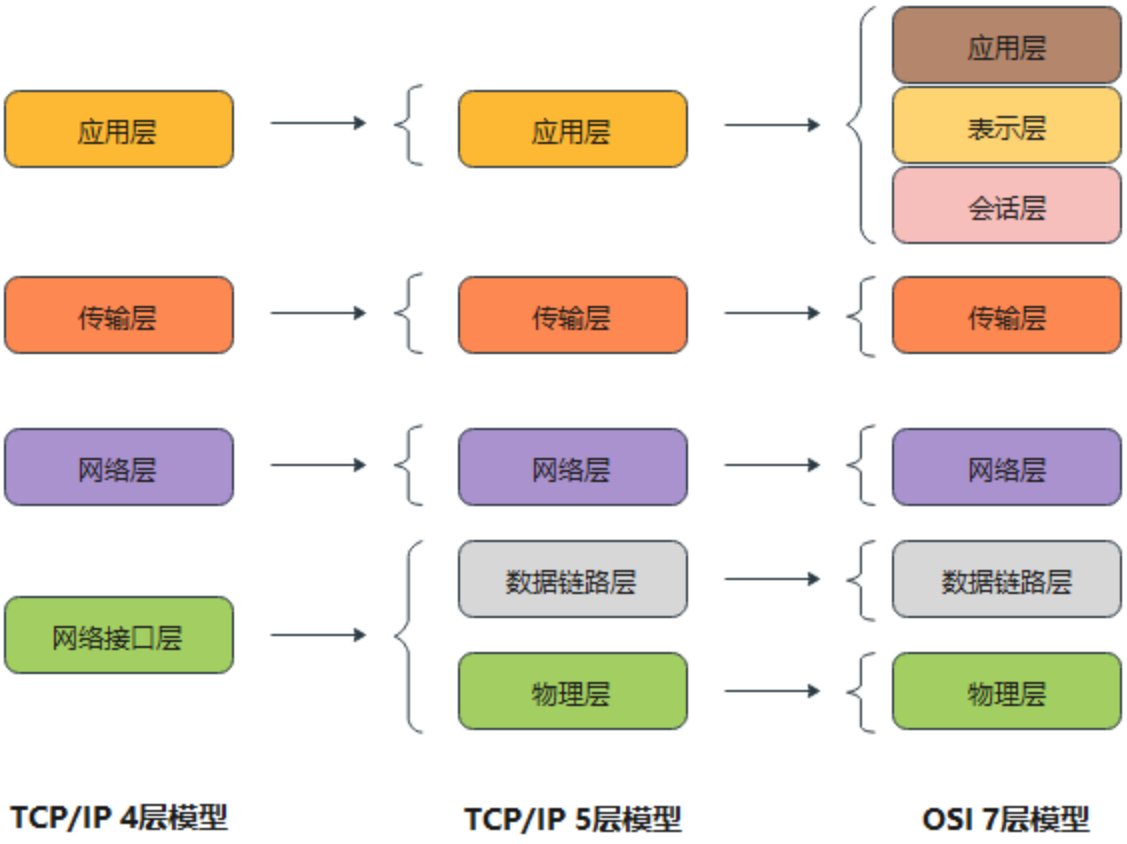
001-Linux数据包接收过程

1. 知识储备

1.1 OSI 网络模型

为了更加了解数据收发过程，有必要将OSI7层模型和TCP/IP4层模型，重新拉出来在鞭尸一遍。



- **物理层**：

解决两个硬件之间怎么通信的问题，常见的物理媒介有光纤、电缆、中继器等。它主要定义物理设备标准，如网线的接口类型、光纤的接口类型、各种传输介质的传输速率等。

它的主要作用是传输比特流（就是由1、0转化为电流强弱来进行传输，到达目的地后在转化为1、0，也就是我们常说的数模转换与模数转换）。这一层的数据叫做比特。

- **数据链路层：**

在计算机网络中由于各种干扰的存在，物理链路是不可靠的。该层的主要功能就是：通过各种控制协议，将有差错的物理信道变为无差错的、能可靠传输数据帧的数据链路。

它的具体工作是接收来自物理层的位流形式的数据，并封装成帧，传送到上一层；同样，也将来自上层的数据帧，拆装为位流形式的数据转发到物理层。这一层的数据叫做帧。

- **网络层：**

计算机网络中如果有多台计算机，怎么找到要发的那台？如果中间有多个节点，怎么选择路径？这就是路由要做的事。

该层的主要任务就是：通过路由选择算法，为报文（该层的数据单位，由上一层数据打包而来）通过通信子网选择最适当的路径。这一层定义的是IP地址，通过IP地址寻址，所以产生了IP协议。

- **传输层：**

当发送大量数据时，很可能会出现丢包的情况，另一台电脑要告诉是否完整接收到全部的包。如果缺了，就告诉丢了哪些包，然后再发一次，直至全部接收为止。

简单来说，**传输层的主要功能就是：监控数据传输服务的质量，保证报文的正确传输**。

- **会话层：**

虽然已经可以实现给正确的计算机，发送正确的封装过后的信息了。但我们总不可能每次都要调用传输层协议去打包，然后再调用IP协议去找路由，所以我们要建立一个自动收发包，自动寻址的功能。于是会话层出现了：它的作用就是建立和管理应用程序之间的通信。

- **表示层：**

表示层负责数据格式的转换，将应用处理的信息转换为适合网络传输的格式，或者将来自下一层的数据转换为上层能处理的格式。

- **应用层：**

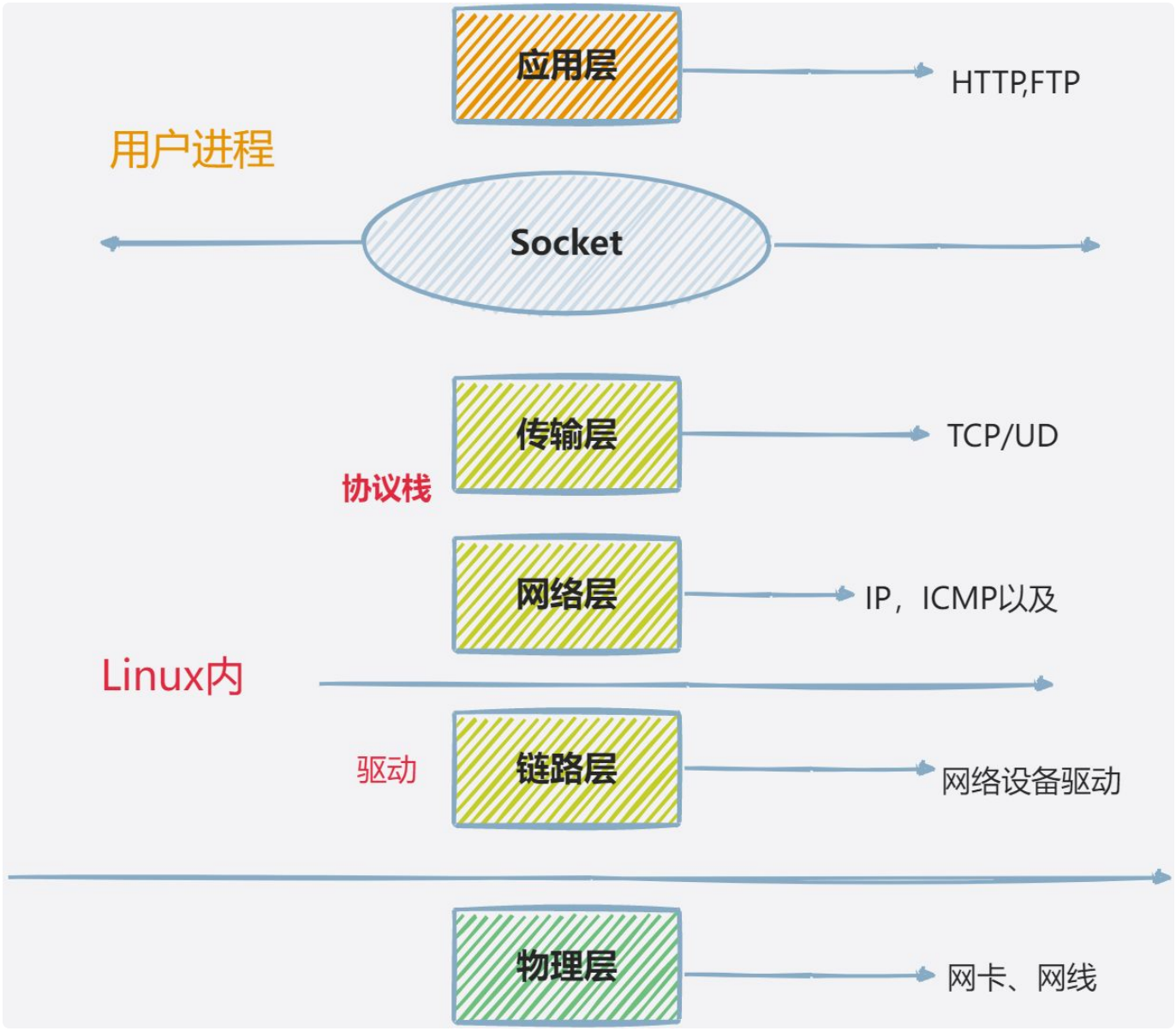
应用层是计算机用户，以及各种应用程序和网络之间的接口，其功能是直接向用户提供服务，完成用户希望在网络上完成的各种工作。前端同学对应用层肯定是最熟悉的。

> 为了简化起见，我们以一个 UDP 数据包在物理网卡上处理流程来介绍 Linux 网络数据包的接收和发送过程

1.2 TCP/IP网络模型

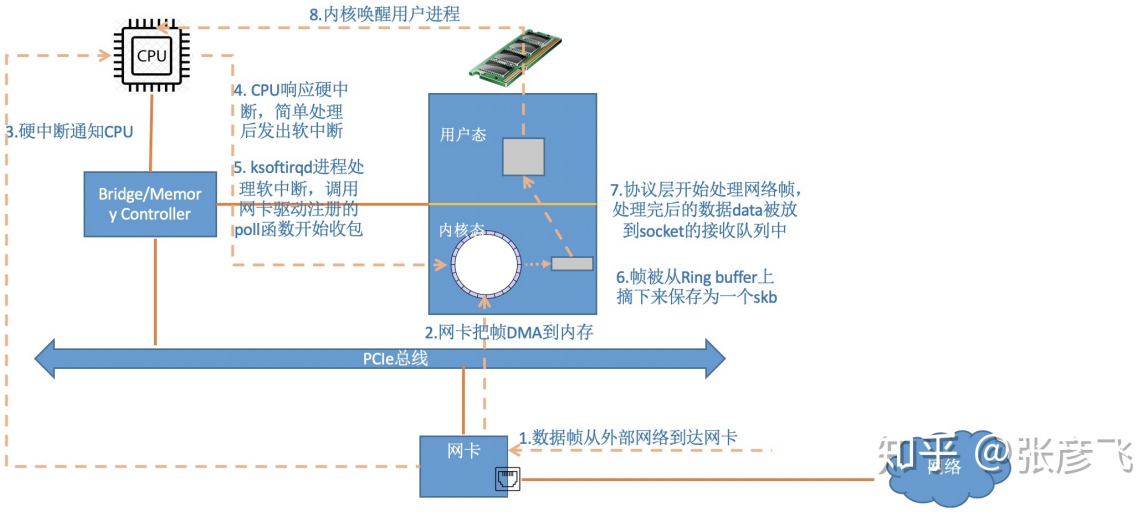
在**TCP/IP5层网络模型**里，整个协议栈被分成了物理层、链路层、网络层，传输层和应用层。**物理层**对应的是网卡和网线，**应用层**对应的是我们常见的Nginx，FTP等等各种应用。**Linux实现的是链路层、网络层和传输层这三层（Linux内核实现）。**

在Linux内核实现中，**链路层协议靠网卡驱动来实现，内核协议栈来实现网络层和传输层**。内核对更上层的应用层提供socket接口来供用户进程访问。我们用Linux的视角来看到的TCP/IP网络分层模型应该是下面这个样子的。



2 Linux网络收包总览

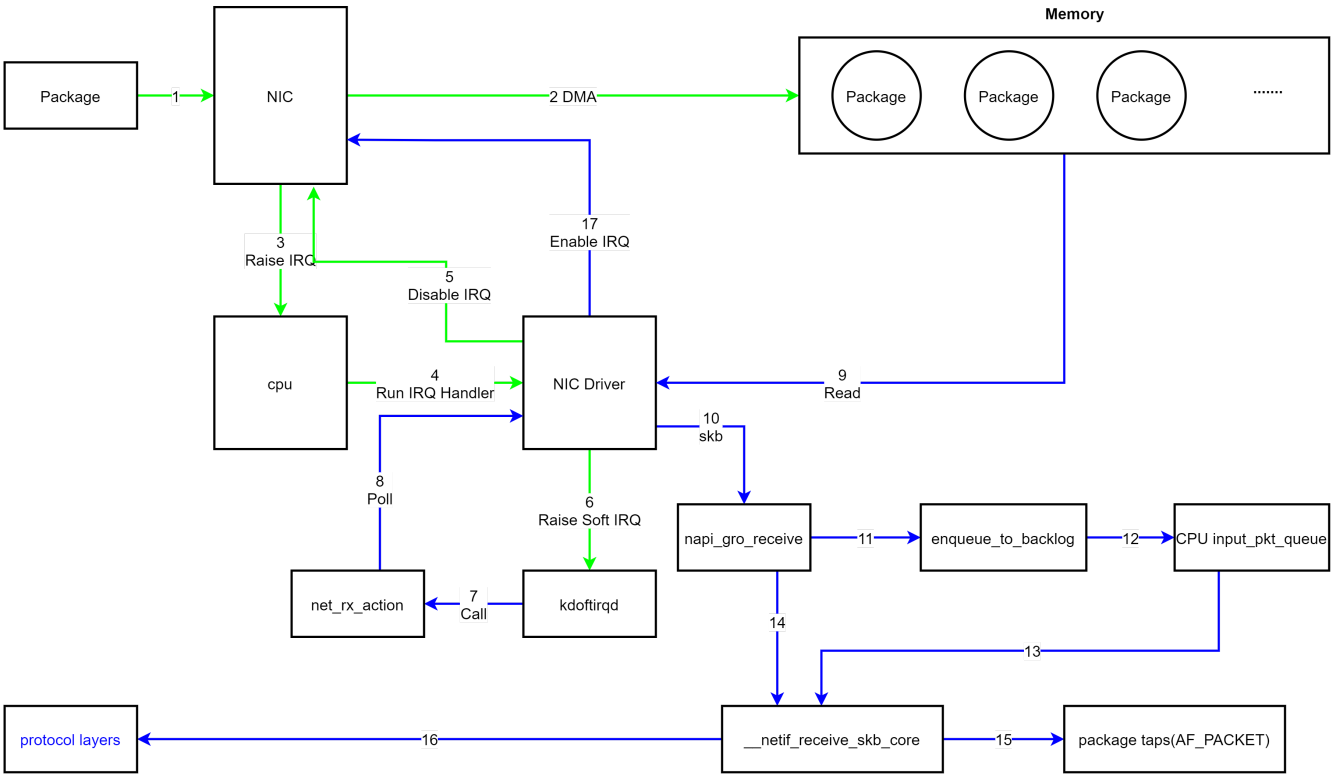
内核和网络设备驱动是通过中断的方式来处理的。当设备上有数据到达的时候，会给CPU的相关引脚上触发一个电压变化，以通知CPU来处理数据。对于网络模块来说，由于处理过程比较复杂和耗时，如果在中断函数中完成所有的处理，将会导致中断处理函数（优先级过高）将过度占据CPU，将导致CPU无法响应其它设备，例如鼠标和键盘的消息。因此Linux中断处理函数是分上半部和下半部的。上半部是只进行最简单的工作，快速处理然后释放CPU，接着CPU就可以允许其它中断进来。剩下将绝大部分的工作都放到下半部中，可以慢慢从容处理。2.4以后的内核版本采用的下半部实现方式是软中断，由ksoftirqd内核线程全权处理。和硬中断不同的是，硬中断是通过给CPU物理引脚施加电压变化，而软中断是通过给内存中的一个变量的二进制值以通知软中断处理程序。



当网卡上收到数据以后，首先会以DMA的方式把网卡上收到的帧写到内存里。再向CPU发起一个中断，以通知CPU有数据到达。当CPU收到中断请求后，会去调用网络驱动注册的中断处理函数。 网卡的中断处理函数并不做过多工作，发出软中断请求，然后尽快释放CPU。ksoftirqd检测到有软中断请求到达，调用poll开始轮询收包，收到后交由各级协议栈处理。对于UDP包来说，会被放到用户socket的接收队列中。

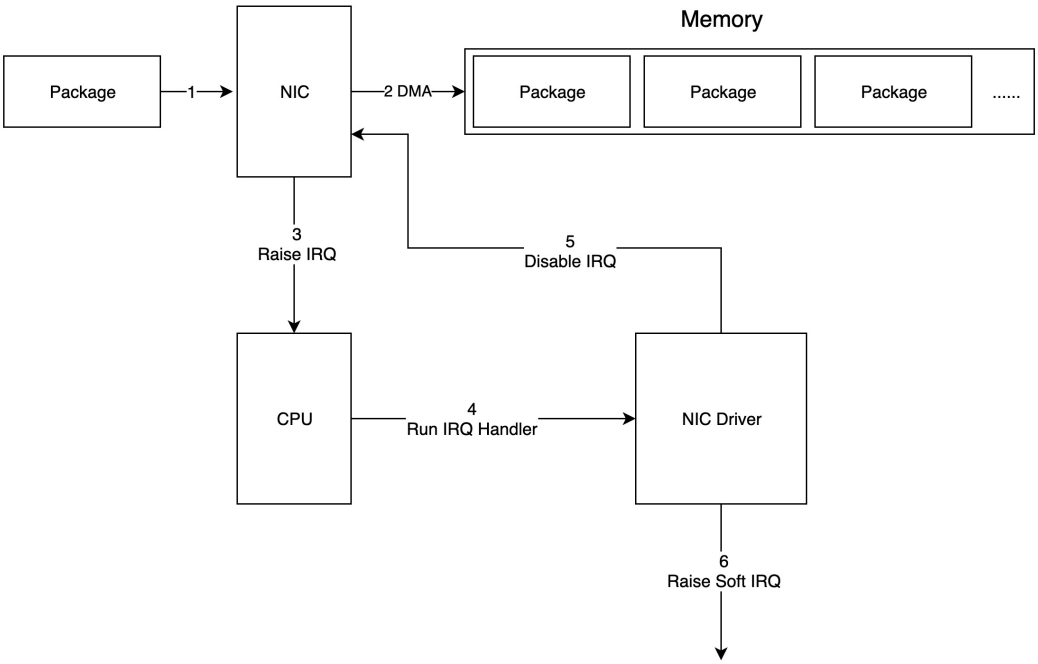
我们从上面这张图中已经从整体上把握到了Linux对数据包的处理过程。

3.Linux收包详细过程



3.1. 从网卡到内存

每个网络设备（网卡）有驱动才能工作，驱动在内核启动时需要加载到内核中。事实上，从逻辑上看，**驱动是负责衔接网络设备和内核网络栈的中间模块，每当网络设备接收到新的数据包时，就会触发中断，而对应的中断处理程序正是加载到内核中的驱动程序。**

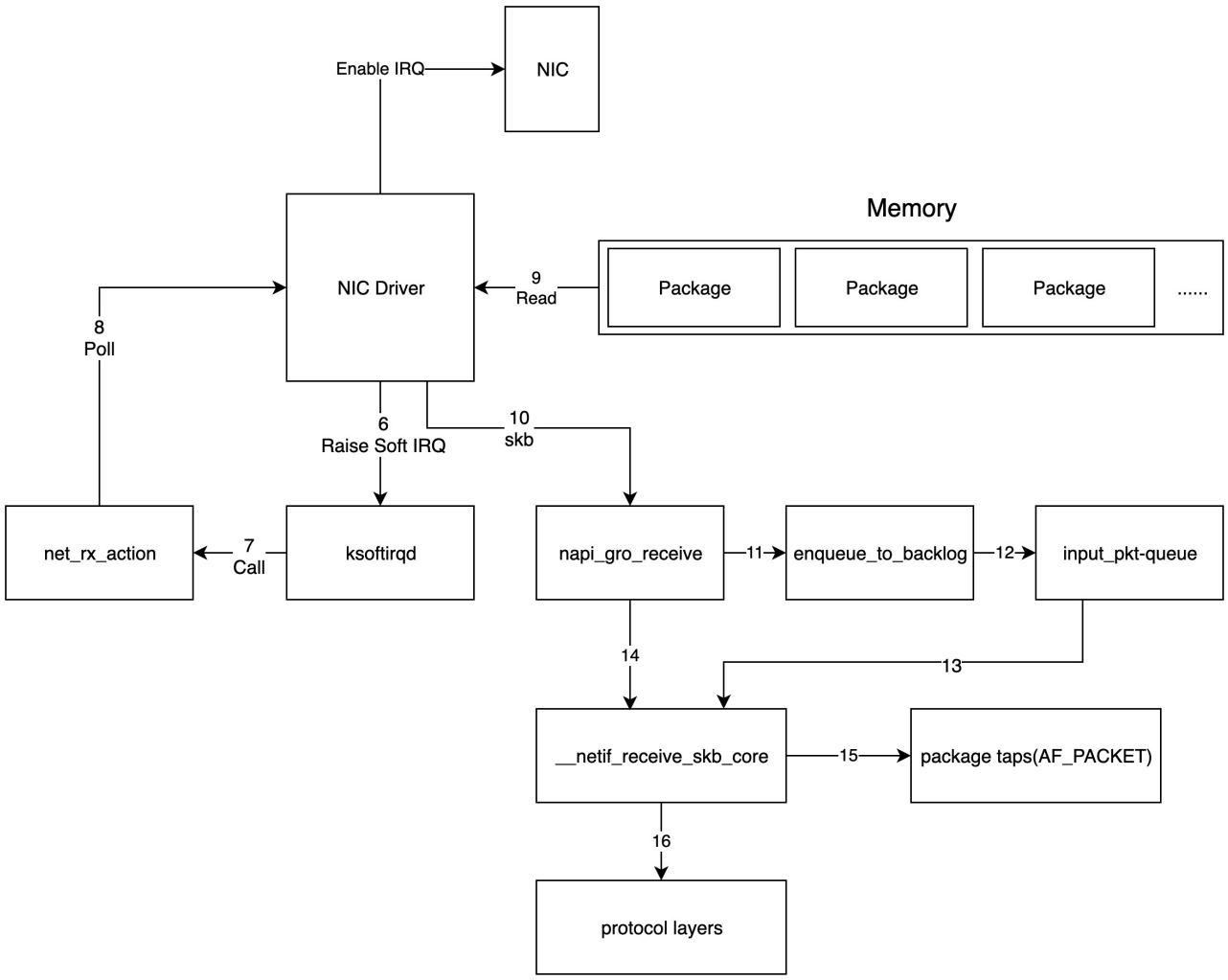


- **1：** 数据包从外面的网络进入物理网卡。如果目的地址不是该网卡，且该网卡没有开启混杂模式，该包会被网卡丢弃。
- **2：** 网卡将数据包通过DMA的方式写入到指定的内存地址，该地址由网卡驱动分配并初始化。注： 老的网卡可能不支持DMA，不过新的网卡一般都支持。
- **3：** 网卡通过硬件中断（IRQ）通知CPU，告诉它有数据来了
- **4：** CPU根据中断表，调用已经注册的中断函数，这个中断函数会调到驱动程序（NIC Driver）中相应的函数
- **5：** 驱动先禁用网卡的中断，表示驱动程序已经知道内存中有数据了，告诉网卡下次再收到数据包直接写内存就可以了，不要再通知CPU了，这样可以提高效率，避免CPU不停的被中断。
- **6：** 启动软中断。这步结束后，硬件中断处理函数就结束返回了。由于硬中断处理程序执行的

过程中不能被中断，所以如果它执行时间过长，会导致CPU没法响应其它硬件的中断，于是内核引入软中断，这样可以将硬中断处理函数中耗时的部分移到软中断处理函数里面来慢慢处理。

3.2 内核的网络模块

软中断会触发内核网络模块中的软中断处理函数，后续流程如下



- 7： 内核中的ksoftirqd进程专门负责软中断的处理，当它收到软中断后，就会调用相应软中断所对应的处理函数，对于上面第6步中是网卡驱动模块抛出的软中断，ksoftirqd会调用网络模块的net_rx_action函数
- 8： net_rx_action调用网卡驱动里的poll函数来一个一个的处理数据包
- 9： 在pool函数中，驱动会一个接一个的读取网卡写到内存中的数据包，内存中数据包的格式

只有驱动知道

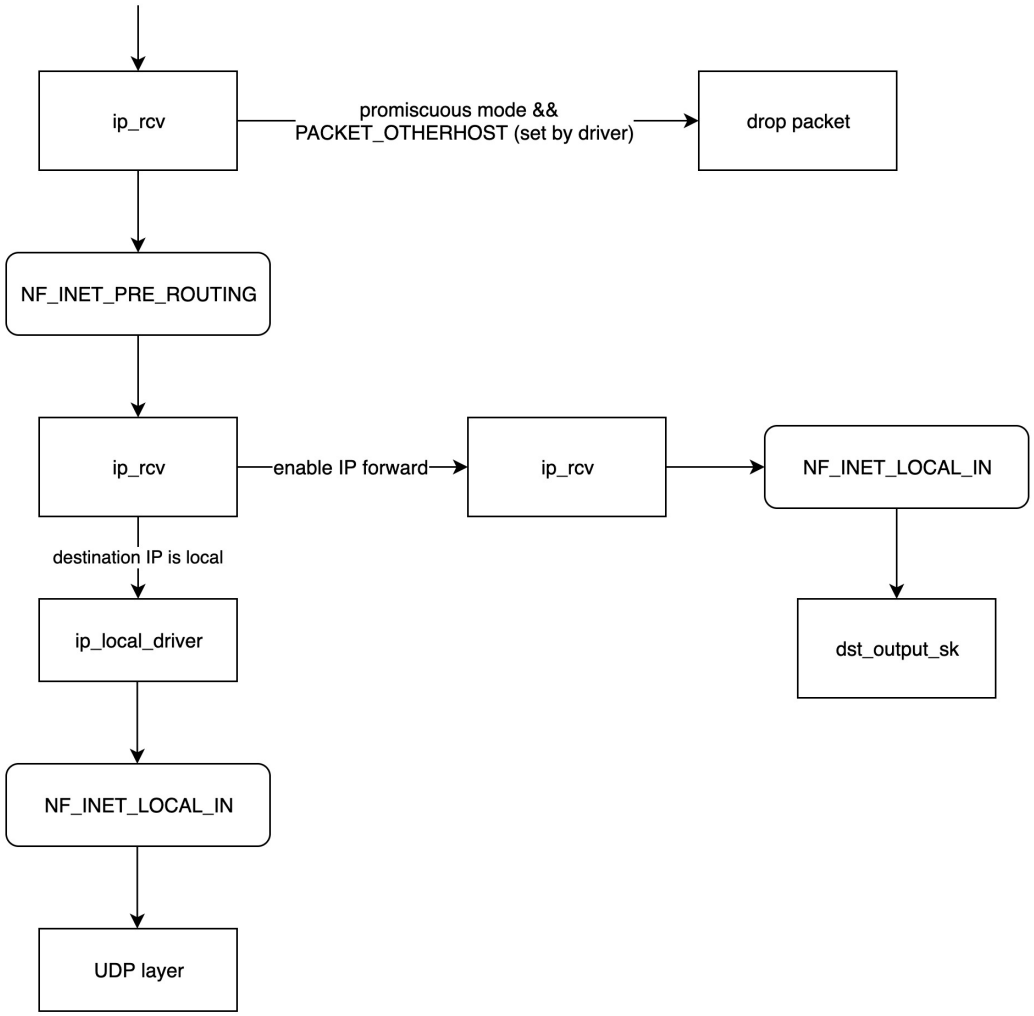
- 10： 驱动程序将内存中的数据包转换成内核网络模块能识别的skb格式，然后调用napi_gro_receive函数
- 11： napi_gro_receive会处理GRO相关的内容，也就是将可以合并的数据包进行合并，这样就只需要调用一次协议栈。然后判断是否开启了RPS，如果开启了，将会调用enqueue_to_backlog
- 12： 在enqueue_to_backlog函数中，会将数据包放入CPU的softnet_data结构体的input_pkt_queue中，然后返回，如果input_pkt_queue满了的话，该数据包将会被丢弃，queue的大小可以通过net.core.netdev_max_backlog来配置
- 13： CPU会接着在自己的软中断上下文中处理自己input_pkt_queue里的网络数据（调用__netif_receive_skb_core）
- 14： 如果没开启RPS，napi_gro_receive会直接调用__netif_receive_skb_core
- 15： 看是不是有AF_PACKET类型的socket（也就是我们常说的原始套接字），如果有的话，拷贝一份数据给它。tcpdump抓包就是抓的这里的包。
- 16： 调用协议栈相应的函数，将数据包交给协议栈处理。
- 17： 待内存中的所有数据包被处理完成后（即poll函数执行完成），启用网卡的硬中断，这样下次网卡再收到数据的时候就会通知CPU

> enqueue_to_backlog函数也会被netif_rx函数调用，而netif_rx正是lo设备发送数据包时调用的函数

3.3 内核网络协议栈

内核 TCP/IP 协议栈此时接收到的数据包其实是三层(网络层)数据包，因此，数据包首先会首先进入到 IP 网络层，然后进入传输层处理。

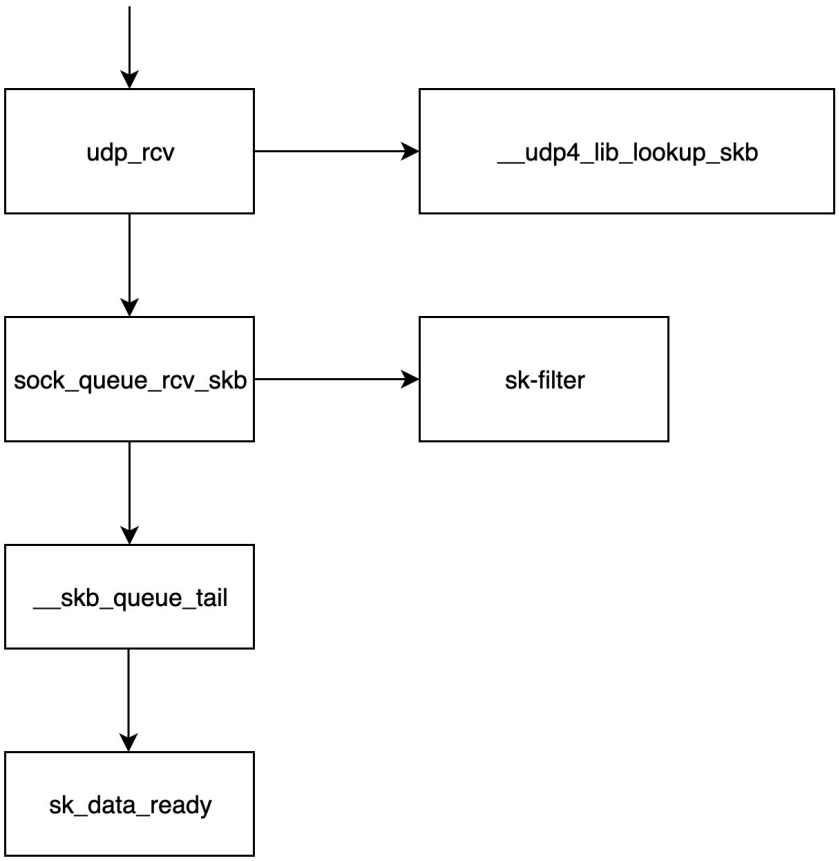
3.3.1 IP 网络层



- **ip_rcv：** ip_rcv函数是IP模块的入口函数，在该函数里面，第一件事就是将垃圾数据包（目的mac地址不是当前网卡，但由于网卡设置了混杂模式而被接收进来）直接丢掉，然后调用注册在NF_INET_PRE_ROUTING上的函数
- **NF_INET_PRE_ROUTING：** netfilter放在协议栈中的钩子，可以通过iptables来注入一些数据包处理函数，用来修改或者丢弃数据包，如果数据包没被丢弃，将继续往下走
- **routing：** 进行路由，如果是目的IP不是本地IP，且没有开启ip forward功能，那么数据包将被丢弃，如果开启了ip forward功能，那将进入ip_forward函数
- **ip_forward：** ip_forward会先调用netfilter注册的NF_INET_FORWARD相关函数，如果数据包没有被丢弃，那么将继续往后调用dst_output_sk函数
- **dst_output_sk：** 该函数会调用IP层的相应函数将该数据包发送出去，同下一篇要介绍的数据包发送流程的后半部分一样。
- **ip_local_deliver**：如果上面**routing**的时候发现目的IP是本地IP，那么将会调用该函数，在该

函数中，会先调用NF_INET_LOCAL_IN相关的钩子程序，如果通过，数据包将会向下发送到UDP层

3.3.2 传输层



- **udp_rcv：** udp_rcv函数是UDP模块的入口函数，它里面会调用其它的函数，主要是做一些必要的检查，其中一个重要的调用是__udp4_lib_lookup_skb，该函数会根据目的IP和端口找对应的socket，如果没有找到相应的socket，那么该数据包将会被丢弃，否则继续
- **sock_queue_rcv_skb：** 主要干了两件事，一是**检查这个socket的receive buffer是不是满了，如果满了的话，丢弃该数据包**，然后就是调用sk_filter看这个包是否是满足条件的包，**如果当前socket上设置了**filter**，且该包不满足条件的话，这个数据包也将被丢弃**（在Linux里面，每个socket上都可以像tcpdump里面一样定义filter，不满足条件的数据包将会被丢弃）
- **__skb_queue_tail：** 将数据包放入socket接收队列的末尾
- **sk_data_ready：** 通知socket数据包已经准备好

> 调用完sk_data_ready之后，一个数据包处理完成，等待应用层程序来读取，上面所有函数的执行过程都在软中断的上下文中。

理解了 Linux 网络数据包的接收和发送流程，我们就可以知道在哪些地方监控和修改数据包，哪些情况下数据包可能被丢弃，特别是了解了 netfilter 中相应钩子函数的位置，对于了解 iptables 的用法有一定的帮助，同时也会帮助我们更好的理解 Linux 下的网络虚拟设备。