《openEuler内核编程》

课程讲稿

第七章 第1讲

内核网络协议栈

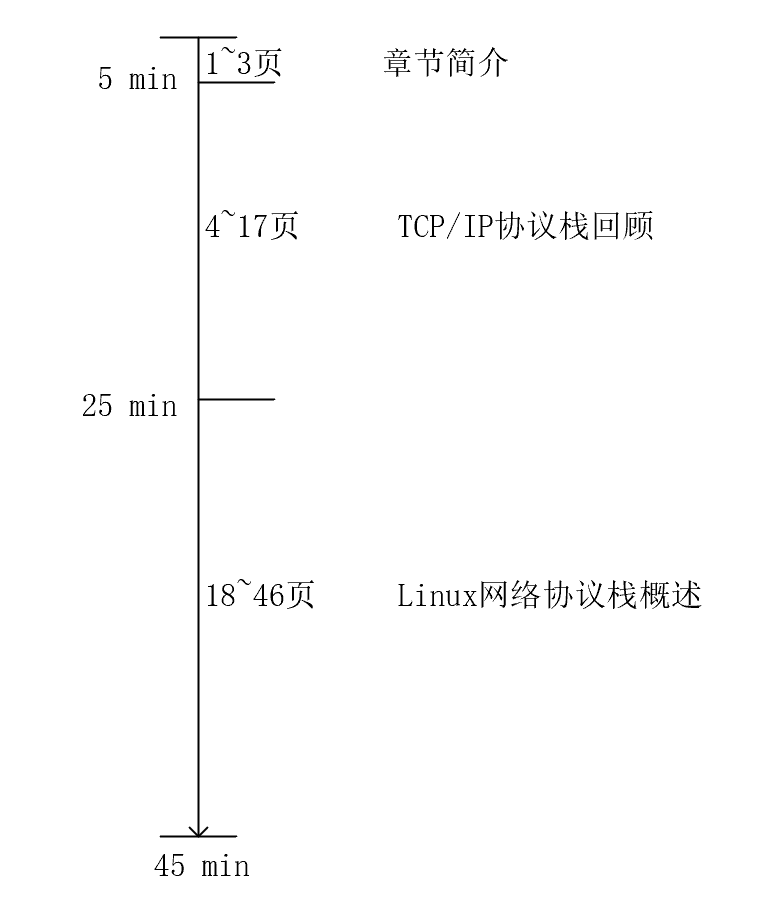
软件所制

第七章 第1讲 内核网络协议栈

**学时：**1学时

**教学目的：**复习TCP/IP协议栈，深入了解Linux内核中的网络协议栈

**课程时间线：**



**课外参考读物：**

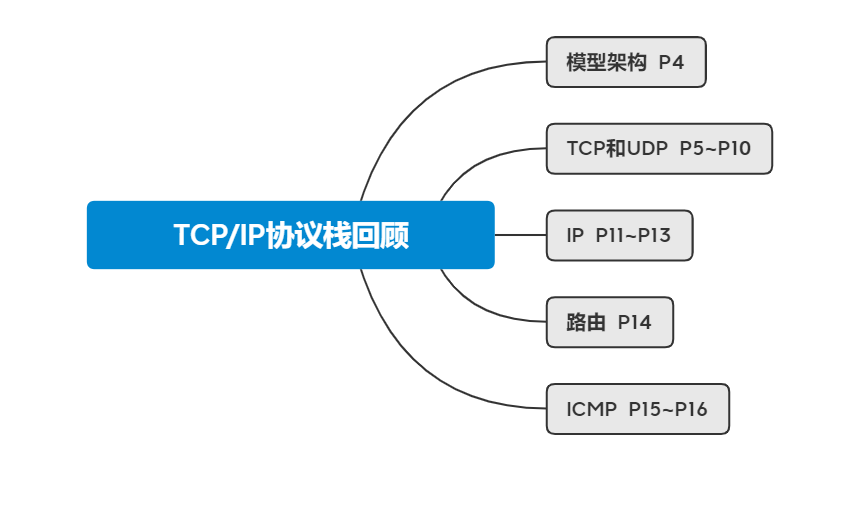
<https://ieevee.com/tech/2017/06/12/kernel-network-stack.html>

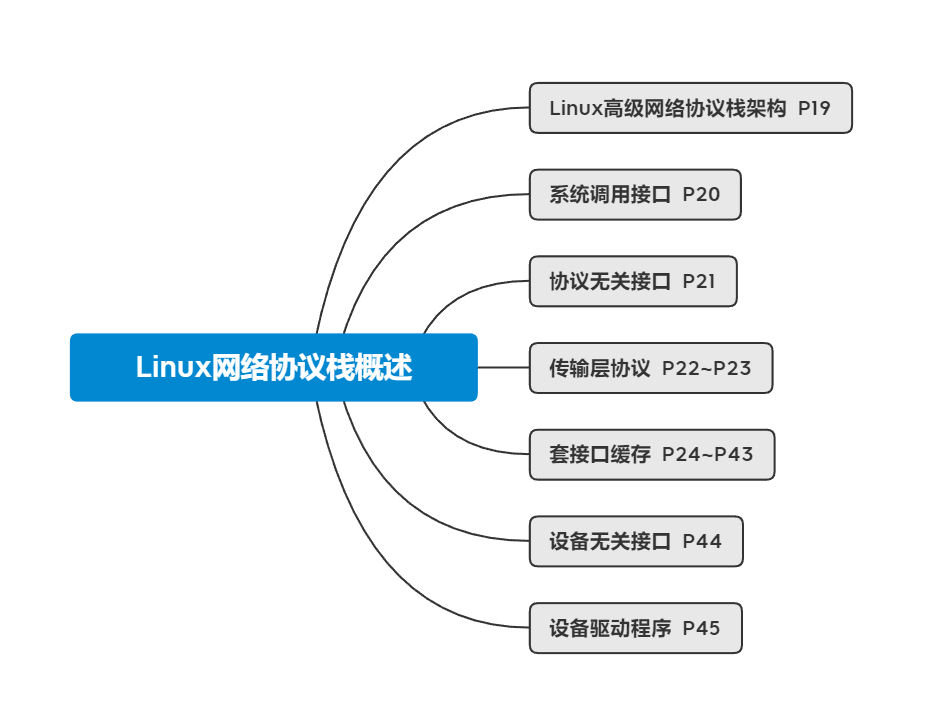
《深入理解Linux网络技术内幕》

《Linux内核源码剖析 TCPIP实现(上册)》

《Linux内核源码剖析 TCPIP实现(下册)》

**知识框图：**





**PPT讲稿：**

1. 这是第七章操作系统课程，我们主要介绍网络相关的知识。
2. 在这一章，我们讲内容主要分成3讲，分别是内核网络协议栈、网络数据在内核中的传递和netfilter和BPF的原理和实现，其中第三讲我们将分为两个课时分别讲解netfilter和BPF。
3. 那么我们进入第七章的第一讲：网络协议栈。该讲我们讲分为两部分内容，第一部分TCP/IP协议栈回顾，该部分主要是对计算机网络基础知识的一个回顾，第二部分Linux网络协议栈将会是这一讲的主要内容，主要介绍Linux网络协议栈的五大组成部分。
4. OSI七层模型是计算机网络中大家比较熟悉的，从上到下分别是应用层、表示层、会话层、传输层、网络层、数据链路层以及物理层，每一层实现各自的功能和协议，并完成与相邻层的接口通信，一般的，某一层的服务就是该层即下层的一种能力，它通过接口提高给上层。而OSI参考模型是学术上和法律上的国际标准，TCP/IP参考模型才是运用到实际的国际标准。与OSI相比，tcp/ip将应用层表示层会话层合并成应用层，并将数据链路层和网络层合并成网络接口层。Tcp/ip相关的协议栈在计算机网络中应该也重点介绍过，而在不同层的协议中数据又以不同的形式存放。例如，在应用层中数据是以报文的形式传递，而到了传输层数据以段的形式传递，网络层以分组形式，数据链路层以帧。
5. 接下来就主要介绍Tcp/ip协议栈，tcp/ip是internet上的标准通信协议集，正如刚才看到的，它并不是一两个协议，而是由数十个具有层次结构的协议组成的一个协议集，而在这个协议集中最为重要的两个协议就是tcp和ip协议。具体协议的内容课后感兴趣的同学可以看RFC文档。
6. Tcp是传输层的协议，传输层除了tcp还有udp协议。比较来看，tcp是面向连接的运输，他建立点到点的连接可靠且有序，还有拥塞控制和流量控制来管理数据的运输，一般重要的数据都是通过tcp来传输的。而udp是无连接的运输，它只能尽力而为的传输数据，因此不可靠且无序，常用作实时数据的传输。
7. 这是tcp报文段的结构，其首部共20字节。首先是32bit的端口号，用来区分服务类别和在同一时间进行多个会话，其中1-1023是周知端口例如我们比较熟悉的http的80端口和ftp的21端口；1024-49151的端口号只有大型公司可以申请例如我们比较熟悉的appach的8080；剩下49151-65535是提供给个人使用的。下来是32位的序列号，数据流的每一个字节都有序号，序列号的值是发送数据的第一个字节的序号。之后是32位的确认号，是发送确认的一端所期望收到的下一个序号。序列号和确认号组合起来保证了tcp的可靠传输。
8. 确认号下面是首部长度和标志位，首部长度是以32比特的字为单位的TCP首部长度，由于选项字段的原因，首部长度是可变的。一般值为20字节。还有六个标志位，我们主要介绍其中的三个，ack标志位只有当ack=1时确认字段才有效；syn标志位在连接建立时用来同步序号；fin标志位用来释放一个连接，这三个标志位都与tcp的三次握手和四次挥手有关。最后是窗口大小用来表示本地接收缓冲区的大小。
9. Udp报文段的首部长度要比tcp多12字节，它只包括端口号，udp长度和检验和。端口号和tcp端口号相似，udp长度是首部长度和数据长度的总长。因为udp是不可靠的传输，所以检验和在这里就显得尤为的重要，我们举个例子，当有三个数据分别为0110011001100000 0101010101010101 1000111100001100，首先需要对这三个求和，值得注意的是这里在计算过程中如果遇到进位溢出需要回卷，然后对结果取反存储，这就是检验和。当接收方收到数据后，将三个数据和检验和相加如果结果全为一则说明没有发生错误。
10. Tcp/ip中的网络层是负责处理主机到主机的通信，主要把数据报通过最佳路径送到目的端，这期间包括寻址、路由选择、分片等等操作。Ip协议是网络层的核心，IP协议与udp一样是无连接的所以只能提供尽力而为的传输，不保证可靠性。
11. 网络层传递数据报是通过IP地址，IP地址分为两部分：网络号和主机号，网络号相同的主机属于同一个网络，主机之间通过主机号区分。为了有效的管理海量的IP地址，将其划分为5类：A类地址，前1个字节为网络号，且最高位是0；B类地址，前2个字节为网络号，且最高位是10；C类地址，前3个字节为网络号，且最高位110；D类地址为多播地址；E类地址为保留地址。我们日常见到的IP地址如128.143.137.144就是通过这种方式计算的。
12. 这是网络层的IP数据报的格式。首部共20字节。前四位是版本号，规定数据报是ipv4还是ipv6；首部长度确定实际数据从哪里开始，因为IPv4数据报可包含一些可变数量的选项，故需要用4比特来确定IP数据报种数据部分实际从哪里开始，大多数IP数据报不包含选项，所以一般的IP数据报 具有20字节的首部。16位总长度包括首部长度和数据长度；标识、标志、和片偏移是比较重要的概念，涉及ip报分片，因为报文长度不能超过MTU，所以需要将超过的IP包分片，其实是数据部分的分片，每一片都要带着ip报头；生存时间TTL在没经过一个路由器都会减一，以确保数据报不会再路由器中循环；协议标识到达终点后确定使用哪个运输层协议，6标志tcp，17标识udp；与udp协议相同首部检验和用于帮助路由器检测收到的ip数据报中的比特错误。源和目的IP地址：当某源生成一个数据报时，他在源IP字段中插入他的IP地址，在目的IP地址字段插入其最终目的地的地址，通常源主机通过DNS查找来决定目的地址。
13. 类似的ipv6数据报的格式和ipv4相似，其中下一个头与ipv4中的协议字段作用相同，跃点限制与ipv4中的生存时间作用相同。
14. 网络层最核心的功能是路由，根据路由表将数据报发送或者转发到正确的网络上，路由表表项中包括了物理地址、网络掩码、网关、接口和跳跃数。路由的基本过程是内存中会维护以个路由表，记录到目的ip地址的下一跳路由器的ip地址，首先当一个数据帧到达某一端口,端口对帧进行CRC校验并检查其目的数据链路层地址是否与本端口符合.如果通过检查,则去掉帧的封装并读出IP数据包中的目的地址信息,查询路由表,决定转发接口与下一跳地址。获得了转发接口与下一跳地址信息后路由器将缓存中是否已经有了在外出接口上进行数据链路层封装所需的信息,如果没有这些信息路由器将通过适当的进程获得这些信息:外出接口如果是以太网,则将通过ARP协议获得下一跳IP地址所对应的MAC地址;而如果外出接口是广域网接口,则将通过手工配置或自动实现的映射过程获得相应的2层地址信息,然后做新的数据链路层封装,并依据外出接口上所做的QOS策略入相应的队列,等待端口空闲进行数据转发。
15. 网络层还有一个协议时icmp协议，主要功能是用于在主机或路由器报告差错和异常情况，它作为IP层数据报的数据，一般加上数据报首部组成ip数据报发送。有两种类型的icmp报文，分别是icmp询问报文和icmp差错报文。Icmp询问报文共四种回送请求和回答报文、时间戳请求和回答报文、掩码地址请求和回答报文、路由器询问和通告报文。
16. 还有五类icmp差错报文类型：终点不可达一般是路由器或主机不能交付数据报时，向原点发送的终点不可达报文；原点抑制一般是当路由器或主机由于拥塞而丢弃数据包时就像远点发送抑制报文从而使原点适当放慢数据报发送速率；时间超时是路由器收到ttl为零的数据报或者终点在预先规定的时间内不能收到一个数据报的全部数据报片时，这两种都会导致数据报片丢失，所以需要向原点发送时间超时报文；参数问题是路由器或主机收到数据报的首部中有字段值不正确而丢弃该报文；改变路由是路由器把改变路由报文发送给主机，让主机知道下次应该发送给另一个路由器。
17. 数据具体的传递过程就如这张图所示，主机A的数据先在用户层增加用户层协议对应的头，然后到达传输层并添加tcp首部，在网络层添加ip首部在数据链路层添加以太网头，最后通过物理层传输，经过路由器的转发到达目标主机，然后进行一次与刚才相反的操作，一层一层的去掉协议首部，最后剩下数据本身交付给主机B。
18. 以上的知识相信大家都在计算机网络课程中有所接触，我们只是稍微回顾一下，为我们接下来的课程奠定基础，下来就要开始Linux网络协议栈的学习。
19. Linux高级网络栈的架构分为三层：用户空间层、内核空间层和物理设备层。而我们学习的重点就在内核空间层。内核空间层的顶部是系统调用接口，它简单地为用户空间的应用程序提供了一种访问内核网络子系统的方法；位于其下面的是一个协议无关层，它提供了一种通用方法来使用底层传输层协议；然后是实际协议，它又可以分为传输层和网络层，在 Linux 中包括内嵌的协议 TCP、UDP还有 IP；然后是另外一个协议无关层，提供了与各个设备驱动程序通信的通用接口；最下面是设备驱动程序本身。
20. 系统调用接口实质是一个面向用户空间应用程序的接口调用库，向用户空间应用程序提供使用网络服务的接口。一般可以从两个角度进行描述，第一个角度是用户发起网络调用，用户发起网络调用时通过系统调用接口进入内核的过程应该是多路的。最终调用sys\_socketcall(./net/socket.c），在sys\_socketcall()中会根据网络系统调用号调用具体的功能，下面是四个比较重要的socket系统调用函数，具体代码我们将在之后具体介绍。第二个角度是普通文件操作作为网络I/O，例如，典型的读写操作可以在网络socket一样执行。对于这类情况，套接口的输入/输出操作可以当作典型的文件读写操作来进行。但也有一些特定于网络的操作如调用socket创建socket，调用connect将socket连接到目的地等等。
21. 协议无关接口层叫socket层，一听名字就知道这个很重要，它提供了一组通用函数来支持各种不同协议。socket 层不但可以支持典型的 TCP 和 UDP 协议，还可以支持 IP、裸以太网和其他传输协议，例如 SCTP。Socket结构是用来存储与该链路有关的信息的，其中包括所使用的协议、协议的状态信息、到达的连接队列、数据缓存和可选标志。再socket结构体中最关键的成员是ops和sk，前者是指向特定传输协议的操作集，后者是指向与该套接口相关的传输控制块。而这里的tcp\_prot就是我们马上要将的传输层。
22. 套接口的sk字段指向与该套接口相关的传输控制块，用来存放套接口所需信息，不同协议对应不同的传输控制块，例如tcp\_sock，支持完整的TCP特性，包含了TCP为各连接维护的所有节点信息，如双方的序列号、窗口大小、重传次数等；udp\_sock，支持完整的UDP特性，包括两端IP地址、两端端口号等；还有raw\_sock
23. 套接口的ops字段指向特定传输协议的操作集接口，因为其是套接口系统调用到传输层调用的入口，因此成员与socket系统调用基本一一对应，struct proto\_ops ops结构就是一张套接口系统调用的跳转表，其中某些操作会继续调用proto结构跳转表中函数进入下一步处理。
24. 我们前面一直在说报文，那么什么是报文，报文在Linux内核里是以什么形式存在的呢？这个就是sk\_buff。套接字缓存是用sk\_buff这个数据结构表示的，sk\_buff是Linux网络代码中最重要的数据结构，用于描述已经接收或待发送的数据报文信息。在多个不同的网络分层中都会使用到这个结构，当该结构从一个分层传到另一个分层时，其不同字段的值会随之发生变化，具体变化会在之后的具体讲解。设计套接字缓存的初衷就是为了方便的处理可变长度缓存，因为接受和发送的数据报长度不是固定的，同时在添加和移除数据时需要尽量避免数据的复制。
25. 这是一张套接口缓冲区结构图与其他各部分的关系图。我们将从两个角度介绍，分别是sk\_buff基本结构和sk\_buff对聚合分散IO的支持。首先我们将从sk\_buff这个结构体讲起。
26. 首先我们要知道c程序中的一种宏的用法，粗略的看过内核配置可以知道，网络代码都提供了大量的选项，如防火墙、多播等等，这些功能选项不一定总是需要。这些选项大多数都要求在内核数据结构中附加字段，所以在struct sk\_buff中可以看到很多用c预处理程序#ifdef指示附加的字段。而只有定义了宏CONFIG\_NET\_SCHED tc\_index字段才有效，在本例中该字段预示着流量控制的打开。
27. sk\_buff字段本身是为了方便搜寻和组织数据结构本身，内核在一个双向链表中维护所有的sk\_buff结构，但其结构其实比一般的双向链表更为复杂。与双向链表相似，每个sk\_buff中都有两个指针next和prev来实现联系。next指向前，prev指向后。但是，这个表还有另一个必要需求，就是每个sk\_buff结构必须能够迅速找到整个表的头。为了实现这项必要需求，在表的开端额外增加一个sk\_buff\_head结构作为哑元元素。sk\_ buff\_head结构中含有四个成员：qlen代表表中元素的数目；lock用于防止对表的并发访问。sk\_buff\_head中也有next和prev指针，一个是指向表头第一个sk\_buff，与i个是指向表尾最后一个sk\_buff。还有一个更重要的，每一个sk\_buff中都有一个list字段来指向sk\_buff\_head。
28. 当然，sk\_buff中还不止上面提到的三个成员，还包括struct sock \*sk为当前sk\_buff的宿主传输控制块，指向拥有此缓冲区的套接字的sock数据结构。当数据在本地产生或者正由本地进程接收时，就需要这个指针。当缓冲区只是被转发时，该指针就是NULL；unsigned char \*head,\*data,\*tail,\*end，head和end分别指向缓存区头和尾，data和tail分别指向数据的头和尾，每一层协议会在head与data之间填充协议首部，还可能在tail和end之间添加数据。
29. 下面四个是与长度有关的成员。unsigned int len是缓冲区数据区块的大小。这个长度包括协议首部的长度（由head所指）和数据部分长度，数据部分又包括线性缓冲区数据长度（由data指向），SG类型的聚合分散I/O的数据以及FRAGLIST类型的聚合分散I/O的数据长度。unsigned int data\_len是数据部分长度，包括线性缓冲区数据长度（由data指向），SG类型的聚合分散I/O的数据以及FRAGLIST类型的聚合分散I/O的数据长度。unsigned int mac\_len是MAC报头的长度。unsigned int truesize是数据缓存区的总长度，包括SKB描述符和数据缓冲区部分，该字段的初始化由alloc\_skb函数设置：truesize = len + sizeof(sk\_buff)。
30. 这三个是与网络设备有关。struct skb\_timeval tstamp为接收时间戳或发送时间戳，通常在网络设备受到一个数据包后，通过netif\_receive\_skb()进行设置。struct net\_device \*dev为网络设备指针，在初始化网络设备驱动，分配接收缓存队列时，该指针指向收到数据包的网络设备，在发送数据包时，该指针指向数据包的网络设备。struct net\_device \*input\_dev指向接收报文的原始网络设备。
31. 还有三个union结构。union{ } 是第四层（传输层）协议首部，包含内核所能解析的所有四层协议的数据结构联合体，如tcphdr结构、udphdr结构。union{ } nh是第三层（网络层）协议首部。union{ } mac是第二层（链路层）协议首部。
32. sk\_buff同时也支持聚合I/O，那么什么是聚合I/O呢？；了解分片的都知道，在网络上创建一个发送报文的过程包括组合多个分片。报文数据必须从用户空间复制到内核空间，同时加上网络协议栈各层的首部，这个组合的过程涉及大量的数据拷贝，就像左边这张图一样，数据首先需要拷贝到一起在发往网络接口。而聚合分散I/O就不需要这个拷贝过程，如果发送报文的网络接口支持聚合分散I/O，则报文就无需组装成一个单块，直接依次发送各个分片即可，从而避免大量数据拷贝。
33. 聚合分散I/O的实现是通过sk\_shared\_info这个结构。结构中包含了frag\_list、nr\_frags、frags三个主要成员。一般的，内核会先检查是否设置了NETIF\_F\_SG，如果没有设置则说明不支持聚合分散I/O，将会对报文进行线性化处理，再聚合成一个单独的报文。而如果设置了该宏，则会坚持nr\_frags的值，确定分片数，这些分散的片段以关联的方式存储在frags数组中。剩下的还有一个frag\_list的作用是用于在接收分片组后链接多个分片，组成一个完整的IP数据报。如果是在UDP数据报的输出中，则将待分片的SKB链接到第一个SKB中，然后在输出过程中能够快速的分片。还用于存放FRAGLIST类型的聚合分散I/O的数据包，如果输出网络设备支持FRAGLIST类型的聚合分散I/O，则可以直接输出。sk\_buff通过这三个成员进行ip分片的存储。
34. 刚才提到了分散片段以关联的方式将存储在frags数组中。frags是一个skb\_frag\_struct结构，其中比较重要的成员是p指针用于指向文件系统缓存页的指针；page\_offset是数据起始地址在文件系统缓存页中的偏移。Size是数据在文件系统缓存页面中使用的长度。
35. 接下来，我们来看一下分散聚合I/O的具体结构。这是在没有启动分片的报文的结构图。可以看出数据全部存储在线性存储区中，数据长度为x，即从data到tail的长度。而在聚合分散I/O存储区中的数据长度data\_len为0，说明这个报文没有聚合分散I/O数据。在这里分片是由skb\_shared\_info结构中的nr\_frags和frag\_list控制的，在此nr\_frags为0，而frag\_list指针为NULL，这两个值再一次说明了这个报文没有分片。
36. 下一个是只启用聚合分散I/O分片的报文。从图中可以看到数据长度len为x+s1+s2，其中x即为data到tail区域的长度，而s1和s2分别为两个分片的长度。此时聚合分散I/O数据存储区的数据长度data\_len就为s1+s2，标识这个报文存在聚合分散I/O数据。同样的我们再来看skb\_shared\_info中的数据，nr\_frags为2，而frag\_list为NULL，这说明不是普通的分片二十聚合分散I/O分片，数量为2.这两个分片指向同一个物理分页，各自在分页中的偏移和长度分别是0/s1和s1/s2。
37. 与上一张图不同，这是启用聚合分散I/O分片的两个报文共享内存。从图中更可以看到报文a和b两个skb数据长度len分别为x+s1和y+s2，各自skb\_shared\_info结构中的page指向同一个页面，但在分页中的偏移和数据长度不同，a的偏移和长度是0和s1，而b的偏移和长度是s1和s2。
38. 最后一张图是包含fraglist类型的分散聚合I/O数据的报文。从图中看到数据长度len为x+s1，其中x为data到tail的长度，而s1为fraglist类型的聚合分散I/O数据的报文长度。Data\_len为s1，表示该报文存在聚合分散I/O数据。Skb\_shared\_info结构中nr\_frags为0，而frag\_list不为NULL，这说明该报文存在fraglist类型的聚合分散I/O数据。
39. 内核中通常提供许多很短、很简单的函数，用以操作sk\_buff元素或元素列表。在这种图中主要介绍的是四种最为重要的函数skb\_put、skb\_push、skb\_pull、skb\_reserve。我们一个一个来看。
40. Skb\_put是向skb尾部添加数据的函数。函数中首先获取当前skb->tail位置，这里还要去检查skb数据区必须为线性。然后再skb尾部增加len长的字节。当然增加完还需要更新skb->len的值，如果增加后尾部tail位置大于end了则表示超出去了报错。最后返回添加数据的第一个字节位置。
41. skb\_push是向skb数据区头部添加数据。首先将数据区data指针往前移len字节，然后更新数据总长度len的值，如果增长后溢出header了报错，最后返回新的data指针位置。
42. skb\_pull是从数据区头部移除数据。在移除头部数据前会先做一个移除数据长度判断，如果移除的长度都大于skb数据总长度了则返回NULL否则继续调用\_\_skb\_pull函数，在\_\_skb\_pull函数中，先将数据总长度减去len字节，然后检查更新后的总长度是否存在异常，最后data指针移动len字节并返回移动后的新data指针位置。
43. 最后一个是skb\_reserver函数是用来保留头部空间，该函数只能对空的skb使用。该操作详见数据区的data指针增加len字节，然后在tail指针增加len字节。
44. 在下面一层是设备无关接口，这是一个硬件无关的接口层，将网络层的不同协议与各种网络设备连接起来，提供一组通用函数供底层网络设备驱动程序和上层协议栈调用。首先设备驱动程序会先对设备进行注册和初始化，之后便可以通过接口层向设备发送sk\_buff了，这个过程中调用了dev\_queue\_xmit函数，该函数对skb进行排队，最终由底层设备驱动程序进行传输。
45. 最后是设备驱动程序层，一般在进行初始化时，设备驱动程序会分配一个 net\_device 结构，然后使用必须的程序对其进行初始化。这些程序中有一个是 dev->hard\_start\_xmit，它定义了上层应该如何对 sk\_buff 排队进行传输。这个程序的参数为 sk\_buff。这个函数的操作取决于底层硬件，但是通常 sk\_buff 所描述的报文都会被移动到硬件环或队列中。最后当设备驱动程序配置其结构中的dev接口后，调用register\_netdevice以后驱动就可以使用了。
46. 本节主要回顾了tcpip协议栈的基础知识，并以Linux网络协议栈的架构为基础，分层讲解了各个层次的内容，下一节，我们将来讲讲网络数据在内核中的传递过程。