超级网络入门与实践(三) 完全理解 ICMP

(益品黄牛 2007/10/30)

目录

1. 基础篇（向 IP 数据包的送信方传递控制信息的信鸽）

“与隔壁村庄往来的桥因为大雨冲走了。”平时用着这个桥的村民 A 用信鸽将这个信息告诉了后面来的村民 B。

对于通信协议，监测通信错误的发生并互相通知的这种构造是需要的。通信途中数据没了，通信对方已经不存在了等等如果不知道的话，这样通信就不能平稳进行了。对于 IP，相当于信鸽的这种构造是 ICMP（internet control message protocol）。这次分四部分的说明来达到完全理解的目标。

首先用这个“基础篇”来了解 ICMP 的分工和框架，在接着的“实现篇”看一下 ICMP 在 IP 的实际通信中是如何被使用的，进一步作为“运用篇”来阐明 ICMP 和安全的关系。“资料篇”使得能参照 ICMP 的所有报文。那么，就马上从基础篇开始吧。

1.1. 作为 IP 不可缺的功能来制作的

实际上，在 IP 通信，经常有数据包到达不了对方的情况。原因是，在通信途中的某处的一个路由器由于不能处理所有的数据包，就将数据包一个一个丢弃了。或者，虽然到达了对方，但是由于搞错了端口号[1]，服务器软件可能不能接受它。

这时，在错误发生的现场，为了联络而飞过来的信鸽就是 ICMP 报文（図 1）。在 IP 网络上，由于数据包被丢弃等原因，为了控制将必要的信息传递给发信方。ICMP 协议是为了辅助 IP 协议，交换各种各样的控制信息而被制造出来的。制定万维网规格的 IETF 在 1981 年将 RFC792[2]作为 ICMP 的基本规格整理出来了。那个 RFC792 的开头部分里写着“ICMP 是 IP 的不可缺少的部分，所有的 IP 软件必须实现 ICMP 协议”。也就是，ICMP 是为了分担 IP 一部分功能而被制定出来的。

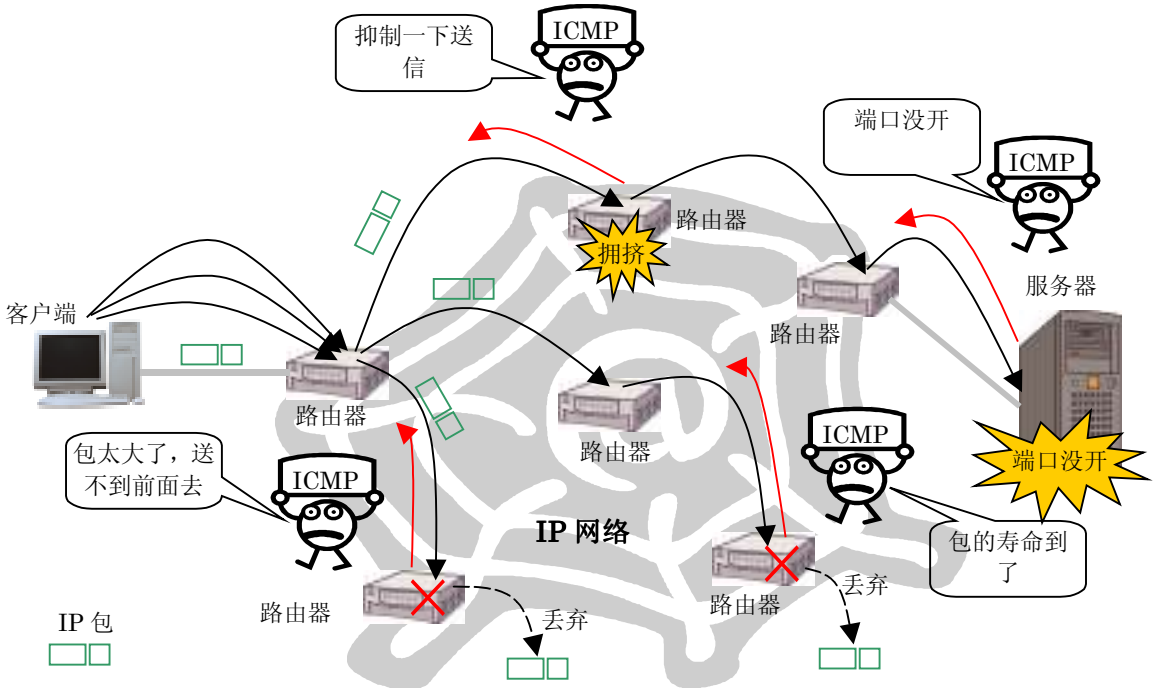

図 1 ICMP 是使 IP 通信平稳运行的辅助协议

[1] 端口号是，在 IP 上工作的 TCP 和 UDP，为了区分同一计算机上跑的不同的应用程序而取的号码。计算机根据收到包的端口号来决定将它传递给哪个应用程序。

[2] IETF 是 Internet Engineering Task Force 的略语。他是发行万维网规格的标准文档 RFC 的机构。RFC 是 Request For Comment 的略语，就是指 IETF 发行的万维网标准技术的文档。

1.2. 使 IP 网络平稳运行的工具

但是，ICMP 协议老是不登到舞台表面上来。即使知道 TCP，UDP，访问网站的 HTTP 协议，但没听说过 ICMP 的读者也不少吧。

那是由 RFC792 决定的规格而引起的。ICMP 只是考虑道：“对于控制 IP 通信需要这些消息吧。”，并准备了用 ICMP 来交流的用于控制的各种各样的消息。但没有具体写“谁，何时，必须要用这些控制”。也就是，RFC 只是准备了使 IP 通信平稳运行的工具[3]。

从信鸽的例子来说，为了传送“桥被冲走了”这个信息，只是放飞了信鸽是没有意义的。接受信鸽，读好信息，寻找别的道路，然后指示别人走这条道路的人（应用程序）也是需要的。在操作系统控制 IP 平稳进行的功能，ping 和 traceroute 等网络命令的背后，工作着 ICMP。单独也运行的，但主要是作为各种功能的幕后劳动者在工作。

1.3. 用途是差错通知和信息查询

这里开始，一点点把老是看不见的 ICMP 暴露出来。从掌握由 RFC 决定的 ICMP 规格开始吧。

首先从使用 ICMP 的方法开始。在 RFC，将 ICMP 大致分成两种功能。那是，[1]给送信者的错误通知；[2]送信者的信息查询（図 2）。[1]是到 IP 数据包被对方的计算机处理的过程中，发生了什么错误时被使用。不仅传送发生了错误这个事实，也传送错误原因等消息。另一方面，[2]的信息询问是在送信方的计算机向对方计算机询问信息时被使用。被询问内容的种类非常丰富，他们有目标 IP 地址的机器是否存在这种基本确认，调查自己网络的子网掩码，取得对方机器的时间信息等。

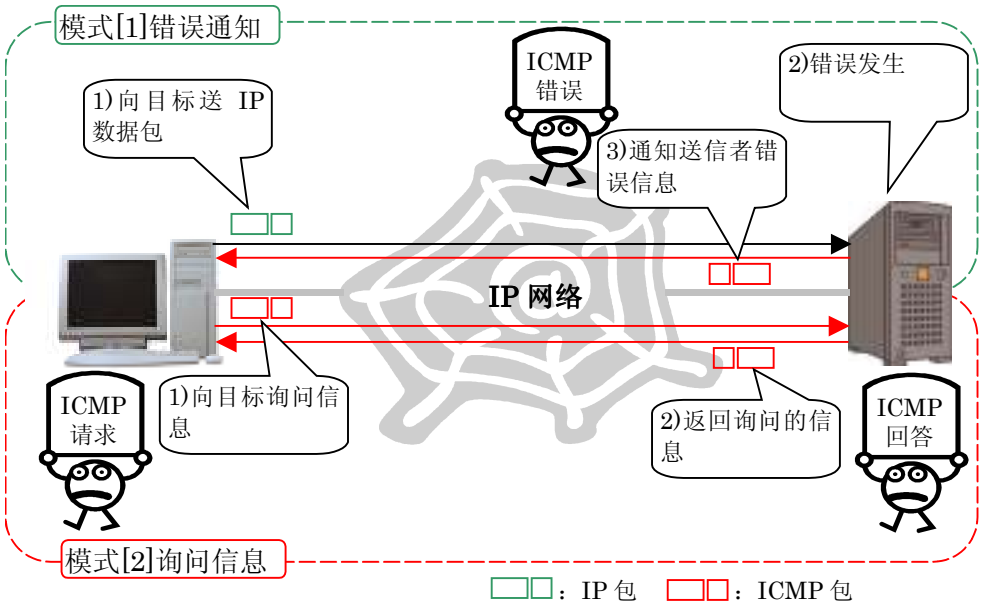

図 2 ICMP 大致两种利用模式

1.4. 作为 IP 的上层协议在工作

那么，ICMP 在错误通知和信息查询的时候，网络上交换着怎样的报文呢？接着，看一下 RFC 规定的数据包格式和报文内容吧。

[3] 实际上，在 RFC792 之后制作的几个 RFC 触及到了 ICMP 的使用。但是，没有深入到具体应用程序的实现。

首先想注视一下，ICMP 的内容是放在 IP 数据包的数据部分里来互相交流的（図 3）。也就是，从 ICMP 的报文格式来说，ICMP 是 IP 的上层协议。但是，正如 RFC 所记载的，ICMP 是分担了 IP 的一部分功能。所以，被认为是与 IP 同层的协议[4]。

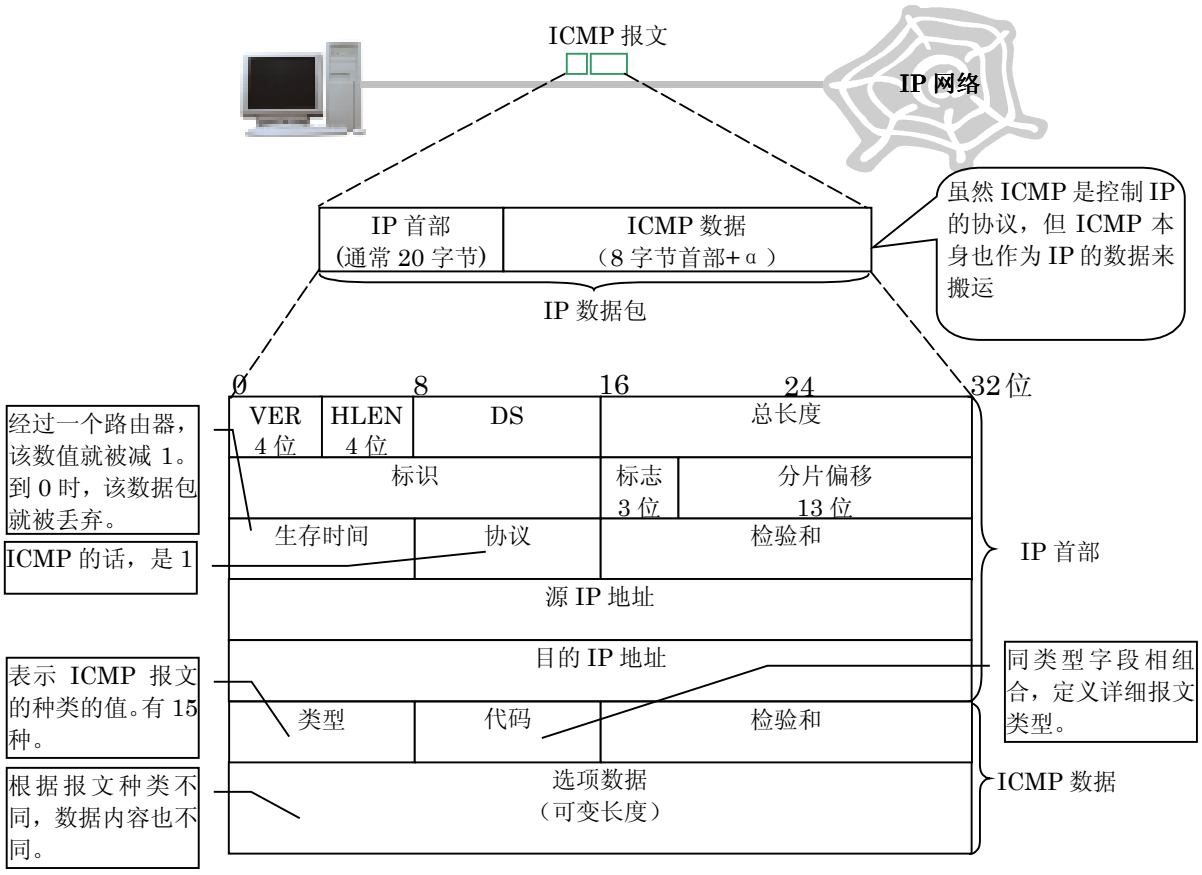

図 3 理解 ICMP 报文的格式

更加详细地看一下数据包的格式吧。看了図 3后应该知道，用来传送 ICMP 报文的 IP 数据包上实际上有不少字段。但是实际上与 ICMP 协议相关的只有 7 个子段。1）协议；2）源 IP 地址；3）目的 IP 地址；4）生存时间[5]；这四个包含在 IP 首部的字段。5）类型；6）代码；7）选项数据；这三个包含在 ICMP 数据部分的字段。这里面，1)协议字段值是 1。2)和 3)是用来交流 ICMP 报文的地址信息，没有特殊意义。4）与运用 ICMP 时相关，详细将在“2实现篇”阐述。

1.5. 用 15 种类的类型来大致分类功能

也就是，对于理解 ICMP 本身，重要的是 5)，6)，7)三个字段。这里面的可以称为核心的重要字段是 5)类型，6)代码这两个字段。

所有 ICMP 用来交流错误通知和信息询问的报文，都是由类型和代码的组合来表示的。RFC 定义了 15

[4] 因为 ICMP 是用于控制 IP 的协议，所以不被称作为 TCP 和 UDP 等完成传送应用程序数据功能的传输层协议。

[5] 生存时间也成为 TTL（time to live）。值为 0～255，每经过一个路由器，该数值就被减 1。到 0 时，该数据包就被认为寿命终结，也就被丢弃。

种类型[6]。“报文不可到达”这样的错误通知和“回送请求”这样的信息查询是由类型字段来区分的。ICMP 报文由类型来表达它的大概意义，需要传递细小的信息时由代码来分类。进一步，需要向对方传送数据的时候，用 7）选项数据字段来放置。

在这里没有必要记忆这 15 种类型。将在“4资料篇”里详细记述类型和代码的可能组合。

RFC 决定的所有东西就是看到这里的框架。但是实际的通信中 ICMP 是怎样被使用的，只靠这些是不能够理解的。因此，在下面的“2实现篇”来详细看一下 ICMP 是如何被使用的。

[6] 类型和代码各自是 8 位字段，理论上各自能定义 256 个值。但 RFC792 和 RFC950 两个上面只定义了 15 个类型。

2. 实现篇（熟知的 ping 也是实现例子之一，理解 RFC 里没有的动作）

突然间不能访问对方了，很多人用 ping 命令来确认是否连通。这个 ping 命令实际上就将 ICMP 的原理作在了自己内部，使它能够被方便地使用的实现例子之一。如基础篇所见，RFC792 基本上只决定了 ICMP 的规格。具体如何使之发挥作用，这个任务交给了 OS，TCP/IP 软件，应用程序等的实现。因此，想要了解 ICMP 是如何在实际的网络中被使用的话，知道应用程序的实现是不可缺的。

2.1. 根据用法实现各种各样的功能

象 ping 这样用户有意识地使用的命令以外，ICMP 也活跃在用户完全看不到的地方。例如，Windows 等操作系统，收到了发给自己的 ICMP 错误通知后，通过实施自动改变 TCP/IP 的设定这样的处理来控制着 IP 通信。

ICMP 的实现方法也是各种各样的。例如，操作系统或 TCP/IP 软件就有，在接收到“快抑制送信”的错误通知后就停止一段时间的通信这样的处理。这可以说是原分不动地使用了 RFC 的定义的实现例子。另一方面，也有象 traceroute 命令一样，一眼看不出使用了 ICMP 这样的复杂实现方法。为了深入理解 ICMP，眺望一下各种各样的实现例子也是必要的。

下面，在这个实现篇，作为代表实现例，1）Windows 的 ICMP 处理；2）ping 命令；3）traceroute 命令；4)端口扫描；举这四个例子看一下原理。

2.2. Windows 的 ICMP 处理（收到错误通知后自动改变设定）

首先以身边的 Windows 为例，窥视一下在用户看不到的地方，操作系统或 TCP/IP 软件使用 ICMP 的样子吧。在这，介绍一下有代表性的三个例子。首先举的是“路径 MTU[7]探索”的功能。

2.2.1. 特意设成分片禁止后送出去

所谓路径 MTU 探索，是探索与通信对方之间不用分片 IP 数据包，就能交流的 MTU 大小的功能[8]。MTU 大小是指计算机一次能够送出去的数据的最大长度，基本上由网路的种类来决定。例如，以太网的话通常是 1500 字节，使用 PPPoE 的 ADSL 通常是 1492 字节。

为了实现这个路径 MTU 探索，ICMP 被使用着。沿着流程，具体看一下 Windows 的 MTU 探索的样子吧（図 4）。

路径 MTU 探索的原理本身是非常简单的。首先，Windows 向通信对方送 IP 数据包时，先设置 IP 首部的分片禁止标志然后再送（図 4的 1）。这是路径 MTU 探索的基本。假如，Windows 将大于 1000 字节的数据包送了出去，但是见図 4，通信路径上有 MTU 从 1500 字节变成 1000 字节的地方。因此，那个路由器将不允许超过 1000 字节的数据包通过，而进入 MTU 是 1000 字节的网路。

路由器尝试着将 IP 数据包分片。但是因为数据包的分片禁止标志是有效的，所以不能分片。该路由器就将该 IP 数据包丢弃，并用 ICMP 通知送信方“想分片，但不能分片”（同 2）。这时路由器发送的 ICMP 的类型字段是 3，代码字段为 4。这是“需要分片但不能分片，不能送至终点”的意思。而且，大多数路由器将在数据选项部里填入不分片就能通过的 MTU 大小。

Windows 收到该 ICMP 报文后就知道了不分片就能够传送的数据大小，并暂时将 MTU 大小更换掉（同 3），然后继续通信（同 4）。

---

[7] MTU 是 maximum transfer unit 的略称。作为一块整体计算机能送出数据的最大值。例如，以太网的场合，MAC 帧的最大值(1518 字节)减去首部和尾部的 1500 字节是可设定 MTU 的最大值。

[8] 为什么需要路径 MTU 探索呢？应为 IP 数据包在途中被分片后，不仅通信效率会下降，还会加很大的负荷在分割数据包的路由器上。

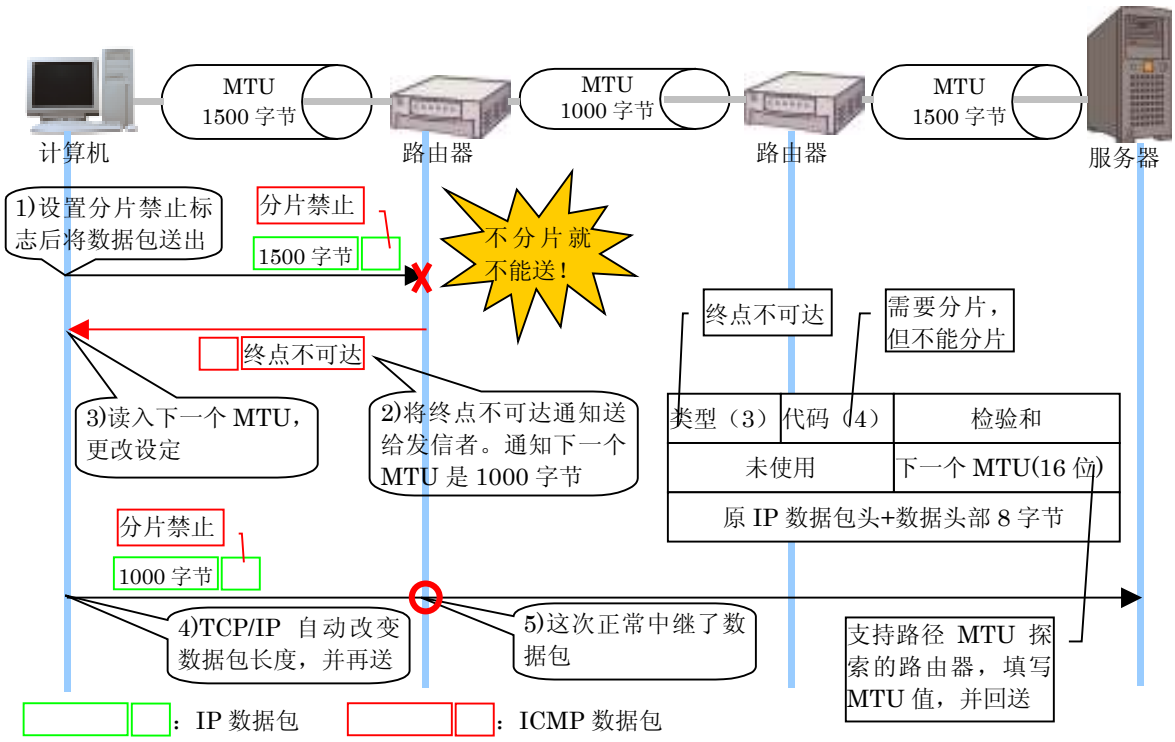

图 4 路径 MTU 探索原理

2.2.2. 向送信方传送路由器的改变

第二个 Windows 基本 ICMP 处理就举“改变路由”吧。改变路由是指路由器向送信方计算机指示路径改变这个功能（图 5的上）。局域网里有复数个路由器被使用。

计算机根据自己的路由信息(路由表)来决定传送目标。不知道发给谁好的时候，就将数据包发给设为默认网关[9]的路由器。被指定为默认网关的路由器接收到数据包，发现将数据包发给局域网内的其它路由器会比较快的时候，将这一信息通过 ICMP 通知发送方。这时使用的是，类型是 5，代码是 1 的 ICMP 改变路由报文。在选项数据部分里写着应该发送给的路由器 IP 地址。Windows 收到这个报文后，重写自己的路由表，与对方的通信将在一段时间里经由被指定的路由器来实行。

2.2.3. 要求送信方调整速度

第三个 ICMP 处理是源点抑制（流控制[10]）报文。数据包集中到达某一路由器后，数据包因为来不及被处理，有可能被丢弃的情况。这时候，向送信方发送的是 ICMP 源点抑制报文（图 5的下），用来使送行方减慢发送速度。

先简单看一下源点抑制的流程。在用来处理到达数据包的缓存将要溢出[11]的时候，路由器将发送 ICMP 报文。这时使用的是，类型是 4，代码是 0 的源点抑制 ICMP 报文。Windows 接到路由器发来的源点抑制报文后，主动扩大数据包的送信间隔来降低送信速度。接着，在报文到达（有效）的期间，持续这一动

9 默认网关是，数据包的目标地址不在路由表的范围内时，用来转发数据包的路由器。作为 TCP/IP 的基本设置被登记着。

10 流控制是，为了不让网络中互相交流的包被丢弃，调整通信速度的功能。除了 IP 的上层协议 TCP，即使以太网，IEEE802.3x 里也规定了流控制等，各种各样的协议具有这一功能。

11 实际情况中，是在缓存溢出后，还是在缓存溢出前通知送信方，和路由器的实现与设置有关。

作。这样就能防止路由器陷入不断丢弃数据包的状态。

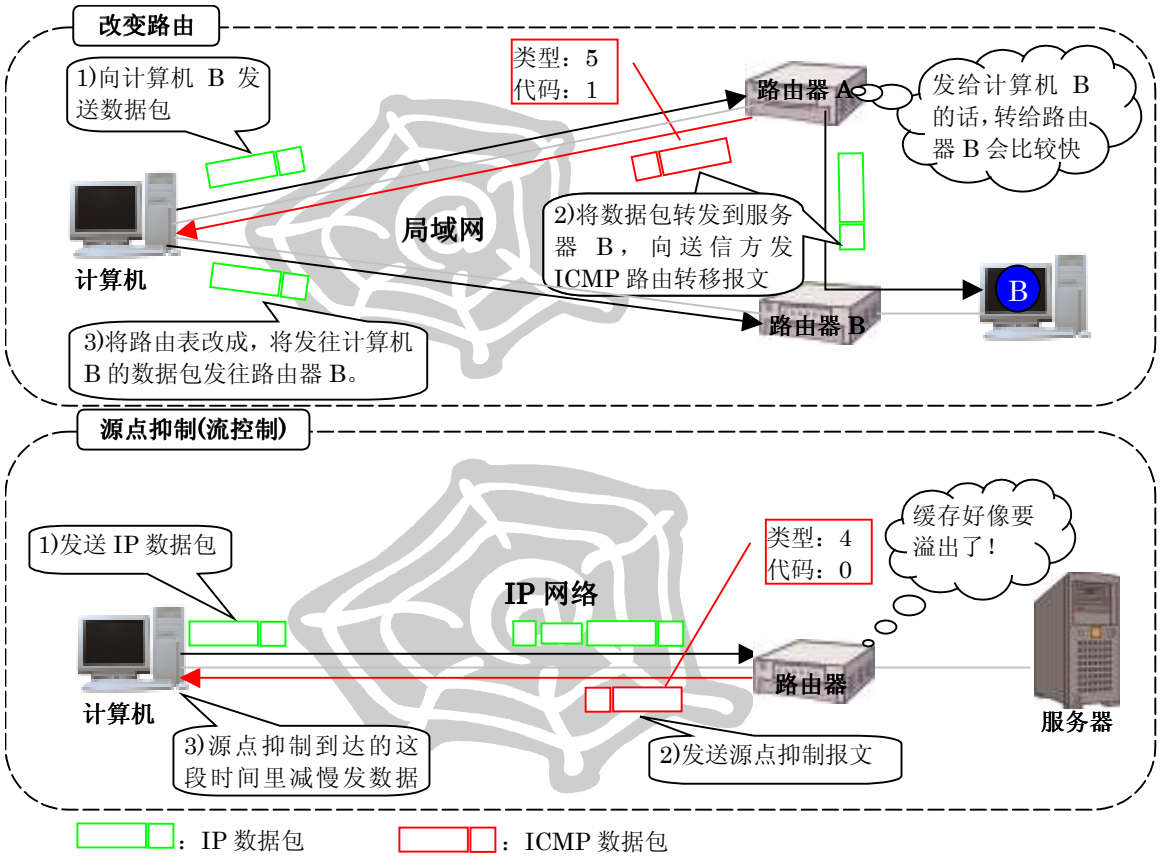

図 5 Windows 的幕后 ICMP 处理例

2.3. ping 命令 用回送请求与回答来确认对方的通信情况

从这里开始，看几个更加在用户身边使用的实现例子。首先要举出的是，网络命令代表中的代表 ping 命令。ping 命令用来，在 IP 层次上调查与指定机器是否连通，调查数据包往复需要多少时间。为了实现这个功能，ping 命令使用了两个 ICMP 报文（图 6）。

2.3.1. 向目标服务器发送回送请求

那么，看一下 Windows 计算机上执行 ping 命令的程序吧。执行 ping 后（図 6的 1）首先，向目标服务器发出回送请求（类型是 8，代码是 0）报文（同 2）。在这个回送请求报文里，除了类型和代码字段，还被追加了标识符和序号字段。

标识符和序号字段分别是 16 位的字段。ping 命令在发送回送请求报文时，在这两个字段里填入任意的值。对于标识符，应用程序执行期间送出的所有报文里填入相同的值。对于序号，每送出一个报文数值就增加 1。而且，回送请求的选项数据部分用来装任意数据。这个任意数据用来调整 ping 的交流数据包的大小。

2.3.2. 鹦鹉学舌一样返回回送回答

计算机送出的回送请求到达目标服务器后，服务器回答这一请求，向送信方发送回送请求（类型是 0，代码是 0）（同 3）。这个 ICMP 回送回答报文在 IP 层来看，与被送来的回送请求报文基本上一样。不同的只是，源和目标 IP 地址字段被交换了，类型字段里填入了表示回送回答的 0，这两点。也就是，从送信方来看，自己送出的 ICMP 报文从目标服务器那里象鹦鹉学舌那样原样返回了。

送信方的计算机可以通过收到回送回答报文，来确认目标服务器在工作着。进一步，记住发送回送请求报文的时间，与接收到回送回答报文的时间一比较，就能计算出报文一去一回往复所需要的时间（同 4）。但是，收到的回送回答报文里写的只是类型和代码的话，发送方计算机将无法判断它是否是自己发出去请求的回答。因此，前面说到的标识符和序号字段就有它的意义了。将这两个值与回送回答报文中的相同字段值一比较，送行方计算机就能够简单地检测回送回答是否正确了。

执行 ping 命令而调查的结果没什么问题的话，就将目标服务器的 IP 地址，数据大小，往复花费的时间打印到屏幕上（图 6的下）。

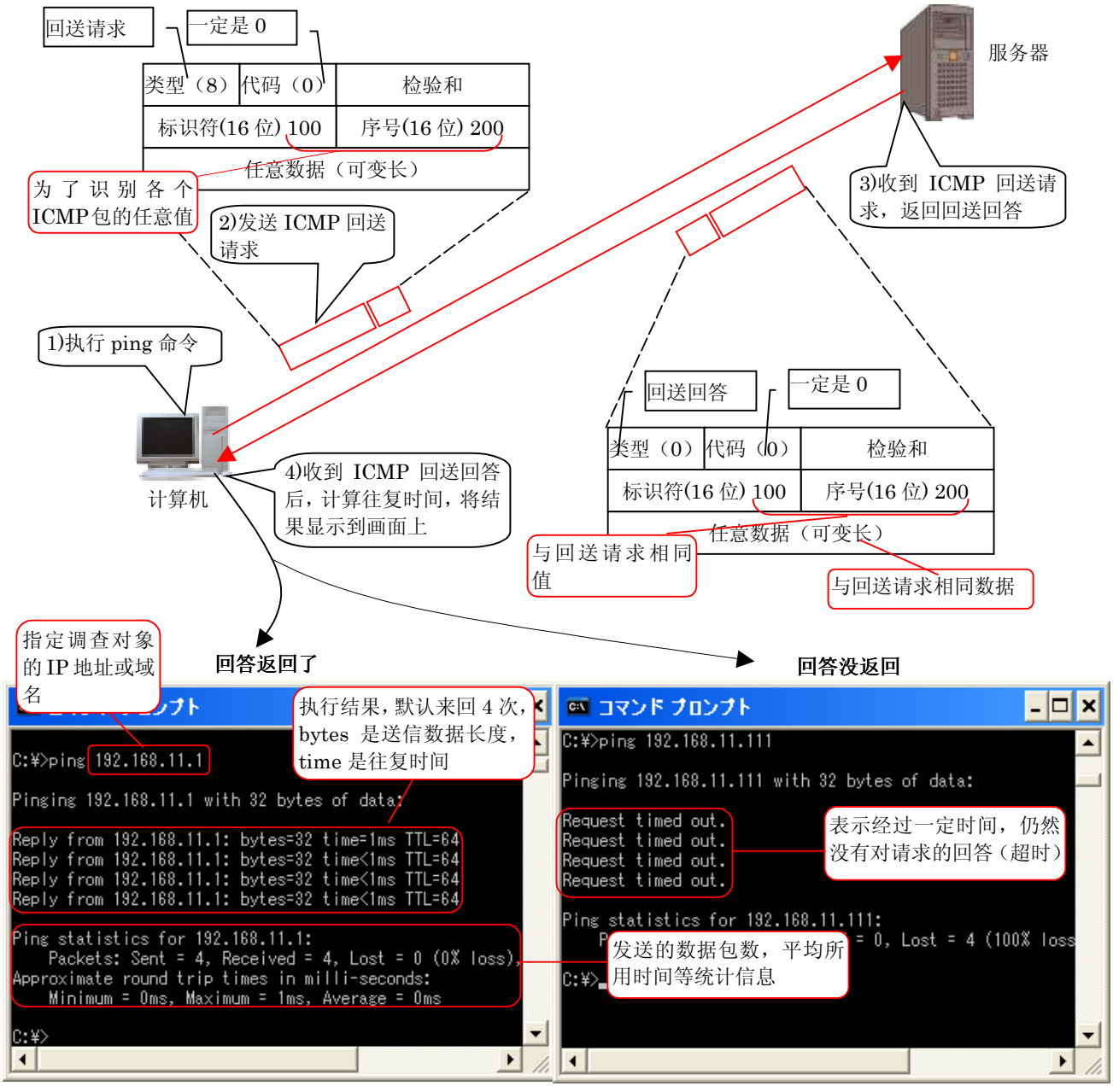

图 6 ping 命令的原理

2.3.3. 不能确定连通的原因有三个

但是，也不是什么时候都返回回送回答的。例如，也有显示“Request timed out”的时候。

用 ping 命令不能确定与对方连通的原因大致有三个。那就是，1）目标服务器不存在；2)花在数据包交流上的时间太长 ping 命令认为超时；3）目标服务器不回答 ping 命令。如果是原因 2），通过 ping 命令的选项来延长到超时的等待时间，就能正确显示结果了。如果原因是 1）或 3）的话，仅凭 ping 命令的结果就不能判断是哪方了。正如这样，ping 命令不一定一定能判断对方是否存在。先记住吧。

2.3.4. 也能利用在负荷分散上

看到这里的 ping 的原理，在用户有意使用的 ping 命令以外也被使用着。典型例子是，准备复数个服务器来分散客户端来的访问时使用的负荷分散装置了。

负荷分散装置将处理分散到复数个服务器时，需要判断各个服务器有多忙。也想知道连接服务器的网路的拥挤程度。通过尽量将处理委托给不忙且网路也空着的服务器，来防止处理集中在一部分服务器上而使服务器掉线的状况。

因此，负荷分散装置定期向服务器发送 ICMP 回送请求，测试数据包往复所花的时间。这个原理本身与 ping 命令是完全相同的。负荷分散装置通过调查回送请求返回所需时间的变化，来知道网路的拥挤程度和服务器的负荷，以这个为判断标准来分配客户端来的访问[12]。

2.4. traceroute 命令（通过超时错误来调查到目标的路径）

为了调查到通信对方的路径现在是怎么样了，使用的是 traceroute 命令。它与 ping 并列，是代表网络命令。这个 traceroute 也是 ICMP 的典型实现之一。以 Windows 上带着的 tracert[13]命令为例，来窥视一下 ICMP 在背后是如何被使用的吧。

2.4.1. 故意使生存时间过期

traceroute 命令是，指定目标主机名或 IP 地址并执行后，就能够报告我中途经过的路由器名或 IP 地址的方便的工具（图 7）。到底如何用 ICMP 的功能来实现的呢？接下来，按序看一下。

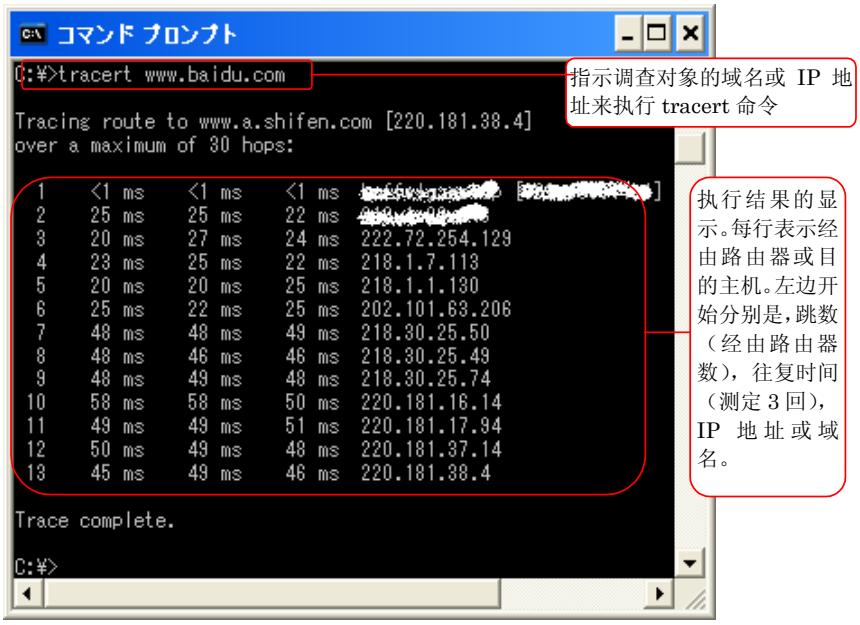

图 7 tracert 命令执行例

[12] 负荷分散装置除了 ping 以外，使用着例如服务器 CPU 的使用率等各种各样的信息来作为负荷分散的判断标准。

[13] Windows 的场合，实际输入的是“tracert”而不是“traceroute”命令。因此，指 Windows 命令的时候用 tracert 命令来标记。

在 Windows 上执行 tracert 命令后，首先计算机向目的服务器发送 IP 数据包。Windows 上使用的是与 ping 同样的 ICMP 回送请求报文。但是，有一点和通常的回送请求不一样。那是，最初将 IP 首部的 TTL(生存时间)字段设为 1 这一点[14]（图 8的 1）。

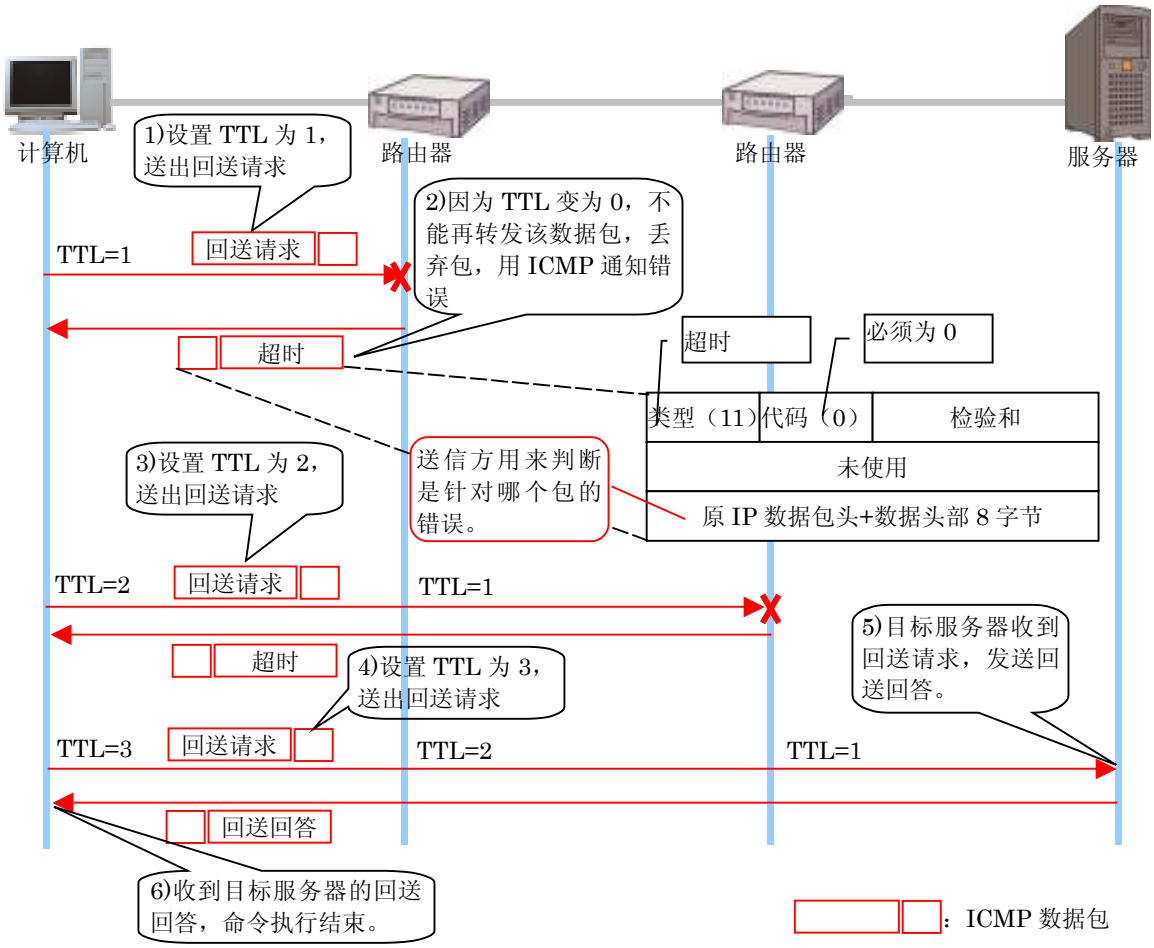

图 8 traceroute 命令的工作原理

路由器每转送一次数据包就将 TTL 的值减 1。当 TTL 变为 0 的时候，按规定将丢弃这个数据包。正如这样，与其说 TTL 是时间，还不如说 TTL 是经过路由器的个数。对于计算机发送出去的数据包，只要它与目标服务器不在同一局域网内，一定会被哪儿的路由器中继。这时如果 TTL 的值是 1，由于路由器的处理会变为 0，则该数据包将会被丢弃（同 2）。

2.4.2. 用超时报文来通知送信方

路由器丢弃数据包的同时，用 ICMP 报文来通知错误。这时使用的 ICMP 报文是，类型为 11，代码为 0 的 ICMP 超时报文。而且在选项数据字段里，将填入原先数据包的 IP 首部和 ICMP 的开始 8 字节。正如 ping 命令的时候看到的，ICMP 回送请求的先头 8 字节里包含了标识符和序号字段。因此，送信方的计算机看了超时报文后，就知道是针对自己发出的回送请求的错误通知。

[14] 通常的通信上，送信时将 TTL 设为 64，128，256 等值。

计算机接到针对第一个数据包的 ICMP 超时报文后，接下来将 TTL 加 1（TTL=2）并同样地送出（同 3）。这次通过第一个路由器，TTL 变为 1，到达第二个路由器。但是第二个路由器象前面一样，由于 TTL 变为 0，将不能转发该包。因此，同第一个路由器一样，将该包丢弃，并返回 ICMP 超时报文。以后，收到错误的发送方计算机将 TTL 加 1，重复同样的工作（同 4）。

2.4.3. 只有目标服务器的反应不同

如此一个一个增加 TTL，某个时候 ICMP 回送请求报文将到达最终的目标服务器。这时，只有目标服务器与途中的路由器不同，不返回 ICMP 超时报文。为什么呢？因为即使目标服务器收到 TTL 为 1 的数据包也不会发生错误。

作为代替处理，服务器针对送信方计算机发出的 ICMP 回送请求报文，返回 ICMP 回送回答报文。也就是，送信方计算机与服务器之间，与 ping 命令的执行一样了（同 5）。得到了 ICMP 回送回答报文的送信方知道了路经调查已经到了目标服务器，就结束了 tracert 命令的执行（同 6）。像这样，通过列出中途路由器返回的错误，就能知道构成到目标服务器路径的所有路由器的信息了。

2.4.4. 操作系统不同则实现方法略微不同

到这里，以 Windows 上的 tracert 命令为例看了原理，有些别的操作系统的 traceroute 命令的原理略微不同。

具体来说，也有用向目标发送 UDP 数据包代替 ICMP 回送请求报文来实现的。虽说是用 UDP，但途中的路由器的处理与图 8完全相同。只是 UDP 数据包到达目标后的处理不同。目标计算机突然收到与通信无关的数据包，就返回 ICMP 错误[15]，因此根据返回数据包的内容来判断命令的中止。

2.5. 端口扫描（发送 UDP 数据包来检查未使用端口）

最后登场的是端口扫描。所谓的端口扫描就是检查服务器不需要的端口是否开着。服务器管理者用来检查有没有安全上有问题的漏洞开着。不是象 ping 和 traceroute 那样是操作系统自带的工具，需要利用网络工具才行。

2.5.1. 用 UDP 数据包使错误发生

端口扫描大致分为“UDP 的端口扫描”和“TCP 的端口扫描”两种。这里面，与 ICMP 相关的是 UDP 一边。使用 TCP 的通信，通信之前必定要先遵循三向握手[16]的程序。因此，只要边错开端口号边尝试 TCP 连接就能调查端口的开闭。不特别需要 ICMP。

与此相对，UDP 没有这样的连接程序。因此，调查端口是否打开需要想点办法。这样，被使用的是 ICMP。根据 ICMP 规格，UDP 数据包到达不存在的端口时，服务器需要返回 ICMP 的“终点不可达”之一的“端口不可达”报文[17]。

具体来说，向希望调查的服务器发送端口号被适当指定了的 UDP 数据包。这样，目标端口没开着的话，服务器就返回 ICMP 端口不可达报文（图 9）。返回的 ICMP 数据包的选项数据字段里放入着，送信方送出的 UDP 数据包的 IP 首部与 UDP 首部的头 8 个字节。送信方通过这个信息来辨别该错误通知是针对哪个 UDP 数据包的，并判断端口是否打开着。

---

[15] traceroute 命令使用 UDP 数据包的时候，为了使对方返回错误，通常使用服务器软件没有使用的巨大端口号。

[16] 三向握手是用 TCP 来通信时与对方的连接程序。双方通过交流 3 个数据包来确立假想的连接。用 TCP 来通信的终端同伴不遵循这个连接程序就不能通信。

[17] 该 ICMP 报文的类型是 3，代码是 3。

2.5.2. 知道的仅仅是关着

UDP 端口扫描一边一个一个错开端口号，一边持续着这个通信。这样，就知道了哪个端口是“好象开着的”了。但是，UDP 端口扫描与 TCP 端口扫描有很大区别的地方。那就是，即使 ICMP 端口不可达报文没有返回，也不能断定端口开着[18]。

端口扫描除了被管理员用来检查服务器上是否有开着的漏洞，作为黑客非法访问的事先调查，对服务器实施的情况也是很多的。需要非常小心地来使用。

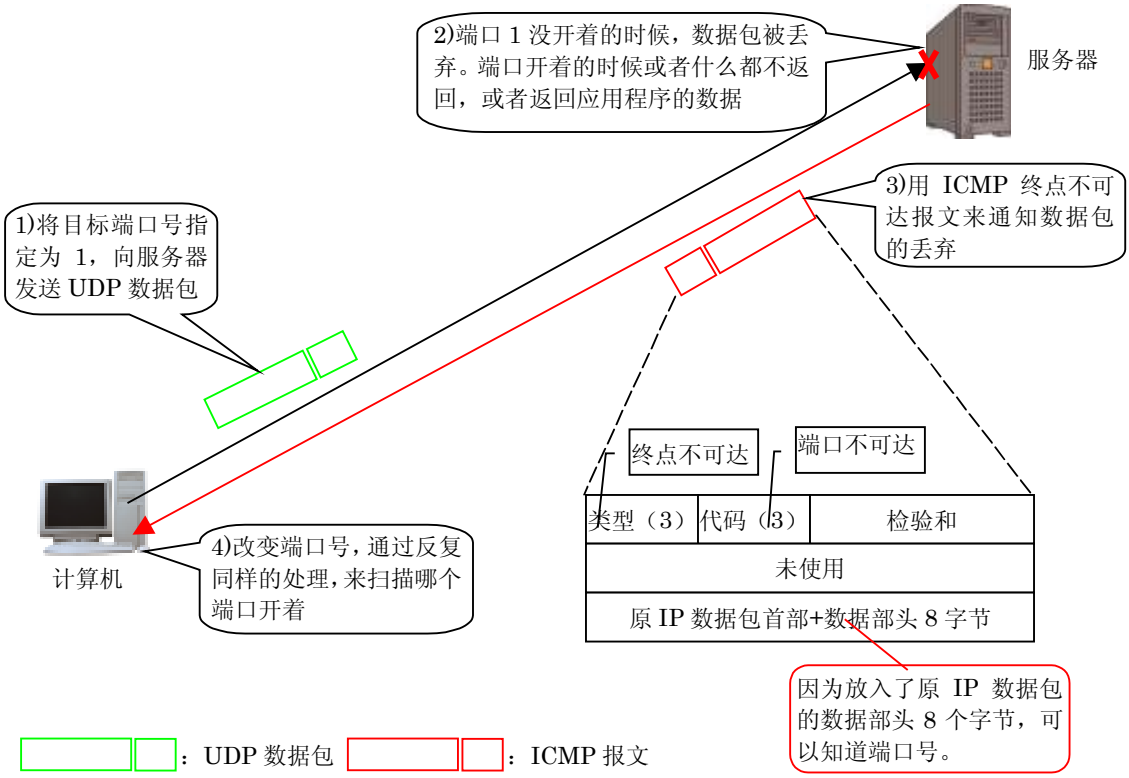

图 9 将 ICMP 里用于端口扫描

[18] 安全上的原因，服务器不处理 ICMP 的情况也是有的。ICMP 和安全间的关系将在“运用篇”阐述。

3. 运用篇（方便性和安全性，现实上的利用是两者的平衡）

看了基础篇和实现篇之后，对于 ICMP 的原理应该基本了解了。但是，为了能拍胸脯说“我掌握了 ICMP 了！”，还有一点需要先知道。那是“ICMP 和安全的关系”。知道这之后，就不得不考虑实际的运用了。首先，挖掘一下一定想知道的安全知识。

3.1. 为什么停止方便的 ICMP？

到实现篇为止看了 ICMP 的原理，实际使用网络的时候，首先用户并没有意识到使用着 ICMP。如果说用户意识到 ICMP 的话，那应该是个人防火墙的设置画面[19]吧。这样的设置画面上有是否停止 ICMP 的项目（図 10）。

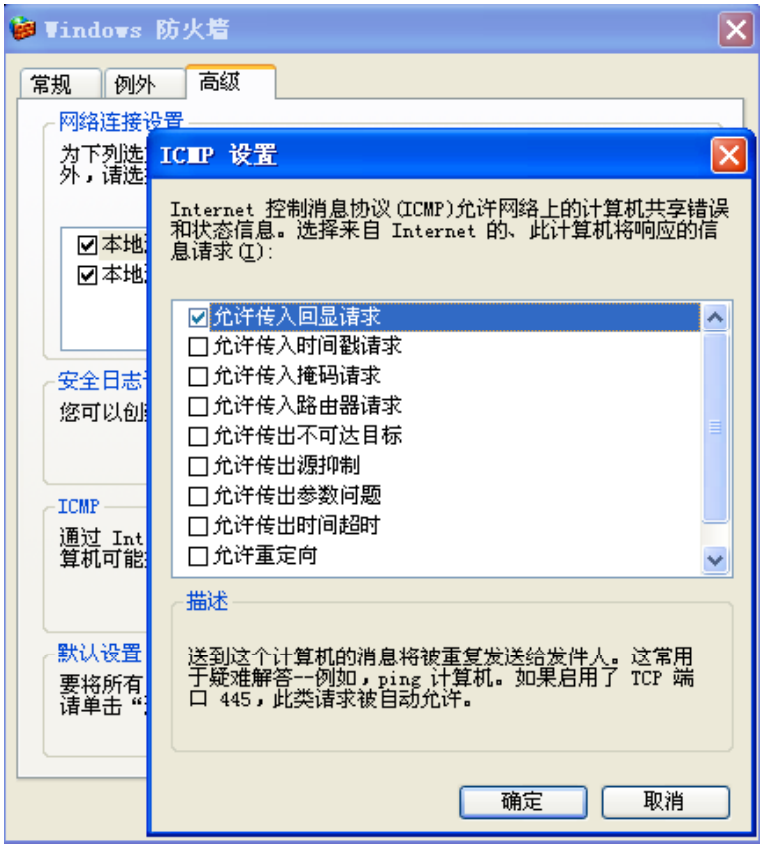

図 10 Windows 个人防火墙画面

为什么有停止 ICMP 使用的设定项目呢？理由只有一个，那就是确保安全。虽然 ICMP 是非常便利的协议，但黑客在尝试非法访问的时候会被恶意利用。由于 ICMP 被恶意使用而遭受损害的用户正在不断增加之中，因此有了限制 ICMP 使用的意见。

3.2. 发送大量的 ICMP 数据包

那么实际上，ICMP 被怎样恶意使用的呢？想考虑安全相关问题，不知道这个就开不了头。看两个典型的恶意使用例子吧。作为恶意使用 ICMP 的最有代表性的例子，也就是所谓的 “ping 洪水”的攻击。它利用 ping 的原理，向目标服务器发送大量的 ICMP 回送请求。这是黑客向特定的机器连续发送大量的 ICMP 回送请求报文。目标机器回答到达的 ICMP 回送请求已经用尽全力了，原来的通信处理就变得很不

[19] 打开 Windows“控制面板”的“网络连接”，选中有效的连接，单击鼠标右键选择属性项目，将会打开属性对话框，选择“高级”就可以看到“Windows 防火墙”项目，单击就会打开图中对话框。

稳定了。进一步，目标机器连接的网络也可能由于大量的 ICMP 数据包而陷入不可使用的状态[20]。

与 ping 洪水相似，以更加恶劣的使用方法而闻名的是称为“smurf”的攻击手法。smurf 同样，黑客恶意的使用 ICMP 回送请求报文。这一点同 ping 洪水是相同的。不过在 smurf，对 ICMP 回送请求实施了一些加工。源 IP 地址被伪装成攻击对象服务器的地址，目标地址也不是攻击对象服务器的地址，而是成为中转台[21]的网络的广播地址[22]。

3.3. 几百倍的 ICMP 回送回答报文的到达

来具体看一下 smurf 攻击的流程吧（図 11）。

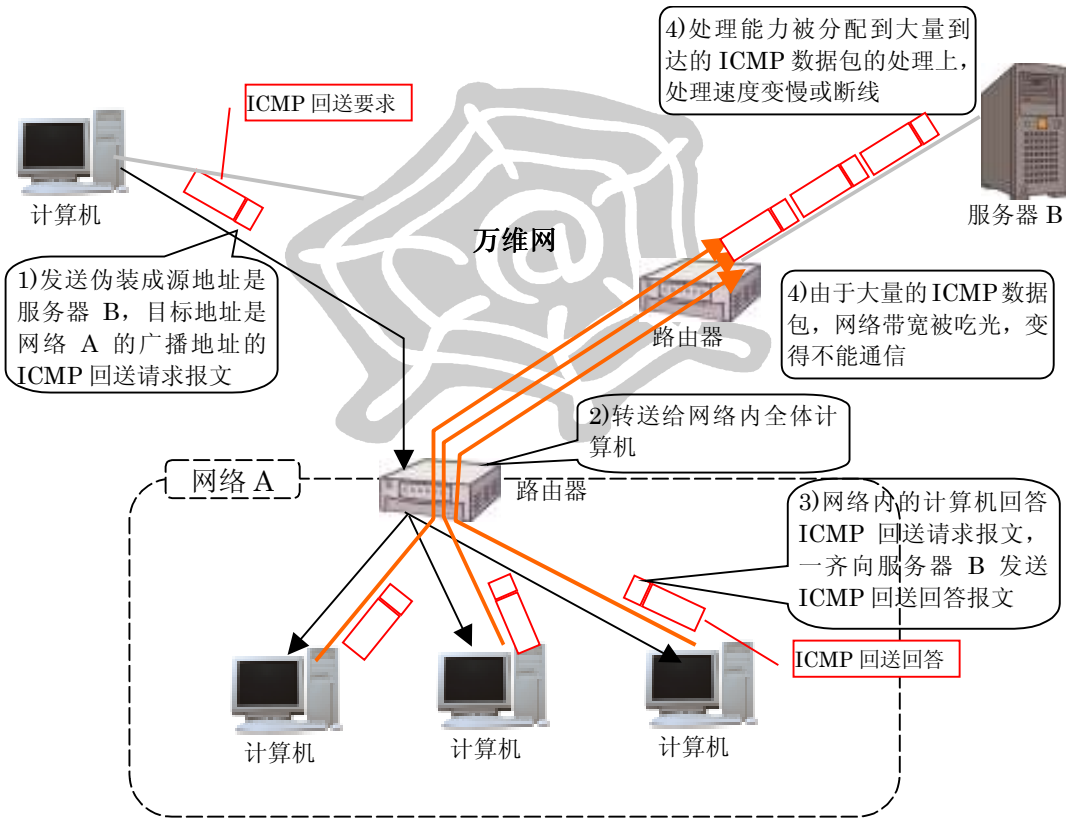

図 11 恶意使用 ICMP 的 smurf

黑客发送伪装了的 ICMP 回送请求后（図 11的 1），到达在作为踏板的网络的入口处的路由器。这样，路由器将回送请求转发给网内所有的计算机（同 2）。假如有 100 台计算机，回送请求将到达 100 台所有的计算机。收到回送请求的计算机对此作出反应，送出回送回答报文（同 3）。这样，黑客送出的一个 ICMP 回送请求报文，一下子增加到了 100 倍。

这样增加的 ICMP 回送回答报文面向的不是黑客的计算机，而是伪装成回送请求的源 IP 地址的攻击对象服务器。变成到达了，从几百台计算机发出的巨大数量的 ICMP 回送回答。

smurf 与 ping 洪水攻击不同，因为到达服务器的是 ICMP 回送回答，服务器不用返回回答。但是为了处理大量的 ICMP，服务器承受了大量的负载。网路被撑爆了也是一样的（同 4）。

---

[20] 使连接在网络上的机器不能提供原来的服务的攻击称为 DoS(denial of service:拒绝服务)。

[21] 中转台是，黑客为了攻击目标服务器而作为踏板来使用的第三方网络或服务器。

[22] IP 地址分成用来识别网络的网络部和用来识别主机的主机部两部分。主机部分都是 1 的地址被用来作为向网络内所有主机发送的广播地址。

3.4. 恶意使用的模式是无限的

除此之外，还有很多各种各样 ICMP 被恶意使用的例子。例如，通知错误或询问信息本身，也有被黑客用来传递谎言的可能性。同用信鸽来扩展谎言的传播，通过传递与事实不同的信息来使人判断错误是一样的。

而且，反过来也有传递错误信息而变成问题的例子。例如，在实现篇里看到的端口扫描，黑客就可以利用它来进行攻击对象的调查。进一步，推翻了“ICMP 是用来控制 IP 的”这一常识的恶意使用方法也登场了。就是将 ICMP 的选项数据部分作为信息搬运工的手法。黑客将这种工具隐藏在服务器里，从外部控制服务器，将用户的个人信息和重要的情报偷盗出来。

如上，仅从安全的方面来说，ICMP 是有百害而无一利的。

3.5. 阻止 ICMP 后将陷入困境

“那阻止所有的 ICMP 不就行了吗!”可能有读者会这样认为。不过那就太轻率了。ICMP 作为支持 IP 的协议是需要的，所以被制作了。即使没有，也不是说 IP 通信本身就完全不行了，实际上会出现几个难办的情况。

它的典型例子就是称为“黑洞路由器”的问题（図 12）。所谓黑洞路由器，就是通信路径上的 IP 数据包不留痕迹的消失了的现象。原因是，实现篇里说明的路径 MTU 探索功能不起作用了。

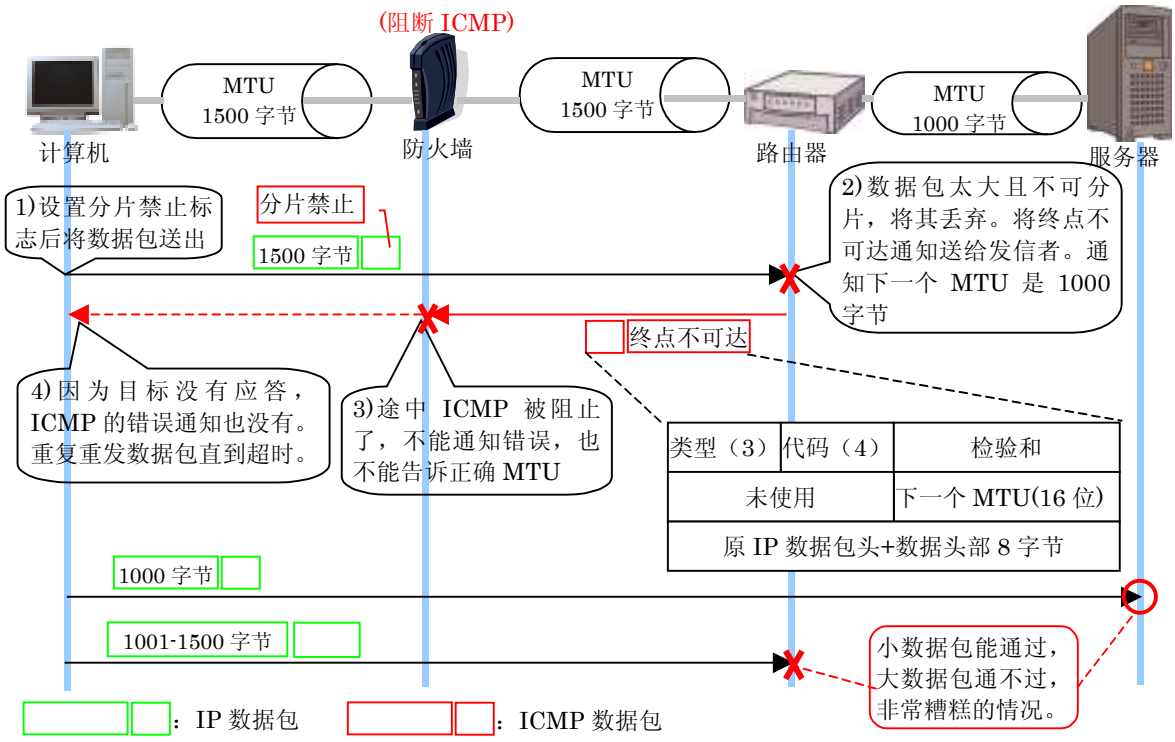

図 12 阻断 ICMP 引发的问题（黑洞路由器）

如図 12，假设通信路径上有因为 MTU 大小不同而需要分片的路由器。而且，计算机和路由器之间，为了安全上的原因，设置了阻止 ICMP 报文通过的防火墙。这种情况下，计算机实行路径 MTU 探索将会怎么样呢？

3.6. 不能调整数据包长度

如果是传送路径上不需要分片大小的 IP 数据包，它将会毫无问题地到达对方。另一方面，数据包的长

度是需要分片的时候，发送就会有问题。

正如实现篇看到的，这样的数据包到达连接在不同大小 MTU 的网络的路由器后，路由器将用 ICMP 终点不可达报文来通知发送方。本来的处理是，送信方接收到该 ICMP 报文，根据路径 MTU 探索处理调整 MTU 大小后继续通信。

但是，这次的例子，ICMP 报文被路经中的防火墙隔断了。路径 MTU 探索功能不起作用，MTU 的大小也就不能调整了[23]。

3.7. 不知道原理就不可能理解

最近从局域网的计算机通过 ADSL 服务访问万维网时，经常看到这个黑洞路由器现象。ADSL 线路的 MTU 大小，宽带路由器的设定，Windows 的路径 MTU 探索功之间互相关联引起了这个现象。

糟糕的是，即使有黑洞路由器，也不是完全不能通信这一点。不管怎么样说，被吸进去的只是长度是需要分片的 IP 数据包。也就是，考虑一下 WEB 访问，连接 WEB 服务器时是没有问题的，以文字为主体的页面也大都能被显示，但是含有比较大图像的页面不能被显示。黑洞路由器就由这种复杂奇怪的现象表现出来了。如果不知道路径 MTU 探索和黑洞路由器的原理的话，碰到这种现象，可能连猜想原因都很困难了。

3.8. 即使阻止了客户端也没问题

如最初所见，在现实的万维网上，如果事先使所有的 ICMP 功能有效的话，就会给了黑客各种各样的机会，安全上就会有问题了。

另一方面，如果一个一个阻止了的话，不仅非常不方便，而且还会发生黑洞路由器等问题。那么，如何充分运用 ICMP 才行呢？客户端，服务器，还有路由器，从各个方面来看一下。

首先从客户端开始。最近的宽带路由器和个人防火墙，通过设置来阻止 ICMP 的很多。但是，初期设置是千差万别的。阻止全部 ICMP 的也有，反过来的也有。其中，只允许 ping 命令等一部分 ICMP 报文通过的也有。

原来，对于安全的考虑方法是根据环境的不同而变化巨大的，并不是一定要这样才行的。但是，最近的倾向是，使连在万维网上的个人计算机不应答没有必要的 ICMP 报文。例如 Windows XP 的情况下，使用操作系统自带的个人防火墙的话，默认是将外部来的所有 ICMP 报文隔断。

那么路由器怎么样呢？万维网中的路由器，不小心阻断了 ICMP 的话，会发生黑洞路由器等问题。还有，大量的数据包涌过来的时候，如果不发送 ICMP 源点抑制报文，处理速度就会跟不上。路由器的话，这样的情况以外，再加上考虑周围网络环境的基础上，再来判断是否阻断不需要的或者可能造成攻击的 ICMP 数据包比较好吧。

服务器就比较难判断了。例如，不让它回应 ping 命令的话，连不上服务器的时候，就缺少了调查的有效手段。但是，有受到 ping 洪水攻击的可能性也是事实。这些只能由管理者来判断了[24]。

---

[23] 不仅这样，计算机连路由器把数据包丢弃的事情也不知道。也就是完全不能判断，是虽然到达了对方计算机但由于某些原因被丢弃了呢，还是在通信路径的哪里被丢弃了呢，还是稍微等一会儿就会到达呢。

[24] 实际上，挑选几个世界上比较著名的网站，ping 通的和不通的，大约是一半一半。

4. 资料篇（只有这些？全部 ICMP 报文一览）

最后作为终结，看一下 RFC 规定的所有 ICMP 报文一览（表 1）。如基础篇所见，ICMP 的报文用“类型”和“代码”两个字段的组合来表现。

首先类型有 15 种，用来定义“终点不可达”这种报文的大概意思。对于各个类型，如果需要传达更加详细的情况，就准备复数个代码，用来传达“主机不可达”等信息。这样的由类型和代码的组合而构成的 ICMP 报文有 35 个[25]。

类型	代码	说明	种类
0	0	回送回答。与回送请求成对被 ping 命令使用	信息查询
3		终点不可达	错误通知
	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	需要分片，但该数据包的 DF（不要分片）已设置	
	5	源点路由选择不能完成	
	6	目的网络未知	
	7	目的主机未知	
	8	源主机是孤立的	
	9	从管理上禁止与目的网络通信	
	10	从管理上禁止与目的主机通信	
	11	对指定的服务类型，网络不可达	
	12	对指定的服务类型，主机不可达	
	13	主机不可达，因为管理机构已经在该主机上放置了过滤器（由 RFC1812 追加）	
	14	主机不可达，因为主机的优先级被违背了（由 RFC1812 追加）	
	15	主机不可达，因为主机的优先级被删除了（由 RFC1812 追加）	
4	0	源点抑制。通知送信方抑制发送数据包	错误通知
5		改变路由	错误通知
	0	对特定网络路由的改变	
	1	对特定主机路由的改变	
	2	基于指明的服务类型对特定网络路由的改变	
	3	基于指明的服务类型对特定主机路由的改变	
6	0	回送请求。与回送回答成对被 ping 命令使用	信息查询
9		路由器通告（由 RFC1256 追加）	信息查询
	0	一般路由器通告（路由器向自己身边通告自己的存在）。	
	1	不能转发一般流量（由 RFC2002 追加）	
10	0	路由器询问（由 RFC1256 追加）	信息查询
11		超时	错误通知
	0	传送中生存时间变为了 0。被 traceroute 命令利用	
	1	规定时间内没有收到所有的分片	
12		参数问题	错误通知
	0	在 IP 首部的某个字段中有差错或两义性	
	1	缺少所需的选项部分（由 RFC1108 追加）	
	2	长度不对	
13	0	时间戳请求	信息查询
14	0	时间戳回答	信息查询

[25] 这里是以 RFC792 和 950 为基准，如果包括试验性质的 RFC，那数字还稍有增加。

15	0	信息请求。要求分配 IP 地址（现在未使用）	信息查询
16	0	信息回答。IP 地址的通知（现在未使用）	信息查询
17	0	地址掩码请求。（由 RFC950 追加）	信息查询
18	0	地址掩码回答。（由 RFC950 追加）	信息查询

表 1 ICMP 报文一览

没有必要详细的看全部 35 个 ICMP 数据包。这里先大略的看一下决定 ICMP 报文功能的 15 个类型。

4.1. 光终点不可达就有 15 种

首先从类型 0 的“回送回答”开始。这就是象在应用篇里看到的，用来作为 ping 命令的应答报文。

下一个是类型 3 的“终点不可达”。由这个类型 3 定义的 ICMP 报文特别的多。代码从 0 到 15 有 16 种报文。看到定义了如此之多的种类，可以知道，将数据包不能到达终点的信息传给送信方，对于平稳地推进 IP 通信是多么的重要了吧。

但是，正如实现篇所看到的，在规格里存在与实际上实现并使用是不同的问题。而且，即使实现了，平常操作系统一直在使用的情况也有，反过来，基本上看不到的情况也有。以类型 3 的报文群来说，“端口不可达”（代码 3）和“需要分片但不可分片”等，如实现篇所见平常频繁地被使用着。还有，代码 0，1，6 的使用频度也相对比较高。另一方面，别的代码基本上看不到，现状是完全没有被使用的也有[26]。

4.2. 默认网关也能寻找

下一个类型 4 的“源点抑制”和类型 5 的“改变路由”，正如实现篇看到的，用来抑制送信方的通信速度，或者让送信方使用别的路由来发送数据包。

类型 8 的“回送要求”与类型 0 的回送回答作为一对被 ping 命令使用。类型 9 的“路由器通告”和类型 10 的“路由器询问”，在计算机启动时用来自动寻找（路由器探索）路由器（默认网关）。Windows 通过操作系统的设置也可使路由器探索有效[27]。

类型 11 的“超时”是实现篇里说明的 traceroute 命令所不可缺少的 ICMP 报文。类型 12 以后的现状是，不怎么被看到或者基本上没有被使用[28]。

以上看了表 1一遍，我们很少有直接看到这些 ICMP 报文的机会。但是，在什么不可理解的问题发生了的时候，用抓包软件[29]来抓一下 ICMP 数据包，从类型和代码上可能会明白些什么。还有，用宽带路由器的设置来详细过滤 ICMP 的时候，表 1可能会帮上忙。

[26] 例如代码 1 和 12 是用来通知，用 IP 首部的 TOS(type of service)字段来控制数据包的优先度时发生的错误。但是，万维网的现状是没有使用 TOS 字段，这些代码也就看不到了。

[27] 实际上，用 DHCP 等其他协议来取得默认网关的情况比较多，路由器探索基本上没有被使用。

[28] 其中类型 15 的“信息请求”和类型 16 的“信息回答”，以前是在计算机启动时用来取得自己的 IP 地址的。现在别的协议所代替，完全没有被使用。

[29] 可以使用免费的抓包软件 ethereal（现名 wireshark）等。

5. 网络脚本语言的实践

介绍网络入门的书籍很多，但往往都是一些理论知识。因为没有实践，对于初学者，往往看完之后心里也不觉得踏实，总是会有“实际协议的运行和我理解的是不是一样呢？如果自己能试一下就好了。”这种想法。尝试协议最好的方法之一就是，实际做一些数据包发着试一试，看看应答结果怎么样。但是现在的一些生成数据包的工具要么只能发送定制好的数据包，不具备动态解析和控制的功能，不能适应需要动态应答的协议[30]。要么需要 C 语言等高级语言和 Linux 等开发环境的知识，不是初学者很快就能上手的。

为了解决这个问题，可以使用称为 Fine Packet Builder（后简称 FPB）的一个工具。它能够通过脚本语言 Jython[31]来实现复杂的数据包的生成和数据包的实时解析，并具有集成开发环境，只需学习很少知识就能快速上手。Fine Packet Builder 和相关资料可以从http://www.fineqt.cn/index_cn.htm免费下载，Jython 的相关知识可以从http://www.jython.com.cn/或http://www.jython.org/获得。下面将使用该工具的脚本语言 API 来实践一些本篇文章里说明的协议。如对脚本语言本身的原理及使用方法不太熟悉的话，可先下载并阅读“超级网络入门与实践(一)　网络脚本语言基础”。

FPB 本身提供了不少脚本语言使用例子，可以在以下场所找到例子的源程序。

D:¥Fine Packet Builder¥plugins¥com.fineqt.pktbuilder.core_1.x.x¥sample¥jython¥en

其中，“D:\Fine Packet Builder”是软件的安装路径。而且，在用户使用手册里有对每个例子的说明。而各个协议的格式描述文件则在如下场所。

D:¥Fine Packet Builder¥plugins¥com.fineqt.pktbuilder.core_1.x.x¥protocol

5.1. 用 ICMP 来实现 Ping

下面根据“2.3ping 命令 用回送请求与回答来确认对方的通信情况”的说明，用脚本语言来验证一下用 ICMP 的回送请求和回送回答报文来实现 ping 命令的通信过程。在 FPB 软件里有 ping 命令的具体实现例子，这里只验证正常的通信过程。ping 的具体例子和它的使用方法分别在“PktBuilderSample.py”和“PktBuilderSampleTest.py”文件里。

5.1.1. 相关协议格式

ICMP 的协议格式在各个功能的相应章节里已经阐述，这里不再重复。Ethernet 和 IP 协议也被用到了，它们的格式已在“超级网络入门与实践(二)　学习 TCP, IP, Ethernet 协作的原理”的网络脚本语言的实践章节里已有说明，请参阅那里。ICMP 协议在 FPB 中，被记述在 ICMP.pst 文件里。下面给出 ICMP 协议描述文件的说明。

XML 描述字段	类型	对应协议格式字段或报文
icmp_root	Sequence	协议根字段。对应 ICMP 协议格式整体。
echo_reply	Sequence	回送回答报文
type0	integer	固定值 0。回送回答的类型值。
code0	integer	固定值 0。回送回答的代码值。
checksum	byte_array	检验和。值字段。
identifier	integer	标识符
sequence_number	integer	序号
data	Payload	数据。用于回送请求和回送回答。
nest_ip_data	Payload	IP 数据的一部分。用于各个错误通知。

[30] 比如象 TCP 那样，需要用对方传送来的序号，来返回包。

[31] Jython 是最流行的脚本语言之一 Python 的 Java 实现，与 Python 相比最大的好处就是可以方便地使用 Java 的所有类库。因为，语法与 Python 基本相同，即使用户完全没有 Java 语言的知识也是没有问题的。

unused32	integer	32 位未使用字段
unused24	integer	24 位未使用字段
destination_unreachable	Sequence	终点不可达报文
type3	integer	固定值 3。终点不可达的类型值。
code3	integer	终点不可达的代码值。
source_quench	Sequence	源点抑制报文
type4	integer	固定值 4。源点抑制的类型值。
redirect	Sequence	改变路由报文。
type5	integer	固定值 5。改变路由报文的类型值。
code5	integer	改变路由报文的代码值。
gateway_internet_address	inet4_address	改变路由报文的目标路由器地址
echo_request	Sequence	回送请求报文。
type8	integer	固定值 8。回送请求报文的类型值。
time_exceeded	Sequence	超时报文。
type11	integer	固定值 11。超时报文的类型值。
code11	integer	超时报文的代码值。
parameter_problem	Sequence	参数问题报文。
type12	integer	固定值 12。参数问题报文的类型值。
pointer	integer	参数问题报文的指针。
timestamp_request	Sequence	时间戳请求报文。
type13	integer	固定值 13。时间戳请求报文的类型值。
originate_timestamp	integer	时间戳请求报文的原始时间戳。
receive_timestamp	integer	时间戳请求报文的接收时间戳。
transmit_timestamp	integer	时间戳请求报文的发送时间戳。
timestamp_reply	Sequence	时间戳回答报文。
type14	integer	固定值 14。时间戳回答报文的类型值。
information_request	Sequence	信息请求报文。
type15	integer	固定值 15。信息请求报文的类型值。
information_reply	Sequence	信息回答报文。
type16	integer	固定值 16。信息回答报文的类型值。

5.1.2. 通信顺序图

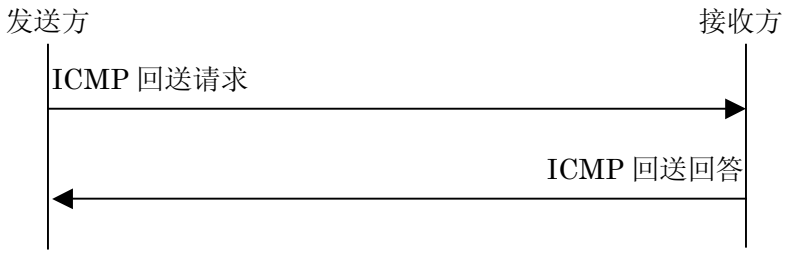

図 13 ping 命令通信顺序图

① 发送 ICMP 回送请求报文。并填写标识符，序号和数据字段。

② 发送 ICMP 回送回答报文。并将受到的 ICMP 回送请求报文的标识符，序号和数据字段，原封不动的返回回去。

5.1.3. 脚本语言实现

```
1  ##########################################
2  #sample for icmp ping
```

```
3       ##########################################
4       from pktbuilder import *
5       import jarray
6       import random
7       import fpformat
8       import time
9
10      def createEchoRequest(srcMac, dstMac, srcIp, dstIp, dataBytes):
11          """ICMP Echo-Request Packet Creation
12          """
13          #protocol stack is ("Ethernet", "IPv4", "ICMP")
14          packet = Packet(["Ethernet", "IPv4", "ICMP"])
15          packet.setTextValue("Ethernet", "destination_address", dstMac)
16          packet.setTextValue("Ethernet", "source_address", srcMac)
17          packet.setIntValue("Ethernet", "ether_type", 0x0800)
18          packet.setIntValue("IPv4", "identification", random.randrange(1, 0xFFFF))
19          packet.setIntValue("IPv4", "protocol", 0x01)
20          packet.setTextValue("IPv4", "source_address", srcIp)
21          packet.setTextValue("IPv4", "destination_address", dstIp)
22          packet.select("ICMP", "icmp_root", "echo_request")
23          data = range(0, dataBytes)
24          packet.setByteArrayValue("ICMP", "data", data)
25          return packet
26
27      #socket factory creation
28      factory = PcapSocketFactory()
29      #get all device(network adapter) name in sequence
30      devices = factory.getDevices()
31      #print "devices:",devices
32      #local machine address
33      LOCAL_MAC = "00:16:D4:17:25:C4"
34      #local machine IP address
35      LOCAL_IP = "192.168.11.2"
36      #create socket
37      socket = factory.createSocket()
38      #set socket parameter. "devices[1]" is selecting the second device.
39      socket.setPcapSetting(devices[1],
40                                          PcapSocket.DEFAULT_SNAPLEN,
41                                          PcapSocket.DEFAULT_PROMISC,
42                                          5)
43      #target IP address(www.baidu.com)
44      dstIp = "220.181.38.4"
45      #gateway mac address
46      gatewayMac = "00:16:01:15:a4:88"
47      #data count
48      dataBytes = 10
49      #buffer creation
50      buffer = jarray.zeros(1600, "b")
51      #Packet Serializer object creation
52      serializer = PacketSerializer()
53      #open the socket
54      socket.open()
55      #receiving filter setting
56      socket.setFilter("ip dst "+ LOCAL_IP + " and icmp")
```

```
57      #create the ID for ICMP
58      icmpId = random.randrange(1, 0xFFFF)
59      seqNo = 1
60      #ICMP Echo-Request packet creation
61      packet = createEchoRequest(LOCAL_MAC, gatewayMac, LOCAL_IP,
62                                              dstIp, dataBytes)
63      #Set the field of id and sequence for protocol of ICMP.
64      packet.setIntValue("ICMP", "identifier", icmpId)
65      packet.setIntValue("ICMP", "sequence_number", seqNo)
66      #AutoUtil().printPacket(packet)
67      #Create the raw sending data.With encoding the packet.
68      rawData = serializer.encode(packet)
69      #Sending
70      startTime = time.time()
71      socket.write(rawData, 0, len(rawData))
72      #Receiving
73      rlen = socket.read(buffer, 0, len(buffer))
74      newPacket = serializer.decodeAuto(buffer, 0, rlen, "Ethernet")
75      if newPacket.contains("ICMP", "echo_reply") :
76          #Check the integration of Packet.The id received must be same as the
77          #ID which sent.
78          recvIcmpId = newPacket.getIntValue("ICMP", "identifier")
79          recvSeqNo = newPacket.getIntValue("ICMP", "sequence_number")
80          if (recvIcmpId != icmpId or recvSeqNo != seqNo):
81              print "Invalid reply with id=", recvIcmpId, " and seq no=", seqNo
82          else :
83              tempIp = newPacket.getTextValue("IPv4", "source_address")
84              dataLen = newPacket.getLength("ICMP", "data")
85              ttl = newPacket.getIntValue("IPv4", "ttl")
86              print "Reply from ", tempIp, " bytes=", dataLen/8, " time<=", ¥
87                  fpformat.fix(time.time()-startTime, 4), "(s)", " TTL=", ttl
88      else :
89          print "Invalid reply without echo_reply"
90      #close socket
91      socket.close()
```

将 33，35，44，46 行的本地 MAC 地址，本地 IP 地址，目标 IP 地址，默认网关 MAC 地址变量改为实际环境的值，在 39 行从 devices（设备名称数组）选择适当设备名称，就能运行了。其中当接收方与发送方在同一网络的时候，网关 MAC 地址就是接收方的 MAC 地址。网关 MAC 地址可以通过“超级网络入门与实践(二) 学习 TCP, IP, Ethernet 协作的原理”内的说明方法来取得。下面是源程序的说明。

4 行　导入网络脚本语言 API 库

5-8 行　导入 Jython 库。jarray 用来生成各类数组，random 用来生成随机数，fpformat 用来格式化文本，time 用来取得时间

10-25 行　定义生成 ICMP 回送请求报文包对象的函数。函数参数分别为源 MAC 地址，目标 MAC 地址，源 IP 地址，目标 IP 地址和 ICMP 的数据字段的字节数。

18 行　使用 jython 的 random 库的 randrange 函数来生成随机数，并赋给 IP 的标识符字段。其中 1 和 0xFFFF 是数值的有效范围。

22 行　选择 ICMP 回送请求报文。

23 行　用 jython 的 range 函数来取得范围为 0 – dataBytes 的数字数组。如果 dataBytes 是 5，则结果为[0，1，2，3，4]

28 行　Socket 工厂对象的生成。PcapSocketFactory 类用来生成 Socket 对象，或取得所在计算机的网卡设备名称数组。

30 行　取得所在计算机的网络设备(网卡)名称数组。

33，35 行　定义本地 MAC 地址和本地 IP 地址

37 行　生成 Socket 对象

39-42 行　设置 Scoket 参数。devices[1]是设备名称，PcapSocket.DEFAULT_SNAPLEN 是截取数据包的最大长度，默认是 1600，PcapSocket.DEFAULT_PROMISC 是是否用 PROMISC 模式，也就是是否收取网络上所有的包包括不是发给自己的，默认是否。5 是收信间隔，单位是 1/1000 秒。

44，46 行　定义目标 IP 地址和默认网关 MAC 地址

48 行　定义 ICMP 回送请求报文的数据字段字节数

50 行　生成放置通信数据的字节数组并全部初始化为 0。1600 时长度，“b”是数组的内容是 byte 的意思。

52 行　包对象串行化器的生成。PacketSerializer 类是用来将包对象转换成可传送的数据块（encode），或将接收到的数据块转换成包对象（decode）。

54 行　打开 Socket 用来通信

56 行　设置收信过滤条件。使用 WinPcap 的过滤条件表达式。“ip dst XXX“是指定目标 IP 地址为 XXX。“icmp”是只收 ICMP 数据包。

58 行　用随机数，定义 ICMP 的标识符

61-62 行　生成 ICMP 回送请求报文包对象

64-65 行　设置 ICMP 的标识符和序号字段

68 行　用包对象串行化器将 ICMP 回送请求包对象转换成送信数据。

70 行　取得通信开始时间。结果是以秒为单位的浮点小数。

71 行　送信

73 行　接收满足过滤条件的数据包。如果经过 1 秒钟，任然没有回信，将抛出异常结束程序执行。

75-87 行　收到 ICMP 回送回答报文后的处理

78，79 行　取得 ICMP 回送回答报文的标识符和序号字段的值

80 行　判断发送和接收到的标识符和序号字段是否一致，来确认 ICMP 回送回答报文的正确性

83-87 行　取得 IP 源地址，ICMP 数据字段的长度，IP 的 ttl（生存时间）字段的值，来打印 ping 命令的正常结果

91 行　关闭 socket

5.2. 用 ICMP 来实现 Traceroute

下面根据“2.4traceroute 命令（通过超时错误来调查到目标的路径)”的说明，用脚本语言来验证一下用 ICMP 的回送请求和回送回答报文来实现 traceroute 命令的通信过程。在 FPB 软件里有 traceroute 命令的具体实现例子，这里只验证正常的通信过程。tracroute 的具体例子和它的使用方法分别在“PktBuilderSample.py”和“PktBuilderSampleTest.py”文件里。

5.2.1. 相关协议格式

traceroute 使用到了 Ethernet，IP，ICMP 协议，各协议已在“5.1用 ICMP 来实现 Ping”章节中说明，这里不再重复。

5.2.2. 通信顺序图

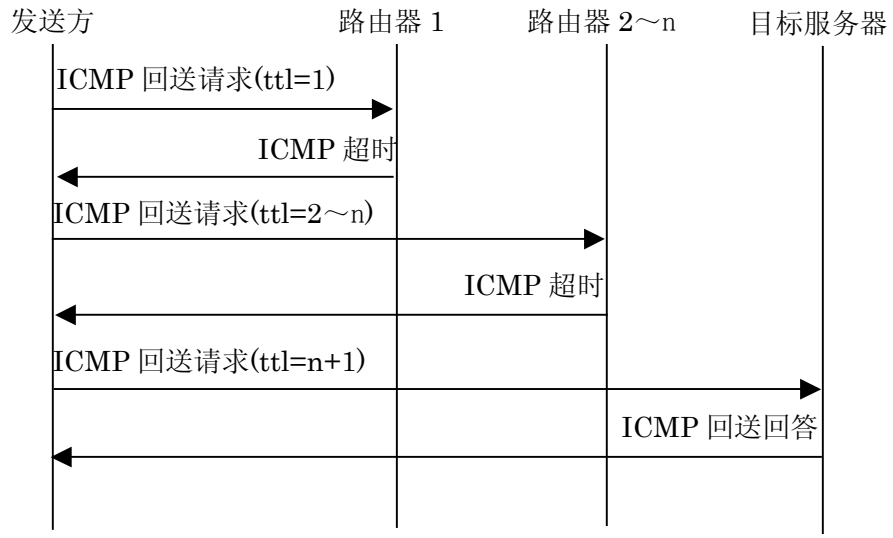

图 14 traceroute 命令顺序图

① 将 ttl 字段设为 1，发送 ICMP 回送请求报文。

② 经过路由器 1，由于 ttl 变为 0，返回 ICMP 超时报文

③ 重复①和②过程直至目标服务器前的一个路由器 n

④ 将 ttl 字段设为 n+1，发送 ICMP 回送请求报文至目标服务器。

⑤ 目标服务器返回 ICMP 回送回答

5.2.3. 脚本语言实现

```
1   ##########################################
2   #sample for icmp traceroute
3   ##########################################
4   from pktbuilder import *
5
6   import jarray
7   import random
8   import fpformat
9   import time
10  import java.io.IOException
11
12  def createEchoRequest(srcMac, dstMac, srcIp, dstIp, dataBytes):
13      """ICMP Echo-Request Packet Creation
14      """
15      #protocol stack is ("Ethernet", "IPv4", "ICMP")
16      packet = Packet(["Ethernet", "IPv4", "ICMP"])
17      packet.setTextValue("Ethernet", "destination_address", dstMac)
18      packet.setTextValue("Ethernet", "source_address", srcMac)
19      packet.setIntValue("Ethernet", "ether_type", 0x0800)
20      packet.setIntValue("IPv4", "identification", random.randrange(1, 0xFFFF))
21      packet.setIntValue("IPv4", "protocol", 0x01)
22      packet.setTextValue("IPv4", "source_address", srcIp)
23      packet.setTextValue("IPv4", "destination_address", dstIp)
24      packet.select("ICMP", "icmp_root", "echo_request")
25      data = range(0, dataBytes)
```

```
26              packet.setByteArrayValue("ICMP", "data", data)
27              return packet
28
29          #socket factory creation
30          factory = PcapSocketFactory()
31          #get all device(network adapter) name in sequence
32          devices = factory.getDevices()
33          #print "devices:",devices
34          #local machine address
35          LOCAL_MAC = "00:16:D4:17:25:C4"
36          #local machine IP address
37          LOCAL_IP = "192.168.11.2"
38          #create socket
39          socket = factory.createSocket()
40          #set socket parameter. "devices[1]" is selecting the second device.
41          socket.setPcapSetting(devices[1],
42                                                PcapSocket.DEFAULT_SNAPLEN,
43                                                PcapSocket.DEFAULT_PROMISC,
44                                                5)
45          #target IP address(www.baidu.com)
46          dstIp = "220.181.38.4"
47          #gateway mac address
48          gatewayMac = "00:16:01:15:a4:88"
49          #data count
50          dataBytes = 10
51          #maxium passed router
52          maxHops = 40
53          #buffer creation
54          buffer = jarray.zeros(1600, "b")
55          #Packet Serializer object creation
56          serializer = PacketSerializer()
57          #open the socket
58          socket.open()
59          #receiving filter setting
60          socket.setFilter("ip dst "+ LOCAL_IP + " and icmp")
61          #create the ID for ICMP
62          icmpId = random.randrange(1, 0xFFFF)
63          #if reached the destination
64          reachedDst = 0
65          for i in xrange(1, maxHops) :
66              #sequence NO. of ICMP
67              seqNo = i
68              #field of TTL in IP, increase 1 each time
69              ttl = i
70              #ICMP Echo-Request packet creation
71              packet = createEchoRequest(LOCAL_MAC, gatewayMac, LOCAL_IP,
72                                                    dstIp, dataBytes)
73              #Set the field of id and sequence for protocol of ICMP.
74              packet.setIntValue("ICMP", "identifier", icmpId)
75              packet.setIntValue("ICMP", "sequence_number", seqNo)
76              packet.setIntValue("IPv4", "ttl", ttl)
77              #AutoUtil().printPacket(packet)
78              #Create the raw sending data.With encoding the packet.
79              rawData = serializer.encode(packet)
```

```
80          #Sending
81          startTime = time.time()
82          socket.write(rawData, 0, len(rawData))
83          #Receiving
84          try:
85              rlen = socket.read(buffer, 0, len(buffer))
86          #ignore the timeout of socket
87          except java.io.IOException:
88              print "Request timeout"
89              continue
90          newPacket = serializer.decodeAuto(buffer, 0, rlen, "Ethernet")
91      #     AutoUtil().printPacket(newPacket)
92          if newPacket.contains("ICMP", "echo_reply"):
93              #Check the integration of Packet.The id received must be same as the
94              #ID which sent.
95              recvIcmpId = newPacket.getIntValue("ICMP", "identifier")
96              recvSeqNo = newPacket.getIntValue("ICMP", "sequence_number")
97              if (recvIcmpId != icmpId or recvSeqNo != seqNo):
98                  print "Invalid reply with id=", recvIcmpId, " and seq no=", seqNo
99              reachedDst = 1
100         tempIp = newPacket.getTextValue("IPv4", "source_address")
101         ttl = newPacket.getIntValue("IPv4", "ttl")
102         print i, " from ", tempIp, " time<=", ¥
103             fpformat.fix(time.time()-startTime, 4), "(s)", " TTL=", ttl
104         if reachedDst:
105             break
106     #print statistics
107     print "Traceroute complete. "
108     if reachedDst:
109         print dstIp , " is reached."
110     else:
111         print dstIp , " is not reached."
112     #close socket
113     socket.close()
```

执行方法与前述 ping 命令基本相同。源程序的构成与原理也基本相同，但有以下区别之处。

64 行　将是否到达目标标志设为假(否)

65 行　重复尝试发送 ICMP 回送请求报文

67，69，75，76 行　将 ICMP 的序号字段与 IP 的 ttl 字段设为循环次数

84-89 行　接收满足过滤条件的报文。try 与 except 是用来捕捉超时异常。异常发生处理在 except 程序块中执行。continue 是跳转语句，执行后程序将直接跳至上一层循环开始处，也就是 65 行。

99 行　如果收到 ICMP 回送回答报文，则认为请求报文已到达目标服务器，所以设置到达目标标志位真。

108-111 行　根据到达目标标志来判断 traceroute 的成功与失败

5.3. 用 UDP 来实现 UDP 的端口扫描

下面根据“2.5.1用 UDP 数据包使错误发生”的说明，来验证用 UDP 数据包来实现端口扫描的通信过程。该通信用到了 Ethernet，IP，UDP 协议。Ethernet 和 IP 协议的格式在“超级网络入门与实践(二)　学习 TCP, IP, Ethernet 协作的原理”的网络脚本语言的实践章节里已有说明，请参阅那里。下面说明 UDP

协议的格式。

5.3.1. UDP 协议格式

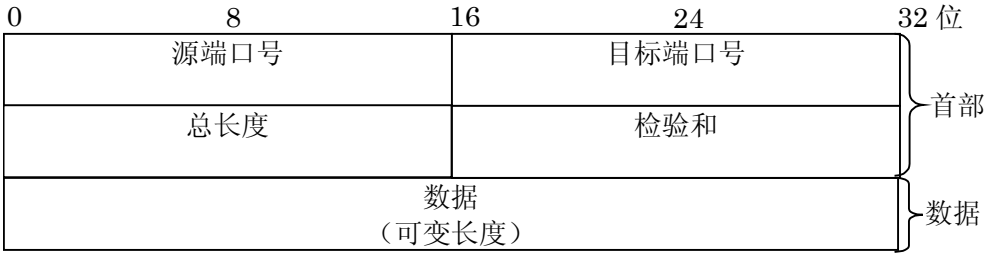

图 15 UDP 协议的格式

上图各个字段的内容：

- **源端口号** 16 位字段。源主机上运行的进程所使用的端口号。
- **目标端口号** 16 位字段。目标主机上运行的进程所使用的端口号。
- **长度** 16 位字段。定义 UDP 数据包的长度。
- **检验和** 16 位字段。用来检验整个数据包是否传输中，发生了损坏。该字段通过生成一个伪首部加上数据部来作为计算对象，与 TCP 的检验和计算方式类似。具体算法，这里不作说明。
- **数据** 可变长字段。用来放置应用程序的数据。

UDP 协议在 FPB 中，由 UDP.pst 文件来描述。下面是描述文件的说明。

XML 描述字段	类型	对应协议格式字段
udpv4_root	Sequence	协议根字段。对应 UDPv4 协议格式整体。 UDP 首部+UDP 数据
udp_header	Sequence	UDP 协议的首部 源端口号+目标端口号+总长度+检验和
source_port	integer	源端口号
destination_port	integer	目标端口号
message_length	integer	总长度
checksum	Select	检验和。可选择不计算检验和与计算检验和
no_checksum	byte_array	不计算检验和。值不自动计算。
computed_checksum	byte_array	计算检验和。值自动计算。
udp_data	Payload	数据

5.3.2. 通信顺序图

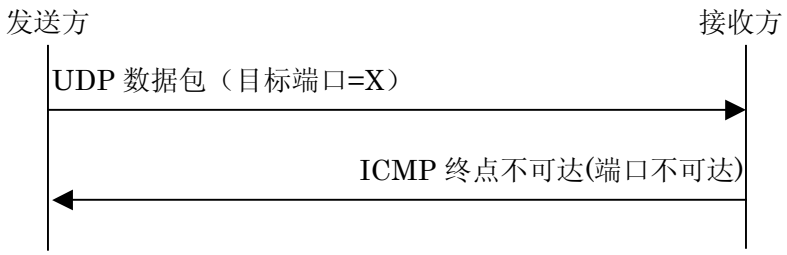

图 16 UDP 端口扫描通信顺序图

① 将测试对象端口号设入 UDP 协议的目标端口号后发送至接收方。

② 当接受方测试对象端口号没有打开时将返回 ICMP 终点不可达（端口不可达）报文。并将接收到的 IP 首部加上 UDP 的前 8 个字节写入 ICMP 报文的数据字段。发送方用返回的 UDP 的前八个字节，也就是源端口号和目标端口号来判断是否针对于自己发送的 UDP 数据包。

5.3.3. 脚本语言实现

```
1  ##########################################
2  #sample for udp port scan
3  ##########################################
4  from pktbuilder import *
5
6  import jarray
7  import random
8  import java.io.IOException
9
10 def createUDPPacket(srcMac, dstMac, srcIp, dstIp, port):
11     #protocol stack is ("Ethernet", "IPv4", "ICMP")
12     packet = Packet(["Ethernet", "IPv4", "UDPv4"])
13     packet.setTextValue("Ethernet", "destination_address", dstMac)
14     packet.setTextValue("Ethernet", "source_address", srcMac)
15     packet.setIntValue("Ethernet", "ether_type", 0x0800)
16     packet.setIntValue("IPv4", "identification", random.randrange(1, 0xFFFF))
17     packet.setIntValue("IPv4", "protocol", 17)
18     packet.setTextValue("IPv4", "source_address", srcIp)
19     packet.setTextValue("IPv4", "destination_address", dstIp)
20     packet.setIntValue("UDPv4", "source_port", port)
21     packet.setIntValue("UDPv4", "destination_port", port)
22     packet.setByteArrayValue("UDPv4", "udp_data", [1, 2, 3, 4, 5])
23     return packet
24
25 #socket factory creation
26 factory = PcapSocketFactory()
27 #get all device(network adapter) name in sequence
28 devices = factory.getDevices()
29 #print "devices:",devices
30 #local machine address
31 LOCAL_MAC = "00:16:D4:17:25:C4"
32 #local machine IP address
33 LOCAL_IP = "192.168.11.2"
34 #create socket
35 socket = factory.createSocket()
36 #set socket parameter. "devices[1]" is selecting the second device.
37 socket.setPcapSetting(devices[1],
38                       PcapSocket.DEFAULT_SNAPLEN,
39                       PcapSocket.DEFAULT_PROMISC,
40                       5)
41 #target IP address(www.baidu.com)
42 #dstIp = "220.181.38.4"
43 dstIp = "192.168.11.1"
44 #gateway mac address
45 gatewayMac = "00:16:01:15:a4:88"
46 #port range
47 portRange = range(40, 50)
48 #portRange.remove(445)
49 #buffer creation
```

```
50      buffer = jarray.zeros(1600, "b")
51      #Packet Serializer object creation
52      serializer = PacketSerializer()
53      #open the socket
54      socket.open()
55      #receiving filter setting
56      socket.setFilter("ip dst "+ LOCAL_IP + " and icmp")
57      #send the udp packet
58      for i in portRange:
59          #create udp packet
60          packet = createUDPPacket(LOCAL_MAC, gatewayMac, LOCAL_IP, dstIp, i)
61          #Create the raw sending data.With encoding the packet.
62          rawData = serializer.encode(packet)
63          socket.write(rawData, 0, len(rawData))
64      closedPorts = []
65      #receive the icmp packet
66      while 1:
67          #Receiving
68          try :
69              rlen = socket.read(buffer, 0, len(buffer))
70          #ignore the timeout of socket
71          except java.io.IOException :
72              print "Finished"
73              break
74          newPacket = serializer.decodeAuto(buffer, 0, rlen, "Ethernet")
75          #get udp port
76          if (newPacket.contains("ICMP", "destination_unreachable")):
77              data = newPacket.getByteArrayValue("ICMP", "nest_ip_data")
78              textBitset = TextBitset(data)
79              #print "textBitset:", textBitset.getAll(TextBitset.HEX)
80              if textBitset.size() >= 24 * 8:
81                  textBitset = textBitset.subset(22*8, 2*8)
82                  port = textBitset.getAsInteger()
83                  closedPorts.append(port)
84      #close socket
85      socket.close()
86      print "port closed for ", dstIp, "is ", closedPorts
```

执行方法与前述 ping 命令基本相同。源程序的构成与原理也基本相同，但有以下区别之处。

10-23 行　定义生成 UDP 包对象的函数

22 行　设置 UDP 协议的数据部分。[1, 2, 3, 4, 5]是 Jython 的数组定义方法，它有从 1 到 5 五个数字成员。

47 行　定义测试端口范围。range 是 Jython 的范围生成函数。range(40, 50)将生成有 40 到 49 十个数字的数组。

58-63 行　用测试范围内的端口号生成 UDP 数据包并发送[32]。

64 行　定义空已关闭端口数组

66-83 行　循环接收 ICMP 的终点不可达报文，判断关闭端口，一直到接收超时。接收超时在 71-73 的异常处理内处理。

[32] 在 WindowsXP 上发送端口号为 445 的 UDP 数据包会失败，可能是操作系统内部的限制，原因不是很清楚。

76 行　判断是否为 ICMP 终点不可达报文

77-83 行　从 ICMP 的数据部分取出错误 UDP 数据包的目标端口字段的值。81 行的 22*8 的意思是偏移 22 个字节，也就是 IP 首部+源端口号的长度，从而取得目标端口号的值。并将关闭了的端口号（目标端口号）加入已关闭端口数组

86 行　打印已关闭端口数组

5.4. 用 TCP 来实现 TCP 的端口扫描

下面根据“2.5.1用 UDP 数据包使错误发生”的说明，来验证用 UDP 数据包来实现端口扫描的通信过程。

5.4.1. 相关协议格式

该通信用到了 Ethernet，IP，TCP 协议。各协议的格式在“超级网络入门与实践(二) 学习 TCP, IP, Ethernet 协作的原理”的网络脚本语言的实践章节里已有说明，请参阅那里。

5.4.2. 通信顺序图

图 17 TCP 的端口扫描通信顺序图

① 将测试端口号填入 TCP 的目标端口字段，发送 TCP SYN 数据包。

② 测试端口如果没有被打开，接收方将发送 TCP RST+ACK 数据包。

5.4.3. 脚本语言实现

```
1   ##########################################
2   #sample for tcp port scan
3   ##########################################
4   from pktbuilder import *
5   import jarray
6   import random
7   def createTCPPacket(srcMac, dstMac, srcIp, dstIp, srcPort, dstPort):
8       """create normal tcp packet"""
9       packet = Packet(["Ethernet", "IPv4", "TCPv4"])
10      packet.setTextValue("Ethernet", "destination_address", dstMac)
11      packet.setTextValue("Ethernet", "source_address", srcMac)
12      packet.setIntValue("Ethernet", "ether_type", 0x0800)
13      packet.setIntValue("IPv4", "identification", random.randrange(1, 0xFFFF))
14      packet.setIntValue("IPv4", "protocol", 0x06)
15      packet.setTextValue("IPv4", "source_address", srcIp)
16      packet.setTextValue("IPv4", "destination_address", dstIp)
17      packet.setIntValue("TCPv4", "source_port", srcPort)
18      packet.setIntValue("TCPv4", "destination_port", dstPort)
19      packet.setIntValue("TCPv4", "window_size", 2000)
20      return packet
21
22  #socket factory creation
```

```
23      factory = PcapSocketFactory()
24      #get all device(network adapter) name in sequence
25      devices = factory.getDevices()
26      #print "devices:",devices
27      #local machine address
28      LOCAL_MAC = "00:16:D4:17:25:C4"
29      #local machine IP address
30      LOCAL_IP = "192.168.11.2"
31      #create socket
32      socket = factory.createSocket()
33      #set socket parameter. "devices[1]" is selecting the second device.
34      socket.setPcapSetting(devices[1],
35                                          PcapSocket.DEFAULT_SNAPLEN,
36                                          PcapSocket.DEFAULT_PROMISC,
37                                          5)
38      #target IP address(www.baidu.com)
39      #dstIp = "220.181.38.4"
40      dstIp = "192.168.11.1"
41      srcPort = random.randrange(10000, 19999)
42      #gateway mac address
43      gatewayMac = "00:16:01:15:a4:88"
44      localSeqNo = random.randrange(1, 0xFFFF)
45      #port range
46      portRange = range(40, 100)
47      #buffer creation
48      buffer = jarray.zeros(1600, "b")
49      #Packet Serializer Creation
50      serializer = PacketSerializer()
51      #open the socket
52      socket.open()
53      #receiving filter setting
54      socket.setFilter("ether dst "+ LOCAL_MAC + " and tcp "+ " and dst host "+
55              LOCAL_IP + " and src host "+ dstIp)
56
57      closedPorts = []
58      for i in portRange:
59          #create syn tcp packet
60          packet = createTCPPacket(LOCAL_MAC, gatewayMac, LOCAL_IP, dstIp, srcPort, i)
61          packet.setIntValue("TCPv4", "syn", 1)
62          packet.setIntValue("TCPv4", "sequence_number", localSeqNo)
63          #AutoUtil().printPacket(packet)
64          #packet encoding
65          rawData = serializer.encode(packet)
66          #sending syn
67          socket.write(rawData, 0, len(rawData))
68
69          #receiving syn+ack
70          rlen = socket.read(buffer, 0, len(buffer))
71          #decode packt
72          resPacket = serializer.decodeAuto(buffer, 0, rlen, "Ethernet")
73          #AutoUtil().printPacket(resPacket)
74          if resPacket.getIntValue("TCPv4", "rst") == 1:
75              closedPorts.append(resPacket.getIntValue("TCPv4", "source_port"))
76      #close socket
```

```
77          socket.close()
78          print "port closed for ", dstIp, "is ", closedPorts
```

执行方法与前述 ping 命令基本相同。源程序的构成与原理也基本相同，但有以下区别之处。

7-20 行　生成 TCP 包对象

46 行　定义测试端口范围。range 是 Jython 的范围生成函数。range(40, 100)是生成有 40 到 99 五十个数字的数组。

54 行　定义 WinPcap 过滤条件。用来限定目标 MAC 地址，目标 IP 地址，源 IP 地址的值。并且必须是 TCP 数据包。

58-78 行　对每一个测试端口进行测试。其中 62 行将发送 TCP 设置成 SYN 类型。75-76 判断是否是 RST 类型的 TCP 数据包，并将已关闭的端口加入关闭端口数组。

79 行　打印关闭端口数组。