bruijn (willemb@google.com)

翻译：

hugulas chen (hugulas@163.com)