# 7.轮询模式驱动

DPDK包括1Gigabit、10Gigabit 及 40Gigabit 和半虚拟化IO的轮询模式驱动程序。

轮询模式驱动程序(PMD)由通过在用户空间中运行的BSD驱动提供的API组成，以配置设备及它们的队列。此外，PMD直接访问 RX 和 TX 描述符，且不会有任何中断（链路状态更改中断除外）产生，这可以保证在用户空间应用程序中快速接收，处理和传送数据包。本节介绍PMD的要求、设计原则和高级架构，并介绍了以太网PMD的对外通用API。

# 7.1.前提与假设

DPDK环境支持两种模式的数据包处理，RTC模式和pipeline模式：

* 在 run-to-completion 模式中，通过调用API来轮询指定端口的RX描述符以获取报文。 紧接着，在同一个core上处理报文，并通过API调用将报文放到接口的TX描述符中以发送报文。
* 在 pipe-line 模式中，一个core轮询一个或多个接口的RX描述符以获取报文。然后报文经由ring被其他core处理。其他core可以继续处理报文，最终报文被放到TX描述符中以发送出去。

在同步的run-to-complete模式中，每个逻辑核处理数据包的流程包括以下步骤：

* 通过PMD报文接收API来获取报文
* 一次性处理每个数据报文，直到转发阶段
* 通过PMD发包API将报文发送出去

相反地，在异步的pipeline模式中，一些逻辑核可能专门用于接收报文，其他逻辑核用于处理前面收到的报文。收到的数据包通过报文ring在逻辑核之间交换。数据包收包过程包括以下步骤：

* 通过PMD收包API获取报文
* 通过数据包队列向逻辑核提供接收到的数据包

数据包处理过程包括以下步骤：

* 从数据包队列中获取数据包
* 处理接收到的数据包，直到重新发送出去

为了避免任何不必要的中断处理开销，执行环境不得使用任何异步通知机制。即便有需要，也应该尽量使用ring来引入通知信息。

在多核环境中避免锁竞争是一个关键问题。为了解决这个问题，PMD旨在尽可能地使用每个core的私有资源。例如，PMD每个端口维护每个core单独的传输队列。同样的，端口的每个接收队列都被分配给单个逻辑核并由其轮询。

为了兼容NUMA架构，内存管理旨在为每个逻辑核分配本地（相同插槽）中的专用缓冲池，以最大限度地减少远程内存访问。数据包缓冲池的配置应该考虑到DIMMs、channels和ranks等底层物理内存架构。应用程序必须确保在内存池创建时给出合适的参数。具体内容参阅内存池库。

## 7.2.设计原则

Ethernet\* PMDs的API和架构设计遵考虑到以下原则。

PMDs必须能够帮助上层的应用实现全局的策略。反之，不能阻止或妨碍上层应用的实施。

例如，PMD的发送和接收函数都有大量的报文或描述符需要轮询。这允许RTC处理协议栈通过不同的全局循环策略静态修护或动态调整其行为如：

* 立即接收，处理并以零碎的方式一次传送数据包。
* 尽可能所的接收数据包，然后处理所有数据包，再发送。
* 接收给定的最大量的数据包，处理接收的数据包，累加，最后将累加的数据包发送出去。

为了实现最优性能，需要考考整体软件设计选择和纯软件优化技术，并与可用的低层次硬件优化功能（如CPU缓存属性、总线速度、NIC PCI带宽等）进行考虑和平衡。报文传输的情况就是突发性网络报文处理是软硬件权衡问题的一个例子。 在初始情况下，PMD只能导出一个 rte\_eth\_tx\_one 函数，以便在给定的队列上一次传输一个数据包。最重要的是，可以轻松构建一个 rte\_eth\_tx\_burst 函数，循环调用 rte\_eth\_tx\_one 函数以便一次传输多个数据包。然而，PMD有效地实现了 rte\_eth\_tx\_burst 函数，以通过以下优化来最小化每个数据包的驱动级传输开销：

* 在多个数据包之间共享调用 rte\_eth\_tx\_one 函数的非摊销成本。
* 启用 rte\_eth\_tx\_burst 函数以利用burst-oriented 硬件特性（缓存数据预取、使用NIC头/尾寄存器）以最小化每个数据包的CPU周期数， 例如，通过避免对环形缓传输描述符的不必要的读取寄存器访问，或通过系统地使用精确匹配告诉缓存行边界大小的指针数组。
* 使用burst-oriented软件优化技术来移除失败的操作结果，如ring索引的回滚。

还通过API引入了Burst-oriented函数，这些函数在PMD服务中密集使用。这些函数特别适用于NIC ring的缓冲区分配器，他们提供一次分配/释放多个缓冲区的功能。例如，一个 mbuf\_multiple\_alloc 函数返回一个指向rte\_mbuf 缓冲区的指针数组，它可以在向ring添加多个描述符来加速PMD的接收轮询功能。

## 7.3.逻辑核、内存和网络接口卡队列的关系

当处理器的逻辑核和接口利用其本地存储时，DPDK提供NUMA支持，以提供更好的性能。因此，与本地PCIE接口相关的mbuf分配应从本地内存中创建的内存池中申请。如果可能，缓冲区应该保留在本地处理器上以获取最佳性能，并且应使用从本地内存中分配的mempool中申请的mbuf来填充RX和TX缓冲区描述符。

如果数据包或数据操作在本地内存中，而不是在远程处理器内存上，则RTC模型也会运行得更好。 只要所有使用的逻辑核位于同一个处理器上，pipeline模型也将获得更好的性能。

所个逻辑核不应共享接口的接收或发送队列，因为这将需要全局上锁保护，而导致性能下降。

## 7.4.设备标识和配置

### 7.4.1.设备标识

每个NIC端口（总线/桥、设备、功能）由其PCI标识符唯一指定。该PCI标识符在DPDK初始化时执行的PCI探测/枚举功能分配。根据PCI标识符，NIC端口被分配了两个其他的表示：

* 一个端口索引，用于在PMD API导出的所有函数中指定NIC端口
* 端口名称，用于在控制消息中指定端口，主要用于管理和调试目的。为了便于使用，端口名称包括端口索引。

### 7.4.2.设备配置

每个NIC端口的配置包括以下步骤：

* 分配 PCI 资源
* 将硬件复位为公知的默认状态
* 设置PHY和链路
* 初始化统计计数器

PMD API还必须导出函数用于启动/终止端口的全部组播功能，并且可以在混杂模式下设置/取消设置端口。

某些硬件卸载功能必须通过特定的配置参数在端口初始化时单独配置。例如，接收侧缩放（RSS）和数据中心桥接（DCB）功能就是这种情况。

### 7.4.3.即时配置

所有可以“即时”启动或停止的设备功能（即不停止设备），无需PMD API来导出函数实现这些功能。

所需要的是只是设备PCI寄存器的映射地址，以在驱动程序之外使用特殊的函数来配置实现这些功能。

为此，PMD API导出一个函数提供可用于在驱动程序外部设置给定设备功能的设备相关联的所有信息。这些信息包括PCI供应商标识符，PCI设备标识符，PCI设备寄存器的映射地址以及驱动程序的名称。

这种方法的主要优点是可以自由地选择API来启动、配置、停止这些设备功能。

例如，testpmd应用程序中的英特尔?82576千兆以太网控制器和英特尔?82599万兆以太网控制器控制器的IEEE1588功能配置。

可以以相同的方式配置端口的L3 / L4 5-Tuple包过滤功能等其他功能。以太网流控（暂停帧）可以在单个端口上进行配置。有关详细信息，请参阅testpmd源代码。此外，只要数据包mbuf设置正确，就可以为单个数据包启用网卡的L4（UDP / TCP / SCTP）校验和卸载。相关详细信息，请参阅硬件offload 。

### 7.4.4.发送队列配置

每个传输队列都独立配置了以下信息：

* 发送环上的描述符数目
* NUMA架构中，用于标识从哪个socket的DMA存储区分配传输环的标识
* 传输队列的Prefetch，Host及Write-Back阈值寄存器的值
* 传输报文释放的最小阈值。当用于传输数据包的描述符数量超过此阈值时，应检查网络适配器以查看是否有回写描述符。在TX队列配置期间可以传递0，以指示应使用默认值。tx\_free\_thresh的默认值为32。这使得PMD不会去检索完成的描述符，直到NIC已经为此队列处理了32个报文。
* RS位最小阈值。在发送描述符中设置报告状态（RS）位之前要使用的最小发送描述符数。请注意，此参数仅适用于Intel 10GbE网络适配器。如果从最后一个RS位设置开始使用的描述符数量（直到用于发送数据包的第一个描述符）超过发送RS位阈值（tx\_rs\_thresh），则RS位被设置在用于发送数据包的最后一个描述符上。简而言之，此参数控制网络适配器将哪些传输描述符写回主机内存。在TX队列配置期间可以传递值为0，以指示应使用默认值。tx\_rs\_thresh的默认值为32。这确保在网络适配器回写最近使用的描述符之前至少使用32个描述符。这样可以节省TX描述符回写所产生的上游PCIe \*带宽。重要的是注意，当tx\_rs\_thresh大于1时，应将TX写回阈值（TX wthresh）设置为0。有关更多详细信息，请参阅英特尔?82599万兆以太网控制器数据手册。

对于tx\_free\_thresh和tx\_rs\_thresh，必须满足以下约束：

* tx\_rs\_thresh必须大于0。
* tx\_rs\_thresh必须小于环的大小减去2。
* tx\_rs\_thresh必须小于或等于tx\_free\_thresh。
* tx\_free\_thresh必须大于0。
* tx\_free\_thresh必须小于环的大小减去3。
* 为了获得最佳性能，当tx\_rs\_thresh大于1时，TX wthresh应设置为0。

TX环中的一个描述符用作哨兵以避免硬件竞争条件，因此是最大阈值限制。当配置DCB操作时，在端口初始化时，发送队列数和接收队列数必须设置为128。

### 7.4.5.按要求释放TX缓冲区

许多驱动程序并没有在数据包传输后立即将mbuf释放回到mempool或本地缓存中。相反，他们将mbuf留在Tx环中，当需要在Tx环中插入，或者 tx\_rs\_thresh 已经超过时，执行批量释放。

应用程序请求驱动程序通过接口 rte\_eth\_tx\_done\_cleanup() 释放使用的mbuf。该API请求驱动程序释放不再使用的mbufs，而不管tx\_rs\_thresh值是否已超过。有两种情况会使得应用程序可能想要立即释放mbuf：

* 当给定的数据包需要发送到多个目标接口（对于第2层洪泛或第3层多播）。 一种方法是复制数据包或者复制需要操作的数据包头部。 另一种方法是发送数据包，然后轮询 rte\_eth\_tx\_done\_cleanup() 接口直到报文引用递减。 接下来，这个报文就可以发送到下一个目的接口。 该应用程序仍然负责管理不同目标接口之间所需的任何数据包操作，但可以避免数据复制。 该API独立于数据包是传输还是丢弃，只是mbuf不再被接口使用。
* 一些应用程序被设计为进行多次运行，如数据包生成器。 为了运行的性能原因和一致性，应用程序可能希望在每个运行之间重新设置为初始状态，其中所有mbufs都返回到mempool。 在这种情况下，它可以为其已使用的每个目标接口调 rte\_eth\_tx\_done\_cleanup() API 以请求它释放所有使用的mbuf。

要确定驱动程序是否支持该API，请检查 Network Interface Controller Drivers 文档中的Free Tx mbuf on demand功能。

### 7.4.6.硬件offload

根据 rte\_eth\_dev\_info\_get() 提供的驱动程序功能，PMD可能支持硬件offload功能，如校验和TCP分段或VLAN插入。

这些offload功能的支持意味着将专用状态位和值字段添加到rte\_mbuf数据结构中，以及由每个PMD导出的接收/发送功能的适当处理。标记列表及其精确含义在mbuf API文档及报文缓冲区库中元数据章节有描述。

## 7.5.轮询模式驱动API

### 7.5.1.概述

默认情况下，PMD提供的所有外部函数都是无锁函数，这些函数假定在同一目标设备上不会在不同的逻辑core上并行调用。例如，PMD接收函数不能再两个逻辑核上并行调用，以轮询相同端口的相同RX队列。当然，这个函数可以由不同的RX队列上的不同逻辑核并行调用。上级应用程序应该保证强制执行这条规则。

如果需要，多个逻辑核到并行队列的并行访问可以通过专门的在线加锁来显式保护，这些加锁函数是建立在相应的无锁API之上的。

### 7.5.2.通用分组表示

数据包由数据结构 rte\_mbuf 表示，这是一个包含所有必要信息的通用元数据结构。这些信息包括与硬件特征相对应的字段和状态位，如IP头部和VLAN标签的校验和。

数据结构rte\_mbuf包括以通用方式表示网络控制器提供的硬件功能对应的字段。对于输入数据包，rte\_mbuf 的大部分字段都由PMD来填充，包括接收描述符中的信息。相反，对于输出数据包，rte\_mbuf的大部分字段由PMD发送函数用于初始化发送描述符。

数据结构 mbuf 的更全面的描述，请参阅 Mbuf Library 章节。

### 7.5.3.以太网设备API

以太网PMD驱动导出的以太网设备API请参阅 《DPDK API 参考手册》 描述。

### 7.5.4.扩展的统计API

扩展的统计API允许每个独立的PMD导出一组唯一的统计信息。每个统计信息狗有三个属性：

* Name：下文描述的用户可读的格式化字符串
* Id：仅表示统计信息的整数
* Value：一个无符号的64bit整数，标识统计结果

请注意，扩展统计信息标识符是驱动程序特定的，因此对于不同的端口可能不一样。API由各种rte\_eth\_xstats \_\*()函数组成，允许应用程序灵活地检索统计信息。

##### 7.5.4.1.用户可读名称命名方案

对于暴露给API的客户端的字符串，存在一个命名方案。这是为了允许刮取API用于感兴趣的统计信息。命名方案使用由单个下划线\_分割的字符串。方案如下：

* direction
* detail 1
* detail 2
* detail n
* unit

常规统计示例字符串如下，符合上面的方案：

* rx\_bytes
* rx\_crc\_errors
* tx\_multicast\_packets

该方案虽然简单，但可以灵活地显示和读取统计字符串中的信息。以下示例说明了命名方案 rx\_packets 的使用。 在这个例子中，字符串被分成两个组件。第一个 rx 表示统计信息与NIC的接收端相关联。 第二个 packets 表示测量单位是数据包。

一个更为复杂的例子是 tx\_size\_128\_to\_255\_packets 。 在这个例子中， tx 表示传输， size 是第一个细节， 128 等表示更多的细节， packets 表示这是一个数据包计数器。

元数据中的一些方案补充如下：

* 如果第一部分不符合 rx 或 tx ，统计计数器与传送或接收不相关。
* 如果第二部分的第一个字母是 q 且这个 q 后跟一个数字，则这个统计数据是特定队列的一部分。

使用队列号的示例如： tx\_q7\_bytes 表示此统计信息适用于队列号7，并表示该队列上传输的字节数。

#### 7.5.4.2.API设计

xstats API使用name，id和value来允许执行查询特定的统计信息。执行查找意味着两件事情：

* 快速路径中的统计信息不执行名称字符串比较
* 允许仅请求感兴趣的统计信息

API通过将统计信息的名称映射到唯一的ID来确保满足这些要求，该唯一ID用作快速路径中的查找Key。API允许应用程序请求一个id值数组，以便PMD只执行所需的计算。预期的用法是应用程序扫描每个统计信息的名称，并且如果对该统计信息感兴趣则缓存该id。在快速路径上，该整数可用于检索id表示的统计信息的实际值。

#### 7.5.4.3. API函数

只输出了少量的函数，用于检索统计信息的数量以及这些统计信息的名称、ID和数值。

* rte\_eth\_xstats\_get\_names\_by\_id()：返回统计信息的名称，当给定一个NULL参数时，函数返回可用的统计数目。
* rte\_eth\_xstats\_get\_id\_by\_name()：搜索与xstat\_name匹配的统计ID。如果找到，则设置id值。
* rte\_eth\_xstats\_get\_by\_id()：根据配提供的ids数组，填充一系列uint64\_t值。如果ids数组为NULL，则返回所有可用的统计信息。

#### 7.5.4.4.API用例

考虑一下，应用程序需要查看丢弃的数据包数目。如果没有数据包被丢弃，由于性能原因，应用程序不会读取任何其他指标。如果数据包丢弃，应用程序将具有一组特定的统计信息。这一组统计信息允许应用程序决定下一步执行的步骤。以下代码片段展示了如何使用xstats API来实现此目标。

第一步就是获取所有统计信息的名称，并列出来。

struct rte\_eth\_xstat\_name \*xstats\_names;uint64\_t \*values;int len, i;

/\* Get number of stats \*/

len = rte\_eth\_xstats\_get\_names\_by\_id(port\_id, NULL, NULL, 0);if (len < 0) {

printf("Cannot get xstats count\n");

goto err;

}

xstats\_names = malloc(sizeof(struct rte\_eth\_xstat\_name) \* len);if (xstats\_names == NULL) {

printf("Cannot allocate memory for xstat names\n");

goto err;

}

/\* Retrieve xstats names, passing NULL for IDs to return all statistics \*/if (len != rte\_eth\_xstats\_get\_names\_by\_id(port\_id, xstats\_names, NULL, len)) {

printf("Cannot get xstat names\n");

goto err;

}

values = malloc(sizeof(values) \* len);if (values == NULL) {

printf("Cannot allocate memory for xstats\n");

goto err;

}

/\* Getting xstats values \*/if (len != rte\_eth\_xstats\_get\_by\_id(port\_id, NULL, values, len)) {

printf("Cannot get xstat values\n");

goto err;

}

/\* Print all xstats names and values \*/for (i = 0; i < len; i++) {

printf("%s: %"PRIu64"\n", xstats\_names[i].name, values[i]);

}

该应用程序可以访问PMD暴露的所有统计信息的名称。应用程序自己决定哪些统计信息是感兴趣的，通过查找名称来缓存这些统计信息的ID。

uint64\_t id;uint64\_t value;const char \*xstat\_name = "rx\_errors";

if(!rte\_eth\_xstats\_get\_id\_by\_name(port\_id, xstat\_name, &id)) {

rte\_eth\_xstats\_get\_by\_id(port\_id, &id, &value, 1);

printf("%s: %"PRIu64"\n", xstat\_name, value);

}else {

printf("Cannot find xstats with a given name\n");

goto err;

}

API为应用程序提供了灵活性，以便它可以使用包含多个ID号的数组查找多个统计信息。这样可以减少检索统计信息的函数调用开销，并使得应用程序更统一查找多个统计信息。

#define APP\_NUM\_STATS 4/\* application cached these ids previously; see above \*/uint64\_t ids\_array[APP\_NUM\_STATS] = {3,4,7,21};uint64\_t value\_array[APP\_NUM\_STATS];

/\* Getting multiple xstats values from array of IDs \*/

rte\_eth\_xstats\_get\_by\_id(port\_id, ids\_array, value\_array, APP\_NUM\_STATS);

uint32\_t i;for(i = 0; i < APP\_NUM\_STATS; i++) {

printf("%d: %"PRIu64"\n", ids\_array[i], value\_array[i]);

}

用于xstats数组查找的API允许应用程序创建多个统计信息组，并使用单个API调用查找这些ID值。作为最终查找结果，应用程序能够实现监视单个统计信息（在这种情况下为rx\_errors）的目标，如果显示数据包被丢弃，则可以使用ID数组轻松地检索组统计信息。