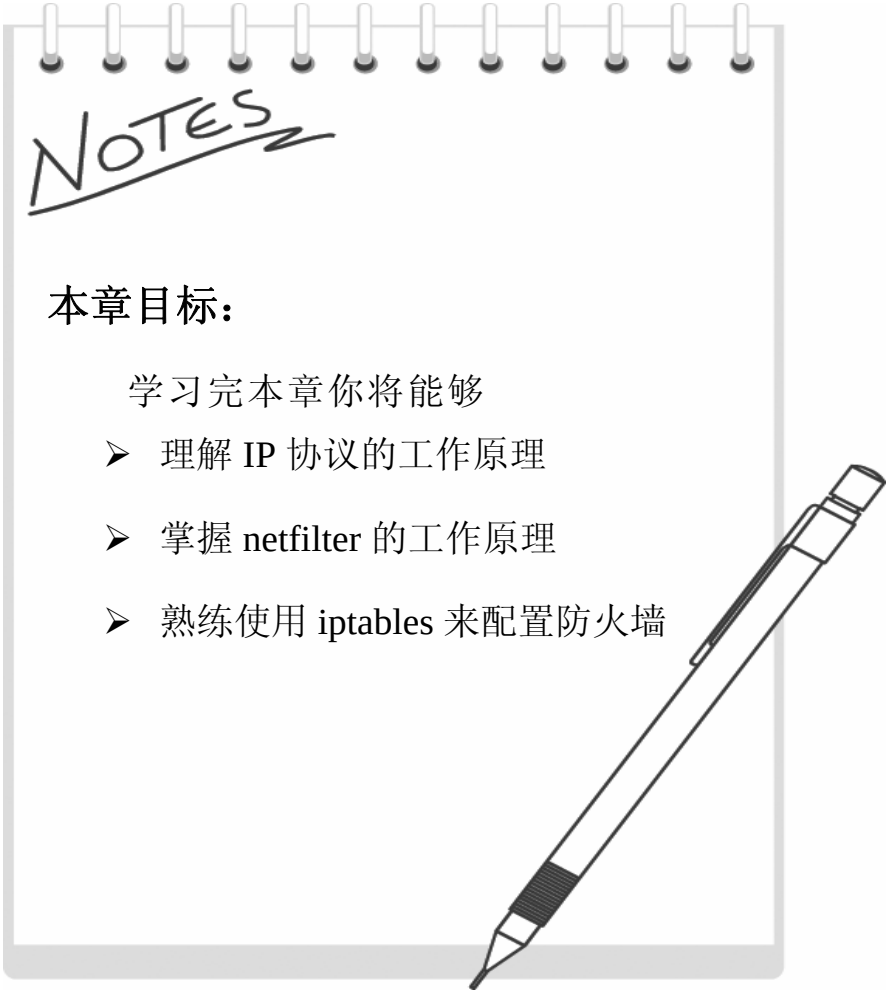
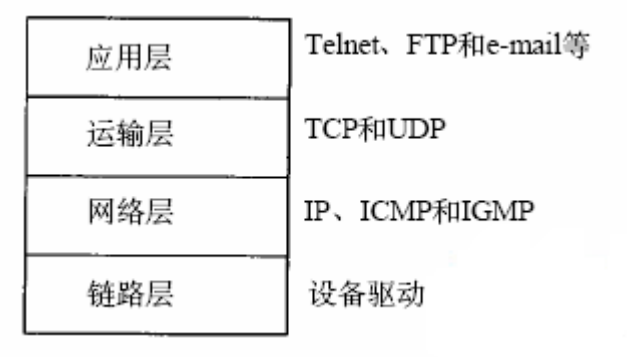
使用 Linux 搭建企业防火墙

网络防火墙是企业网络的安全保障。但是高额的硬件投入使企业难以接受。Linux 系统的出现，为企业低成本地解决企业网络的安全提供了一个实惠的解决方案。要使用，Linux 系统来搭建企业防火墙，首先需要了解 TCP/IP 网络的基本原理，理解 Linux 防火墙的工作原理，并且熟悉各个工具的集体使用。本章将深入浅出第介绍 Linux 防火墙的配置与使用。



1. IP 协议介绍

IP（Internet Protocol）是 TCP/IP 协议族中最为核心的协议。所有的 TCP、UDP、ICMP 及 IGMP 数据都以 IP 数据报格式传输（见图 2-1）。Linux 的防火墙 netfilter 就是工作在 TCP/IP 的网络层和传输层。



2-1

由于 Linux 防火墙是基于 IP 数据包来进行数据的过滤，因此对 IP 协议必须要熟悉。如果您对 TCP/IP 协议族已经熟悉，可以跳过此节

1.1 IP 数据包的头部信息

数据包就像我们平时邮寄包裹一样，需要在包裹的外面包含一些信息，IP 数据报的头部就类似于我们邮寄包裹的包裹信息。

IP 数据报的格式如图 2-2 所示。如果选项字段不包含内容，普通的 IP 首部长为 20 个字节。

分析图 2-2 中的首部。最高位在左边，记为 0 bit；最低位在右边，记为 31 bit。4 个字节的 32 bit 值以下面的次序传输：首先是 0～7 bit，其次 8～15 bit，然后 1 6～23 bit，最后是 24~31 bit。这种传输次序称作 big endian 字节序。

由于 TCP /IP 首部中所有的二进制整数在网络中传输时都要求以这种次序，因此它又称作网络字节序。以其他形式存储二进制整数的机器，如 little endian 格式，则必须在传输数据之前把首部转换成网络字节序。下面对各段进行简单描述：

（1）目前广泛应用的 IP 协议版本号是 4，因此 IP 有时也称作 IPv4。

（2）首部长度指的是首部占 32 bit 字的数目，包括任何选项。由于它是一个 4 比特字段，因此首部最长为 60 个字节。

（3）服务类型（TOS）字段包括一个 3 bit 的优先权子字段（现在已被忽略），4 bit 的 TOS 子字段和 1 bit 未用位但必须置 0。4 bit 的 TOS 分别代表：最小时延、

最大吞吐量、最高可靠性和最小费用。4 bit 中只能置其中 1 bit。如果所有 4 bit 均为 0，那么就意味着是一般服务。

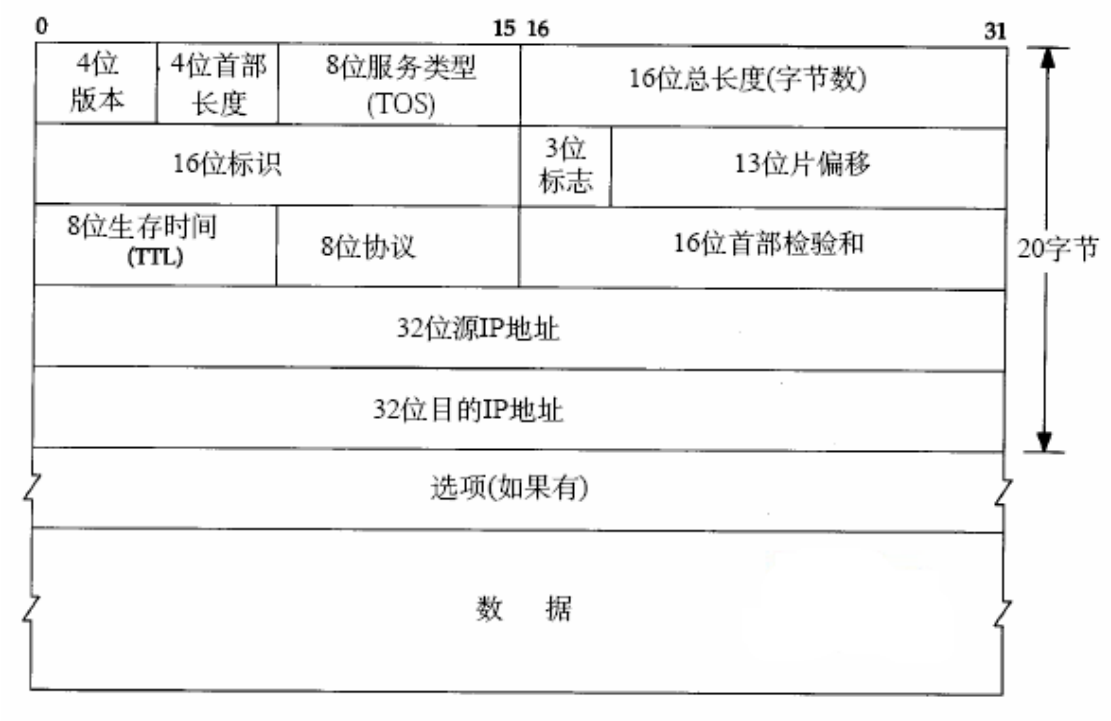


图 2-2

（4）总长度字段是指整个 I P 数据报的长度，以字节为单位。利用首部长度字段和总长度字段，就可以知道 IP 数据报中数据内容的起始位置和长度。由于该字段长 16 比特，所以 IP 数据报最长可达 65535 字节。尽管可以传送一个长达 65535 字节的 IP 数据报，但是大多数的链路层都会对它进行分片。

（5）标识字段唯一地标识主机发送的每一份数据报（即给每个数据报取一个唯一编号）。通常每发送一份报文它的值就会加 1。

（6）标志字段用于标识 IP 包是否被分片。刚才提到 IP 数据报最大可以到 65535 字节，但是为了方便传输，较大的数据报通常被分成多个小数据包来传输。对于发送端发送的每份 IP 数据报来说，其标识字段都包含一个唯一值。该值在数据报分片时被复制到每个片中（我们现在已经看到这个字段的用途）。

标志字段（总共 3 位）用其中第三个比特来表示“更多的片”。除了最后一片外，其他每个组成数据报的片都要把该比特置 1。另外，当数据报被分片后，每个片的总长度值要改为该片的长度值。

标志字段中第二个比特称作“不分片”位。如果将这一比特置 1，I P 将不对数据报进行分片。而第一个比特保留，必须置为 0。

（7）片偏移字段指的是该片偏移原始数据报开始处的位置。

（8）TTL（time-to-live）生存时间字段设置了数据报可以经过的最多路由器数。它指定了数据报的生存时间。当该字段的值为 0 时，数据报就被丢弃，并发送 ICMP

报文通知源主机。例如在 windows 系统上发出的数据报的 TTL 初始值为 128，而 Linux 系统上为 64。

（9）协议字段，用来标识该 IP 包的下一层协议（next level protocol），例如：TCP、UDP、ICMP 等。

（10）首部检验和字段是根据 IP 首部计算的检验和码。它不对首部后面的数据进行计算。ICMP、IGMP、UDP 和 TCP 在它们各自的首部中均含有同时覆盖首部和数据检验和码。当然校验的目的是看收到的数据报是否有错。

（11）源 IP 地址记录了发送数据包的主机的 IP 地址

（12）目的 IP 地址记录了应该接收数据报的主机的 IP 地址。

对于选项字段，是在一些特殊情况使用，在此不再介绍。深入理解 IP 协议，可以参考 rfc971 和其它的相关资料。

1.2 IP 路由选择

从概念上说，IP 路由选择是简单的，特别对于主机来说。如果目的主机与源主机直接相连（如点对点链路）或都在一个共享网络上（以太网或令牌环网），那么 IP 数据报就直接送到目的主机上。否则，主机把数据报发往一默认的路由器上，由路由器来转发该数据报。大多数的主机都是采用这种简单机制。

当数据报来自某个网络接口时，IP 首先检查目的 IP 地址是否为本机的 IP 地址之一或者 IP 广播地址。如果确实是这样，数据报就被送到由 IP 首部协议字段所指定的协议模块进行处理。如果数据报的目的不是这些地址，那么：（1）如果 IP 层被设置为路由器的功能，那么就检查路由表并对数据报进行转发（也就是说，像下面对待发出的数据报一样处理）；否则（2）数据报被丢弃。

路由表中的每一项都包含下面这些信息：

• 目的 IP 地址。它既可以是一个完整的主机地址，也可以是一个网络地址，由该表目中的标志字段来指定。

• 下一站（或下一跳）路由器（ next-hop router）的 IP 地址，或者有直接连接的网络 IP 地址。下一站路由器是指一个在直接相连网络上的路由器，通过它可以转发数据报。下一站路由器不是最终的目的，但是它可以把传送给它的数据报转发到最终目的。

• 标志。其中一个标志指明目的 IP 地址是网络地址还是主机地址，另一个标志指明下一站路由器是否为真正的下一站路由器。

• 为数据报的传输指定一个网络接口。

下面是一台 Linux 主机输出的路由信息：

```
[root@vfast linux-2.6.18]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.100.0   0.0.0.0         255.255.255.240 U     0      0        0 eth0
211.152.14.0    0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.10.10.0      0.0.0.0         255.255.255.0   U     0      0        0 eth1
169.254.0.0     0.0.0.0         255.255.0.0     U     0      0        0 eth1
0.0.0.0         192.168.100.1   0.0.0.0         UG    0      0        0 eth0
```

I P 路由选择主要完成以下这些功能：

1) 搜索路由表，寻找能与目的 I P 地址完全匹配的表目（网络号和主机号都要匹配）。如果找到，则把报文发送给该表目指定的下一站路由器或直接连接的网络接口（取决于标志字段的值）。

2) 搜索路由表，寻找能与目的网络号相匹配的表目。如果找到，则把报文发送给该表目指定的下一站路由器或直接连接的网络接口（取决于标志字段的值）。目的网络上的所有主机都可以通过这个表目来处置。例如，一个以太网上的所有主机都是通过这种表目进行寻径的。

3) 搜索路由表，寻找标为“默认（default）”的表目。如果找到，则把报文发送给该表目指定的下一站路由器。

如果上面这些步骤都没有成功，那么该数据报就不能被传送。如果不能传送的数据报来自本机，那么一般会向生成数据报的应用程序返回一个“主机不可达”或“网络不可达”的错误。

1.3 子网寻址

现在所有的主机都要求支持子网编址。不是把 IP 地址看成由单纯的一个网络号和一个主机号组成，而是把主机号再分成一个子网号和一个主机号。

这样做的原因是因为 A 类和 B 类地址为主机号分配了太多的空间，可分别容纳的主机数为 224-2 和 216-2。事实上，在一个网络中人们并不安排这么多的主机。由于全 0 或全 1 的主机号都是无效的，因此我们把总数减去 2。

在 InterNIC 获得某类 IP 网络号后，就由当地的系统管理员来进行分配，由他（或她）来决定是否建立子网，以及分配多少比特给子网号和主机号。例如，这里有一个 B 类网络地址（140.252），在剩下的 16 bit 中，8 bit 用于子网号，8 bit 用于主机号，格式如图 2-35 所示。这样就允许有 254 个子网，每个子网可以有 254 台主机。

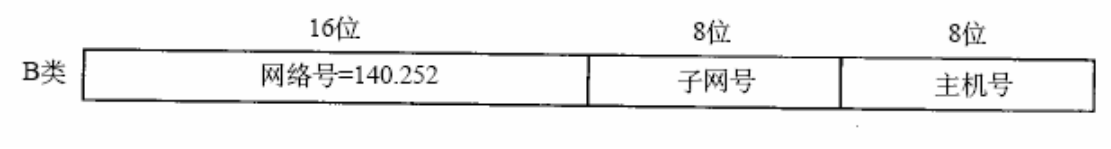


图 2-3

子网对外部路由器来说隐藏了内部网络组织（一个校园或公司内部）的细节。子网对于子网内部的路由器是不透明的。

1.4 子网掩码

任何主机在引导时进行的部分配置是指定主机 IP 地址。大多数系统把 IP 地址存在一个磁盘文件里供引导时读用。除了 IP 地址以外，主机还需要知道有多少比特用于子网号及多少比特用于主机号。这是在引导过程中通过子网掩码来确定的。这个掩码是一个 32 bit 的值，其中值为 1 的比特留给网络号和子网号，为 0 的比特留给主机号。图 2-4 是一个 B 类地址的两种不同的子网掩码格式。

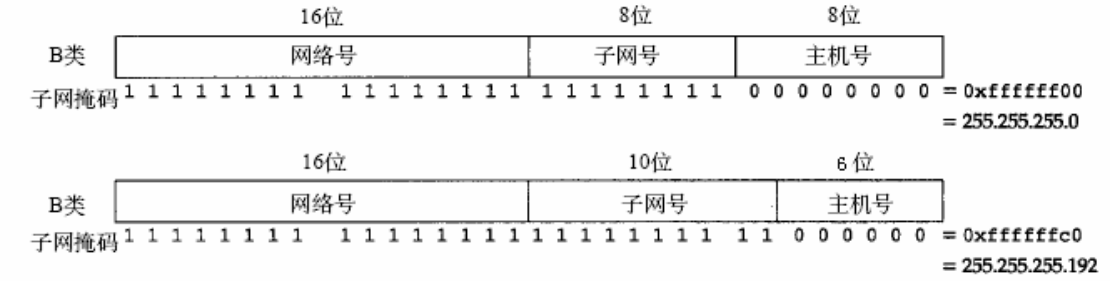


图 2-4

第一个例子子网号和主机号都是 8 bit 宽。

第二个例子是一个 B 类地址划分成 10 bit 的子网号和 6 bit 的主机号。

在路由中，假设我们的主机地址是 140.252.1.1（一个 B 类地址），而子网掩码为 255.255.255.0（其中 8 bit 为子网号，8 bit 为主机号）。那么：

• 如果目的 IP 地址是 140.252.4.5，那么我们就知道 B 类网络号是相同的（140.252），但是子网号是不同的（1 和 4）。用子网掩码在两个 IP 地址之间的比较如图 2-5 所示。

图 2-5

• 如果目的 IP 地址是 140.252.1.22，那么 B 类网络号还是一样的（140.252），且子网号也是一样的（1），但是主机号是不同的。

• 如果目的 IP 地址是 192.43.235.6（一个 C 类地址），那么网络号是不同的，因而进一步的比较就不用再进行了。

通过子网的比较，立即就可以决定了下一步应该将数据包发给谁。

2. TCP 协议简介

TCP（Transmission Control Protocol，传输控制协议）提供一种面向连接的、可靠的字节流服务。它工作在传输层。

2.1 TCP 的服务

面向连接意味着两个使用 TCP 的应用（通常是一个客户和一个服务器）在彼此交换数据之前必须先建立一个 TCP 连接。这一过程与打电话很相似，先拨号振铃，等待对方摘机说“喂”，然后才说明是谁。

在一个 TCP 连接中，仅有两方进行彼此通信。TCP 通过下列方式来提供可靠性：

• 应用数据被分割成 TCP 认为最适合发送的数据块。这和 UDP 完全不同，应用程序产生的数据报长度将保持不变。由 TCP 传递给 IP 的信息单位称为报文段或段（segment）。

• 当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。

• 当 TCP 收到发自 TCP 连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒。

• TCP 将保持它首部和数据的检验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP 将丢弃这个报文段和不确认收到此报文段（希望发端超时并重发）。

• 既然 TCP 报文段作为 IP 数据报来传输，而 IP 数据报的到达可能会失序，因此 TCP 报文段的到达也可能会失序。如果必要，TCP 将对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层。

• 既然 IP 数据报会发生重复，TCP 的接收端必须丢弃重复的数据。

• TCP 还能提供流量控制。TCP 连接的每一方都有固定大小的缓冲空间。TCP 的接收端只允许另一端发送接收端缓冲区所能接纳的数据。这将防止较快主机致使

较慢主机的缓冲区溢出。

2.2 TCP 的头部信息

TCP 数据被封装在一个 IP 数据报中，如图 2-6 所示。

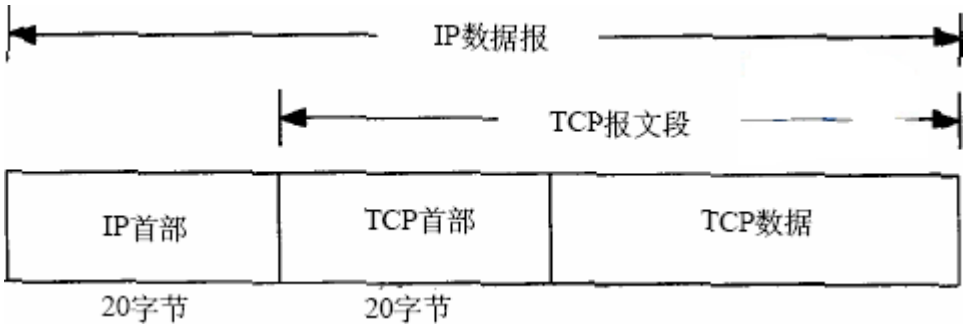


图 2-6

图 2-7 显示 TCP 首部的数据格式。如果不计任选字段，它通常是 20 个字节。

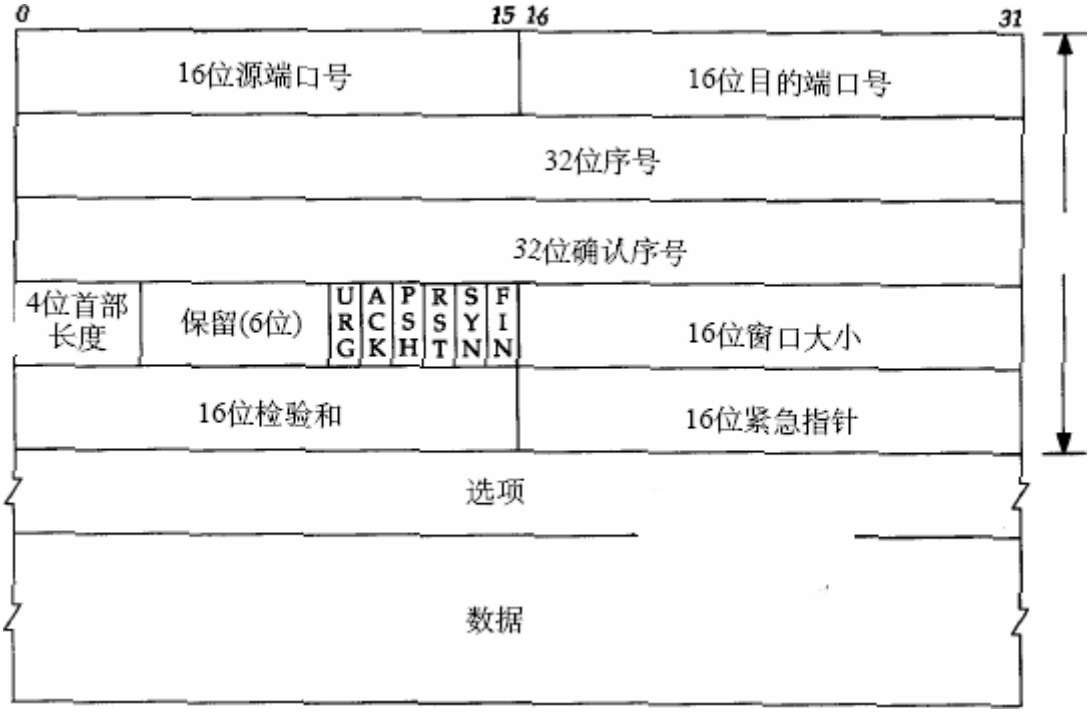


图 2-7

下面分别对 TCP 头部的各部分进行说明：

（1）源端口号记录了发起连接的主机的端口号。

（2）目的端口号记录了接受连接（提供服务）方的端口号。

（3）序号用来标识从 TCP 发端向 TCP 收端发送的数据字节流，它表示在这个报文段中的的第一个数据字节。如果将字节流看作在两个应用程序间的单向流动，则 TCP 用序号对每个字节进行计数。序号是 32 bit 的无符号数，序号到达 2 的 32-1 次幂后又从 0 开始。

当建立一个新的连接时，SYN 标志变 1。序号字段包含由这个主机选择的该连接的初始序号 ISN（Initial Sequence Number）。该主机要发送数据的第一个字节序号为这个 ISN 加 1，因为 SYN 标志消耗了一个序号。

（4）确认序号包含发送确认的一端所期望收到的下一个序号。因此，确认序号应当是上次已成功收到数据字节序号加 1。只有 ACK 标志（下面介绍）为 1 时确认序号字段才有效。

（5）首部长度给出首部中 32 bit 字的数目。需要这个值是因为任选字段的长度是可变的。这个字段占 4 bit，因此 TCP 最多有 60 字节的首部。然而，没有任选字段，正常的长度是 20 字节。

（6）保留，自然就是没有定义了，全置 0。

（7）6 个标志比特，从第一到第六个比特分别代表：

URG：如果被置为 1，紧急指针（urgent pointer）有效。
ACK：如果被置为 1，确认序号有效。
PSH：如果被置为 1，接收方应该尽快将这个报文段交给应用层。
RST：如果被置为 1，重新建立 TCP 连接。
SYN：如果被置为 1，发起一个新连接。
FIN：如果被置为 1，发端完成发送任务。

（8）TCP 的流量控制由连接的每一端通过声明的窗口大小来提供。窗口大小为字节数，起始于确认序号字段指明的值，这个值是接收端正期望接收的字节。窗口大小是一个 16 bit 字段，因而窗口大小最大为 65535 字节。

（9）检验和覆盖了整个的 TCP 报文段：TCP 首部和 TCP 数据。这是一个强制性的字段，一定是由发送端计算和存储，并由收端进行验证。

（10）只有当 URG 标志置 1 时紧急指针才有效。紧急指针是一个正的偏移量，和序号字段中的值相加表示紧急数据最后一个字节的序号。TCP 的紧急方式是发送端向另一端发送紧急数据的一种方式。

最常见的可选字段是最长报文大小，又称为 MSS (Maximum Segment Size)。每个连接方通常都在通信的第一个报文段（为建立连接而设置 SYN 标志的那个段）中指明这个选项。它指明本端所能接收的最大长度的报文段。

2.3 TCP 的连接与终止

要建立一条 TCP 连接，需要经过以下几个阶段（图 2-8 所示）：

1) 请求端（通常称为客户）发送一个 S Y N 段指明客户打算连接的服务器的端口，以及初始序号（ISN，在这个例子中为 1415531521）。这个 SYN 段为报文段 1。

2) 服务器发回包含服务器的初始序号的 SYN 报文段（报文段 2）作为应答。同时，将确认序号设置为客户的 ISN 加 1 以对客户的 SYN 报文段进行确认。一个 SYN 将占用一个序号。

3) 客户必须将确认序号设置为服务器的 ISN 加 1 以对服务器的 SYN 报文段进行确认（报文段 3）。

这三个报文段完成连接的建立。这个过程也称为三次握手（three-way handshake）。三次握手完成之后就可以进行数据传输了。

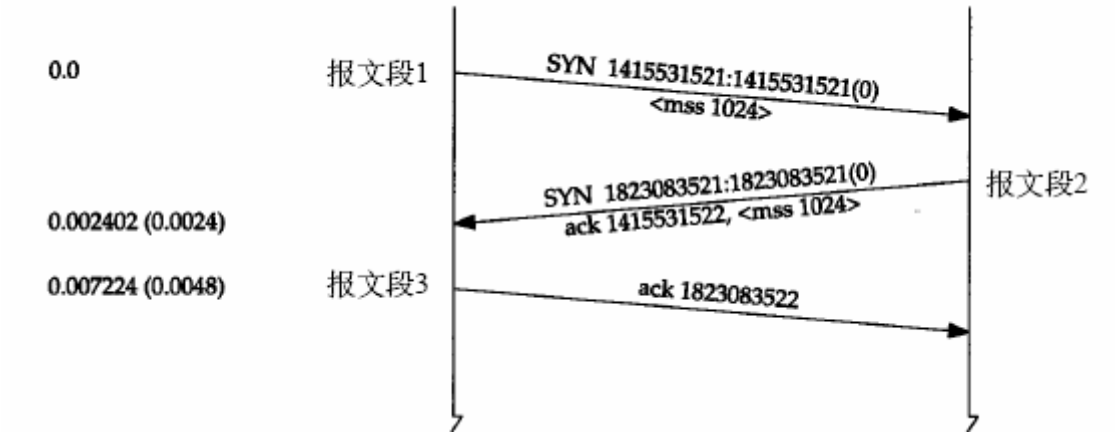


图 2-8

当数据传输完成之后，请求端会发出终止连接命令。而终止一个连接要经过 4 次握手（如图 2-9 所示）。这由 TCP 的半关闭（half -close）造成的。

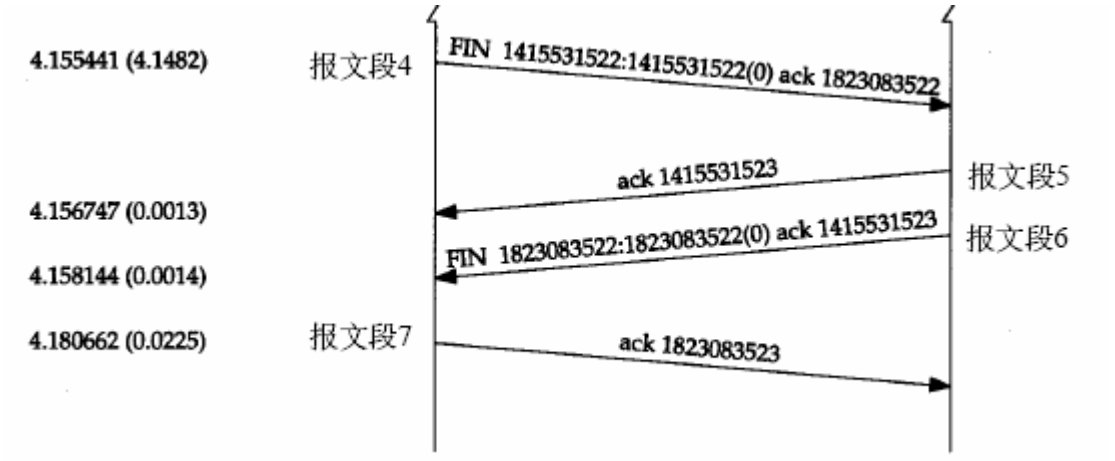


图 2-9

终止连接的过程（图 2-9 所示）如下：

（1）客户端发出一个 FIN 报文（图中的报文段 4），告诉另一方我的数据已经发送完毕，我要关闭连接了。

（2）服务器端发送 ACK 应答（图中的报文段 5），表示我知道了，我的数据已经接收完毕，你那个方向上的连接可以关闭了。（此时的连接就进入了半关闭状态了，即单向的连接。）

（3）服务器端如果没有数据需要发送，它也发出 FIN（图中的报文段 6），告诉对方我的数据也发送完了，我要关闭连接了。

（4）客户端发送 ACK 应答（图中的报文段 7），连接完全关闭。

2.4 TCP 的连接状态

TCP 连接有以下 11 种状态（严格来讲是 10 种）：

LISTEN － 等待从任何远端 TCP 和端口的连接请求。

SYN_SENT －发送完一个连接请求后等待一个匹配的连接请求。

SYN_RECEIVED － 发送连接请求并且接收到匹配的连接请求以后等待连接请求确认。

ESTABLISHED － 表示一个打开的连接，接收到的数据可以被投递给用户。连接的数据传输阶段的正常状态。

FIN_WAIT_1 － 等待远端 TCP 的连接终止请求，或者等待之前发送的连接终止请求的确认。

FIN_WAIT_2 － 等待远端 TCP 的连接终止请求

CLOSE_WAIT － 等待本地用户的连接终止请求

CLOSING － 等待远端 TCP 的连接终止请求确认

LAST_ACK － 等待先前发送给远端 TCP 的连接终止请求的确认（包括它字节的连接终止请求的确认）

TIME_WAIT － 等待足够的时间过去以确保远端 TCP 接收到它的连接终止请求的确认

CLOSED － 不在连接状态（这是为方便描述假想的状态，实际不存在）

这些连接状态如果出现，在命令行用 netstat –an 命令可以看到。

图 2-10 显示了正常打开和关闭一个 TCP 连接时的状态变化。变化的过程描述如下：

（1）客户端首先发起连接（第一次握手），处于 SYN_SEND 状态。

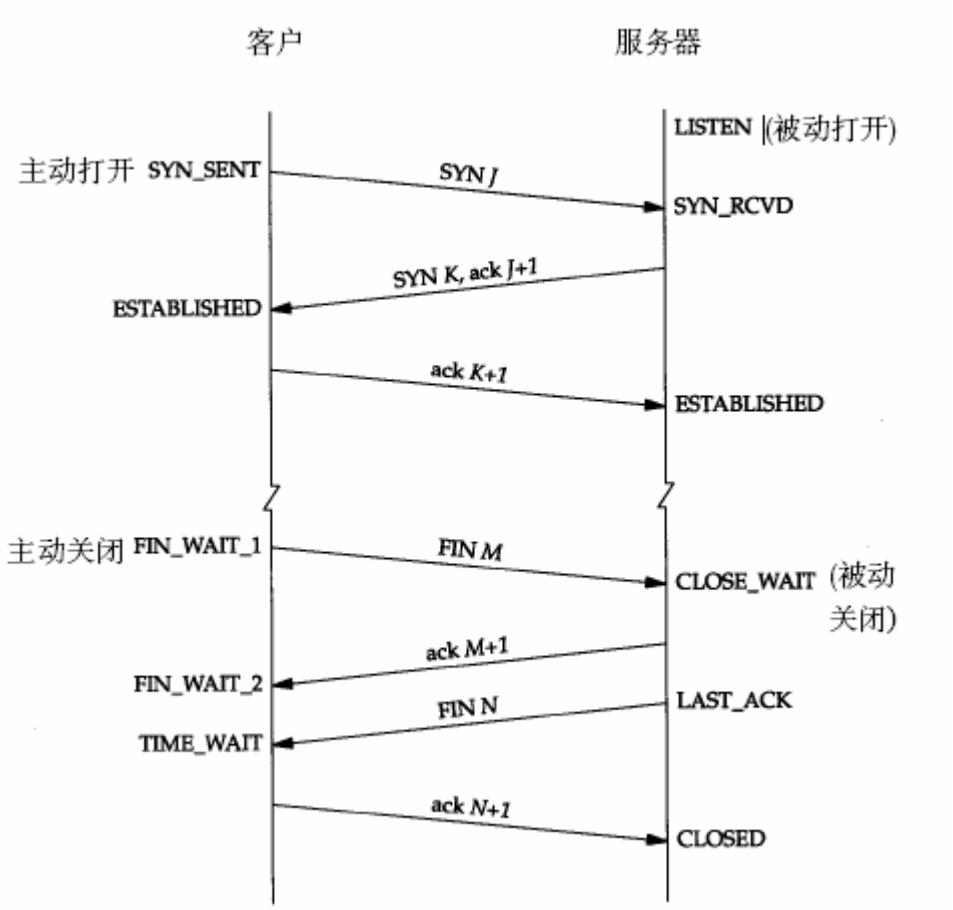


图 2-10

（2）服务器端收到 SYS 数包，立即发送 ACK 应到，并发送自己的 SYN（第二次握手）；状态被置为 SYN_RECEIVED。

（3）客户端接收到服务器端的应答，将自己置于 ESTABLISHED，表示已经准备就绪，同时发出 ACK 应答（第三次握手）。

（4）服务器端接收到应答，立即将自己置于 ESTABLISHED 状态，开始发送数据。

（5）客户端接收完数据后，发送 FIN，同时将自己置于 FIN_WAIT_1 状态，等待服务器端的应答。

（6）服务器端收到关闭请求，将自己置于 CLOSE_WAIT，等待本地用户（进程）的关闭命令。同时发送应答告诉客户端可以关闭，并且自己也向客户端请求关闭（前面已经介绍过），发出后将自己从 CLOSE_WAIT 变迁到 LAST_ACK 状态，等待客户端应答后由本地用户（进程）执行关闭操作。

（7）客户端在接到服务的应答后，将自己的状态置于 FIN_WAIT_2，等到服务器段发出 FIN（关闭请求），当接收到服务器端发来的 FIN 之后，将自己置于 TIME_WAIT（因为是客户端发起的关闭所以是 TIME_WAIT，如果是服务器端发起的关闭，那么我们看到的将是 CLOSE_WAIT。CLOSE_WAIT 是我们在浏览网页时经

常看到的状态，因为网页传输完成后通常由服务器发起关闭命令。)，等待一定的时间以确保服务器接收到应答。

3. UDP 与 ICMP 协议简介

3.1 UDP 协议

UDP（User Datagram Protocol，用户数据报协议）是一个简单的面向数据报的运输层协议：进程的每个输出操作都正好产生一个 UDP 数据报，并组装成一份待发送的 IP 数据报。这与面向流字符的协议不同，如 TCP，应用程序产生的全体数据与真正发送的单个 IP 数据报可能没有什么联系。

UDP 数据报封装成一份 IP 数据报的格式，如图 2-11 所示。

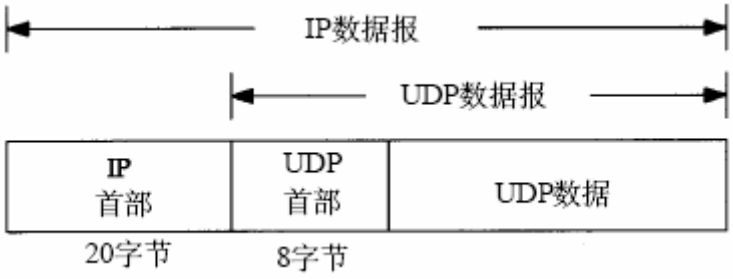


图 2-11

UDP 首部的各字段如图 2-12 所示：

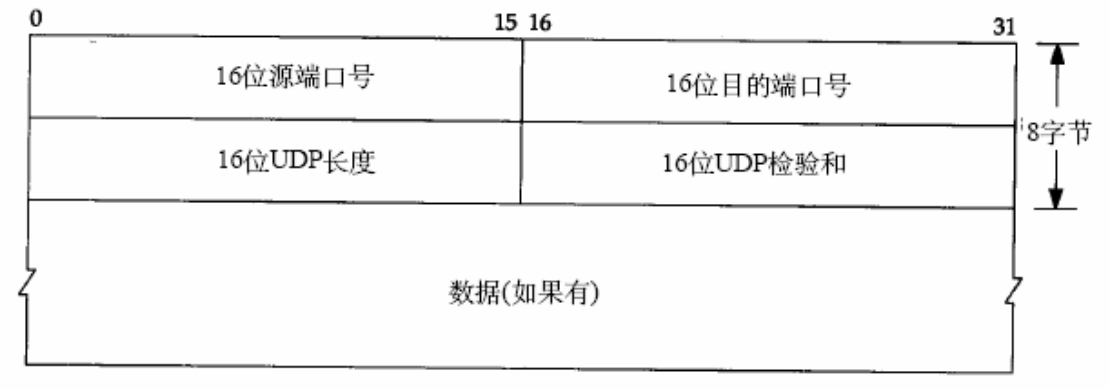


图 2-12

在防火墙中，对于 UDP 数据报，我们主要关心它的源端口号和目的端口号，因此这里就不再一一介绍了。

3.2 ICMP 协议

ICMP（Internet Control Message Protocol，Internet 控制报文协议）经常被认为

是 IP 层的一个组成部分。它传递差错报文以及其他需要注意的信息。

ICMP 报文通常被 IP 层或更高层协议（ TCP 或 UDP）使用。一些 ICMP 报文把差错报文返回给用户进程。

ICMP 报文是在 IP 数据报内部被传输的，如图 2-13 所示。

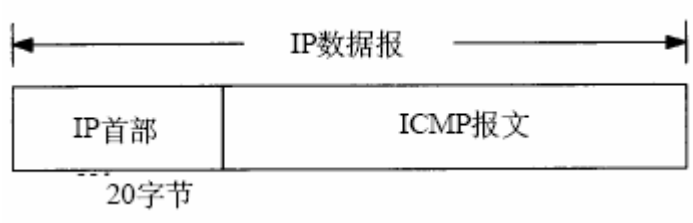


图 2-13

ICMP 报文的格式如图 2-14 所示。

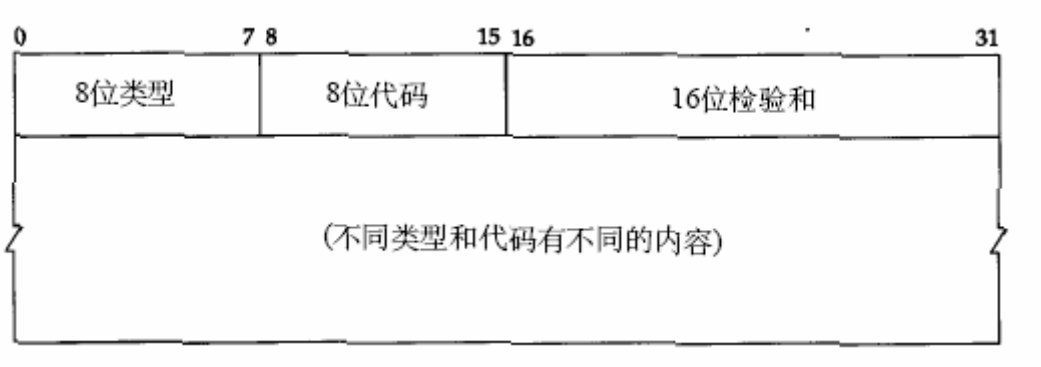


图 2-14

所有报文的前 4 个字节都是一样的，但是剩下的其他字节则互不相同。下面我们将逐个介绍各种报文格式。

类型字段可以有 15 个不同的值，以描述特定类型的 ICMP 报文。某些 ICMP 报文还使用代码字段的值来进一步描述不同的条件。

检验和字段覆盖整个 ICMP 报文。使用的算法与 IP 首部检验和算法相同。ICMP 的检验和是必需的。

各种类型的 ICM P 报文如图 2-15 所示，不同类型由报文中的类型字段和代码字段来共同决定。图中的最后两列表明 ICMP 报文是一份查询报文还是一份差错报文。因为对 ICMP 差错报文有时需要作特殊处理，因此我们需要对它们进行区分。例如，在对 ICMP 差错报文进行响应时，永远不会生成另一份 ICMP 差错报文（如果没有这个限制规则，可能会遇到一个差错产生另一个差错的情况，而差错再产生差错，这样会无休止地循环下去）。

当发送一份 ICMP 差错报文时，报文始终包含 IP 的首部和产生 ICMP 差错报文的 IP 数据报的前 8 个字节。这样，接收 I C M P 差错报文的模块就会把它与某个特定的协议（根据 IP 数据报首部中的协议字段来判断）和用户进程（根据包含在 IP

数据报前 8 个字节中的 TCP 或 UDP 报文首部中的 TCP 或 UDP 端口号来判断）联系起来。

类　型	代　码	描　述	查　询	差　错
0	0	回显应答	•	
3		目的不可达:		
	0	网络不可达		•
	1	主机不可达		•
	2	协议不可达		•
	3	端口不可达		•
	4	需要进行分片但设置了不分片比特		•
	5	源站选路失败		•
	6	目的网络不认识		•
	7	目的主机不认识		•
	8	源主机被隔离（作废不用）		•
	9	目的网络被强制禁止		•
	10	目的主机被强制禁止		•
	11	由于服务类型 TOS，网络不可达		•
	12	由于服务类型 TOS，主机不可达		•
	13	由于过滤，通信被强制禁止		•
	14	主机越权		•
	15	优先权中止生效		•
4	0	源端被关闭		•
5		重定向		•
	0	对网络重定向		•
	1	对主机重定向		•
	2	对服务类型和网络重定向		•
	3	对服务类型和主机重定向		•
8	0	请求回显	•	
9	0	路由器通告	•	
10	0	路由器请求	•	
11		超时:		
	0	传输期间生存时间为 0		•
	1	在数据报组装期间生存时间为 0		•
12		参数问题:		
	0	坏的IP首部（包括各种差错）		•
	1	缺少必需的选项		•
13	0	时间戳请求	•	
14	0	时间戳应答	•	
15	0	信息请求（作废不用）	•	
16	0	信息应答（作废不用）	•	
17	0	地址掩码请求	•	
18	0	地址掩码应答	•	

图 2-15

下面各种情况都不会导致产生 ICMP 差错报文：

1) ICMP 差错报文（但是，ICMP 查询报文可能会产生 ICMP 差错报文）。

2) 目的地址是广播地址或多播地址的 I P 数据报。

3) 作为链路层广播的数据报。

4) 不是 IP 分片的第一片。

5) 源地址不是单个主机的数据报。这就是说，源地址不能为零地址、环回地址、广播地址或多播地址。

这些规则是为了防止过去允许 ICMP 差错报文对广播分组响应所带来的广播风暴。

4. Linux 内核防火墙的工作原理

Linux 的内核提供的防火墙功能是通过 netfiter 框架实现的，通过 iptables 工具来修改防火墙的规则。

4.1 netfilter 的体系结构

netfilter 只是为不同的网络协议栈（例如 IPv4, IPv6 和 DECnet）定义了一套钩子（hook）函数，对 IPv4 协议栈就定义了 5 个 hook 函数，具体可以看 Linux 源代码目录下的 include/linux/ netfilter_ipv4.h，这 5 个函数分别是：

（1）NF_IP_PRE_ROUTING（0）：网络数据包进入系统，经过了简单的检测（例如，没有被截断，IP 校验和正确）后，就会交给该函数进行处理。读取管理员设置的规则对数据包进行处理，如果数据包不被丢弃，然后交给路由函数进行处理。

在该函数中可以替换 IP 包的目的地址，即做 DNAT。

（2）NF_IP_LOCAL_IN（1）： 所有发送给本机（local computer）的数据包都要通过该函数的处理，该函数通过读取管理员设置的规则对数据包进行处理，如果数据包不被丢弃，然后交给本地的应用程序。

（3）NF_IP_FORWARD（2）：所有不是发送个本机的数据包都要通过该函数碱性处理，该函数通过读取管理员设置的规则对数据包进行处理，如果数据包不被丢弃，交给 NF_IP_POST_ROUTING 进行处理。

（4）NF_IP_LOCAL_OUT （3）：所有从本地应用程序出来的数据包，必须通过该函数的处理，该函数通过读取管理员设置的规则对数据包进行处理，如果数据包不被丢弃，然后交给路由函数进行处理。

（5）NF_IP_POST_ROUTING（4）：所有数据包在发给其它主机之前需要通过该函数的处理，该函数通过读取管理员设置的规则对数据包进行处理，如果数据包不被丢弃，将数据包发给数据链路层。

在该函数中可以替换 IP 包的源地址，即做 SNAT。

上面只是对5个hook函数进行了简要的说明，图2-16显示了数据包在通过Linux防火墙时的处理过程：

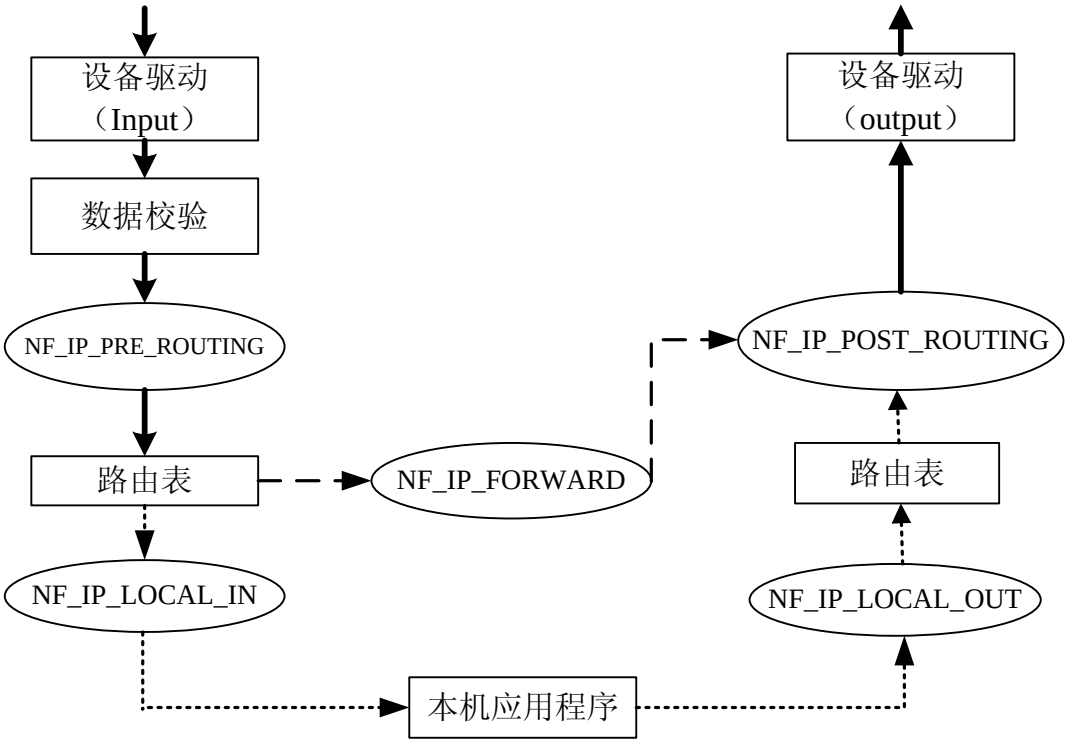


图 2-16

4.2 包过滤

刚才已经介绍，每个函数都可以对数据包进行处理，处理的最基本的操作就是对数据包进行过滤。

管理员可以通过 iptables 工具来向内核模块注册多个过滤规则，并且指名过滤规则的优先权，这样，么个 hook 按照规则进行匹配，如果与规则匹配，函数就会进行一些过滤操作，这些操作主要是以下几个：

（1）NF_ACCEPT：继续正常的传递包。

（2）NF_DROP：丢弃包，停止传送。

（3）NF_STOLEN：我已经接管了包，不要继续传送。

（4）NF_QUEUE：排列包（将包放入队列）。

（5）NF_REPEAT：再次使用该 hook。

4.3 包选择: IP 表(IP Tables)

在 netfilter 框架上已经创建了一个包选择系统。这是一个 ipchains(来自 ipfwadm, 来自 BSD 的 ipfw IIRC)的直接可扩展的派生产物。内核可以在该系统上注册一个新的表，并要求包在新表中传送。这个包选择工具默认已经注册了 3 个表，分别是：过滤(“filter”表)， 网络地址翻译(“NAT”表)。 和 general pre-route packet mangling (“'mangle”表)。hook 与 IP Tables 同时注册的表情况如图 2-17 所示：

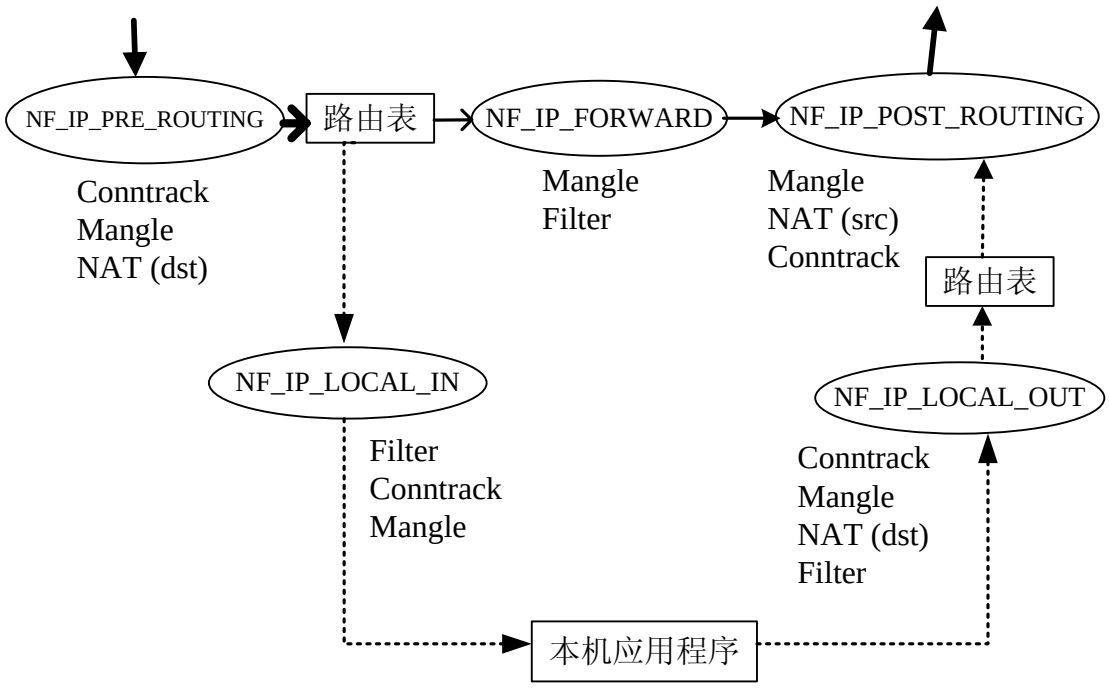


图 2-17

注意表中各个 IP 表的顺序，在调用 hook 函数时是按照表的顺序来调用的。例如在执行 NF_IP_PRE_ROUTING 时，首先检查 Conntrack，然后检查 Mangle 表，最后检查 NAT 表。

下面分别对各个表做一个简单说明：

（1）包过滤表（Filter）

这个表“filter”，绝不能改变包，而应该仅仅过滤它们。实际中由网络过滤框架来提供 NF_IP_FORWARD 挂钩（hook）的输出和输入接口使得很多过滤工作变得非常简单。从图中可以看出，NF_IP_LOCAL_IN 和 NF_IP_LOCAL_OUT 也可以做过滤，但是只是针对本机。

（2）网络地址转换（NAT）

它分别服务于两套不同的网络过滤挂钩的包， 对于非本地包，NF_IP_PRE_ROUTING 和 NF_IP_POST_ROUTING 挂钩可以完美的解决源地址和目的地址的变更。

这个表有与“filter”表有一点点不同，只有新连接的第一个包会在表中传送，结果将被用于用于以后所有来自这一连接的包。例如某一个连接的第一个数据包在这个表中被替换了源地址，那么以后这条连接的所有包都将被替换源地址。

（3）mangle 表

包的 mangling 表(“mangle”表)被用于真正的改变包的信息。例如 TOS 和 TCP MSS。 mangle 表和所有的 5 个网络过滤的挂钩都有关。 (注意这只在 2.4.18 后的内核才有效，以前的内核并没有关联 mangle 和挂钩)

5. 配置包过滤防火墙

Linux 防火墙的基本功能之一就是数据包的过滤。这一节介绍数据包的过滤。

5.1 iptables 介绍

Iptalbes 是用来设置、维护和检查 Linux 内核的 IP 包过滤规则的。

iptables 与定义了三个表(table)，即前面提到的 filter、nat 和 mangle 表；每个表下面包含了多个链（chain)；每个链的下面又包含多条规则（rule)。简而言言之，iptables 主要对表（table)、链（chain）和规则（rule）进行管理。

iptables 预定义了 5 个链，分别对应 netfilter 的 5 个 hook，这 5 个链分别是：

INPUT、FORWARD、OUTPUT、PREROUTING、POSTROUTING。

iptables 命令的基本语法是：

iptables –t 表 命令 选项 动作

例如：iptables –t filter –A FORWARD –s 192.168.1.0/24 –j DROP

其中 –A 就是命令， –s 就是选项， –j 指定动作。

要查看某个表下的各个链的信息可以使用 iptables -L –n。

要使您的 Linux 系统成为网络防火墙，当然除了内核支持之外，还需要启用 Linux 的网络转发功能：

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

要使系统启动时就具有该功能，可以将上面的命令写入/etc/rc.d/rc.local 文件中。

5.2 数据包穿过表和链的顺序

当数据包到达防火墙时，如果 MAC 地址符合，就会由内核里相应的驱动程序

接收，然后会经过一系列操作，从而决定是发送给本地的程序，还是转发给其他机子。数据包通过防火墙时分下面的三种情况：

（1）以本地为目标的包

当一个数据包进入防火墙后，如果目的地址是本机，被防火墙进行检查的顺序如表 2-1 所示。如果在某一个步骤数据包被丢弃，当然就不会执行后面的检查了。

表 2-1

步骤	表	链	注释
1			在线路上传输(例如，Internet)
2			进入接口 (例如， eth0)
3	mangle	PREROUTING	这个链用来 mangle 数据包，例如改变 TOS 等
4	nat	PREROUTING	这个链主要用来做 DNAT。不要在这个链做过虑操作，因为某些情况下包会溜过去。
5			路由判断，例如，包是发往本地的，还是要转发的。
6	mangle	INPUT	在路由之后，被送往本地程序之前，mangle 数据包。
7	filter	INPUT	所有以本地为目的的包都要经过这个链，不管它们从哪儿来，对这些包的过滤条件就设在这里。
8			到达本地程序了(例如，服务程序或客户程序)

（2）以本地为源的包

本地应用程序发出的数据包，被防火墙进行检查的顺序如表 2-2 所示。

步骤	表	链	注释
1			本地程序（例如，服务程序或客户程序）
2			路由判断，要使用源地址，外出接口，还有其他一些信息。
3	mangle	OUTPUT	在这儿可以 mangle 包。建议不要在这儿做过滤，可能有副作用。
4	nat	OUTPUT	这个链对从防火墙本身发出的包进行 DNAT 操作。
5	filter	OUTPUT	对本地发出的包过滤。
6	mangle	POSTROUTING	进行数据包的修改
7	nat	POSTROUTING	在这里做 SNAT。但不要在这里做过滤，因为有副作用，而且有些包是会溜过去的，即使你用了 DROP 策略。
8			离开接口(例如： eth0)
9			在线路上传输(例如，Internet)

（3）被转发的包

需要通过防火墙转发的数据包，被防火墙进行检查的顺序如表 2-3 所示。

表 2-3

步骤	表	链	注释
1			在线路上传输(例如，Internet)
2			进入接口（例如， eth0）
3	mangle	PREROUTING	mangle 数据包，，例如改变 TOS 等。
4	nat	PREROUTING	这个链主要用来做 DNAT。不要在这个链做过虑操作，因为某些情况下包会溜过去。稍后会做 SNAT。
5			路由判断，例如，包是发往本地的，还是要转发的。
6	mangle	FORWARD	包继续被发送至 mangle 表的 FORWARD 链，这是非常特殊的情况才会用到的。在这里，包被 mangle。这次 mangle 发生在最初的路由判断之后，在最后一次更改包的目的之前。
7	filter	FORWARD	包继续被发送至这条 FORWARD 链。只有需要转发的包才会走到这里，并且针对这些包的所有过滤也在这里进行。注意，所有要转发的包都要经过这里，不管是外网到内网的还是内网到外网的。
8	mangle	POSTROUTING	这个链也是针对一些特殊类型的包。这一步 mangle 是在所有更改包的目的地址的操作完成之后做的，但这时包还在本地上。
9	nat	POSTROUTING	这个链就是用来做 SNAT 的，当然也包括 Masquerade（伪装）。但不要在这儿做过滤，因为某些包即使不满足条件也会通过。
10			离开接口(例如： eth0)
11			又在线路上传输了(例如，LAN)

5.3 包过滤的动作

在对包进行过滤时，常用的有以下 3 个动作（其它的特殊操作后面会有介绍）：

（1）ACCEPT：一旦包满足了指定的匹配条件，就会被 ACCEPT，并且不会再去匹配当前链中的其他规则或同一个表内的其他规则，但它还要通过其他表中的链。

（2）DROP：顾名思义，如果包符合条件，数据包就会被丢掉，也就是说包的生命到此结束，不会再向前走一步，效果就是包被阻塞了。在某些情况下，这个动作会引起意外的结果，因为它不会向发送者返回任何信息，也不会向路由器返回信息，这就可能会使连接的另一方的 sockets 因苦等回音而亡。 解决这个问题的较好的办法是使用 REJECT 动作（没有特殊需求的话尽量使用 DROP）。

（3）REJECT：和 DROP 基本一样，区别在于它除了阻塞包之外，还向发送者返回错误信息。现在，此操作还只能用在 INPUT、FORWARD、OUTPUT 和它们的子链里，而且包含 REJECT 的链也只能被它们调用，否则不能发挥作用。它只有一个选项，是用来控制返回的错误信息的种类的。虽然有很多种类，但如果你有 TCP/IP 方面的基础知识，就很容易理解它们。

5.4 链中规则（rule）的匹配顺序

下面这一段是 filter 表中的 FORWARD 链的输出：

```
Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  192.168.100.0/28     0.0.0.0/0
ACCEPT     all  --  0.0.0.0/0            211.99.0.0/24
ACCEPT     all  --  172.16.0.0/16        0.0.0.0/0
mychain    tcp  --  10.0.0.0/24          0.0.0.0          tcp dpt:80
```

现有一数据包，源地址为 192.168.1.58，目的地址为 202.127.124.110，协议为 TCP，源端口为 2038，目的端口为 80。当该数据包通过 FORWARD 链时，从上往下开始匹配：

（1）与第 1 条规则：源为 192.168.100.0/28，源不匹配。

（2）与第 2 条规则：目的为 211.99.0.0/24，目标不匹配。

（3）与第 3 条规则：源为 172.16.0.0/16，源不匹配。

（4）与第 4 条规则：源为 10.0.0.0/24，源不匹配。

所有规则都不匹配，最后交给缺省规则来处理，在本实例中，缺省的动作是 DROP（输出中的 policy DROP），因此该数据包被丢弃。

再看另一个数据包，源地址为 192.168.1.58，目的地址为 211.99.0.110，协议为 TCP，源端口为 2038，目的端口为 80。当该数据包通过 FORWARD 链时，从上往下开始匹配：

（1）与第 1 条规则：源为 192.168.100.0/28，源不匹配。

（2）与第 2 条规则：源地址为任意（0.0.0.0/0），匹配；目的地址为 211.99.0.0/24，211.99.0.110 在范围内，匹配；源端口为任意，匹配；目的端口为任意，匹配；协议为任意（all），匹配；规则链对该数据包的动作为 ACCEPT，因此该数据包通过。

如果数据包的源地址为 10.0.0.35，目的地址为 211.99.0.110，协议为 TCP，源端口为 2038，目的端口为 80。当该数据包通过 FORWARD 链时，从上往下开始匹配，当匹配到第 4 条规则时匹配，动作为 mychain，此时数据包会被传递到用户自己定义的规则链 mychain。

5.5 配置实例

实验的网络拓扑图如图 2-18 所示。

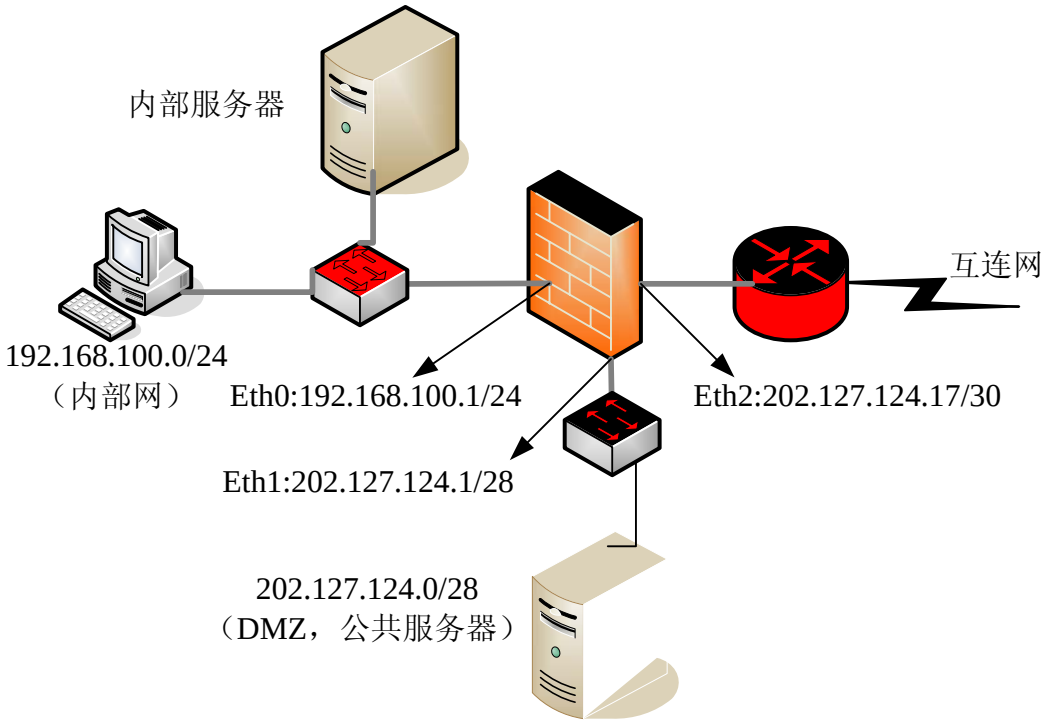


图 2-18

5.5.1 基于 IP 和端口的过滤

下面列举几个实例来对基于 IP 和端口的过滤进行简要说明。

（1）凡是来自内网的访问公共服务器的 http 服务的数据包允许通过：

```
iptables -t filter –F    #清除原来的所有规则
iptables -t filter -P FORWARD DROP        #设置缺省策略
iptables –t filter –A FORWARD –p tcp –s 192.168.100.0/24    -d 202.127.124.0/28
--dport 80 –j ACCEPT
```

在该案例中，“-A”指定向某个链（chain）中添加一条规则；-p 指定协议（TCP，UDP，ICMP 等。如果该参数忽略，则匹配所有的协议。）；-s 指定源地址段（如果该参数忽略或者为 0.0.0.0/0，源地址为任何地址）；-d 指定目的地址；--dport 指定目的端口。

（2）从任何位置可以访问公共服务器的 FTP 服务：

对于 FTP 服务，需要允许访问两个端口 21（命令传输）、20（数据传输）：

```
iptables -t filter –F
iptables -t filter -P FORWARD DROP
iptables -t filter -A FORWARD -p tcp -d 202.127.124.0/28 --dport 20 -j ACCEPT
iptables -t filter -A FORWARD -p tcp -d 202.127.124.0/28 --dport 21 -j ACCEPT
```

（3）禁止 IP 为 192.168.100.78 的主机访问任何 FTP 服务器：

```
iptables -t filter –F
iptables -t filter -P FORWARD ACCEPT
iptables -t filter -A FORWARD -p tcp -s 192.168.100.78 --dport 21 -j DROP
```

（4）除网管主机 192.168.100.100 外，所有主机都禁止访问公共服务器的 22 端口：

```
iptables -t filter –F
iptables -t filter -P FORWARD ACCEPT
iptables -t filter -A FORWARD -p tcp -s ! 192.168.100.100 -d 202.127.124.110 --dport 22 -j DROP
```

5.5.2 基于接口（网卡）的过滤

从互连网可以访问公共服务器的 http 服务：

```
iptables -t filter –F
iptables -t filter -P FORWARD DROP
iptables -t filter -A FORWARD -p tcp -i eth2 -o eth1 --dport 80 -j ACCEPT
```

使用-i 参数来指定数据包的来源网卡，使用-o 来指定数据包将从哪个网卡出去。从图 2-18 中互连网的数据包从 eth2 进入，从 eth1 出去。

需要注意的是在 INPUT 链中不能使用-o 选项，OUTPUT 链中不能使用-i 选项。

5.5.3 基于 tcp 标志位的过滤（防止 syn-flood 攻击）

在 TCP 协议中已经介绍了 TCP 头部的 6 个标志比特的作用，Linux 防火墙可以基于这些标志位来对数据包进行过滤。

例如：

```
iptables -t filter -A FORWARD -p tcp -i eth2 --tcp-flags ALL    SYN,ACK -j DROP
```

上面这条规则的含义是：当数据包从 eth2 接口进入，并协议为 TCP 时，检测 TCP 的所有（ALL）标志位，如果 SYN 和 ACK 标志位都设置了（都为 1），那么丢弃该数据包。

但是在实际应用中，如果单纯的对标志位进行匹配意义不大。可能我们也听说过 SYS 洪水攻击，SYS 洪水攻击是利用 TCP/IP 协议 3 次握手的原理，发送大量的建立连接的网络包，但不实际建立连接，最终导致被攻击服务器的网络队列被占满，无法被正常用户访问。

因此，为了防止过多的空连接，我们可以对每秒钟同一 IP 地址发送来的 SYS 数据包的个数进行限制：

```
iptables -A FORWARD -p tcp -i eth2 --syn -m limit --limit 1/s -j ACCEPT
```

--syn 是--tcp-flags SYN,RST,ACK SYN 的缩写。上面这条规则的含义就是在每秒钟内只接受一个 TCP 连接，其余的连接将被丢弃。

```
iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --limit 1/s -j ACCEPT
```

上面这条规则的意思是检测数据包的 SYN,ACK,FIN,RST 标志位，如果 RST 被设置，这样的数据包在 1 秒钟内只接受一个，其余的将被丢弃。

```
iptables -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s -j ACCEPT
```

在一秒钟内只接受一个类型为 echo-request 的 ICPMP 包。

5.5.4 基于 Mac 地址的过滤

例如：

```
iptables -A FORWARD -m mac --mac-source 00:0C:29:B7:8B:E0 -j DROP
```

上面这条规则的作用就是阻止 mac 地址为 00:0C:29:B7:8B:E0 的数据包通过防火墙。

5.5.5 基于状态的过滤检测

在这里我必须要解释一下防火墙的状态(state)。例如，用 ssh 远程访问，你的主机和远程主机会进行通信。如果使用静态的防火墙会这样处理：

检查时入机器的数据包，发现数据是目的端口是 22 端口，当允许时，连接之后相互通信的数据也一样，检查每个数据，发现数据来源于 22 端口，允许通过。也就是说每个数据包都需要走规则表。

如果用有状态的防火墙如何处理呢？

当连接远程主机成功之后，防火墙会把这个连接记录下来，当有数据从远程 ssh 服务器再进入你的机器时，检查自己连接状态表，发现这个数据来源于一个已经建立的连接，允许这个数据包进入。

以上两种处理，我们明显的发现静态防火墙比较生硬，而有状态的防火墙则显得要智能一些！Linux 防火墙就支持这种动态检测的功能。

在 iptables 中，可以使用的连接状态有：

（1）NEW：一个新连接。

（2）ESTABLISHED：正在传输数据的连接。

（3）RELATED：像 ftp 这样的服务，用 21 端口传送命令，而用 20 端口(port 模式)或其他端口(PASV 模式)传送数据。在已有 21 端口上建立好连接后发送命令，用 20 传送的数据，状态是 RELATED

（4）INVALID：表示包是未知连接。

例如：

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

上面这条规则的含义就是当有数据包通过防火墙时，首先会检查连接状态，如果连接状态是 ESTABLISHED 或 RELATED，那么就不会再通过防火墙规则的检查。这样做可以减少防火墙的工作量。

5.5.6 ICMP 数据包的过滤

要对 ICMP 协议的数据包进行过滤，在设定规则时，使用-p 选项来指定协议即可；要进行更详细的控制，可以使用--icmp-type 来指定 ICMP 数据报的类型。例如，如果要禁止别人 ping 公共服务器区域的主机，可以加入下面的规则：

```
iptables -A FORWARD -p icmp --icmp-type echo-request -d 202.127.124.0/28 -j DROP
```

在 iptables 中可以使用的 icmp 数据报的类型名称，可以通过“iptables -p icmp –h”来查看。

6. 配置 NAT

通常，网络中的（IP）包从他们的源（地址）出发（比如你家的电脑），到他们的目的地（比如 www.vfast.com ），会经过很多不同的连接，这些连接不会真去修

改你的包，他们只是照原样传出去。这样一来，如果发出数据包的主机使用的源地址是私用网络地址，该数据包就不能在互连网上传输。要能够使用私有网络访问互连网，就必须要做 NAT（Network Address Translation，网络地址转换）。

我把 NAT 分为两种不同的类型：源 NAT(SNAT)和目标 NAT(DNAT)。（注：以下不再翻译 SNAT 和 DNAT，直接用 Source NAT 和 Destination NAT）

Source NAT 是指修改第一个包的源地址：也就是说，改变连接的来源地。Source NAT 会在包送出之前的最后一刻做好 post-routing（动作），伪装是 SNAT 的一种特殊形式。SNAT 通常用于使使用了私网地址的局域网络能够访问互连网络。

Destination NAT 是指修改第一个包的目标地址：也就是说，改变连接的目的地。Destination NAT 总是在包进入以后（马上）进行 before routing（动作）。端口转发、负载均衡和透明代理都属于 DNAT。

6.1 SNAT 的配置

在图 2-18 所示的实例中，如果要让 192.168.100.0/24 的主机能够连接到互连网，就需要在防火墙上做 SNAT：

```
iptables -t nat -A POSTROUTING -s 192.168.100.0/28 -o eth2 -j SNAT --to 202.127.124.17
```

这样只有到互连网络的数据才做 SNAT，而访问公共服务器不会做 SNAT。

需要注意的是，做 SNAT 时只能在 POSTROUTING 中进行。

另外，SNAT 还可以这样进行：

```
# 改变源地址为 1.2.3.4、1.2.3.5 或者 1.2.3.6
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.3.4-1.2.3.6

# 改变源地址为 1.2.3.4，端口 1-1023
# iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to 1.2.3.4:1-1023
```

如果是拨号网络，这时拨号服务器（此时拨号服务器应该是防火墙本身）获取的 IP 地址是不固定的。因此无法指定要转换的 IP 地址。Linux 防火墙给我们提供了另一种实现 SNAT 的方法，那就是伪装（MASQUERADE）：

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

6.2 DNAT 的配置

DNAT 的目的主要是使互连网络上的主机可以访问局域网络内部的服务器或主

机。例如在图 2-18 的实例中，假设内部服务器安装了 Oracle 数据库，其 IP 地址为 192.168.100.66，监听的端口为 1521，要想使员工在家中也可以直接访问公司内部的 Oracle 数据库，此时必须在防火墙上做 DNAT。加入我们让员工在家时要访问 oracle 数据库，就访问 202.127.124.5（实际上服务器是在局域网内）那么我们在防火墙上需要做下面的设置：

```
iptables -t nat -A PREROUTING -p tcp -i eth2 -d 202.127.124.5 --dport 1521 -j DNAT --to 192.168.100.66:1521
```

需要注意的是，如果做 DNAT，只能在 PREROUTING 内进行。

整台主机都影射：

```
iptables -t nat -A PREROUTING -i eth2 -d 202.127.124.5-j DNAT --to 192.168.100.66
```

7. iptables 常用命令选项说明

Iptalbes 是用来设置、维护和检查 Linux 内核的 IP 包过滤规则的。

可以定义不同的表，每个表都包含几个内部的链，也能包含用户定义的链。每个链都是一个规则列表，对对应的包进行匹配：每条规则指定应当如何处理与之相匹配的包。这被称作'target'（目标），也可以跳向同一个表内的用户定义的链。

TARGETS

防火墙的规则指定所检查包的特征，和目标。如果包不匹配，将送往该链中下一条规则检查;如果匹配,那么下一条规则由目标值确定.该目标值可以是用户定义的链名，或是某个专用值，如 ACCEPT[通过]，DROP[删除]，QUEUE[排队]，或者 RETURN[返回]。

ACCEPT 表示让这个包通过。DROP 表示将这个包丢弃。QUEUE 表示把这个包传递到用户空间。RETURN 表示停止这条链的匹配，到前一个链的规则重新开始。如果到达了一个内建的链(的末端)，或者遇到内建链的规则是 RETURN，包的命运将由链准则指定的目标决定。

TABLES

当前有三个表（哪个表是当前表取决于内核配置选项和当前模块)。

-t table

这个选项指定命令要操作的匹配包的表。如果内核被配置为自动加载模块，这时若模块没有加载，(系统)将尝试(为该表)加载适合的模块。这些表如下：filter,这是默认的表，包含了内建的链 INPUT（处理进入的包）、FORWORD（处理通过的包）和 OUTPUT（处理本地生成的包）。nat,这个表被查询时表示遇到了产生新的连接的包,由三个内建的链构成：PREROUTING (修改到来的包)、OUTPUT（修改路由之前本地的包）、POSTROUTING（修改准备出去的包）。mangle 这个表用来对指定的包进行修改。它有两个内建规则：PREROUTING（修改路由之前进入的包）和 OUTPUT（修改路由之前本地的包）。

COMMANDS

这些选项指定执行明确的动作：若指令行下没有其他规定,该行只能指定一个选项.对于长格式的命令和选项名,所用字母长度只要保证 iptables 能从其他选项中区分出该指令就行了。

-A -append

在所选择的链末添加一条或更多规则。当源（地址）或者/与 目的（地址）转换为多个地址时，这条规则会加到所有可能的地址(组合)后面。

-D -delete

从所选链中删除一条或更多规则。这条命令可以有两种方法：可以把被删除规则指定为链中的序号(第一条序号为 1),或者指定为要匹配的规则。

-R -replace

从选中的链中取代一条规则。如果源（地址）或者/与 目的（地址）被转换为多地址，该命令会失败。规则序号从 1 开始。

-I -insert

根据给出的规则序号向所选链中插入一条或更多规则。所以，如果规则序号为 1，规则会被插入链的头部。这也是不指定规则序号时的默认方式。

-L -list

显示所选链的所有规则。如果没有选择链，所有链将被显示。也可以和 z 选项一起使用，这时链会被自动列出和归零。精确输出受其它所给参数影响。

-F -flush

清空所选链。这等于把所有规则一个个的删除。

--Z -zero

把所有链的包及字节的计数器清空。它可以和 -L 配合使用，在清空前察看计数器，请参见前文。

-N -new-chain

根据给出的名称建立一个新的用户定义链。这必须保证没有同名的链存在。

-X -delete-chain

删除指定的用户自定义链。这个链必须没有被引用，如果被引用，在删除之前你必须删除或者替换与之有关的规则。如果没有给出参数，这条命令将试着删除每个非内建的链。

-P -policy

设置链的目标规则。

-E -rename-chain

根据用户给出的名字对指定链进行重命名，这仅仅是修饰，对整个表的结构没有影响。TARGETS 参数给出一个合法的目标。只有非用户自定义链可以使用规则，而且内建链和用户自定义链都不能是规则的目标。

-h Help.

帮助。给出当前命令语法非常简短的说明。

PARAMETERS（参数）

以下参数构成规则详述，如用于 add、delete、replace、append 和 check 命令。

-p -protocal [!]protocol

规则或者包检查(待检查包)的协议。指定协议可以是 tcp、udp、icmp 中的一个或者全部,也可以是数值,代表这些协议中的某一个。当然也可以使用在/etc/protocols 中定义的协议名。在协议名前加上"!"表示相反的规则。数字 0 相当于所有 all。Protocol all 会匹配所有协议，而且这是缺省时的选项。在和 check 命令结合时，all 可以不被使用。

-s -source [!] address[/mask]

指定源地址，可以是主机名、网络名和清楚的 IP 地址。mask 说明可以是网络掩码或清楚的数字，在网络掩码的左边指定网络掩码左边"1"的个数，因此，mask 值为 24 等于 255.255.255.0。在指定地址前加上"!"说明指定了相反的地址段。标志 --src 是这个选项的简写。

-d --destination [!] address[/mask]

指定目标地址，要获取详细说明请参见 -s 标志的说明。标志 --dst 是这个选项的简写。

-j --jump target

-j 目标跳转

指定规则的目标；也就是说，如果包匹配应当做什么。目标可以是用户自定义链（不是这条规则所在的），某个会立即决定包的命运的专用内建目标，或者一个扩展（参见下面的 EXTENSIONS）。如果规则的这个选项被忽略，那么匹配的过程不会对包产生影响，不过规则的计数器会增加。

-i -in-interface [!] [name]

i -进入的（网络）接口 [!][名称]

这是包经由该接口接收的可选的入口名称，包通过该接口接收（在链 INPUT、FORWORD 和 PREROUTING 中进入的包）。当在接口名前使用"!"说明后，指的是相反的名称。如果接口名后面加上"+"，则所有以此接口名开头的接口都会被匹配。如果这个选项被忽略，会假设为"+"，那么将匹配任意接口。

-o --out-interface [!][name]

-o --输出接口[名称]

这是包经由该接口送出的可选的出口名称，包通过该口输出（在链 FORWARD、OUTPUT 和 POSTROUTING 中送出的包）。当在接口名前使用"!"说明后，指的是相反的名称。如果接口名后面加上"+"，则所有以此接口名开头的接口都会被匹配。如果这个选项被忽略，会假设为"+"，那么将匹配所有任意接口。

[!] -f, --fragment

[!] -f --分片

这意味着在分片的包中，规则只询问第二及以后的片。自那以后由于无法判断这种把包的源端口或目标端口（或者是 ICMP 类型的），这类包将不能匹配任何指定对他们进行匹配的规则。如果"!"说明用在了"-f"标志之前，表示相反的意思。

OTHER OPTIONS（其他选项）

还可以指定下列附加选项：

-v --verbose

-v --详细

详细输出。这个选项让 list 命令显示接口地址、规则选项（如果有）和 TOS（Type of Service）掩码。包和字节计数器也将被显示，分别用 K、M、G(前缀)表示 1000、1,000,000 和 1,000,000,000 倍（不过请参看-x 标志改变它），对于添加,插入,删除和替换命令，这会使一个或多个规则的相关详细信息被打印。

-n --numeric

-n --数字

数字输出。IP 地址和端口会以数字的形式打印。默认情况下，程序试显示主机名、网络名或者服务（只要可用）。

-x -exact

-x -精确

扩展数字。显示包和字节计数器的精确值，代替用 K,M,G 表示的约数。这个选项仅能用于 -L 命令。

--line-numbers

当列表显示规则时，在每个规则的前面加上行号，与该规则在链中的位置相对应。

MATCH EXTENSIONS（对应的扩展）

iptables 能够使用一些与模块匹配的扩展包。以下就是含于基本包内的扩展包，而且他们大多数都可以通过在前面加上!来表示相反的意思。

tcp

当 --protocol tcp 被指定,且其他匹配的扩展未被指定时,这些扩展被装载。它提供以下选项：

--source-port [!] [port[:port]]

源端口或端口范围指定。这可以是服务名或端口号。使用格式端口：端口也可以指定包含的（端口）范围。如果首端口号被忽略，默认是"0"，如果末端口号被忽略，默认是"65535"，如果第二个端口号大于第一个，那么它们会被交换。这个选项可以使用 --sport 的别名。

--destionation-port [!] [port:[port]]

目标端口或端口范围指定。这个选项可以使用 --dport 别名来代替。

--tcp-flags [!] mask comp

匹配指定的 TCP 标记。第一个参数是我们要检查的标记，一个用逗号分开的列表，第二个参数是用逗号分开的标记表,是必须被设置的。标记如下：SYN ACK FIN RST URG PSH ALL NONE。因此这条命令：iptables -A FORWARD -p tcp --tcp-flags SYN, ACK, FIN, RST SYN 只匹配那些 SYN 标记被设置而 ACK、FIN 和 RST 标记没有设置的包。

[!] --syn

只匹配那些设置了 SYN 位而清除了 ACK 和 FIN 位的 TCP 包。这些包用于 TCP 连接初始化时发出请求；例如，大量的这种包进入一个接口发生堵塞时会阻止进入的 TCP 连接，而出去的 TCP 连接不会受到影响。这等于 --tcp-flags SYN, RST, ACK

SYN。如果"--syn"前面有"!"标记，表示相反的意思。

--tcp-option [!] number

匹配设置了 TCP 选项的。

udp

当 protocol udp 被指定,且其他匹配的扩展未被指定时,这些扩展被装载,它提供以下选项：

--source-port [!] [port:[port]]

源端口或端口范围指定。详见 TCP 扩展的--source-port 选项说明。

--destination-port [!] [port:[port]]

目标端口或端口范围指定。详见 TCP 扩展的--destination-port 选项说明。

icmp

当 protocol icmp 被指定,且其他匹配的扩展未被指定时,该扩展被装载。它提供以下选项：

--icmp-type [!] typename

这个选项允许指定 ICMP 类型，可以是一个数值型的 ICMP 类型，或者是某个由命令 iptables -p icmp -h 所显示的 ICMP 类型名。

mac

--mac-source [!] address

匹配物理地址。必须是 XX:XX:XX:XX:XX 这样的格式。注意它只对来自以太设备并进入 PREROUTING、FORWORD 和 INPUT 链的包有效。

limit

这个模块匹配标志用一个标记桶过滤器一一定速度进行匹配,它和 LOG 目标结合使用来给出有限的登陆数.当达到这个极限值时,使用这个扩展包的规则将进行匹

配.(除非使用了"!"标记)

--limit rate

最大平均匹配速率：可赋的值有'/second', '/minute', '/hour', or '/day'这样的单位，默认是 3/hour。

--limit-burst number

待匹配包初始个数的最大值:若前面指定的极限还没达到这个数值,则概数字加1.默认值为 5

multiport

这个模块匹配一组源端口或目标端口,最多可以指定 15 个端口。只能和-p tcp 或者 -p udp 连着使用。

--source-port [port[, port]]

如果源端口是其中一个给定端口则匹配

--destination-port [port[, port]]

如果目标端口是其中一个给定端口则匹配

--port [port[, port]]

若源端口和目的端口相等并与某个给定端口相等,则匹配。

mark

这个模块和与 netfilter 过滤器标记字段匹配（就可以在下面设置为使用 MARK 标记）。

--mark value [/mask]

匹配那些无符号标记值的包（如果指定 mask，在比较之前会给掩码加上逻辑的标记）。

owner

此模块试为本地生成包匹配包创建者的不同特征。只能用于 OUTPUT 链，而且即使这样一些包（如 ICMP ping 应答）还可能没有所有者，因此永远不会匹配。

--uid-owner userid

如果给出有效的 user id，那么匹配它的进程产生的包。

--gid-owner groupid

如果给出有效的 group id，那么匹配它的进程产生的包。

--sid-owner seessionid

根据给出的会话组匹配该进程产生的包。

state

此模块，当与连接跟踪结合使用时，允许访问包的连接跟踪状态。

--state state

这里 state 是一个逗号分割的匹配连接状态列表。可能的状态是:INVALID 表示包是未知连接，ESTABLISHED 表示是双向传送的连接，NEW 表示包为新的连接，否则是非双向传送的，而 RELATED 表示包由新连接开始，但是和一个已存在的连接在一起，如 FTP 数据传送，或者一个 ICMP 错误。

unclean

此模块没有可选项，不过它试着匹配那些奇怪的、不常见的包。处在实验中。

tos

此模块匹配 IP 包首部的 8 位 tos（服务类型）字段（也就是说，包含在优先位中）。

--tos tos

这个参数可以是一个标准名称，（用 iptables -m tos -h 察看该列表），或者数值。

TARGET EXTENSIONS

iptables 可以使用扩展目标模块：以下都包含在标准版中。

LOG

为匹配的包开启内核记录。当在规则中设置了这一选项后，linux 内核会通过 printk()打印一些关于全部匹配包的信息（诸如 IP 包头字段等）。

--log-level level

记录级别（数字或参看 syslog.conf(5)）。

--log-prefix prefix

在纪录信息前加上特定的前缀：最多 14 个字母长，用来和记录中其他信息区别。

--log-tcp-sequence

记录 TCP 序列号。如果记录能被用户读取那么这将存在安全隐患。

--log-tcp-options

记录来自 TCP 包头部的选项。

--log-ip-options

记录来自 IP 包头部的选项。

MARK

用来设置包的 netfilter 标记值。只适用于 mangle 表。

--set-mark mark

REJECT

作为对匹配的包的响应，返回一个错误的包：其他情况下和 DROP 相同。

此目标只适用于 INPUT、FORWARD 和 OUTPUT 链，和调用这些链的用户自定义链。这几个选项控制返回的错误包的特性：

--reject-with type

Type 可以是 icmp-net-unreachable、icmp-host-unreachable、icmp-port-nreachable、icmp-proto-unreachable、 icmp-net-prohibited 或者 icmp-host-prohibited，该类型会返回相应的 ICMP 错误信息（默认是 port-unreachable）。选项 echo-reply 也是允许的；它只能用于指定 ICMP ping 包的规则中，生成 ping 的回应。最后，选项 tcp-reset 可以用于在 INPUT 链中,或自 INPUT 链调用的规则，只匹配 TCP 协议：将回应一个 TCP RST 包。

TOS

用来设置 IP 包的首部八位 tos。只能用于 mangle 表。

--set-tos tos

你可以使用一个数值型的 TOS 值，或者用 iptables -j TOS -h 来查看有效 TOS 名列表。

MIRROR

这是一个试验示范目标，可用于转换 IP 首部字段中的源地址和目标地址，再传送该包,并只适用于 INPUT、FORWARD 和 OUTPUT 链，以及只调用它们的用户自定义链。

SNAT

这个目标只适用于 nat 表的 POSTROUTING 链。它规定修改包的源地址（此连接以后所有的包都会被影响），停止对规则的检查，它包含选项：

--to-source <ipaddr>[-<ipaddr>][:port-port]

可以指定一个单一的新的 IP 地址，一个 IP 地址范围，也可以附加一个端口范围（只能在指定-p tcp 或者-p udp 的规则里）。如果未指定端口范围，源端口中 512 以下的（端口）会被安置为其他的 512 以下的端口；512 到 1024 之间的端口会被安置为 1024 以下的，其他端口会被安置为 1024 或以上。如果可能，端口不会被修改。

--to-destiontion <ipaddr>[-<ipaddr>][:port-port]

可以指定一个单一的新的 IP 地址，一个 IP 地址范围，也可以附加一个端口范围（只能在指定-p tcp 或者-p udp 的规则里）。如果未指定端口范围，目标端口不会被修改。

MASQUERADE

只用于 nat 表的 POSTROUTING 链。只能用于动态获取 IP（拨号）连接：如果你拥有静态 IP 地址，你要用 SNAT。伪装相当于给包发出时所经过接口的 IP 地址设置一个映像，当接口关闭连接会终止。这是因为当下一次拨号时未必是相同的接口地址（以后所有建立的连接都将关闭）。它有一个选项：

--to-ports <port>[-port>]

指定使用的源端口范围，覆盖默认的 SNAT 源地址选择（见上面）。这个选项只适用于指定了-p tcp 或者-p udp 的规则。

REDIRECT

只适用于 nat 表的 PREROUTING 和 OUTPUT 链，和只调用它们的用户自定义链。它修改包的目标 IP 地址来发送包到机器自身（本地生成的包被安置为地址 127.0.0.1）。它包含一个选项：

--to-ports <port>[<port>]

指定使用的目的端口或端口范围：不指定的话，目标端口不会被修改。只能用于指定了-p tcp 或 -p udp 的规则。