



# Wireshark数据包 分析实战（第2版）

PRACTICAL PACKET ANALYSIS 2ND EDITION

[美] Chris Sanders 著 诸葛建伟 陈霖 许伟林 译



人民邮电出版社  
POSTS & TELECOM PRESS

潇湘雨制作：QQ：362027666

读者可通过<http://nostarch.com/packet2.htm>或<http://vdisk.weibo.com/s/nxSYU>下载本书中使用的捕获文件。



特别说明：  
作者从本书获得的版税收入都将捐赠给农村科技基金会 (<http://ruraltechfund.org>)

译者从本书获得的稿费收入都将捐赠给随手公益基金 (<http://weibo.com/ssgyjj>)

借助 Wireshark 这款世界上最流行的网络嗅探器，可以很容易地捕获到网络中的数据包，而不管是有线网络还是无线网络。但是，如何使用这些数据包来理解网络状况呢？

本书在上一版的基础之上进行了大幅修订，提供了 45 个全新的场景，并讨论了更多的网络协议，以帮助读者掌握理解 PCAP 数据的方法。本书新增了对网络变慢进行排错以及出于安全目的而对数据包进行分析的章节，旨在帮助读者更好地理解当今的安全漏洞和恶意软件在数据包层面上的行为。此外，本书还全面介绍了 TCP/IP 网络协议栈，以帮助读者提升数据包分析能力。

本书是任何网络技术人员、网络管理员和网络工程师的必备手册。别再凭空揣测，还是赶紧利用本书中的知识来解决网络中遇到的问题吧！

### 您将学到：

- 使用数据包分析来识别和解决常见网络问题，如网络中断、连接丢失、DNS 故障、网速超慢、感染恶意代码等；
- 构建自定义的捕获和显示过滤器；
- 实时监控网络并监听网络通信；
- 绘制流量模式，使流经网络的数据可视化；
- 使用 Wireshark 的高级特性来理解捕获的数据包；
- 生成统计数据和报告，以更好地向非专业人员解释技术层面的网络信息。

**Chris Sanders** 身兼计算机安全咨询人员、作家和研究人员等多种角色。他还是一名 SANS 导师，并持有 CISSP、GCIA、GCIH、GREM 等行业证书。Sanders 定期在 WindowSecurity.com 网站和自己的博客 ChrisSanders.org 上发表文章。

人民邮电出版社-信息技术分社  
<http://weibo.com/ptpitbooks>

ISBN 978-7-115-30236-6

A standard linear barcode representing the ISBN 978-7-115-30236-6.

9 787115 302366 >

ISBN 978-7-115-30236-6

定价：49.00 元

潇湘雨制作：QQ：362027666

美术编辑：王建国

分类建议：计算机/网络技术/网络安全

人民邮电出版社网址：[www.ptpress.com.cn](http://www.ptpress.com.cn)



# Wireshark数据包 分析实战（第2版）

PRACTICAL PACKET ANALYSIS 2ND EDITION

【美】Chris Sanders 著 谷葛建伟 陈霖 许伟林 译

人民邮电出版社  
北京

## 图书在版编目 (C I P) 数据

Wireshark数据包分析实战 : 第2版 / (美) 桑德斯  
(Sanders, C.) 著 ; 诸葛建伟, 陈霖, 许伟林译. -- 北  
京 : 人民邮电出版社, 2013. 3  
ISBN 978-7-115-30236-6

I. ①W… II. ①桑… ②诸… ③陈… ④许… III. ①  
计算机网络—数据—分析—应用软件 IV. ①TP393. 09

中国版本图书馆CIP数据核字(2012)第294560号

## 版 权 声 明

Copyright © 2011 by Chris Sanders. Title of English-language original: Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems(2nd Edition), ISBN 978-1-59327-266-1, published by No Starch Press. Simplified Chinese-language edition copyright © 2012 by Posts and Telecom Press. All rights reserved.

本书中文简体字版由美国 No Starch 出版社授权人民邮电出版社出版。未经出版者书面许可，对本书任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

## Wireshark 数据包分析实战 (第 2 版)

- 
- ◆ 著 [美] Chris Sanders
  - 译 诸葛建伟 陈 霖 许伟林
  - 责任编辑 傅道坤
  - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号  
邮编 100061 电子邮件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>
  - 三河市海波印务有限公司印刷
  - ◆ 开本: 800×1000 1/16  
印张: 17.75  
字数: 355 千字 2013 年 3 月第 1 版  
印数: 1 - 4 000 册 2013 年 3 月河北第 1 次印刷

著作权合同登记号 图字: 01-2012-2971 号  
ISBN 978-7-115-30236-6

---

定价: 49.00 元

读者服务热线: (010) 67132692 印装质量热线: (010) 67129223  
反盗版热线: (010) 67171154

# 内 容 提 要

本书从网络嗅探与数据包分析的基础知识开始，渐进地介绍 Wireshark 的基本使用方法及其数据包分析功能特性，同时还介绍了针对不同协议层与无线网络的具体实践技术与经验技巧。在此过程中，作者结合一些简单易懂的实际网络案例，图文并茂地演示使用 Wireshark 进行数据包分析的技术方法，使读者能够顺着本书思路逐步地掌握网络数据包嗅探与分析技能。最后，本书使用网络管理员、IT 技术支持、应用程序开发者们经常遇到的实际网络问题（包括无法正常上网、程序连接数据库错误、网速很卡，以及遭遇扫描渗透、ARP 欺骗攻击等），来讲解如何应用 Wireshark 数据包分析技术和技巧，快速定位故障点，并找出原因以解决实际问题。本书覆盖了无线 WiFi 网络中的嗅探与数据包分析技术，同时也给出了嗅探与数据包分析领域丰富的参考技术文档、网站、开源工具与开发库等资源列表。

本书适合网络管理员、安全工程师、软件开发工程师与测试人员，以及网络工程、信息安全等专业学生与网络技术爱好者阅读。

## 关于作者

Chris Sanders 是一名计算机安全咨询顾问、作家和研究人员。他还是一名 SANS 导师，持有 CISSP、GCIA、GCIH、GREM 等行业证书，并定期在 WindowsSecurity.com 网站和自己的博客 ChrisSanders.org 发表文章。Sanders 每天都会使用 Wireshark 进行数据包分析。他目前居住在美国南卡罗米纳州查尔斯顿，以国防承包商的身份工作。

## 联系作者

我非常期盼能够收到本书读者的反馈，如果你出于任何原因想联系我，你可以将你的疑问、评论、批评，甚至是求婚信，直接发送到我的电子邮箱 chris@chrissanders.org，我经常在 <http://www.chrissanders.org/> 发表博客，也欢迎你在 Twitter 上成为 @chrissanders88 的粉丝。

# 致 谢

本书的成功出版离不开很多人的直接贡献和间接支持。

爸爸，我从很多来源获得动力，但比起听到你说你因为我而骄傲，没有其他任何事情能够让我更加高兴。对此，我对你真是感谢不尽。

妈妈，本书第 2 版将在你过世 10 周年纪念日的前夕出版，我知道你一直在天堂注视着我，并为我而自豪，我希望我能够继续努力，能够让你更加为我自豪。

Debi 叔叔和 Randy 阿姨，你们从我写书的第一天起就是我最大的支持者。我没有一个大家庭，但我非常珍视我所拥有的亲情，特别是你们。尽管我们并没有能像我所期望的那样经常会面，但我深深地感谢你们，对我来说，你们和我的亲生父母没有差异。

Tina Nance，虽然最近我们并没有像以前那样有很多的交谈，但我一直将你视作我第二个母亲。如果没有你的支持和信任，我今天不会在做我自己想做的事情。

Jason Smith，你比其他任何人都承受了更多我的牢骚，仅仅这样对我来说都已经是很大的帮助了。谢谢你成为我的好朋友和合作者，在很多项目中为我提供了建议，并让我占用了你家的车库长达 6 个月之久。

感谢我的同事们（过去的和现在的），我一直相信如果一个人周围都是好人的话，他也会成为一个好人。我非常幸运能够和一些优秀人士一起工作，你们

是最棒最聪明的，你们是我的家人。

Mike Poor，你是我毋庸置疑的数据包分析技术的崇拜偶像。你的工作与达成目标的方法一直在给我启迪，并帮助我做我想要做的事情。

Tyler Reguly，非常感谢你为本书承担技术编辑的角色，我知道这并不是个有趣的活儿，但是绝对必要并需要感谢。

同样需要特别感谢 Gerald Combs 和 Wireshark 开发团队，正是 Gerald 和其他几百位开发者的无私投入，才让 Wireshark 能够成为如此强大的分析平台。如果没有他们的贡献，本书也不会存在……即使存在，也可能是基于 tcpdump，那么就不会像现在这样有趣。

感谢 Bill 和 No Starch 出版社的同仁给予我这位来自肯塔基州的小人物不仅仅是一次，而是两次的机会，谢谢你们对我的容忍和耐心，并帮助我让我的梦想成真。

## 关于译者

诸葛建伟，博士，现为清华大学网络与信息安全实验室副研究员、狩猎女神科研团队负责人、CCERT 紧急响应组成员、著名开源信息安全研究团队 The Honeynet Project 的正式成员，以及中国蜜网项目组（[www.honeynet.org.cn](http://www.honeynet.org.cn)）团队发起人和现任负责人、信息安全领域培训讲师和自由撰稿人，编写与参与翻译的畅销图书有《网络攻防技术与实践》、《Metasploit 渗透测试指南》等。新浪微博：@清华诸葛建伟。

陈霖，现就读于洛桑联邦理工学院硕士项目，于北京大学获得计算机科学学士学位。新浪微博：@Larch 爱刀刀。

许伟林，现为清华大学网络与信息安全实验室助理工程师，于北京邮电大学获得计算机科学学士学位，曾两次参与 Google Summer of Code 计划，分别为 Nmap 和 The Honeynet Project 贡献 IPv6 主机发现模块与 IPv6 攻击检测器的开源代码，并负责新浪微博 @Nmap 网络扫描的维护工作。新浪微博：@许伟林。

# 译者序

谨以此书献给中国农村渴望得到更多计算机基础教育的孩子们！

本书是一本非常实用的网络技能培训技术图书，以最为流行和强大的开源数据包抓包与分析工具——Wireshark——作为基础软件，通过大量生动的真实场景案例来讲解数据包分析的基础技术、高级特性与实际应用技能。

Wireshark 在 2011 年最新的 SecTools 安全社区流行软件排行榜中，超越 Metasploit、Nessus、Aircrack 与 Snort，占据榜首位置，这充分体现了 Wireshark 软件在网络安全与取证分析方面的重要作用与流行度。与此同时，Wireshark 更是一个通用化的网络数据嗅探器与协议分析器，在网络运行管理、网络故障诊断、网络应用开发与调试等各个方面都作为基本手头工具，而被网络管理员、软件开发工程师与测试人员所广泛使用。

尽管网络数据包分析技术是网络管理员、安全工程师、软件开发工程师与测试人员必备的基本技能，然而可惜的是国内却没有优秀的实用技术培训教材与书籍。在这广大技术人群遭遇一些实际网络问题，或是对 Wireshark 等网络嗅探器与协议分析器的使用存在疑问时，他们并没有系统性的资料去学习掌握基础知识与技能，并掌握相关技能与方法来解决工作中的实际挑战，而往往只能寄希望于在网络上搜索一些零散并且可能过时的信息来探索尝试，从而花费了大量宝贵时间，并对数据包分析技术的自我修炼不得要领。

本书（英文版）原是国外一本非常优秀、高度注重基础性与实用性的网络数据包分析技术教程，细致地讲解了网络数据包嗅探与分析的技术原理，并以目前最流行的网络数据包嗅探与分析开源工具 Wireshark 作为平台软件，一步步引导读者们掌握使用 Wireshark 进行实际网络中数据包分析、故障定位与问题解决的实践技术方法。

在本书的章节设计上，作者从网络嗅探与数据包分析的基本知识背景开始，以简练清晰的语言帮助读者首先建立起网络协议与数据包结构、不同网络场景下的嗅探方法等方面的基础知识，然后渐进地介绍 Wireshark 的基本使用方法、

Wireshark 的数据包分析功能特性，以及针对不同协议层与无线网络的具体实践技术和经验技巧，在此过程中，作者使用了简单易懂的一些实际网络案例，图文并茂地结合这些具体案例来演示使用 Wireshark 进行数据包分析的技术方法，使读者能够顺着本书思路逐步地掌握网络数据包嗅探与分析的技能。最后，本书以网络管理员、IT 技术支持、应用程序开发者经常遇到的实际网络问题，包括无法正常上网、部署分部网络、程序连接数据库错误、网速很卡，以及遭遇扫描渗透与 ARP 欺骗攻击等，来讲解如何应用 Wireshark 数据包分析技术和技巧，快速定位故障点与原因并解决实际问题。本书也覆盖了无线 WiFi 网络中的嗅探与数据包分析技术，同时也给出了在嗅探与数据包分析领域丰富的参考技术文档、网站、开源工具与开发库等资源列表。

本书原版取得了国外读者的一致好评，在 Amazon 网络安全领域图书销售排名位居 Top 20，以及 4 星半的良好评价，也验证了本书在网络安全领域的重要程度。此外，本书在 2007 年第一版基础上又增加了更多的技术内容和案例，是通过了市场检验的一本好书。

更加难能可贵的是，本书作者 Chris Sanders 是带着一颗善良感恩的心完成了本书的创作。他出生于美国的乡村地区，在通过自身努力成为一位网络技术专家后，成立了一家乡村科技基金会，并将本书的所有版税捐赠出来，为乡村学生设立奖学金，致力于减少乡村学生与城市同龄学生们之间的数字鸿沟。

本书和译者也非常有缘，在 2011 年 10 月为电子工业出版社进行书籍评估时，译者就已经对本书赞誉有加，并给出了引进的建议。之后，在本书中文版权的激烈竞争中，人民邮电出版社最终胜出，这也让译者一度认为无缘这本优秀的作品。机缘巧合的是，在新浪微博上回复一位读者@过来的微博评论时，译者向几个出版社编辑微博发出本书“通缉令”，意外地从人民邮电出版社编辑那获知了本书下落，也非常高兴地接受了他们的翻译邀请。为了平衡翻译质量和进度，我们组织起了 3 人的译者团队，由诸葛建伟译序、前言和第 1~3 章，陈霖译第 4~7 章，许伟林译第 8~11 章与附录。全书内容由诸葛建伟进行全面、仔细的统稿与审校，并由陈爱华、陈建军、刘跃、崔丽娟进行了试读。本书翻译正值学校暑假，因此译者团队也都投入了充分的时间来保障翻译质量，仔细推敲和统一全书技术词汇的译法，确保对翻译内容的技术掌控，从而能够忠实地描述出原书作者期望传递给读者的知识与技能。

在翻译到前言部分，获知本书作者的成长经历以及对乡村学生所做的公益事

业，译者团队感同身受，三位译者中的两位也是从中国的农村地区成长起来的，也都还深刻地记忆着第一次接触到学习机和电脑时那种兴奋异常的感觉。在走向小康社会的今日中国，仍然有这样一个被社会边缘化的群体，他们渴望了解这个世界更多，渴望在互联网世界里遨游，但他们却因为贫困而无法跨越数字鸿沟。这就是农民工子弟学校计算机教育的现状，而在广大的农村地区，问题更加严重。1.5亿农村中小学生，竟然有77.3%从来没有见过电脑！但与此同时，我们看到，社会上存在着大量的废旧电脑资源，城市里每年淘汰的电脑就达500万台。

当你要淘汰掉手中“没有价值”的旧电脑，你能否记起那些求知若渴的孩子们？科技更新换代越来越快，许多旧电脑只要经过简单的整修，便可继续使用至少2年。

因此译者团队达成共识，将本书的公益特性进行到底，决定将本书的译者稿费捐赠给清华大学学生教育扶贫公益协会，通过@电脑传爱活动，将旧电脑维修之后，为打工子弟小学建立电脑室，将公益的精神传递下去。我们也非常欢迎读者能够参与公益事业，事实上，各位读者在购买本书的同时，就已经为公益做出了一份贡献。如果你愿意捐赠淘汰的电脑和计算机基础书籍，欢迎通过新浪微博@电脑传爱，也可以@清华诸葛建伟。如果你是高校学生，希望通过电脑维修做些公益并积累社会实践经验，也非常欢迎加入到@电脑传爱组织的高校电脑维修公益志愿者活动。

公益是举手投足之间的事，我们倡议，每个人为社会公益奉献一点点，让它汇成可以改变世界的力量！



一台电脑开启一扇窗，一次行动传递一份爱！

清华大学教育扶贫电脑传爱公益活动，邀你同行。

诸葛建伟 陈霖 许伟林

2013年1月于北京清华园

# 前 言

本书从 2009 年底开始编写，在 2011 年中期完成，总计历时一年半。而在本书出版之日，已经距离本书第 1 版发布的时间有 4 年之久。本书的所有内容几乎都经过了重写，并采用了完全重新设计的网络捕获数据包文件和场景。如果你喜欢本书第 1 版，那么你也会喜欢本书，因为本书采用了与第 1 版同样的写作方式，以一种简单容易理解的风格来展示技术。如果你不喜欢第 1 版，你也会喜欢本书，因为新版拥有全新设计的场景和扩展后的充实内容。

## 为什么购买本书

你一定很想知道为什么应该买这本书，而不是其他关于数据包分析的书籍。答案在于本书的书名：Practical Packet Analysis。让我们面对这样的现实——没有比实际的经验更加重要的了，而本书通过大量的真实场景中的数据包分析案例，让你获得最贴近实际的经验。

本书的前半部分将为你提供理解数据包分析技术和 Wireshark 软件的必备前置条件。后半部分则完全是一些你在日常网络管理中很容易遇到的一些实际案例。无论你是一位网络技术员、网络管理员、首席信息官、IT 技术支持，还是一位网络安全分析师，你都可以从本书描述的理解与使用数据包分析技术中

获得很多的收获。

## 概念与方法

我是一个非常随意的人，所以，当我教授你一个概念时，我也会尝试用非常随意的方式来进行解释。而本书的语言也会同样的随意，因此你可能比较容易在一些处理技术概念的行话中迷失，但我已经尽我所能地保持行文的一致与清晰，让所有的定义更加明确、直白，没有任何繁文缛节。然而我终究是从伟大的肯塔基州来的，所以我不得不收起我们的一些夸张语气，但你如果在本书中看到一些粗野的乡村土话，请务必原谅我。

如果你真地想学习并精通数据包分析技术，你应该首先掌握本书前几章中介绍的概念，因为它们是理解本书其余部分的前提。本书的后半部分将是纯粹的实战内容，或许你在工作中并不会遇到完全相同的场景，但你在学习本书后应该可以应用所学到的概念与技术，来解决你所遇到的实际问题。

接下来让我们快速浏览本书各个章节的主要内容。

- 第 1 章：“数据包分析与网络基础”，什么是数据包分析技术？这种技术的基本原理是什么样的？你该如何使用这项技术？本章将覆盖这些网络通信与数据包分析的基础知识。
- 第 2 章：“监听网络线路”，本章将覆盖你在网络中放置数据包嗅探器时可以使用的各种不同技术方法。
- 第 3 章：“Wireshark 入门”，从本章起，我们将开始进入 Wireshark 软件的世界，我们将介绍 Wireshark 软件的入门知识——哪里可以下载到，如何使用它，它完成什么功能，为什么它广受好评与关注，以及其他各种好东西。
- 第 4 章：“玩转捕获数据包”，在你运行 Wireshark 软件之后，你会需要知道如何与捕获的数据包进行交互，而这是你开始学习基础实践方法的起始点。
- 第 5 章：“Wireshark 高级特性”，一旦你已经学会了缓慢地爬行，那是时候来学习跑步了。本章将深入钻研 Wireshark 的高级特性，带你揭开引擎的盖子，来了解一些比较少见的操作。
- 第 6 章：“通用底层网络协议”，本章将为你展示那些最常见的通用底层

网络通信协议——比如 TCP、UDP 和 IP——从数据包的层次上来看。为了解决这些网络协议中发生的故障，你首先需要理解它们是如何工作的。

- 第 7 章：“常见高层网络协议”，继续讲解网络协议的相关内容，本章将带你了解 3 种最为常见的高层网络通信协议——HTTP、DNS 与 DHCP——并从数据包的层次上来看。
- 第 8 章：“基础的现实世界场景”，这章中将包含一些常见的网络流量，以及最初的现实世界场景中的案例。每个案例将采用一种容易跟随的方式进行展示，包括问题、分析和解决方法。这些基础场景案例仅仅涉及到少量几台计算机，以及很少的分析——仅仅能够将你的脚打湿。
- 第 9 章：“让网络不再卡”，网络技术人员最普遍遇到的网络问题是网络性能很慢的情况，本章便是专门为解决这一问题而设计的。
- 第 10 章：“安全领域的数据包分析”，网络安全是信息技术领域中最大的热点问题，本章将带给你几个关于如何使用数据包分析技术解决安全相关问题的实际案例。
- 第 11 章：“无线网络数据包分析”，本章是无线网络数据包分析技术启蒙，讨论了无线数据包分析与有线数据包分析技术的差异，并包含了无线网络流量分析的几个案例。
- 附录 A：“延伸阅读”，本书附录给出了其他一些参考工具和网站列表，你可以从中找到继续使用你所学到的数据包分析技术的进一步资料。

## 如何使用本书

我期待本书按照如下两种方式进行使用。

- 作为一本教学书籍，你可以按一章接着一章的顺序阅读，来获得对数据包分析技术的理解与掌握。这种方式会特别关注后面几章中的现实世界场景案例。
- 作为一本参考索引资料，有些 Wireshark 软件的特性是你不会经常使用到的，所以你可能会忘记它们是如何工作的。数据包分析实用技术是你的书柜中一本非常有用的参考书，当你需要快速重温如何使用 Wireshark 软件的某个特

性时，你可以从本书中获得参考。我已经提供了很多独特的图表、图解和方法说明，已经被证明能够作为你进行数据包分析的有用索引参考。

## 关于示例网络数据包捕获文件

所有本书中使用的网络数据包捕获文件都在本书的官方网站 (<http://www.nostarch.com/packet2.htm><sup>1</sup>) 下载到，为了最大化本书的价值，我强烈建议你下载这些文件，并在你学习每个真实案例时使用它们。

## 乡村科技基金会

在我编写本书的前言时，我无法不提及由数据包分析实用技术书籍而衍生出的这一美好事物。在本书第一版出版后不久，我创办了一个 501(c)(3)的非营利性组织，而这正是我最大梦想成为现实的巅峰时刻。

比起城市与市郊的学生们，乡村学生即使拥有很优秀成绩，仍然很少有机会能够接触到最新的科技。创办于 2008 年的乡村科技基金会 (RTF) 致力于能够减少乡村学生与城市同龄学生们之间的数字鸿沟，主要通过针对性的奖学金项目、社区参与计划，以及一些在乡村地区的科技促进与提倡项目来达成。我们的奖学金项目专门提供给生活在乡村，但对计算机技术拥有着热情并希望在这个方向得到进一步教育的学生们。我非常高兴地宣布本书所有的作者版税将提供给乡村科技基金会，用于设立这些奖学金。如果你需要了解更多关于乡村科技基金会的信息，或者想了解你如何可以参与贡献，请访问我们的网站 <http://www.ruraltechfund.org/>。

<sup>1</sup> 为了方便国内读者下载，将在 <http://netsec.ccert.edu.cn/hacking/book/> 提供备份链接。——译者注

# 目 录

|                               |    |
|-------------------------------|----|
| <b>第1章 数据包分析技术与网络基础</b> ..... | 1  |
| 1.1 数据包分析与数据包嗅探器 .....        | 2  |
| 1.1.1 评估数据包嗅探器 .....          | 2  |
| 1.1.2 数据包嗅探器工作原理 .....        | 3  |
| 1.2 网络通信原理.....               | 4  |
| 1.2.1 协议 .....                | 4  |
| 1.2.2 七层 OSI 参考模型 .....       | 5  |
| 1.2.3 数据封装 .....              | 8  |
| 1.2.4 网络硬件 .....              | 10 |
| 1.3 流量分类.....                 | 15 |
| 1.3.1 广播流量 .....              | 15 |
| 1.3.2 多播流量 .....              | 16 |
| 1.3.3 单播流量 .....              | 16 |
| 1.4 小结.....                   | 17 |
| <b>第2章 监听网络线路</b> .....       | 19 |
| 2.1 混杂模式.....                 | 20 |
| 2.2 在集线器连接的网络中进行嗅探 .....      | 21 |
| 2.3 在交换式网络中进行嗅探 .....         | 23 |
| 2.3.1 端口镜像 .....              | 23 |
| 2.3.2 集线器输出 .....             | 25 |

|                             |           |
|-----------------------------|-----------|
| 2.3.3 使用网络分流器 .....         | 26        |
| 2.3.4 ARP 欺骗 .....          | 29        |
| 2.4 在路由网络环境中进行嗅探 .....      | 34        |
| 2.5 部署嗅探器的实践指南 .....        | 36        |
| <b>第3章 Wireshark入门.....</b> | <b>39</b> |
| 3.1 Wireshark简史 .....       | 39        |
| 3.2 Wireshark的优点 .....      | 40        |
| 3.3 安装Wireshark .....       | 41        |
| 3.3.1 在微软Windows系统中安装 ..... | 41        |
| 3.3.2 在Linux系统中安装 .....     | 43        |
| 3.3.3 在Mac OS X系统中安装 .....  | 45        |
| 3.4 Wireshark初步入门 .....     | 45        |
| 3.4.1 第一次捕获数据包 .....        | 45        |
| 3.4.2 Wireshark主窗口 .....    | 46        |
| 3.4.3 Wireshark首选项 .....    | 48        |
| 3.4.4 数据包彩色高亮 .....         | 49        |
| <b>第4章 玩转捕获数据包.....</b>     | <b>53</b> |
| 4.1 使用捕获文件 .....            | 53        |
| 4.1.1 保存和导出捕获文件 .....       | 54        |
| 4.1.2 合并捕获文件 .....          | 55        |
| 4.2 分析数据包 .....             | 55        |
| 4.2.1 查找数据包 .....           | 56        |
| 4.2.2 标记数据包 .....           | 57        |
| 4.2.3 打印数据包 .....           | 57        |
| 4.3 设定时间显示格式和相对参考 .....     | 58        |
| 4.3.1 时间显示格式 .....          | 58        |
| 4.3.2 数据包的相对时间参考 .....      | 59        |
| 4.4 设定捕获选项 .....            | 60        |
| 4.4.1 捕获设定 .....            | 61        |
| 4.4.2 捕获文件设定 .....          | 61        |
| 4.4.3 停止捕获选项 .....          | 62        |
| 4.4.4 显示选项 .....            | 62        |
| 4.4.5 名字解析选项 .....          | 63        |

|                             |           |
|-----------------------------|-----------|
| 4.5 使用过滤器                   | 63        |
| 4.5.1 捕获过滤器                 | 63        |
| 4.5.2 显示过滤器                 | 69        |
| 4.5.3 保存过滤器                 | 72        |
| <b>第 5 章 Wireshark 高级特性</b> | <b>75</b> |
| 5.1 网络端点和会话                 | 75        |
| 5.1.1 查看端点                  | 76        |
| 5.1.2 查看网络会话                | 77        |
| 5.1.3 使用端点和会话窗口进行问题定位       | 78        |
| 5.2 基于协议分层结构的统计数据           | 79        |
| 5.3 名字解析                    | 81        |
| 5.3.1 开启名字解析                | 81        |
| 5.3.2 名字解析的潜在弊端             | 82        |
| 5.4 协议解析                    | 82        |
| 5.4.1 更换解析器                 | 82        |
| 5.4.2 查看解析器源代码              | 85        |
| 5.5 跟踪 TCP 流                | 85        |
| 5.6 数据包长度                   | 86        |
| 5.7 图形展示                    | 88        |
| 5.7.1 查看 IO 图               | 88        |
| 5.7.2 双向时间图                 | 90        |
| 5.7.3 数据流图                  | 91        |
| 5.8 专家信息                    | 92        |
| <b>第 6 章 通用底层网络协议</b>       | <b>95</b> |
| 6.1 地址解析协议                  | 96        |
| 6.1.1 ARP 头                 | 97        |
| 6.1.2 数据包 1: ARP 请求         | 98        |
| 6.1.3 数据包 2: ARP 响应         | 99        |
| 6.1.4 无偿的 ARP               | 100       |
| 6.2 互联网协议                   | 101       |
| 6.2.1 IP 地址                 | 102       |
| 6.2.2 IPv4 头                | 103       |
| 6.2.3 存活时间                  | 104       |

|              |                        |            |
|--------------|------------------------|------------|
| 6.2.4        | IP 分片 .....            | 107        |
| 6.3          | 传输控制协议 .....           | 109        |
| 6.3.1        | TCP 头 .....            | 109        |
| 6.3.2        | TCP 端口 .....           | 110        |
| 6.3.3        | TCP 的三次握手 .....        | 113        |
| 6.3.4        | TCP 终止 .....           | 116        |
| 6.3.5        | TCP 重置 .....           | 117        |
| 6.4          | 用户数据报协议 .....          | 118        |
| 6.5          | 互联网控制消息协议 .....        | 119        |
| 6.5.1        | ICMP 头 .....           | 119        |
| 6.5.2        | ICMP 类型和消息 .....       | 120        |
| 6.5.3        | Echo 请求与响应 .....       | 120        |
| 6.5.4        | 路由跟踪 .....             | 122        |
| <b>第 7 章</b> | <b>常见高层网络协议 .....</b>  | <b>127</b> |
| 7.1          | 动态主机配置协议 DHCP .....    | 127        |
| 7.1.1        | DHCP 头结构 .....         | 128        |
| 7.1.2        | DHCP 续租过程 .....        | 129        |
| 7.1.3        | DHCP 租约内续租 .....       | 134        |
| 7.1.4        | DHCP 选项和消息类型 .....     | 134        |
| 7.2          | 域名系统 .....             | 135        |
| 7.2.1        | DNS 数据包结构 .....        | 135        |
| 7.2.2        | 一次简单的 DNS 查询过程 .....   | 136        |
| 7.2.3        | DNS 问题类型 .....         | 138        |
| 7.2.4        | DNS 递归 .....           | 139        |
| 7.2.5        | DNS 区域传送 .....         | 142        |
| 7.3          | 超文本传输协议 .....          | 145        |
| 7.3.1        | 使用 HTTP 浏览 .....       | 145        |
| 7.3.2        | 使用 HTTP 传送数据 .....     | 147        |
| 7.4          | 小结 .....               | 149        |
| <b>第 8 章</b> | <b>基础的现实世界场景 .....</b> | <b>151</b> |
| 8.1          | 数据包层面的社交网络 .....       | 152        |
| 8.1.1        | 捕获 Twitter 流量 .....    | 152        |
| 8.1.2        | 捕获 Facebook 流量 .....   | 156        |

|                                 |            |
|---------------------------------|------------|
| 8.1.3 比较 Twitter 和 Facebook 的方法 | 158        |
| 8.2 捕获 ESPN.com 流量              | 159        |
| 8.2.1 使用会话窗口                    | 159        |
| 8.2.2 使用协议分层统计窗口                | 160        |
| 8.2.3 查看 DNS 流量                 | 161        |
| 8.2.4 查看 HTTP 请求                | 162        |
| 8.3 现实世界问题                      | 163        |
| 8.3.1 无法访问 Internet: 配置问题       | 163        |
| 8.3.2 无法访问 Internet: 意外重定向      | 166        |
| 8.3.3 无法访问 Internet: 上游问题       | 169        |
| 8.3.4 打印机故障                     | 172        |
| 8.3.5 分公司之困                     | 175        |
| 8.3.6 生气的开发者                    | 179        |
| 8.4 小结                          | 184        |
| <b>第 9 章 让网络不再卡</b>             | <b>185</b> |
| 9.1 TCP 的错误恢复特性                 | 186        |
| 9.1.1 TCP 重传                    | 186        |
| 9.1.2 TCP 重复确认和快速重传             | 189        |
| 9.2 TCP 流控制                     | 194        |
| 9.2.1 调整窗口大小                    | 195        |
| 9.2.2 用零窗口通知停止数据流               | 196        |
| 9.2.3 TCP 滑动窗口实战                | 197        |
| 9.3 从 TCP 错误控制和流量控制中学到的         | 200        |
| 9.4 定位高延迟的原因                    | 201        |
| 9.4.1 正常通信                      | 202        |
| 9.4.2 慢速通信——线路延迟                | 202        |
| 9.4.3 慢速通信——客户端延迟               | 203        |
| 9.4.4 慢速通信——服务器延迟               | 204        |
| 9.4.5 延迟定位框架                    | 204        |
| 9.5 网络基线                        | 205        |
| 9.5.1 站点基线                      | 206        |
| 9.5.2 主机基线                      | 207        |
| 9.5.3 应用程序基线                    | 208        |

|                                    |            |
|------------------------------------|------------|
| 9.5.4 基线的其他注意事项 .....              | 209        |
| 9.6 小结 .....                       | 209        |
| <b>第 10 章 安全领域的数据包分析 .....</b>     | <b>211</b> |
| 10.1 网络侦察 .....                    | 212        |
| 10.1.1 SYN 扫描 .....                | 212        |
| 10.1.2 操作系统指纹术 .....               | 216        |
| 10.2 漏洞利用 .....                    | 219        |
| 10.2.1 极光行动 .....                  | 219        |
| 10.2.2 ARP 缓存中毒攻击 .....            | 225        |
| 10.2.3 远程访问特洛伊木马 .....             | 229        |
| 10.3 小结 .....                      | 236        |
| <b>第 11 章 无线网络数据包分析 .....</b>      | <b>237</b> |
| 11.1 物理因素 .....                    | 237        |
| 11.1.1 一次嗅探一个信道 .....              | 238        |
| 11.1.2 无线信号干扰 .....                | 239        |
| 11.1.3 检测和分析信号干扰 .....             | 239        |
| 11.2 无线网卡模式 .....                  | 240        |
| 11.3 在 Windows 上嗅探无线网络 .....       | 242        |
| 11.3.1 配置 AirPcap .....            | 242        |
| 11.3.2 使用 AirPcap 捕获流量 .....       | 243        |
| 11.4 在 Linux 上嗅探无线网络 .....         | 244        |
| 11.5 802.11 数据包结构 .....            | 246        |
| 11.6 在 Packet List 面板增加无线专用列 ..... | 247        |
| 11.7 无线专用过滤器 .....                 | 248        |
| 11.7.1 筛选特定 BSS ID 的流量 .....       | 249        |
| 11.7.2 筛选特定的无线数据包类型 .....          | 249        |
| 11.7.3 筛选特定频率 .....                | 250        |
| 11.8 无线网络安全 .....                  | 251        |
| 11.8.1 成功的 WEP 认证 .....            | 251        |
| 11.8.2 失败的 WEP 认证 .....            | 253        |
| 11.8.3 成功的 WPA 认证 .....            | 253        |
| 11.8.4 失败的 WPA 认证 .....            | 255        |
| 11.9 小结 .....                      | 256        |
| <b>附录 A 延伸阅读 .....</b>             | <b>257</b> |

# 第1章

## 数据包分析技术与网络基础



在计算机网络中，每天都可能生成千上万的问题，从简单的间谍软件感染，到复杂的路由器配置错误。我们永远也不可能立即解决所有问题，而只能期盼充分地准备好相关的知识和工具，从而能够快速地响应各种类型的错误。

所有的网络问题都源于数据包层次，即使是有最漂亮外表的应用程序，它们也可能是“金玉其外”但“败絮其中”，有着混乱的设计与糟糕的实现，又或是看起来是可信的，但背地里在搞些恶意的行为。为了更好地了解网络问题，我们需要进入到数据包层次。在这一层次，没有任何东西能够逃出我们的视线范围——这里不再有那些令人误解的菜单栏、用来吸引眼球的动画、以及无法让人信赖的员工。在数据包层次上，就不再有真正的秘密（加密通信除外），我们在数据包层次上做得越多，那我们就能够对网络有更好的控制，就能够更好更快地解决网络问题。这就是数据包分析的世界。

本书将与你一起进入到神奇的网络数据包世界里，你将学习如何解决网络通信速度慢的问题，识别出应用程序的性能瓶颈，甚至在真实世界的场景中追踪黑客。当你读完这本书后，你应该能够使用先进的数据包分析技术来解决自己网络中的实际问题，即便它们看起来是那么复杂与难以解决。

在这一章中，我们将开始学习一些网络通信方面的基础知识，这样你可以获得阅读和学习后续章节所需的基础知识。

## 1.1 数据包分析与数据包嗅探器

数据包分析，通常也被称为数据包嗅探或协议分析，指的是捕获和解析网络上在线传输数据的过程，通常目的是为了能更好地了解网络上正在发生的事情。数据包分析过程通常由数据包嗅探器来执行。而数据包嗅探器则是一种用来在网络媒介上捕获原始传输数据的工具。

数据包分析技术可以通过以下方法来达到目标。

- 了解网络特征。
- 查看网络上的通信主体。
- 确认谁或是哪些应用在占用网络带宽。
- 识别网络使用的高峰时间。
- 识别可能的攻击或恶意活动。
- 寻找不安全以及滥用网络资源的应用。

目前市面上有着多种类型的数据包嗅探器，包括免费的和商业的。每个软件的设计目标都会有一些差异。流行的数据包分析软件包括 `tcpdump`、`OmniPeek` 和 `Wireshark`（我们在这本书中将只使用此款软件）。`tcpdump` 是个命令行程序，而 `Wireshark` 和 `OmniPeek` 都拥有图形用户界面（GUI）。

### 1.1.1 评估数据包嗅探器

当你需要选择一款数据包嗅探器时，需要考虑的因素很多，包括以下内容。

**支持的协议：**数据包嗅探器对协议解析的支持范围各不相同，大部分通常都能解析常见的网络协议（如 IPv4 和 ICMP）、传输层协议（如 TCP 和 UDP），甚至一些应用层协议（如 DNS 和 HTTP）。然而，它们可能并不支持一些非传统

协议或新协议（如 IPv6、SMBv2、SIP 等）。在选择一款嗅探器时，需要确保它能够支持你所要用到的协议。

**用户友好性：**考虑数据包嗅探器的界面布局、安装的容易度，以及操作流程的通用性。你选择的嗅探器应该适合你的专业知识水平。如果你的数据包分析经验还很少的话，你可能需要避免选择那些命令行的嗅探器，比如 `tcpdump`。另一方面，如果你拥有丰富的经验，你可能会觉得这类命令行程序会更具有吸引力。在你逐步积累数据包分析经验时，你甚至会发现组合使用多种数据包嗅探器软件将更有助于适应特定的应用场景。

**费用：**关于数据包嗅探器最伟大的事情是有着很多能够与任何商业产品相媲美的免费工具。商业产品与其他替代品之间最显著的区别是它们的报告引擎，商业产品通常包括各种花哨的报告生成模块，而在免费软件中则通常缺乏，甚至没有该模块。

**技术支持：**即使你已经掌握了嗅探软件的基本用法，但是你还是偶尔会在遇到一些新问题时需要技术支持。在评估技术支持时，你可以寻找开发人员文档、公众论坛和邮件列表。虽然对于一些像 Wireshark 这样的免费软件可能缺乏一些开发人员文档，但使用这些应用软件的社区往往可以填补这些空白。使用者和贡献者社区会提供一些讨论区、维基、博客，来帮助你获得更多关于数据包嗅探器的使用方法。

**操作系统支持：**遗憾的是，并不是所有的数据包嗅探器都支持所有的操作系统平台。你需要选择一款嗅探器，能够支持所有你将要使用的操作系统。如果你是一位顾问，你可能需要在大多数操作系统平台上进行数据包捕获和分析，那么你就需要一款能够在大多数操作系统平台上运行的嗅探器。你还需要留意，你有时会在一台机器上捕获数据包，然后在另一台机器上分析它们。操作系统之间的差异，可能会迫使你在不同的设备上使用不同的嗅探器软件。

## 1.1.2 数据包嗅探器工作原理

数据包嗅探过程中涉及到软件和硬件之间的协作。这个过程可以分为成 3 个步骤。

**第一步：收集**，数据包嗅探器从网络线缆上收集原始二进制数据。通常情况下，通过将选定的网卡设置成混杂模式来完成抓包。在这种模式下，网卡将抓取一个网段上所有的网络通信流量，而不仅是发往它的数据包。

第二步：转换，将捕获的二进制数据转换成可读形式。高级的命令行数据包嗅探器就支持到这一步骤。到这步，网络上的数据包将以一种非常基础的解析方式进行显示，而将大部分的分析工作留给最终用户。

第三步，也是最后一步：分析，对捕获和转换后的数据进行真正的深入分析。数据包嗅探器以捕获的网络数据作为输入，识别和验证它们的协议，然后开始分析每个协议的特定属性。

## 1.2 网络通信原理

为了充分理解数据包分析技术，你必须准确掌握计算机是如何通过网络进行相互通信的。在本节中，我们将研究网络协议、开放系统互连模型（OSI 模型）、网络数据帧的基础知识，以及支持网络通信的硬件知识。

### 1.2.1 协议

现代网络是由多种运行在不同平台上的异构系统组成的。为了使它们之间能够相互通信，我们使用了一套共同的网络语言，并称之为协议。常见的网络协议包括传输控制协议（TCP）、互联网协议（IP）、地址解析协议（ARP）和动态主机配置协议（DHCP）。协议栈是一组协同工作的网络协议的逻辑组合。

理解网络协议的最佳途径之一是将它们想象成人类口头或书面语言的使用规则。每一种语言都有规则，如动词应该如何结合，人们该如何问候，甚至该如何礼貌地致谢。网络协议大多也是以同样方式进行工作的。它帮助我们定义如何路由数据包，如何发起一个连接，以及如何确认收到的数据。一个网络协议可以非常简单，也可能非常复杂，这取决于它的功能。尽管各种协议往往有着巨大的差异，但它们通常用来解决以下问题。

**发起连接：**是由客户端还是服务器发起连接？在真正通信之前必须要交换哪些信息？

**协商连接参数：**通信需要进行协议加密吗？加密密钥如何在通信双方之间进行传输？

**数据格式：**通信数据在数据包中如何排列？数据到达接收设备时以什么样的顺序进行处理？

**错误检测与校正：**当数据包花了太长的时间才到达目的地时如何处理？当

客户端暂时无法和服务器建立通信时，该如何恢复连接？

· **连接终止：**一台主机如何告知另一台主机通信已经结束？为了礼貌地终止通信，应该传送什么样的最终信息？

## 1.2.2 七层 OSI 参考模型

网络协议是基于它们在行业标准 OSI 参考模型中的职能进行分层的。OSI 模型将网络通信过程分为 7 个不同层次，如图 1-1 所示。这个分层模型使得我们更容易理解网络通信。

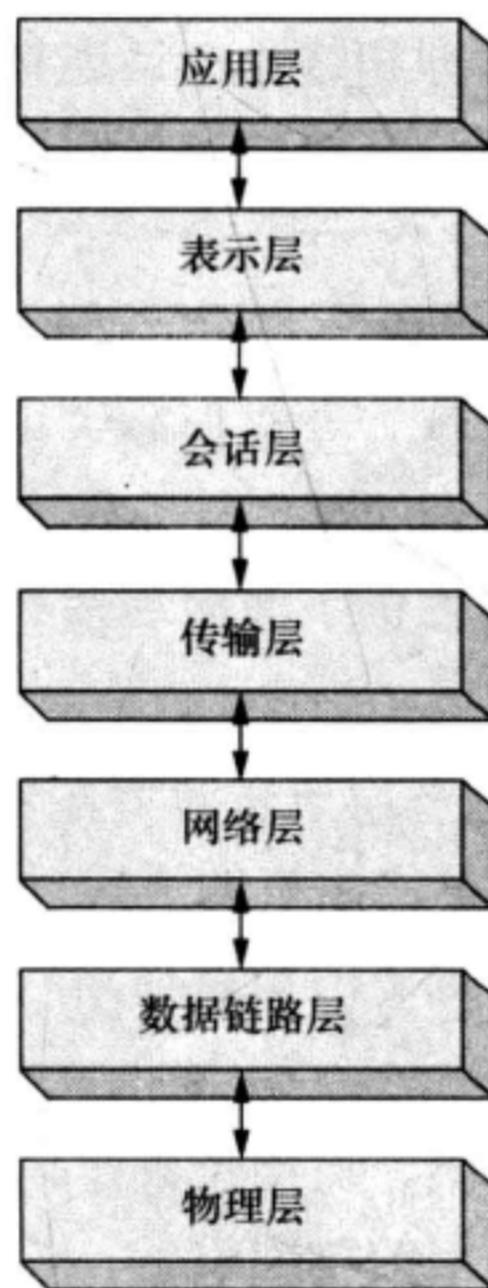


图 1-1 OSI 参考模型的七层协议视图

顶端的应用层表示用来访问网络资源的实际程序。底层则是物理层，通过它来进行实际的网络数据传播。每一层次上的网络协议共同合作，来确保通信数据在协议上层或下层中得到妥善处理。

注意

OSI 参考模型最初是在 1983 年由国际标准化组织出版，标准号为 ISO 7498。OSI 参考模型只是一个行业建议标准，协议开发并不需要严格地遵守它。OSI 参考模型也并不是现有唯一的网络模型，例如，有些人更推崇美国国防部（DoD）的网络模型，也被称为 TCP/IP 模型。

OSI 参考模型中的每层都具有特定功能，具体如下。

**应用层（第 7 层）：**OSI 参考模型的最上层，为用户访问网络资源提供一种手段。这通常是唯一一层能够由最终用户看到的协议，因为它提供的接口是最终用户所有网络活动的基础。

**表示层（第 6 层）：**这一层将接收到的数据转换成应用层可以读取的格式。在表示层完成的数据编码与解码取决于发送与接收数据的应用层协议。表示层同时进行用来保护数据的多种加密与解密操作。

**会话层（第 5 层）：**这一层管理两台计算机之间的对话（会话），负责在所有通信设备之间建立、管理和终止会话连接。会话层还负责以全双工或者半双工的方式来创建会话连接，在通信主机间礼貌地关闭连接，而不是粗暴地直接丢弃。

**传输层（第 4 层）：**传输层的主要目的是为较低层提供可靠的数据传输服务。通过流量控制、分段 / 重组、差错控制等机制，传输层确保网络数据端到端的无差错传输。因为确保可靠的数据传输极为烦琐，因此 OSI 参考模型将其作为完整的一层。传输层同时提供了面向连接和无连接的网络协议。某些防火墙和代理服务器也工作在这一层。

**网络层（第 3 层）：**这一层负责数据在物理网络中的路由转发，是最复杂的 OSI 层之一。它除了负责网络主机的逻辑寻址（例如通过一个 IP 地址）外，还处理数据包分片和一些情况下的错误检测。路由器工作在这一层上。

**数据链路层（第 2 层）：**这一层提供了通过物理网络传输数据的方法，其主要目的是提供一个寻址方案，可用于确定物理设备（例如 MAC 地址）。网桥和交换机是工作在数据链路层的物理设备。

**物理层（第 1 层）：**OSI 参考模型的底层是传输网络数据的物理媒介。这一层定义了所有使用的网络硬件设备的物理和电气特性，包括电压、集线器、网络适配器、中继器和线缆规范等。物理层建立和终止连接，并提供一种共享通信资源的方法，将数字信号转换成模拟信号传输，并反过来将接收的模拟信号转换回数字信号。

表 1-1 列出了 OSI 参考模型各个层次上的一些常见网络协议。

虽然 OSI 参考模型仅仅是一个建议标准，你还是应该将其牢记在心。当我们阅读本书时，你发现，对不同层网络协议进行交互才能解决你面对的网络问题。例如遇到路由器问题，你该很快确认这是个“第 3 层上的问题”，而应用软

件问题则被识别为“第 7 层上的问题”。

表 1-1 OSI 参考模型各个层次上的典型网络协议

| 层次    | 协议                                 |
|-------|------------------------------------|
| 应用层   | HTTP、SMTP、FTP、Telnet               |
| 表示层   | ASCII、MPEG、JPEG、MIDI               |
| 会话层   | NetBIOS、SAP、SDP、NWLink             |
| 传输层   | TCP、UDP、SPX                        |
| 网络层   | IP、IPX                             |
| 数据链路层 | Ethernet、Token Ring、FDDI、AppleTalk |

#### 注意

在讨论我们的工作时，一位同事告诉我，他曾处理过一位用户的投诉，说他不能访问网络资源，而实际原因是用户输入的密码不正确。我的同事将这个案例标成了“第 8 层的问题”，第 8 层是对用户层的一种非官方说法，通常是由那些整天工作在数据包层次上的网络工程师们使用。

那网络数据是如何流经 OSI 参考模型的各个层次呢？在网络上传输的初始数据首先在传输网络的应用层开始，沿着 OSI 参考模型的七层逐层向下，直到物理层。在物理层上，传输系统将数据发送到接收系统。接收系统从它的物理层获取传输数据，然后向上逐层处理，直到最高的应用层。

在 OSI 参考模型任意层次上由不同协议提供的服务并不是多余的。例如，如果某层上的一个网络协议提供了一种服务，那么再没有任何其他层的协议将提供与之完全相同的服务。不同层次的协议可能有目标类似的功能，但它们会以不同的方式来实现。

发送和接收计算机相同层上的网络协议是相互配合的。例如，发送系统在第 7 层的某个协议是负责对传输数据进行加密的，那么往往在接收系统的第 7 层有着相应的网络协议，负责对网络数据进行解密。

图 1-2 中连接了两个通信端图形化地说明了 OSI 参考模型。你可以看到，通信数据会从一个通信端的顶部流向底部，然后当它到达另一通信端时，将反向从底部流向顶部。

OSI 参考模型中的每一层只能和直接的上层与下层进行通信。例如，第 2 层只能从第 1 层与第 3 层发送和接收数据。

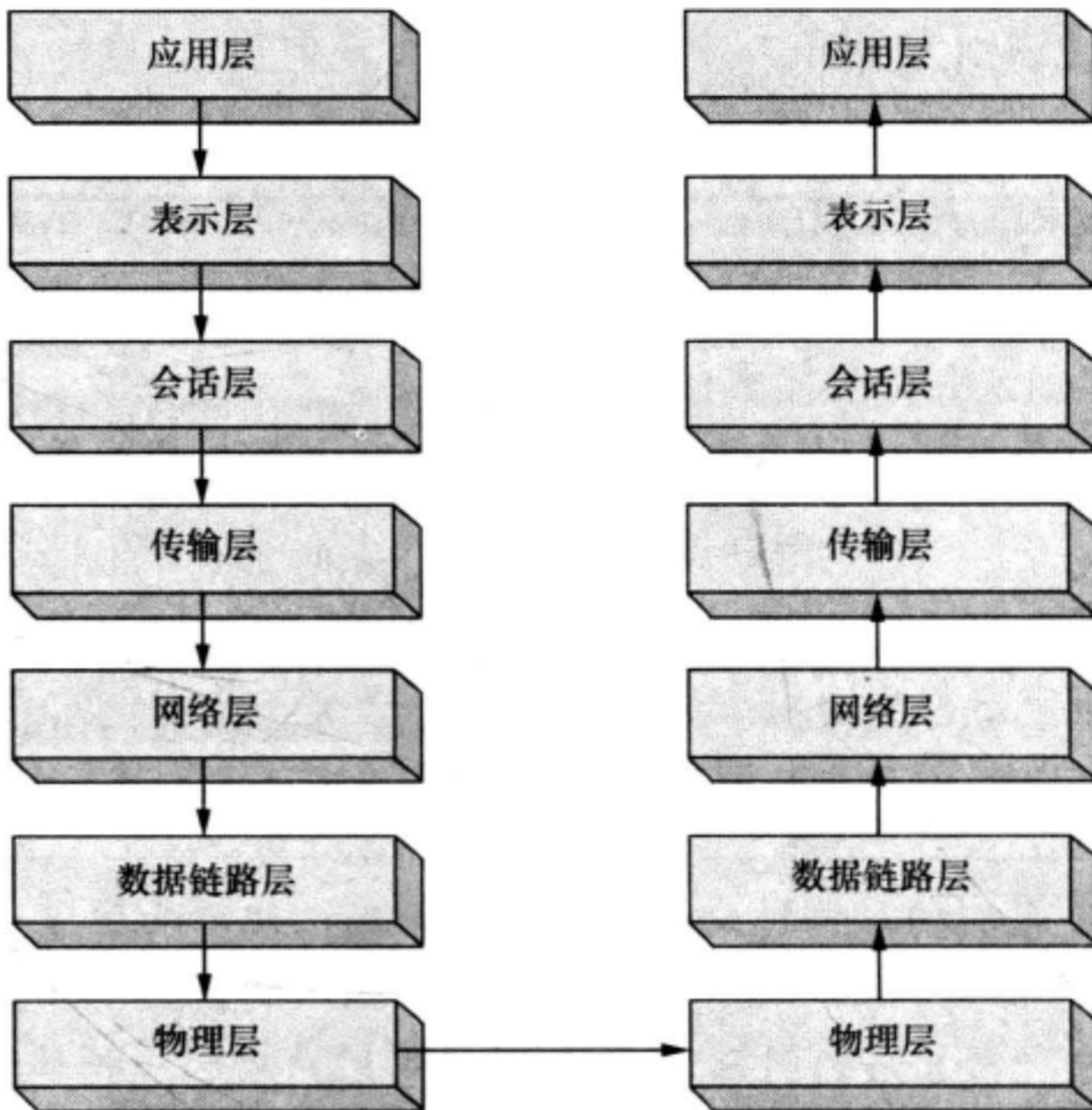


图 1-2 处于发送系统和接收系统同一层的协议

### 1.2.3 数据封装

OSI 参考模型不同层次上的协议在数据封装的帮助下进行通信传输。协议栈中的每层协议都负责在传输数据上增加一个协议头部或尾部，其中包含了使协议栈之间能够进行通信的额外信息。例如，当传输层从会话层接收数据时，它会在将数据传递到下一层之前，增加自己的头部信息数据。

数据封装过程将创建一个协议数据单元（PDU），其中包括正在发送的网络数据，以及所有增加的头部与尾部协议信息。随着网络数据沿着 OSI 参考模型向下流动，PDU 逐渐变化和增长，各层协议均将其头部或尾部信息添加进去，直到物理层时达到其最终形式，并发送给目标计算机。接收计算机收到 PDU 后，沿着 OSI 参考模型往上处理，逐层剥去协议头部和尾部，当 PDU 到达 OSI 参考模型的最上层时，将只剩下原始传输数据。

#### 注意

数据包这个术语指的是一个完整的 PDU，包括 OSI 参考模型所有层次协议的头部与尾部信息。

理解数据封装过程可能会有点困难，让我们看一个实际的例子，看数据包是如何在 OSI 参考模型中被创建、传输和接收的。作为数据包分析师，你需要

了解，我们经常会忽略掉会话层和表示层，所以它们将不会在这个例子中出现（包括本书的其余部分）。<sup>1</sup>

假设这样一个情形：我们试着在计算机上浏览 <http://www.google.com/>。在这个过程中，我们首先必须产生一个请求数据包，从客户端计算机传输到目标服务器上。这里我们假设 TCP/IP 通信会话已经被建立，图 1-3 所示为此案例中的数据封装处理过程。

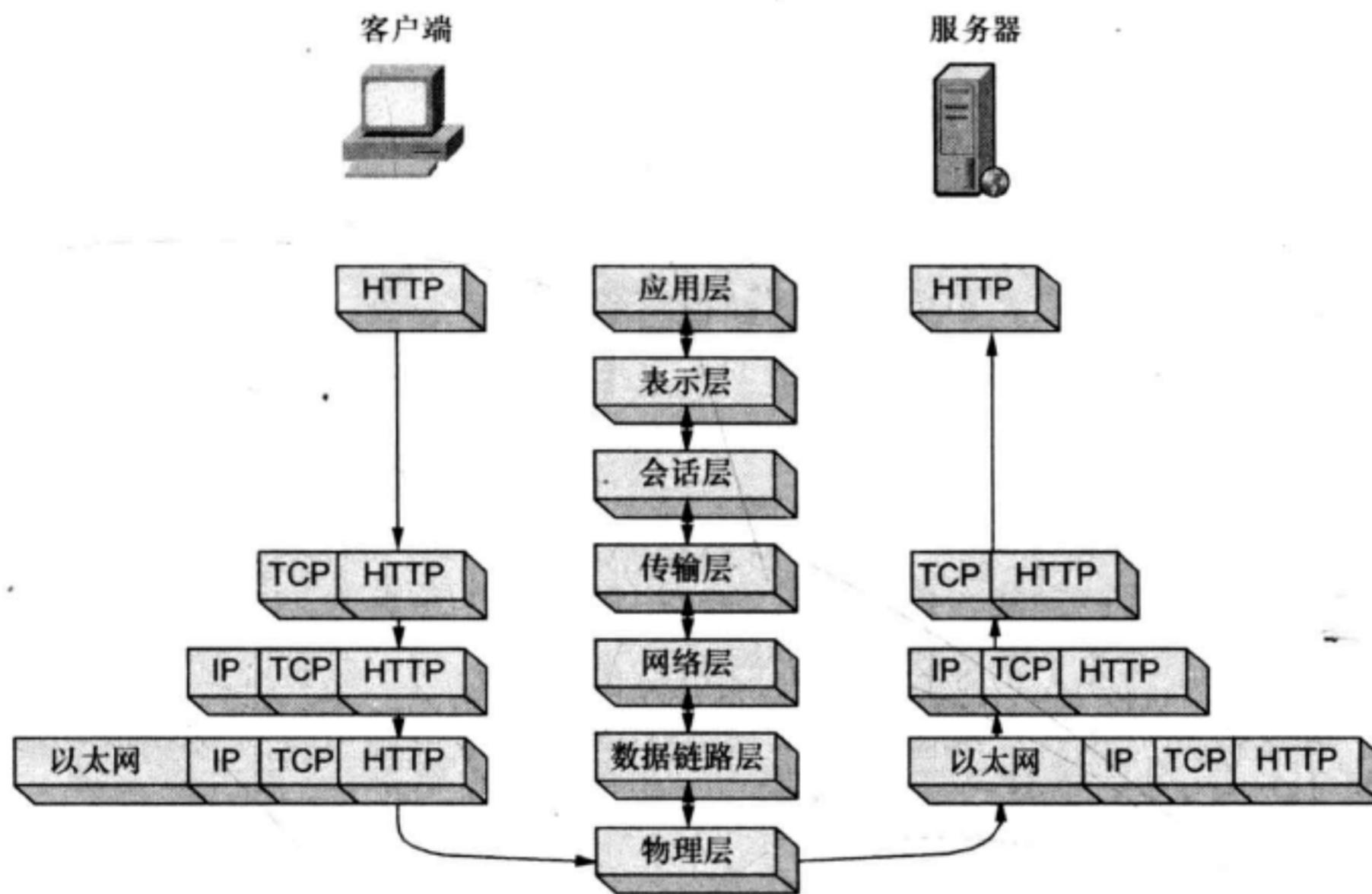


图 1-3 客户端和服务器之间数据封装过程示意图

我们从客户端计算机的应用层开始，在我们浏览一个网站时，所使用的应用层协议是 HTTP，通过此协议发出请求命令，从 google.com 下载 index.html 文件。一旦我们的应用层协议已经确定我们要完成的任务，我们现在关心的是数据包如何发送到目的地。数据包中封装的应用层数据将沿着协议栈传递给传输层。HTTP 是一个使用 TCP（或在 TCP 协议之上）的应用层协议，因此传输层中将使用 TCP 协议来确保数据包的可靠投递。所以一个包括序列号和其他数据的 TCP 协议头部将被创建，并被添加到数据包中，以确保数据包能够被正确交付。

#### 注意

我们常说一个协议在其他协议之上，是因为 OSI 参考模型的上下层设计。例如 HTTP 等应用层协议提供了一个特定的服务，并依靠 TCP 协议来保证服务的可靠交付。正如你学习到的，DNS 协议构架于 UDP 上，而 TCP 构架在 IP 之上。

<sup>1</sup> 译者注：TCP/IP 模型中并没有会话层和表示层，因此实际的 TCP/IP 协议栈中并没有单独设计会话层和表示层网络协议。

在完成这项工作之后，TCP 协议将数据包交给 IP 协议，也就是在第 3 层上负责为数据包进行逻辑寻址的协议。IP 协议创建一个包含有逻辑寻址信息的头部，并将数据包传递给数据链路层上的以太网，然后以太网物理地址会被添加并存储在以太网帧头中。现在数据包已经完全封装好，然后传递给物理层，在这里数据包变成 0、1 信号通过网络完成传输。

封装好的数据包将穿越网络线缆，最终到达 Google 的 Web 服务器。Web 服务器开始读取数据包，从下往上，这意味着它首先读取数据链路层，从中提取到所包含的物理以太网寻址信息，确定数据包是否是发往这台服务器的。一旦处理完这些信息，第 2 层头部与尾部的信息将被剥除，并进入第 3 层的信息处理过程中。

读取 IP 寻址信息的方式和第 2 层相同，目的是确认数据包被正确转发，以及数据包未进行分片处理。这些数据也同样被剥离，并交到下一层进行处理。

现在第 4 层 TCP 协议信息被读取，以确保数据包是按序到达的。然后第 4 层报头信息被剥离，留下的只有应用层数据。这些数据会被传递到 Web 服务器应用程序。为了响应客户端发过来的这个数据包，服务器应该发回一个 TCP 确认数据包，使客户端知道它的请求已经被接收，并可以等待获取 index.html 文件内容了。

所有数据包都会以这个例子中描述的过程进行创建和处理，而无论使用的是哪种协议。

但同时，请牢记并非每个网络数据包都是从应用层协议产生的，所以你会进一步看到只包含第 2 层、第 3 层或第 4 层协议信息的数据包。

## 1.2.4 网络硬件

现在是时候来看看网络硬件了，至此脏活累活都已经完成，接下来的内容都很简单了。我们将专注于几种较为常见的网络硬件：集线器、交换机和路由器。

- **集线器**

集线器一般是提供了多个 RJ-45 端口的机盒，就像图 1-4 所示的 NETGEAR 集线器。集线器从非常小的 4 端口的设备，到企业环境中安装机架设计的 48 端口机盒设备，变化很大。

因为集线器会产生很多不必要的网络流量，并仅在半双工模式下运行（不能在同一时间发送和接收数据），所以你通常不会在现代或高密度的网络中再看到它们的身影了（用交换机来代替）。然而，你应该知道集线器的工作机制，因为它们对于数据包分析技术非常重要，特别在实施我们将于第 2 章介绍的“枢纽”技术时。



图 1-4 一台典型的 4 端口以太网集线器

一台集线器无非就是工作在 OSI 参考模型物理层上的转发设备。它从一个端口接收到数据包，然后将数据包传输（中继）到设备的其他每个端口上。例如，如果一台计算机连接到一个 4 端口集线器的 1 号端口上，需要发送数据到连接在 2 号端口的计算机，那么集线器将会把数据发送给端口 1、2、3、4。连接到 3 号端口与 4 号端口上的客户端计算机通过检查以太网帧头字段中的目标媒体访问控制（MAC）地址，判断出这些数据包并不是给它们的，便丢弃这些数据包。图 1-5 所示为从计算机 A 发送数据到计算机 B 的例子，当计算机 A 发送出数据时，所有连接到集线器的计算机都将接收到数据，但只有计算机 B 会实际接受数据，而其他计算机则将丢弃它。

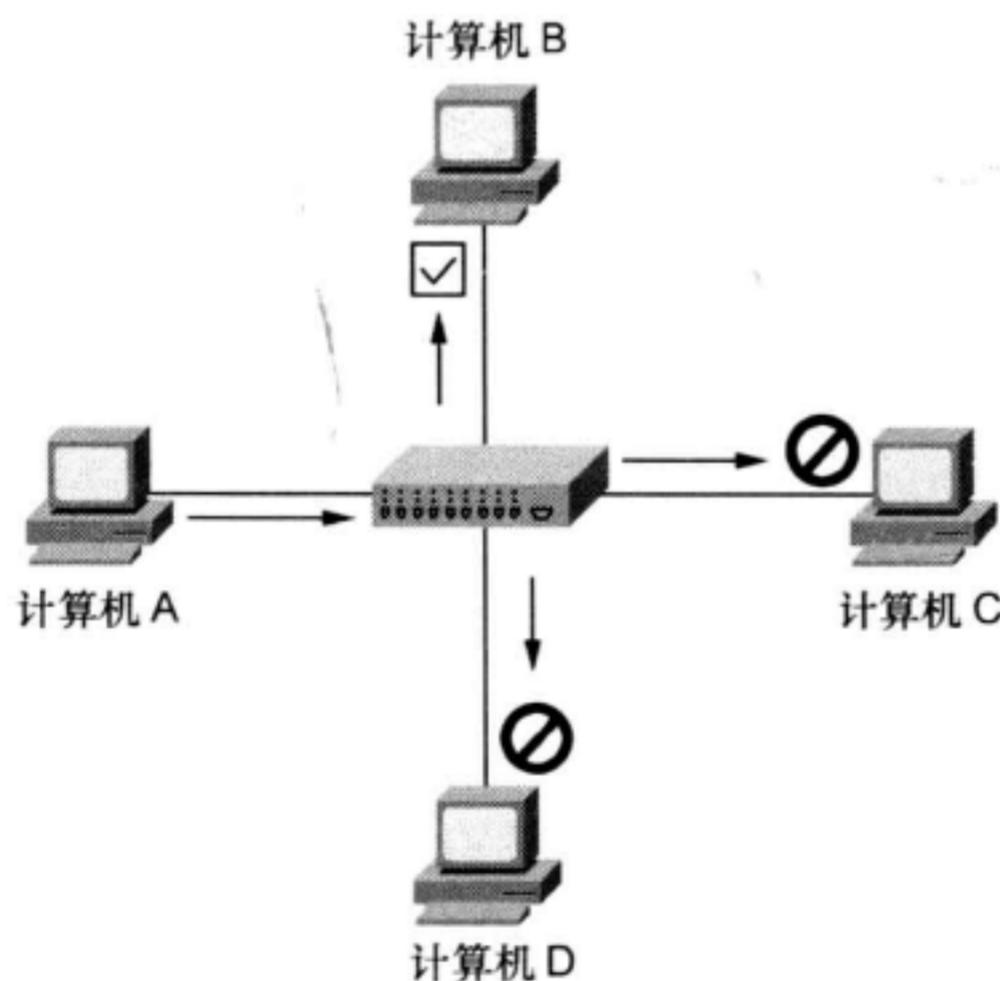


图 1-5 计算机 A 通过集线器传输数据到计算机 B 的通信流

做一个比喻，假设你发送一封主题为“所有的营销人员请注意”的电子邮件给贵公司所有雇员，而不只是那些在营销部门工作的人。市场营销部门的员工会知道这封邮件是给他们的，他们很可能会打开它，而其他员工将看到这封邮件并不是给他们的，则很可能会选择丢弃。你可以看到这会导致很多不必要的通信和时间浪费，然而这正是集线器的工作原理。在高密度的实际网络中，集线器最好的替代产品是交换机，它们是支持全双工的设备，可以同步地发送和接收数据。

- 交换机

与集线器相同，交换机也是用来中继数据包的。但与集线器不同的是，交换机并不是将数据广播到每一个端口，而是将数据发送到目的计算机所连接的端口上。如同你在图 1-6 中看到的那样，交换机的外表与集线器没什么两样。

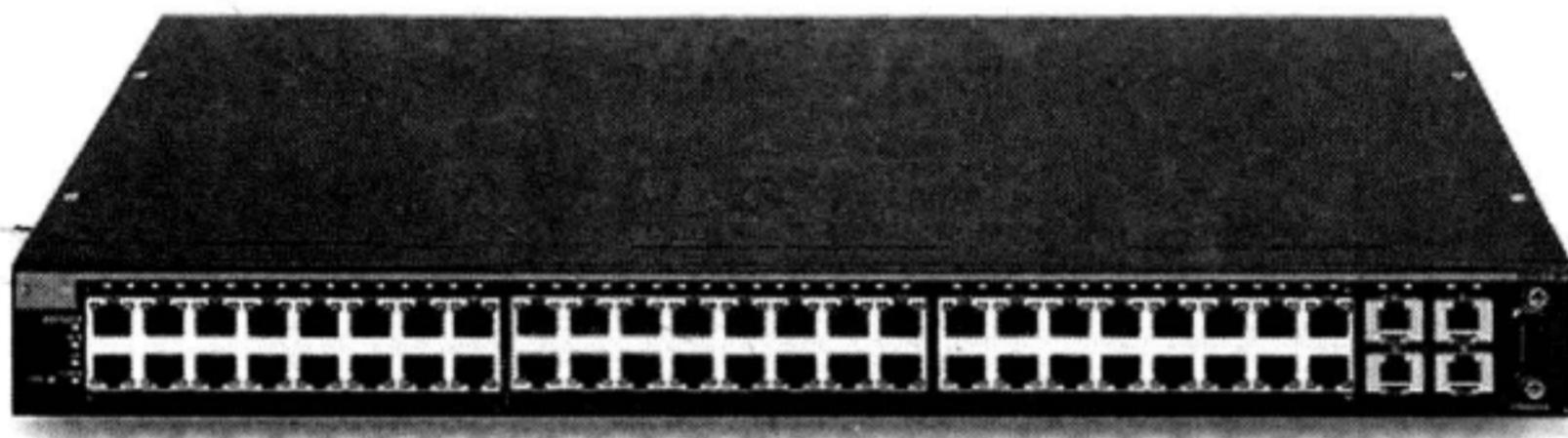


图 1-6 一个机架式 24 端口以太网交换机

市场上几个大牌公司的交换机，比如思科品牌的，能够通过专业化的供应商特定软件或 Web 接口进行远程管理。这些交换机通常被称为管理型交换机。管理型交换机提供了多种在网络管理中非常有用的功能特性，包括启用或禁用特定端口、查看端口细节参数、远程修改配置、远程重启等。

交换机在涉及处理传输数据包时，还提供一些先进的功能。为了能够直接与一些特定设备进行通信，交换机必须能够通过 MAC 地址来唯一标识设备，这意味着它们必须工作在 OSI 参考模型的数据链路层上。

交换机将每个连接设备的第 2 层地址都存储在一个 CAM (Content Addressable Memory 即内容寻址寄存器) 表中，CAM 表充当着一种类似交通警察的角色。当一个数据包被传输时，交换机读取数据包中的第 2 层协议头部信息，并使用 CAM 表作为参考，决定往哪个或哪些端口发送数据包。交换机仅仅将数据包发送到特定端口上，从而大大降低了网络流量。

图 1-7 说明了流量经过交换机进行传输的过程。在这个图示中，计算机 A 发送数据到唯一的目地：计算机 B，虽然同一时间网络上可能会有很多会话，但信息将会直接通过交换机向目地接收者进行传输，而不会被传递到与交换机相连的所有计算机。

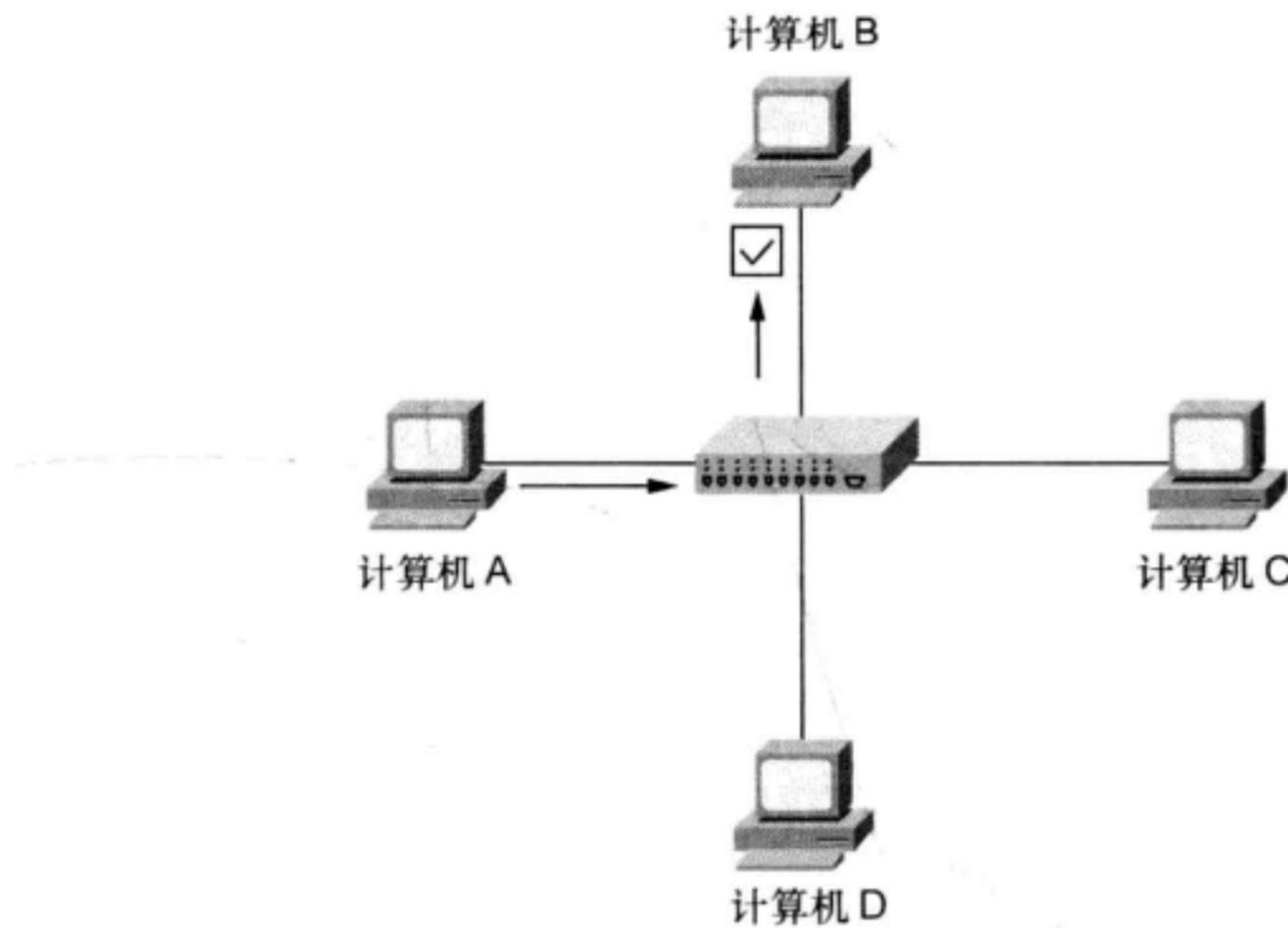


图 1-7 当计算机 A 通过交换机传输数据到计算机 B 时的通信流示意图

#### • 路由器

路由器是一种较交换机或集线器具有更高层次功能的先进网络设备。一个路由器可以有许多种不同的形状和外形，但大多数路由器在前面板上会有几个 LED 指示灯，在背板上会有一些网络端口，个数则取决于网络的大小。图 1-8 所示为一款路由器的示例。

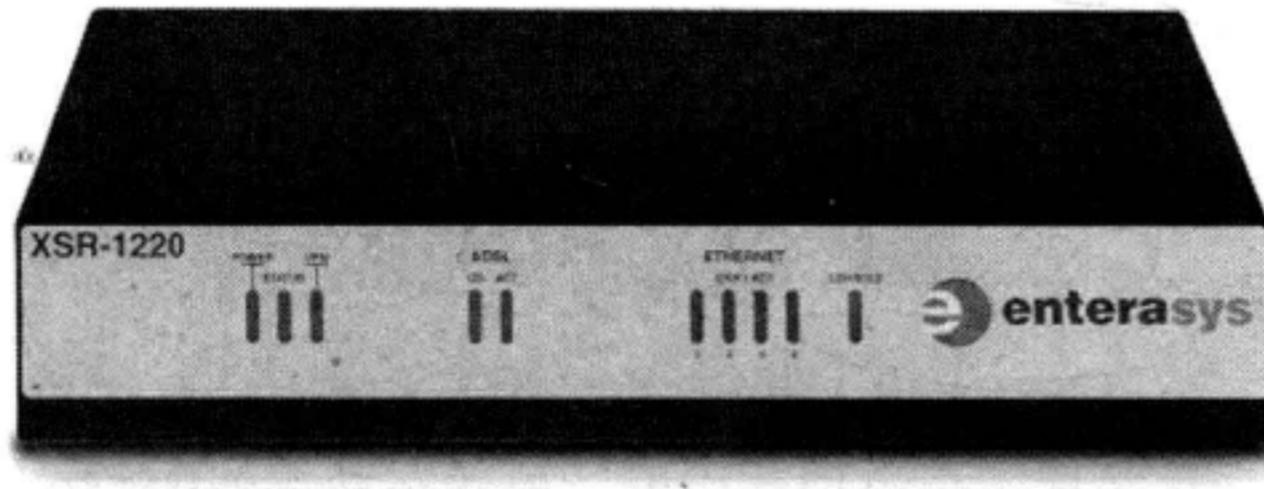


图 1-8 一款低端的 Cisco 路由器，适合在一个中小型网络使用

路由器工作在 OSI 参考模型的第 3 层，它负责在两个或多个网络之间转发数据包。路由器在网络间引导数据包流向的这一过程被称为路由。几种不同类

型的路由协议定义了不同目的的数据包如何被路由到其他网络。路由器通常使用第3层地址（如IP地址）来唯一标识网络上的设备。

为了更清楚地解释路由的概念，我们以一个拥有几条街道的街区进行类比。假设有一些房子，它们都有自己的地址，就好比网络上的计算机一样，而每条街道就好比网段，如图1-9所示。从你所在街道上的某个房子，你可以很容易地与同一街道中居住的邻居进行沟通交流，这类似于交换机的操作，能够允许在同一网段中的所有计算机进行相互通信。然而，与其他街道上居住的邻居进行沟通交流，就像是与不同网段中的计算机进行通信。

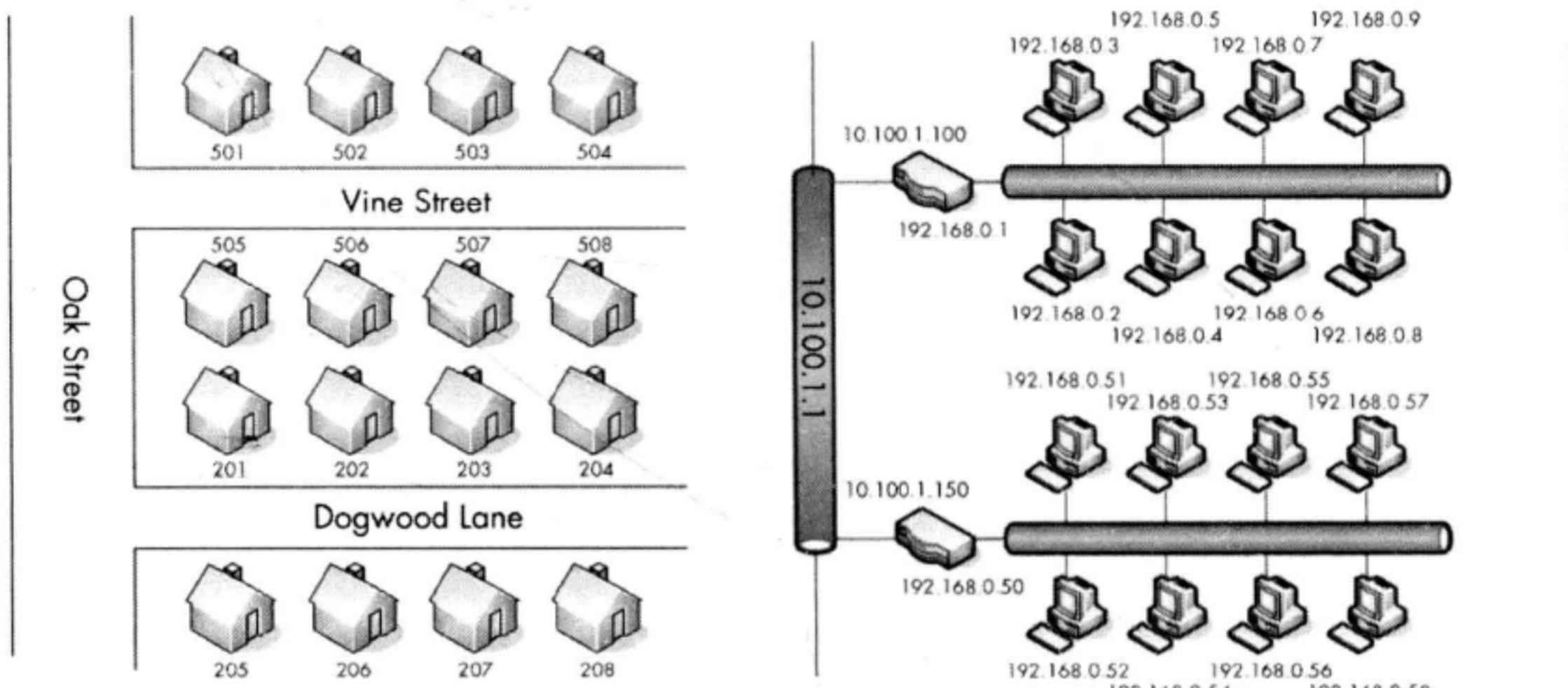


图1-9 一个路由网络与邻街区的类比

参照图1-9，假设你住在Vine Street 503号，需要到Dogwood Lane 202号。如果想要过去，你必须先到Oak Steer上，然后再到达Dogwood Lane。现在请对应到跨越网段的场景中，如果在192.168.0.3地址的设备需要和192.168.0.54地址的设备进行通信，它必须经由路由器到10.100.1.1网络上，然后再经过连接目的网段的路由器才可以到达目标网段上。

网络上路由器的数量与大小通常取决于网络的规模与功能。个人和家庭办公网络可能只需要一个小型路由器，放置在网络的中心。而大型企业网络则可能有几个路由器分布在不同的部门，都连接到一个大型的中央路由器或三层交换机上（具有内置功能，可以充当一台路由器的先进型交换机）：

当你开始查看越来越多的网络图时，你会更加了解网络数据流是如何流经这些不同类型的网络设备节点，图1-10所示为路由网络中一个非常常见的

布局形式。在这个例子中，两个单独的网络通过一个路由器进行连接。如果网络 A 上的计算机希望与网络 B 上的计算机进行通信，传输数据将必须通过路由器。

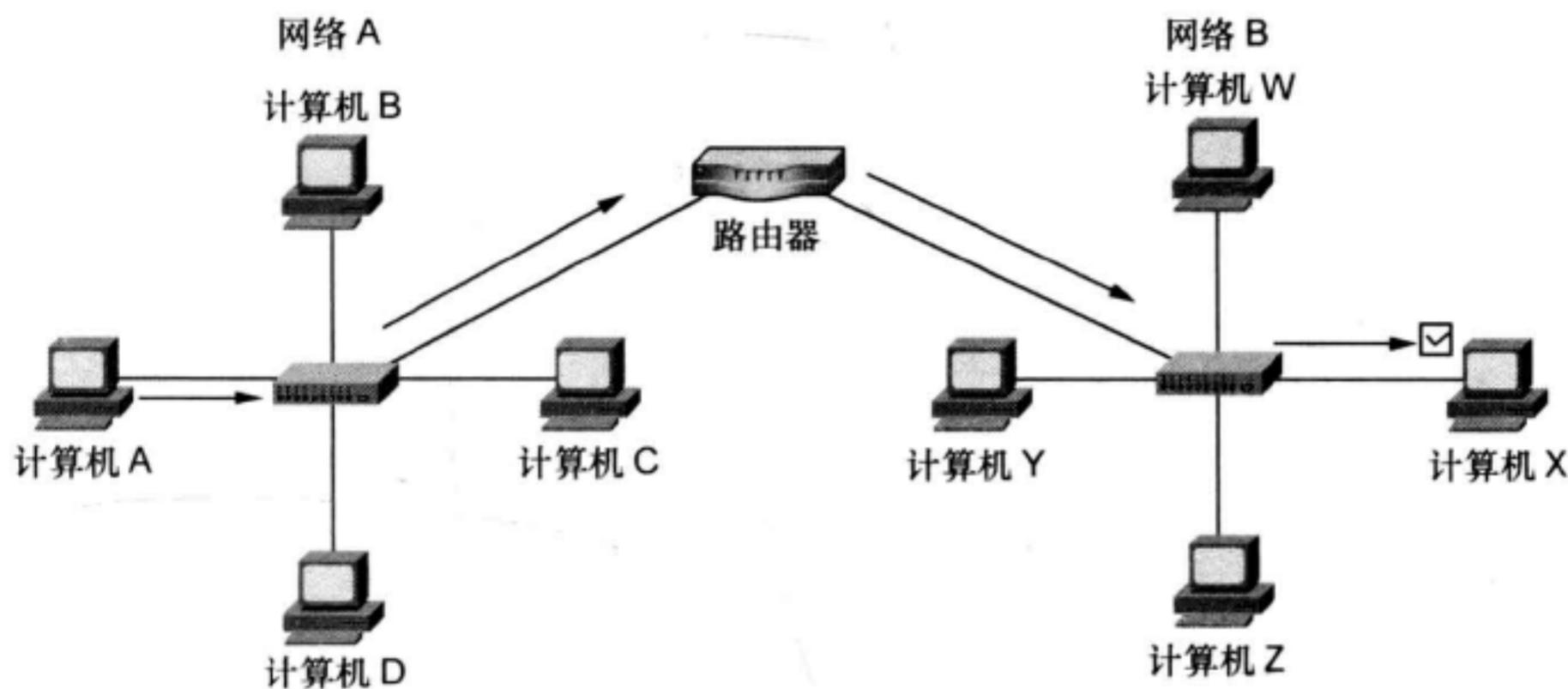


图 1-10 计算机 A 通过路由器将数据传送到计算机 X 的通信流示意图

## 1.3 流量分类

网络流量可以分为三大类：广播、多播和单播。每个分类都具有不同特点，决定着这一类的数据包该如何通过网络硬件进行处理。

### 1.3.1 广播流量

广播数据包会被发送到一个网段上的所有端口，而无论这些端口连接在集线器还是交换机上。但并非所有的广播流量都是以相同方式构建的，而是包括第 2 层广播流量和第 3 层广播流量两种主要形式。例如，在第 2 层，MAC 地址 FF:FF:FF:FF:FF 是保留的广播地址，任何发送到这一地址上的流量将会被广播到整个网段。第 3 层也有一些特定的广播地址。

在一个 IP 网络范围内最大的 IP 地址是被保留作为广播地址使用的。例如，在一个配置了 192.168.0.XXX 的 IP 范围，以及子网掩码是 255.255.255.0 的地址网络中，广播 IP 地址是 192.168.0.255。

在通过多个集线器或交换机连接多种媒介的大型网络中，广播数据包将被一直从一个交换机被中继到另一交换机上，从而传输到网络连接的所有网段上。广播数据包能够到达的区域被称为“广播域”，也就是任意计算机可以不用经由

路由器即可和其他计算机进行直接传输的网段范围。图 1-11 显示了一个小型网络上存在两个广播域的例子。因为每个广播域会一直延伸直到路由器，而广播数据包只能在它特定的广播域中传输。

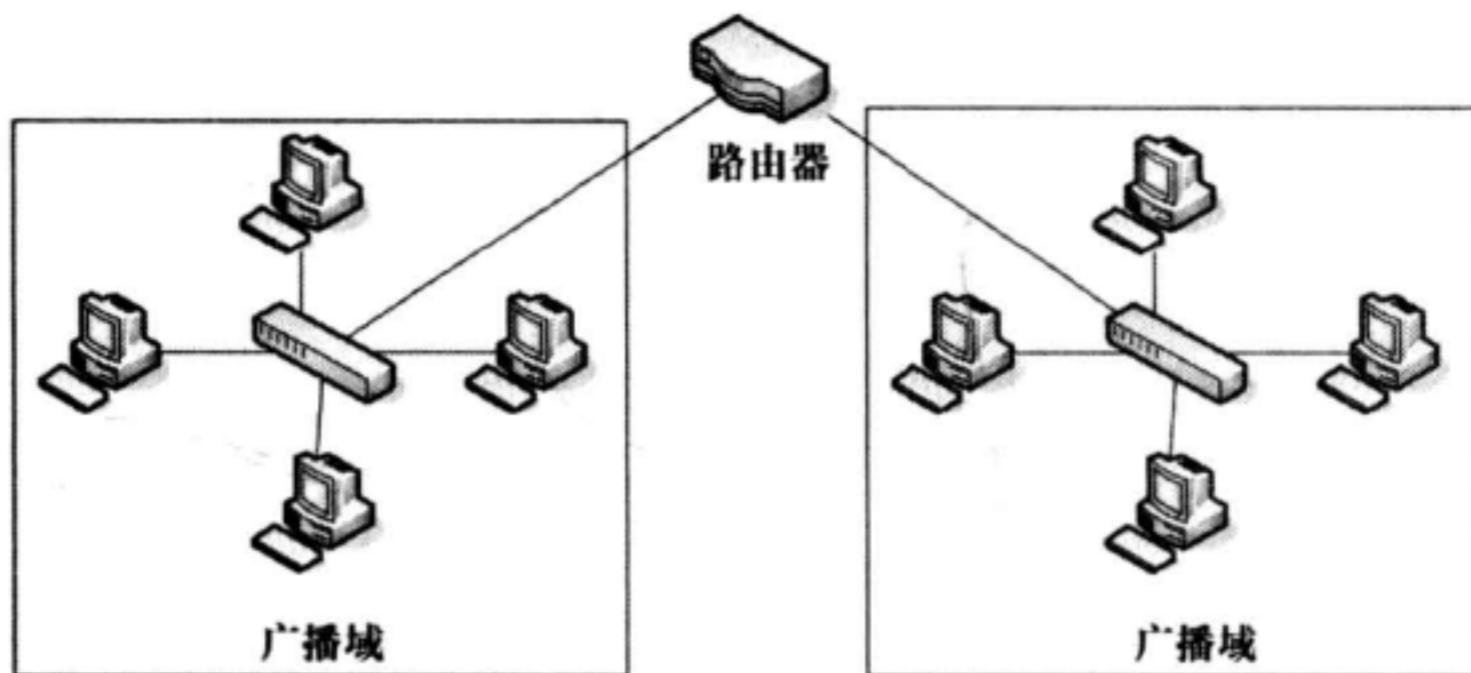


图 1-11 一个广播域一直延伸到路由器后面的网段

我们前面的类比也能很好地说明广播域是如何工作的。你可以将一个广播域想象成一条街道。如果你站在你家门口叫喊，只有街道上的人才能够听到你的声音。而如果你想与不同街道上的人说话，那么你需要找到一种与他进行直接交流的方式，而不是在你的家门口大喊大叫（广播）。

### 1.3.2 多播流量

多播是一种将单一来源数据包同时传输给多个目标的通信方式。多播的目的是为了简化这个过程，并使用尽可能少的网络带宽。多播流量通过避免数据包的大量复制来达到优化效果，而处置多播流量的方式则高度依赖于不同网络协议的实现细节。

实施多播的主要方法是通过一种将数据包接收者加入多播组的编址方案，这也是 IP 多播的工作原理。这种编址方案确保数据包不会被传送到未预期的目的地。事实上 IP 协议将一整段的地址都赋予了多播，如果你在网络上看到在 224.0.0.0 到 239.255.255.255 IP 范围内的地址，它很有可能就是多播流量。

### 1.3.3 单播流量

单播数据包会从一台计算机直接传输到另一台计算机。单播机制的具体实现方式取决于使用的协议。例如，一台设备希望与一个 Web 服务器进行通信，

这便是一个端到端的连接，所以通信过程将由客户端设备发送数据包到这台 Web 服务器开始。这种类型的通信就是单播流量的典型例子。

## 1.4 小结

本章涵盖了你学习数据包分析技术所必须掌握的基础知识。在你开始解决网络故障问题之前，你必须明白网络通信到底是怎么回事。在下一章中，我们将基于这些概念，来讨论更高级的网络通信准则。



# 第2章

## 监听网络线路



进行高效的数据包分析的一个关键决策是在哪里放置数据包嗅探器，以恰当地捕捉网络数据。数据包分析师通常把这个过程称为监听网络线路。简而言之，这是将数据包嗅探器安置在网络上恰当物理位置的过程。

然而遗憾的是，嗅探数据包并不像是将一台笔记本电脑连入网络中那么的简单。事实上，在有些时候，在网络布线系统上放置一个数据包嗅探器，要比实际分析数据包更难一些。

安置嗅探器的挑战是要考虑到种类繁多的用来连接网络的硬件设备。图 2-1 为一种典型的情况。由于网络上的 3 种主要设备（集线器、交换机、路由器）对网络流量的处理方式都不相同，因此你必须非常清楚你所分析的网络使用的是哪些硬件设备。

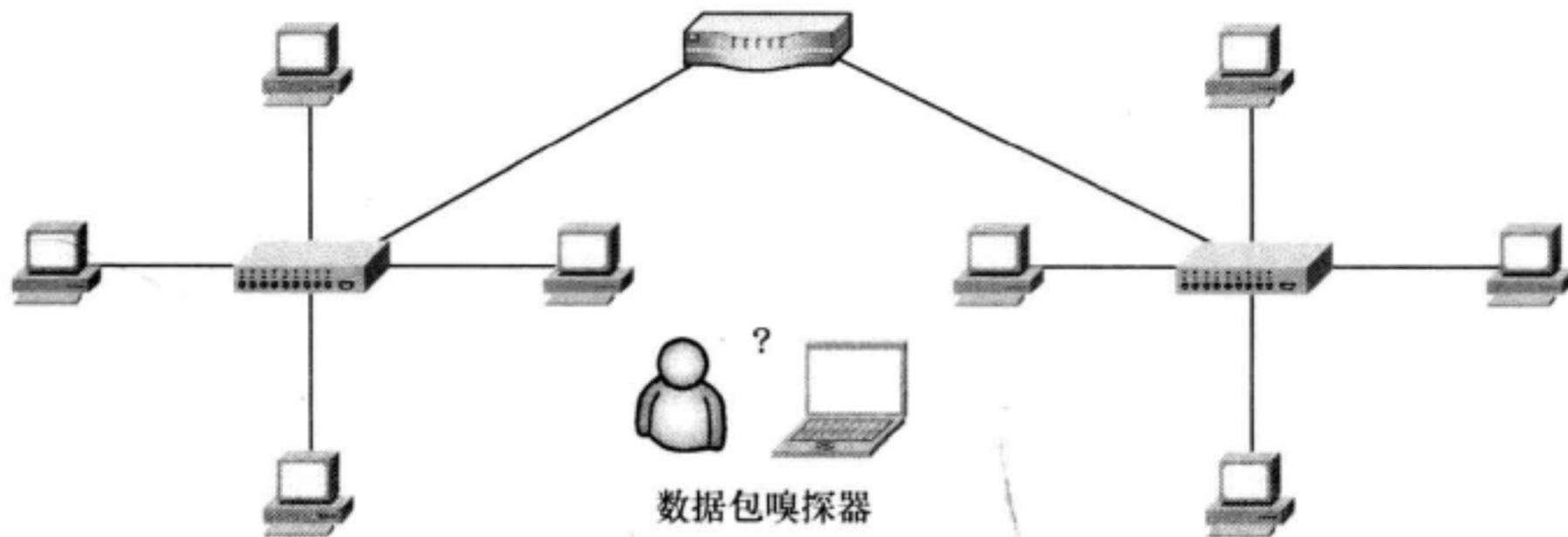


图 2-1 将嗅探器安置在你的网络上，有时是你面对的最大挑战

本章的目标是帮助你理解如何在各种不同网络拓扑结构中安置数据包嗅探器。首先，让我们来看看，我们实际上是如何看到网络线路上所有传递的数据包的。

## 2.1 混杂模式

你需要一个支持混杂模式驱动的网卡，才可能在网络上嗅探数据包。混杂模式是什么样的模式呢？实际上它是一种允许网卡能够查看到所有流经网络线路数据包的驱动模式。

正如你在第 1 章了解到的那样，在网络上有一类广播流量，因此对于客户端来说，接收到并非以它们的地址作为目标的数据包是非常常见的。用来将给定 IP 地址解析成对应 MAC 地址的 ARP 协议，在任何网络上都是一个关键组成部分，且是一个很好的例子，能够说明有些网络流量并非是发送到指定的目标地址。为了找到对应的 MAC 地址，ARP 协议会发出一个广播包并发送到广播域中的每个设备，然后期望正确的客户端做出回应。

一个广播域（也就是一个网络段，其中任何一台计算机都可以无需经过路由器，直接传送数据到另一台计算机）是由几台电脑所组成的，但广播域中仅仅只有一个客户端应该对传输的 ARP 广播请求包感兴趣。一旦网络上的每台电脑都处理和回应 ARP 广播包，那网络的性能将变得非常的糟糕。

因此，其他网络设备上的网卡驱动会识别出这个数据包对于它们来说没有任何用处，于是选择将数据包丢弃，而不是传递给 CPU 进行处理。将目标不是这台接收主机的数据包进行丢弃可以显著地提高网络处理性能，但这对数据包分析师来说并不是个好消息。作为分析师，我们通常需要看到线路上传输的每

一个数据包，这样我们才不用担心会丢失掉任何关键的信息。

我们可以使用网卡的混杂模式来确保能够捕获所有的网络流量。一旦在混杂模式下工作，网卡将会把每一个它所看到的数据包都传递给主机的处理器，而无论数据包的目的地址是什么。一旦数据包到达 CPU 之后，它就可以被一个数据包嗅探软件捕获并进行分析。

现在的网卡一般都支持混杂模式，Wireshark 软件包中也包含了 libpcap / WinPcap 驱动，这让你能够很方便地在 Wireshark 软件界面上就将网卡直接切换到混杂模式上（我们将在第 3 章里介绍更多的 libpcap / WinPcap 的内容）。

为了能够学习本书中的数据包分析技术，你必须要有一个支持使用混杂模式的网卡与操作系统。只有在你只想看到发往你运行嗅探软件主机 MAC 地址的网络数据包的时候，你才不需要以混杂模式来进行嗅探。

---

#### 注意

在大多数操作系统（包括 Windows）上，要想使用一个混杂模式的网卡，你就必须要提升用户权限到管理员级别。如果你不能在一个系统上合法地获得这些权限，那你就不应该在这台系统上对所在网络进行任何方式的数据包嗅探。

---

## 2.2 在集线器连接的网络中进行嗅探

在使用集线器连接的网络中进行嗅探，对于任何数据包分析师来说，都是一个梦想。正如你在第 1 章中了解到的那样，流经集线器的所有网络数据包都会被发送到每一个集线器连接的端口。因此，要想分析一台连接到集线器上的电脑的网络通信，你所需要做的所有事情就是将数据包嗅探器连接到集线器的任意一个空闲端口上。你就能看到所有从那台电脑流入流出的网络通信，以及其他接入集线器的任意电脑之间的通信。

如图 2-2 所示，当你的嗅探器连接到一个集线器连接的网络后，你对本地网络的可视范围是不受限制的。

---

#### 注意

可视范围，这个术语将在整本书中很多图示中显示，表示你在数据包嗅探器中能够看到通信流量的主机范围。

---

令人遗憾的消息是，集线器网络已经是非常罕见的了，因为它们曾经给网络管理员们带来了很大的困扰，已经基本被淘汰了。因为在集线器网络中，在

任一时刻里，只有一个设备可以通信。因此，通过集线器连接的设备必须与其他设备进行竞争，才能取得带宽进行通信，当两个或多个设备同时通信时，数据包就会产生冲突碰撞，如图 2-3 所示。结果可能是丢包，然后通信设备需要承担重新传输数据包带来的性能损失，而这又增加了网络拥塞和碰撞。当通信流量水平和碰撞概率增加之后，设备需要传输每个数据包 3 次甚至 4 次，这大大降低了网络性能。因此很容易理解为什么现在各种规模的网络都已经转而使用交换机了。

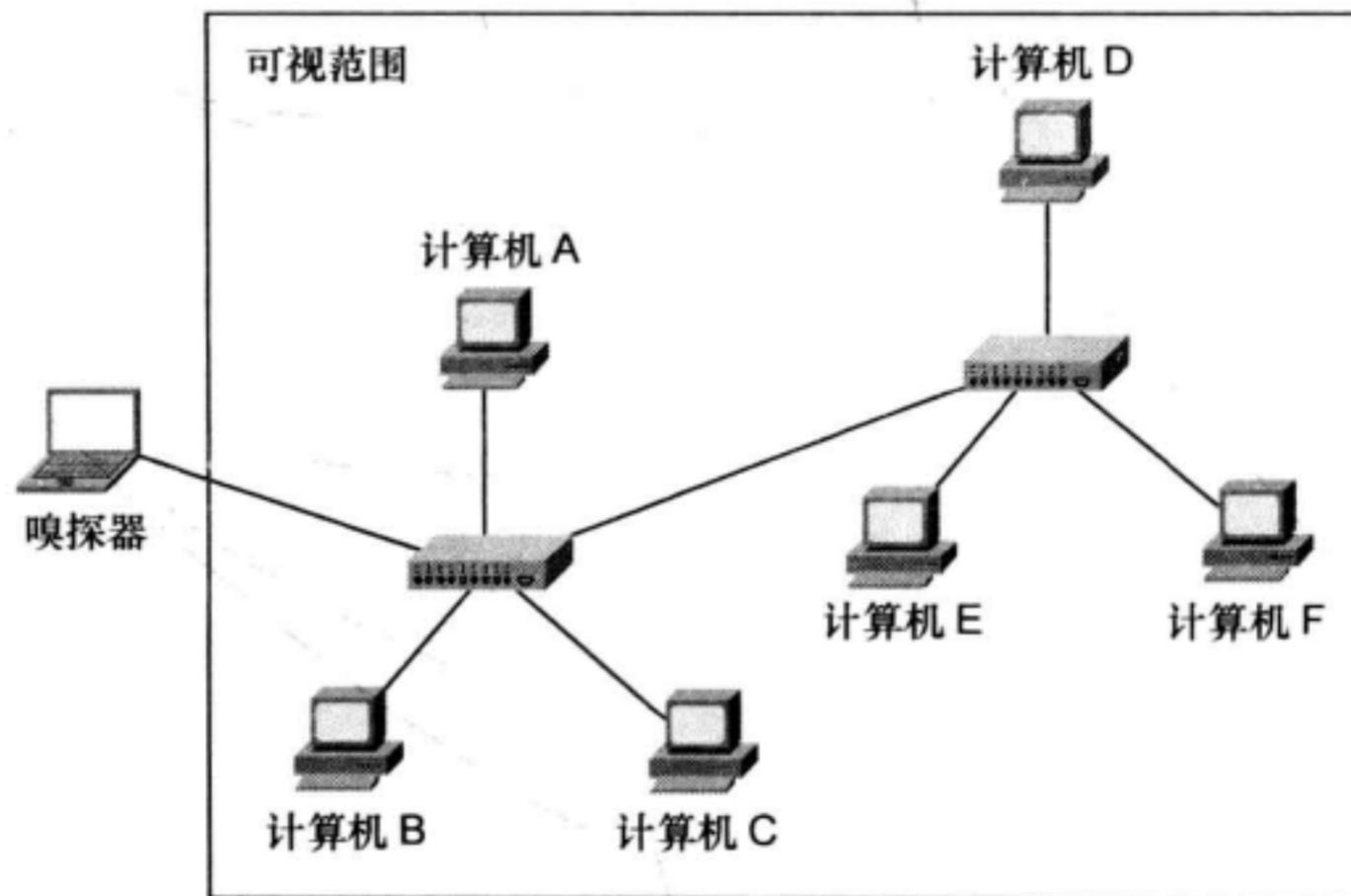


图 2-2 在集线器网络中嗅探将提供一个不受限制的可视范围

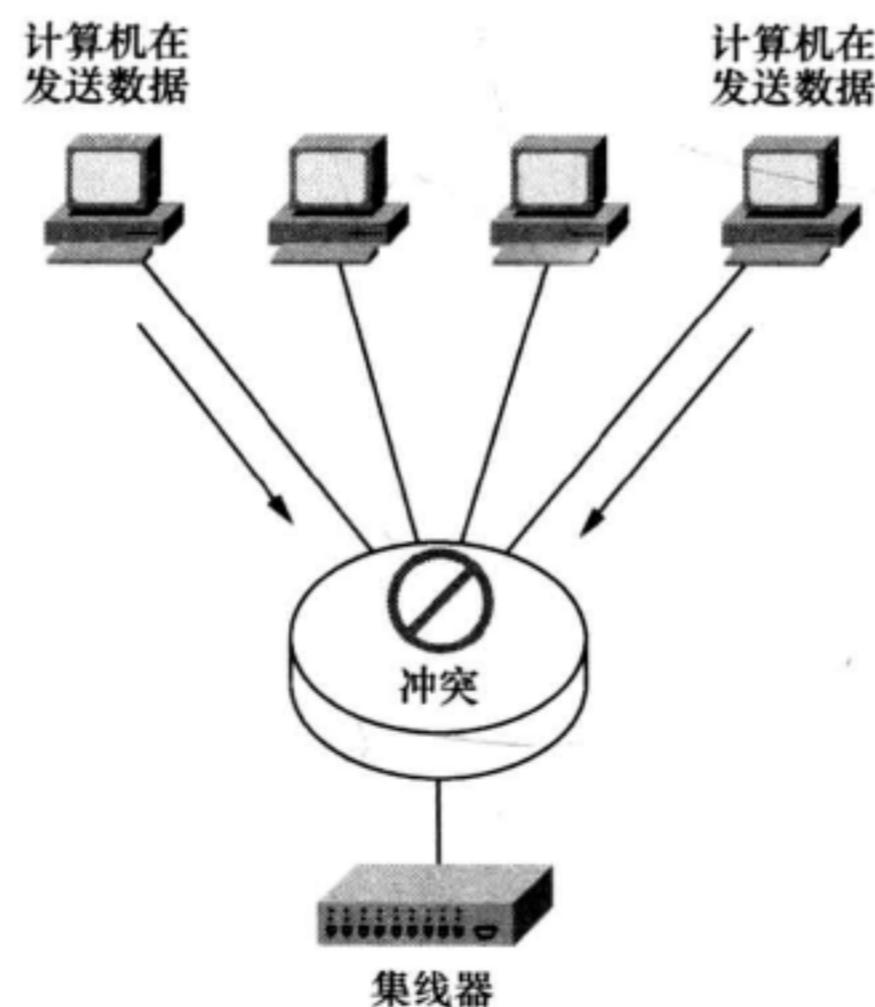


图 2-3 当集线器网络上两个设备在同一时间通信时产生的碰撞

## 2.3 在交换式网络中进行嗅探

正如第 1 章中所讨论的，交换机是在现在网络环境中最常见的连接设备类型。它们为通过广播、单播与多播方式传输数据提供了高效的方法。额外的，一些交换机还允许全双工通讯，也就是说，设备可以同时发送和接收数据。

而这对数据包分析师来说是不幸的，交换机给数据包嗅探带来了一些复杂因素。当你将嗅探器连接到交换机上的一个端口时，你将只能看到广播数据包，及由你自己电脑传输与接收的数据包，如图 2-4 所示。

在一个交换式网络中从一个目标设备捕获网络流量的基本方法有如下 4 种：端口镜像、集线器接出（hubbing out）、使用网络分流器、ARP 欺编攻击。

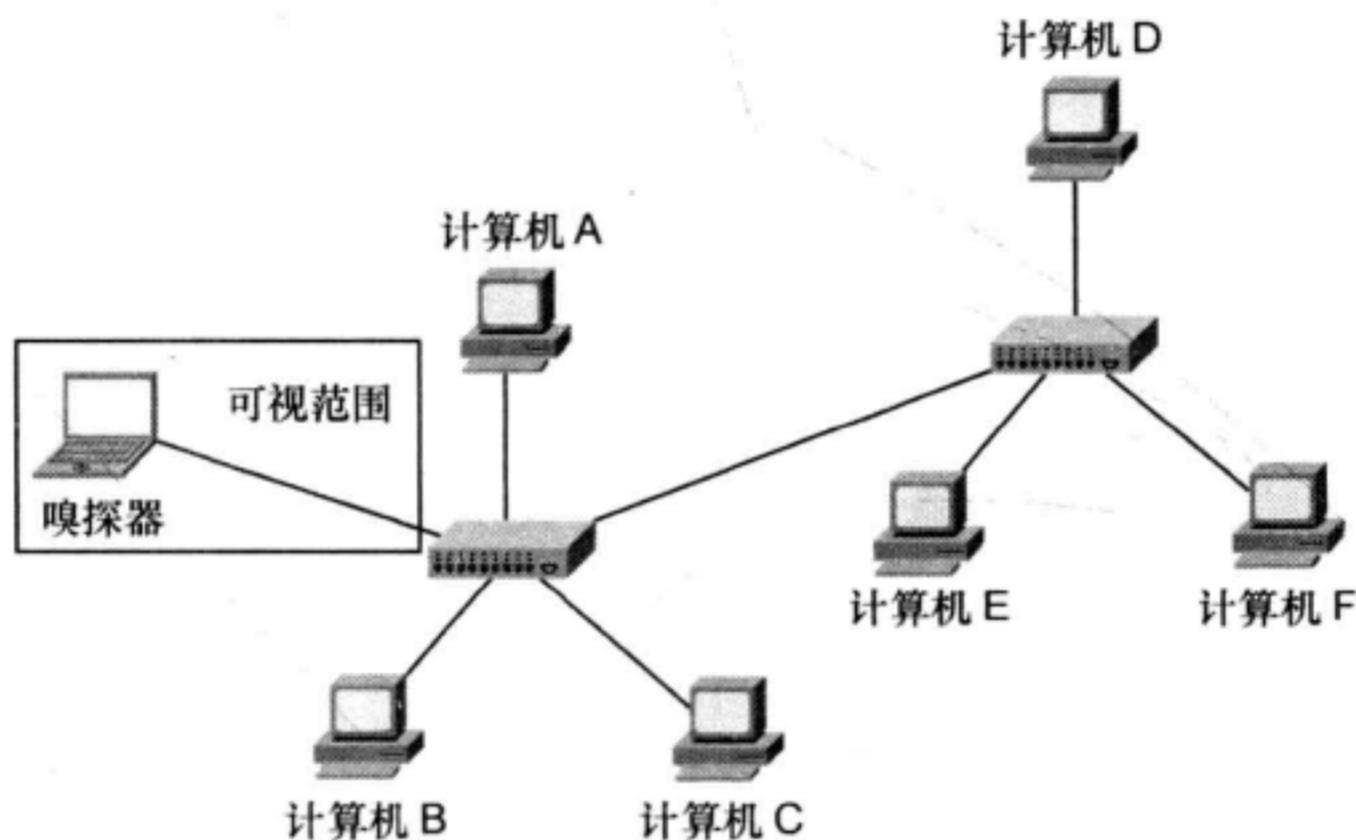


图 2-4 交换式网路上的可视范围仅限于你所接入的端口

### 2.3.1 端口镜像

端口镜像，也许是在交换式网络中捕获一个目标设备所有的网络通信最简单的方法。为了使用端口镜像，你必须能够通过命令行或 Web 管理界面来访问目标设备所连接的交换机。此外，这个交换机还必须支持端口镜像的功能，以及有一个空闲的端口，让你可以插入你的嗅探器。

要启用端口镜像，你需要发出一个命令，来强制交换机将一个端口上的所有通信都镜像到另一端口上。例如，为了捕获交换机 3 号端口连接的设备发出的所有流量，你只需要简单地将你的嗅探分析器接入 4 号端口，然后将 3 号端

口镜像复制到 4 号端口，这就可以让你看到你的目标设备传输与接收的所有网络流量。图 2-5 显示了端口镜像的原理。

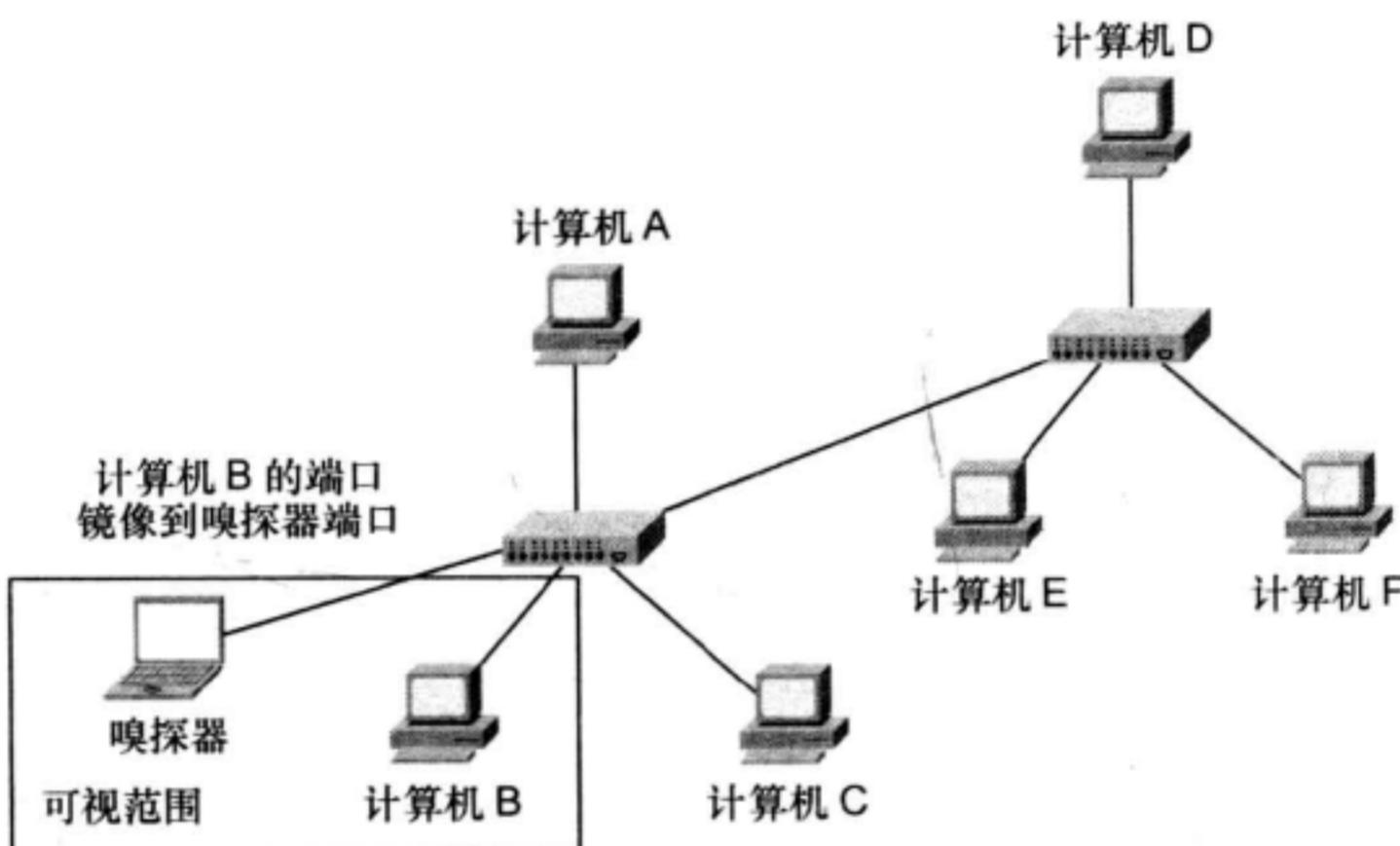


图 2-5 端口镜像可以让你在交换式网络上扩大可视范围

设置端口镜像的具体方法取决于不同的交换机制造商。对于大多数的交换机，你需要登录到命令行界面，然后输入端口镜像命令。你可以在表 2-1 中找到一些通用的端口镜像命令。

#### 注意

某些交换机提供基于 Web 的图形用户管理界面，并提供端口镜像作为一个选项，但这种配置方式不像命令行那么普遍和标准。但是，如果你的交换机提供了一种通过图形化界面可以高效配置端口镜像的方法，那么你也可以使用。

在进行端口镜像时，需要留意你所镜像端口的流量负载。有些交换机厂商允许你将多个端口的流量镜像到一个单独端口上，这在分析一个交换机上两个或多个设备的网络通信时可能是非常有用的。

然而，我们使用一些简单的算术来考虑会发生什么事情。比如你有一个 24 端口交换机，你将 23 个全双工的 100Mbit/s 端口流量都镜像到一个端口上，那么你可能在这个端口上就会有 4600Mbit/s 的流量。这将会远远超出一个单独端口的物理承受能力，因此在网络流量达到一定数量级后，将可能会导致数据包丢失，甚至网络速度变慢。

在这种情况下，交换机会丢弃所有多余的数据包，甚至“暂停”它们内部交换电路，从而造成通信中断。当你开始执行你的数据包捕获时，请务必小心不要让这种情况在你的网络中发生。

### 2.3.2 集线器输出

另一种在交换式网络中捕获目标设备通信流量的方式是集线器输出。使用这种技巧，你需要将目标设备和分析系统分段到同一网络段中，然后把它们直接插入到一个集线器上。

许多人认为集线器输出根本就是一种作弊方法，但是它在你不能进行端口镜像但仍对目标设备接入的交换机有着物理访问的时候，真的是一个完美的解决方案。为了进行集线器输出，你所需要的就是一个集线器和几根网线。当你找齐了硬件之后，就可以按照如下操作步骤进行连接。

1. 找到目标设备所连接的交换机，并将目标设备连接网线从交换机上拔掉。
2. 将目标设备的网线插入到你的集线器上。
3. 使用另一根网线，将你的嗅探分析器也连接到集线器上。
4. 从你的集线器连接一根网线到交换机上，将集线器连接到网络上。

表 2-1 用于启用端口镜像的命令

| 制造商 | 命令   |
|-----|--|
| 思科  | set span <source port> <destination port>                                    |
| 凯创  | set port mirroring create <source port> <destination port>                   |
| 北电  | port-mirroring mode mirror-port <source port> monitor-port<destination port> |

现在你已经将目标设备和你的嗅探分析器连接到了同一个广播域中，所有从你的目标设备流入流出的网络流量都将在集线器中广播，从而让你的嗅探分析器可以捕获到这些数据包，如图 2-6 所示。

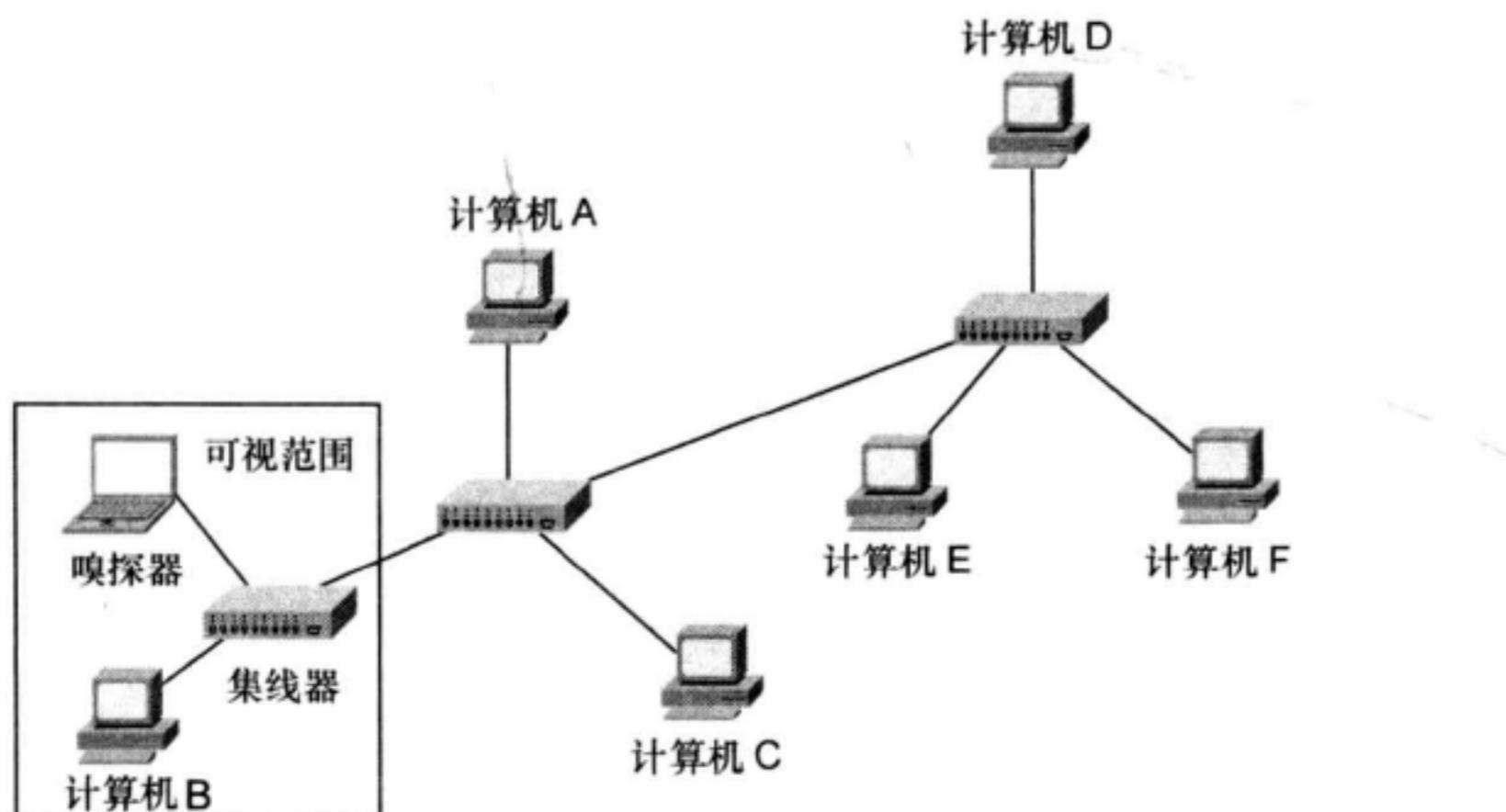


图 2-6 将你的目标设备通过集线器输出，与嗅探分析器连接在一起

在大多数情况下，集线器输出会将目标设备的全双工变成半双工。尽管这种方法并不是进行网络线路监听最彻底的方法，但在你的交换机不支持端口镜像时它可能是你唯一的选择。但是，请记住，你的集线器同样需要一个电源线连接，而某些情况下你却很难找到。

#### 注意

作为友情提示，在拔掉用户的设备时，你应该事先给用户一个善意的提醒，否则如果碰巧这位用户是公司 CEO，你就惨了。

### 找到“真正的”集线器

当进行集线器输出时，你需要确保你使用的设备是一个真正的集线器，而不是虚假标记的交换机。有几家网络硬件厂商有着营销的坏习惯，会把一些低级别的交换机当做集线器进行出售。如果你使用的并不是一个可信的经过测试的集线器，你将只会看到你自己的流量，而不是目标设备的流量。

当你找到一个集线器时，你需要对它进行测试，来确保它确实是一个集线器。如果是的话，它绝对值得你收藏！确定一个设备是否真的是集线器的最好方法，就是连上两台电脑，然后看这两台电脑是否能嗅探对方与网络其他设备之间的网络通信。如果能监听到的话，那它就是一款真正的集线器。

由于集线器已经是老古董，它们早就不再进行大规模的生产了。你几乎不可能从市面上买到真正的集线器了。所以你需要设法来找到一个。一个很好的来源往往是在当地学校的二手交易市场。公立学校在处理老旧设备之前，都必须尝试进行二手拍卖交易，而它们经常有一些很古老的硬件设备。我曾经见过有人从二手交易市场上仅仅花了不到一顿快餐的钱买到了好几个集线器。此外，eBay 也是一个集线器的好来源，但你也需要留意，你有可能也会遇到将交换机错误标识成集线器的情况。

### 2.3.3 使用网络分流器

大家都知道这句谚语：“有牛排可以吃的时候，为什么要选择鸡肉呢？”（或者是美国南方的谚语，“有炸腊肠吃的时候，为什么要选择火腿呢？”）。这种选择，也适用于使用网络分流器与集线器输出的对比上。

网络分流器是一个硬件设备，你可以将它放置在网络布线系统的两个端点之间，来捕获这两个端点之间的数据包。与集线器输出类似，你可以在网络上

放置一个硬件以捕获你需要的数据包。所不同的是，这次你并不是使用集线器，而是使用一个专门为了网络分析而设计的特殊硬件。

网络分流器又分为两种基本类型：聚合的和非聚合的。这两种分流器都安置在两个设备之间，来嗅探所有流经的网络通信。它们之间最基本的区别在于：非聚合的网络分流器有 4 个端口，如图 2-7 所示，而聚合分流器则只有 3 个端口。

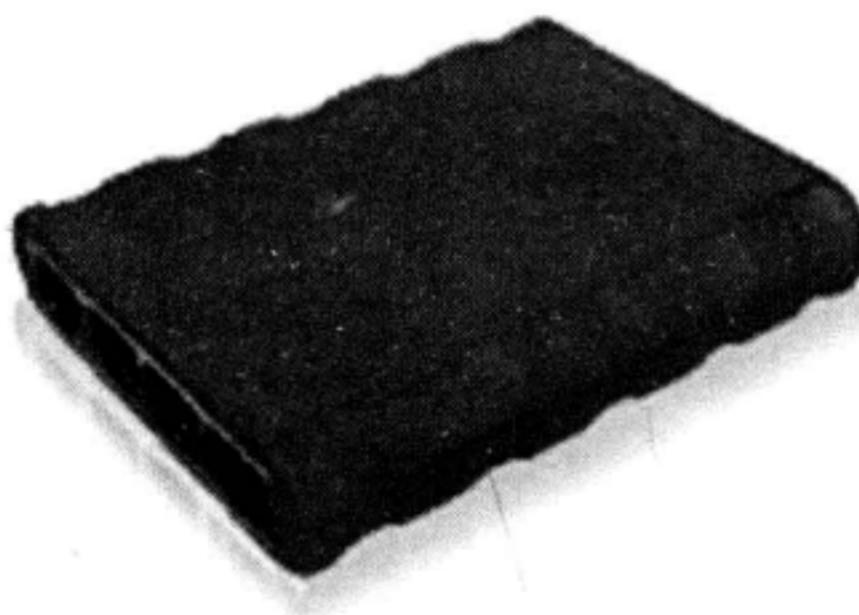


图 2-7 一款 Barracuda 的非聚合网络分流器

网络分流器通常还需要一个电源连接，但也有一些是带电池的，它们可以不需要插入一个电源插座就可以进行短暂的数据包嗅探。

- 聚合的网络分流器

聚合的网络分流器的使用方法是最简单的。它只有一个物理的流量监听口，来对双向通信进行嗅探。

为了使用聚合的网络分流器来捕获一台接入交换机的电脑流入流出的所有流量，你只需要按照如下步骤进行操作。

1. 从交换机上拔下目标电脑的网线。
2. 将连接目标电脑网线的另一端插入到网络分流器的“in”端口中。
3. 将另一根网线的一端插入到网络分流器的“out”端口，并将另一端插入到网络交换机。
4. 将最后一根网线的一端插入网络分流器的“monitor”端口，并将另一端插入到你作为嗅探器使用的电脑上。

聚合网络分流器的连接应该如图 2-8 所示的那样，一旦连好之后，你的嗅探器就能够捕获到你接入网络分流器的所有流入流出的网络流量。

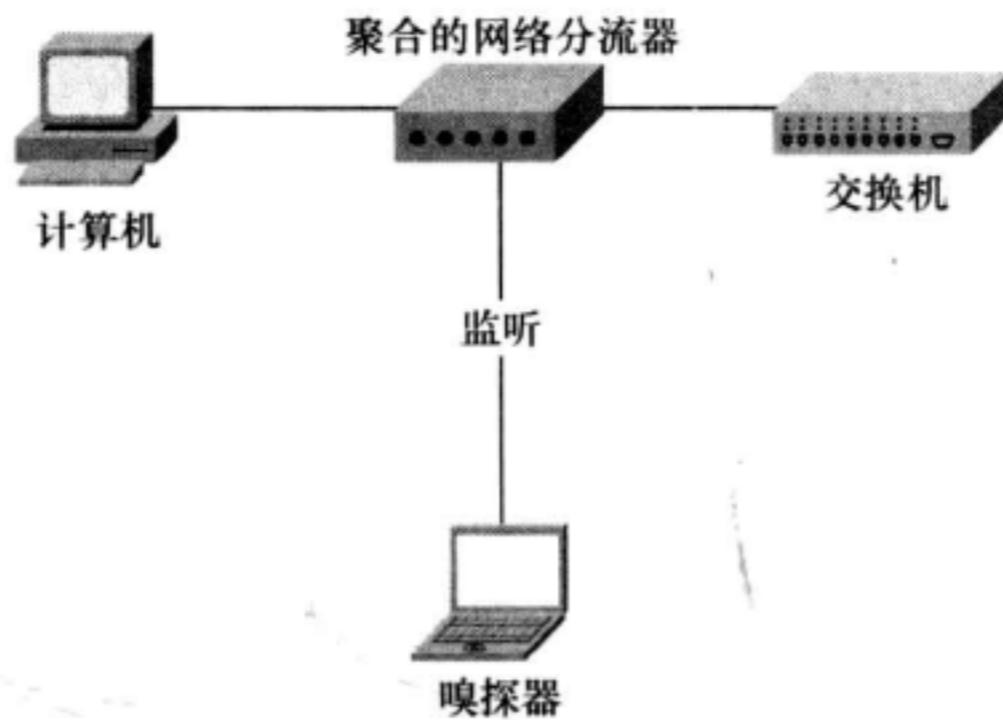


图 2-8 使用聚合的网络分流器来嗅探网络流量

- 非聚合的网络分流器

非聚合的网络分流器比起聚合的稍微复杂一些，但它在进行流量捕获时有着更好的灵活性。与聚合网络分流器仅仅只有一个监听端口来嗅探双向通信流量不同的是，非聚合的网络分流器有着两个监听端口。一个监听端口是用来嗅探流出方向的网络流量（从电脑到分流器端口的方向），另一个监听端口是用来嗅探流入方向的网络流量（从分流器端口到电脑的方向）。

为了捕获一台连接交换机的电脑的所有流入流出网络流量，你则需要按照如下步骤进行配置。

1. 从交换机上拔下电脑连接网线。
2. 将电脑连接网线的另一端插入到网络分流器的“in”端口上。
3. 将另一根网线的一端插入到网络分流器的“out”端口，然后将另一端插入到网络交换机上。
4. 将第三根网线插入到网络分流器的“Monitor A”端口，并将另一端插入到你作为嗅探器使用的电脑的一块网卡接口上。
5. 将最后一根网线插入到网络分流器的“Monitor B”端口，并将另一端插入到你作为嗅探器使用的电脑的第二块网卡接口上。

非聚合网络分流器的连接方式如图 2-9 所示。

- 选择一款网络分流器

网络分流器拥有两种不同的类型，那么哪一种会更好一些呢？在大多数情况下，聚合的网络分流器是首选，因为它们需要较少的网线，同时在嗅探器电脑上也不需要两块网卡。然而，在一些你需要捕获高带宽的流量，或是只需要

关注一个方向上的流量时，非聚合的网络分流器会更加适用。

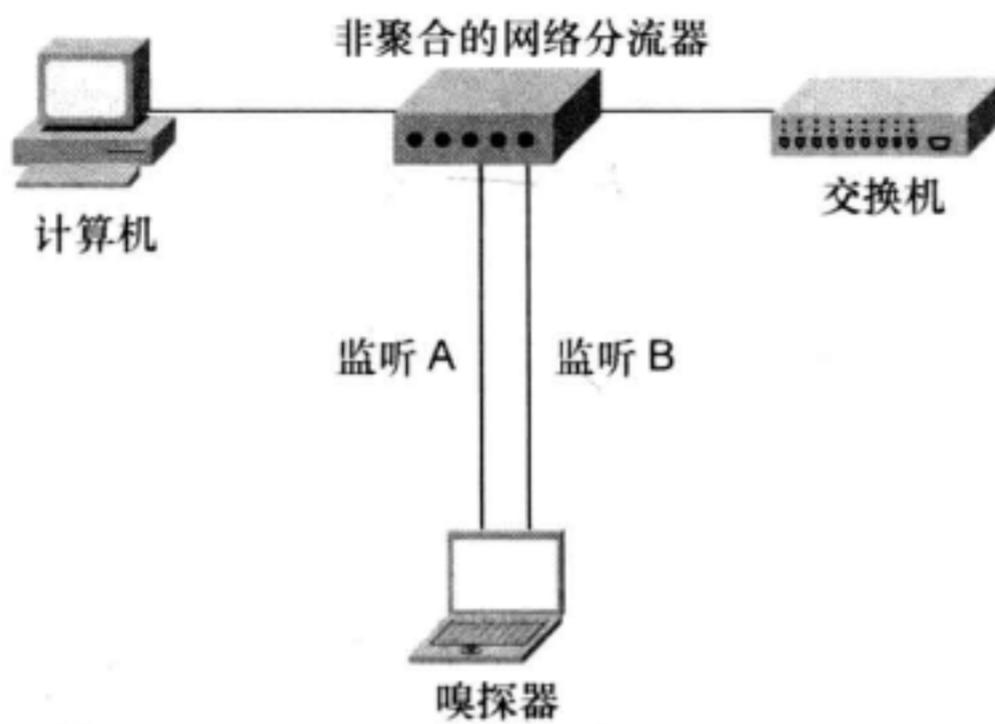


图 2-9 使用非聚合的网络分流器来嗅探网络流量

你可以购买到各种规格的网络分流器，从最简单 150 美元左右就能买到的以太网分流器，到需要数万美元的企业级光纤分流器。我曾经使用过 Net Optics 和 Barracuda 网络的网络分流器，它们的产品都非常不错。我敢肯定，市面上还有很多其他非常不错的选择。

### 2.3.4 ARP 欺骗

进行网络线路监听最让人喜欢的技术，就是 ARP 欺骗。我们将在第 6 章中详细介绍 ARP 协议，但在这里会进行简要解释，以帮助了解这种技术是如何工作的。

- ARP 查询过程

在第 1 章里，我们介绍了 OSI 参考模型中在第 2 层与第 3 层上数据包寻址的两种主要方式。这些第 2 层地址，或称为 MAC 地址，在无论你使用哪种第 3 层寻址方案时，都会与之协同工作。

在本书中，按照行业标准术语，我们将第 3 层寻址方案称为 IP 寻址系统。网络上的所有设备相互通信时在第 3 层上均使用 IP 地址。由于交换机在 OSI 模型的第 2 层上工作，它们只认识第 2 层上的 MAC 地址，因此网络设备必须在它们创建的数据包中包含这些信息。当这些设备在不知道通信对方的 MAC 地址时，必须要通过已知的第 3 层 IP 地址来进行查询，这样才可能通过交换机将流量传输给相应的设备。

这些翻译过程就是通过第 2 层上的 ARP 协议来进行实施的。连接到以太网

的计算机的 ARP 查询过程，是从一台计算机想要与另一台进行通信时开始的。发起通信的计算机首先检查自己的 ARP 缓存，查看它是否已经有对方 IP 地址对应的 MAC 地址。

如果不存在，它将往数据链路层广播地址 FF:FF:FF:FF:FF:FF 发送一个 ARP 广播请求包。作为一个广播数据包，它会被这个特定的以太网广播域上的每台计算机接收，这个请求包问道：“某某 IP 地址的 MAC 地址是什么？”

不匹配目标 IP 地址的计算机会简单地选择丢弃这个请求包。而目标计算机则选择答复这个数据包，通过 ARP 应答告知它的 MAC 地址。此时，发起通信的计算机就获取到了数据链路层的寻址信息，便可以利用它与远程计算机进行通信，同时将这些信息保存在 ARP 缓存中，来加速以后的网络访问。

- ARP 欺骗是如何工作的

ARP 欺骗，有时也被称为 ARP 缓存中毒，是通过发送包含虚假 MAC 地址的 ARP 消息，以劫持其他计算机流量的过程。图 2-10 显示了 ARP 欺骗的具体过程。

ARP 欺骗是一种在交换式网络中进行监听的高级技术。它通常由攻击者用于向客户端系统发送虚假地址的数据包，以截获特定的网络流量或者对目标进行拒绝服务（DoS）攻击。然而，它也可以是一种在交换式网络中捕获目标系统数据包的合法方式。

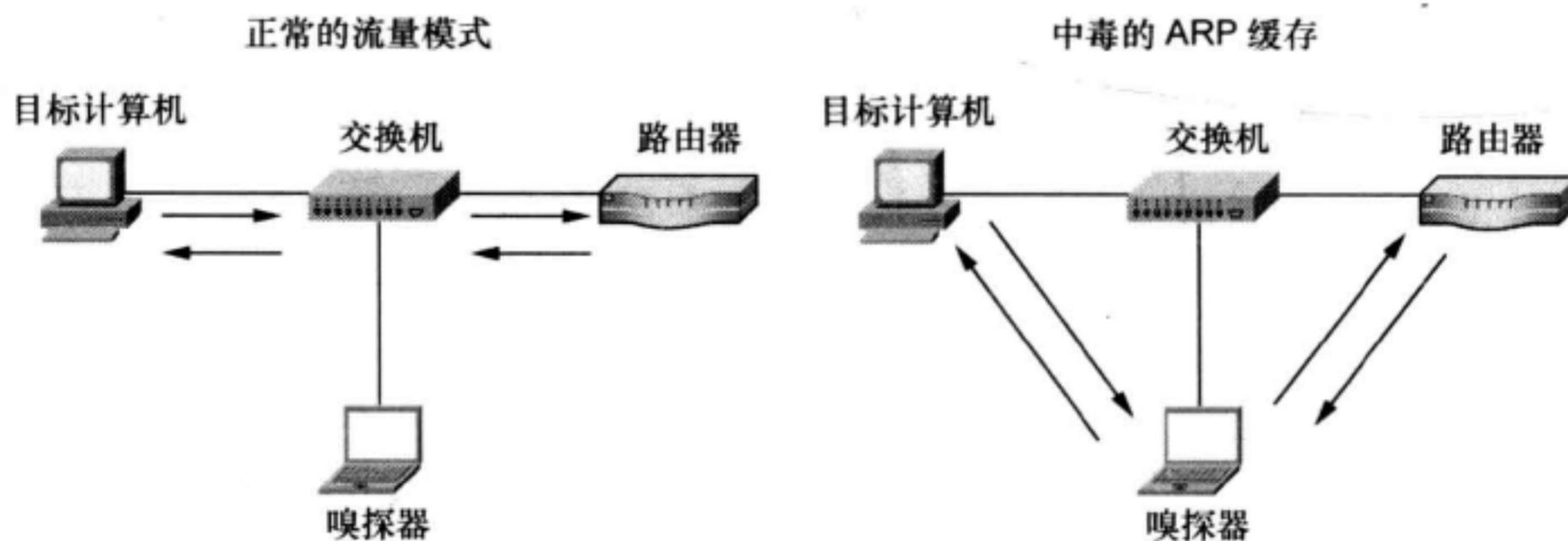


图 2-10 ARP 欺骗允许你拦截目标计算机的流量

- 使用 Cain & Abel 软件

当试图进行 ARP 欺骗时，第一步你需要获得一些必要的工具来搜集相关信息。在我们的演示中，我们将使用一款流行的安全工具 Cain & Abel，可以从

oxid.it (<http://www.oxid.it/>) 下载获得。这款软件也支持 Windows 系统。你可以根据网站上的指引来下载和安装这款软件。

在你使用 Cain & Abel 软件之前，你需要收集某些信息，包括嗅探分析器系统的 IP 地址，你所希望嗅探网络流量的远程计算机的 IP 地址，以及远程计算机所连接的上游路由器 IP 地址。

当你第一次打开 Cain & Abel 软件后，你会发现在软件窗口的顶端有着一系列的标签页（ARP 缓存中毒攻击只是强大的 Cain & Abel 软件其中一个功能）。为了演示我们的例子，我们将切换到“嗅探器”选项卡上。当你单击此选项卡，你应该会看到一个空表，如图 2-11 所示。

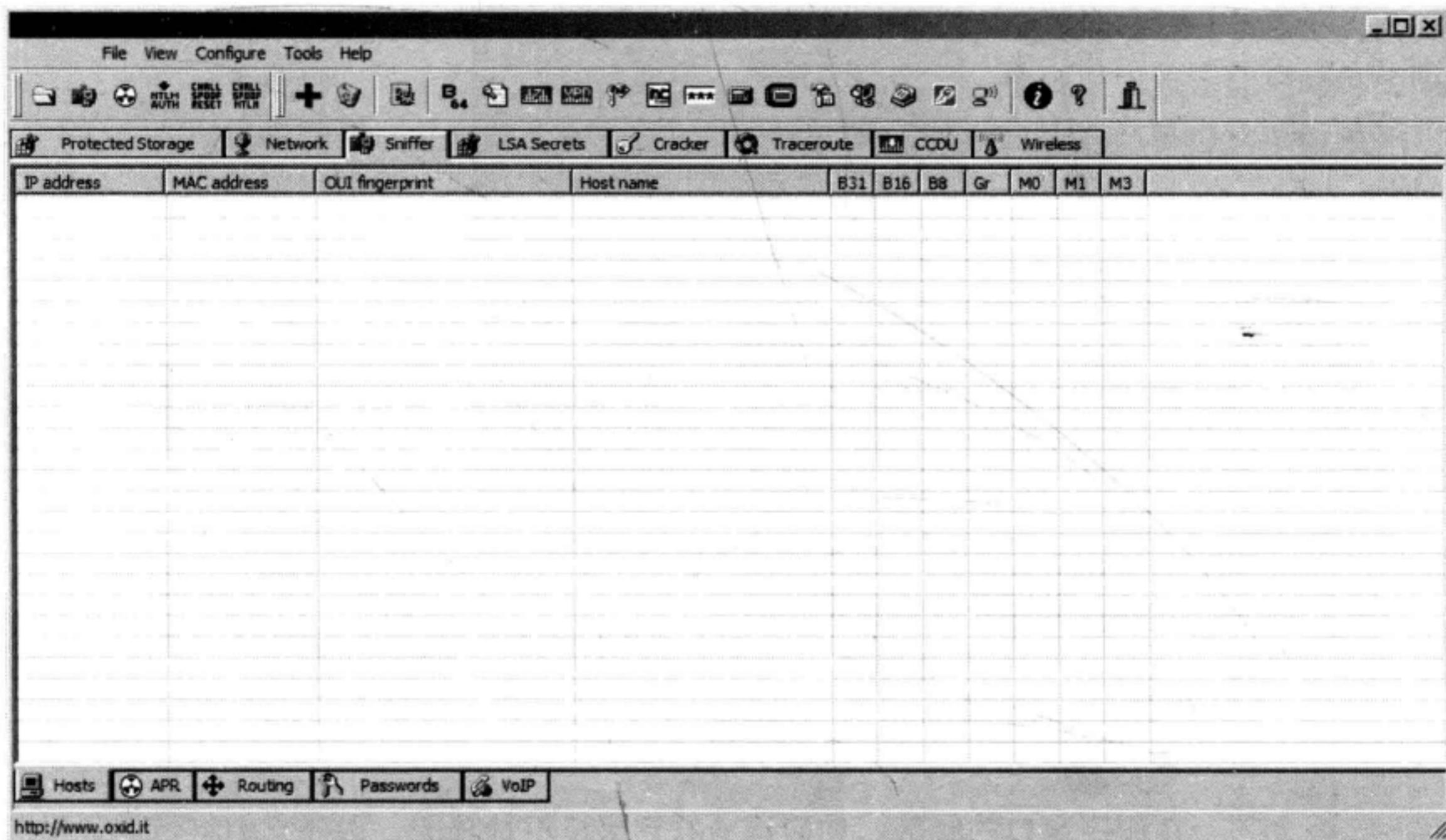


图 2-11 Cain & Abel 软件主窗口中的“嗅探器”选项卡

要完成此表，你需要激活这款软件的内置嗅探器，扫描你的网络并找出活跃主机。请按以下步骤进行操作以完成上述目标。

1. 单击工具栏上左起第二个图标，类似网卡形状的那个。
2. 你会被要求选择你希望进行嗅探的网络接口。这个接口应该连接到你所希望进行 ARP 欺骗的网络。选择这个网络接口，然后单击 OK 按钮（要确保按下这个按钮，以激活 Cain & Abel 软件内置的嗅探器）。

3. 要建立在你的网络上可用主机的列表，单击加号（+）图标。MAC 地址扫描器对话框将会出现，如图 2-12 所示。请选择“*The All hosts in my subnet*”圆形按钮（或者你可以选择特定的地址范围），单击 OK 继续。

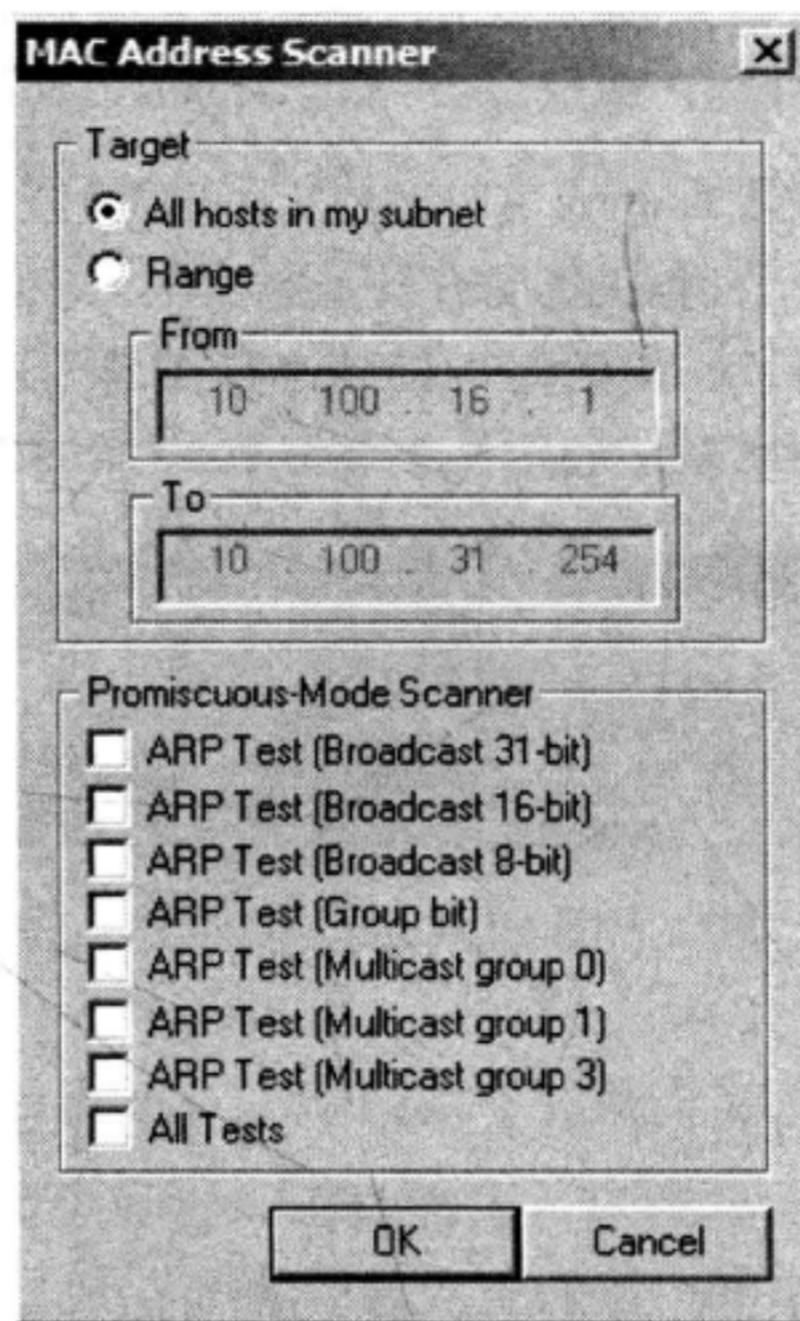


图 2-12 Cain & Abel 网络发现工具

现在表格中应该填满了你所在网络中的所有主机的信息，包括它们的 MAC 地址、IP 地址和供应商信息等。这是你开始进行 ARP 欺骗的目标主机列表。

在程序窗口的底部，你应该会看到另一组选项卡，选择它们将带你到嗅探器标题下的其他窗口。现在，你已经创建了主机列表，你就可以选择 ARP 选项卡，切换至 ARP 窗口中。

在 ARP 窗口中，你会看到两个空的表格。当你完成下面的操作步骤之后，上方的表格中将显示出你的 ARP 欺骗过程涉及的设备列表，而下方表格则会显示出在你进行中毒攻击的计算机之间的所有通信内容。

进行 ARP 欺骗攻击，请按照下列步骤进行操作。

1. 在屏幕上方的空白区域中单击，然后单击程序标准工具栏中的加号（+）

图标。

2. 出现的窗口中会有两个选择栏。在左侧，你可以看到网络上的所有活跃主机的列表。单击你希望进行网络流量嗅探的目标系统 IP 地址，右边的选择栏中将会显示出网络中除了你所选择的目标主机 IP 地址之外的所有主机列表。

3. 在右边的选择栏中，单击目标计算机的直接上游路由器（即网关）IP 地址，如图 2-13 所示，然后单击“OK”。这两个设备的 IP 地址现在应该会被显示在主程序窗口上方的表格中。

4. 完成这个过程的最后一步，单击标准工具栏中黄黑相间的辐射符号，这个操作将激活 Cain & Abel 软件的 ARP 欺骗功能。让你的嗅探分析器作为从目标系统到它的上游路由器之间所有通信的中间人。你现在应该就能启动你的数据包嗅探器，并开始分析过程。当你完成流量捕获之后，只需再次单击黄黑相间的辐射图标，便可以停止 ARP 欺骗过程。

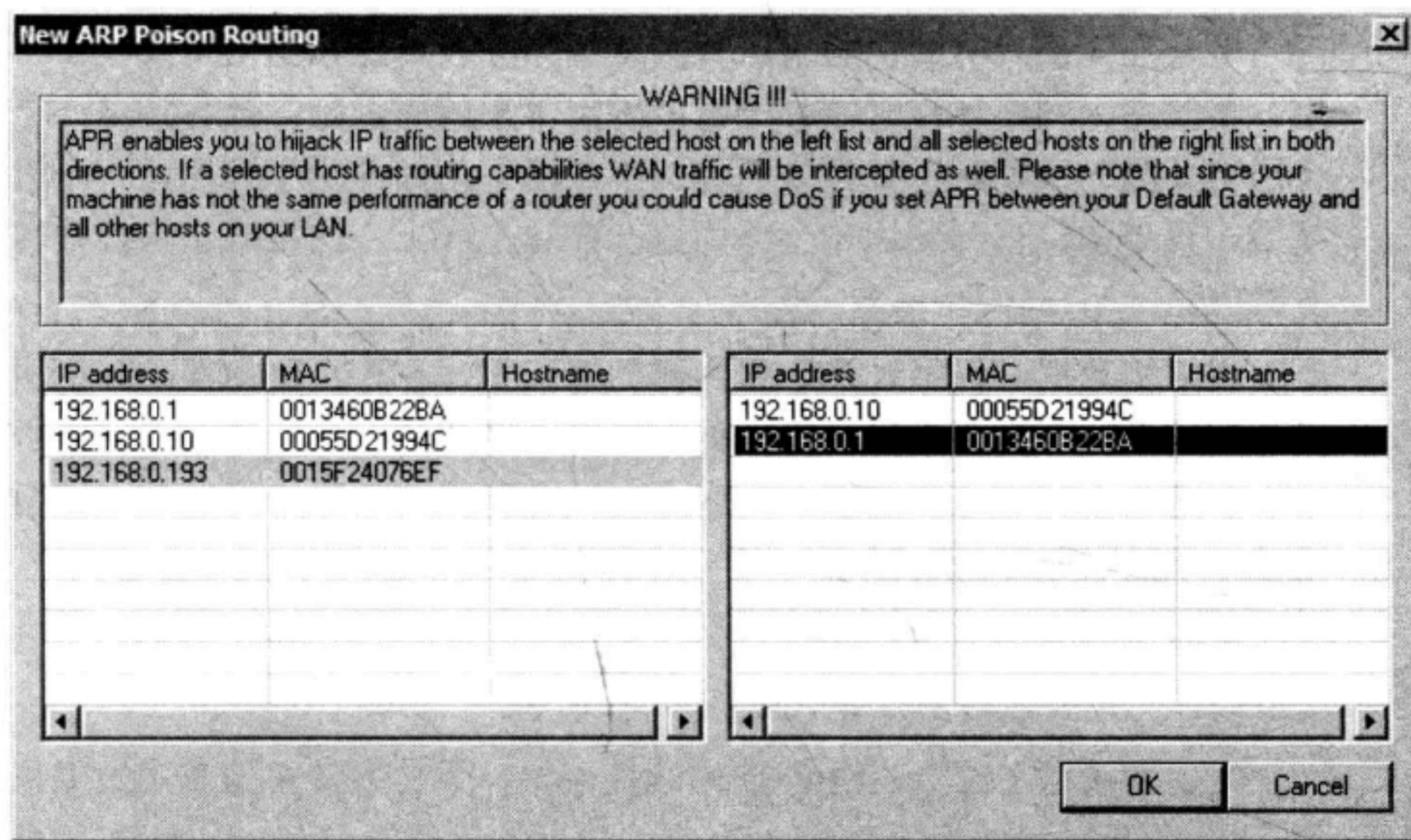


图 2-13 选择你要启用 ARP 欺骗的目标系统

- 关于 ARP 欺骗的警示

作为 ARP 欺骗过程的最后警示，你必须要非常清楚实施这个过程中每个系统的角色与作用。在目标设备拥有很高的网络使用流量时，例如一台有着 1Gbit/s 联网线路的文件服务器，不要使用这项技术（尤其是当你的嗅探分析系统只提

供了一条 100Mbit/s 的链路时)。

当你使用在这个例子中演示的这项技术对网络流量进行重路由时，所有目标系统发送和接收的流量都必须先通过你的嗅探分析系统，因此，你的嗅探分析系统可能成为整个通信过程中的瓶颈。这种流量重路由会对你进行分析的系统造成一种拒绝服务攻击式的影响，将导致网络性能下降以及分析数据不完全。

#### 注意

你可以使用一个称为非对称路由的功能，来避免所有的网络流量经过你的嗅探分析器，所谓非对称路由。对于这种技术的更多信息，请参阅 oxid.it 用户手册 ([http://www.oxid.it/ca\\_um/topics/apr.htm](http://www.oxid.it/ca_um/topics/apr.htm))。

## 2.4 在路由网络环境中进行嗅探

所有在交换式网络中用来监听网络线路的技术在路由网络环境中都同样适用。面对路由网络环境时，唯一需要重点考虑的问题是：当你调试一个涉及多个网络分段的故障时，如何安置你的嗅探器？正如你所学到的，一个设备的广播域一直延伸，直到到达一个路由器，在这个点上，网络流量将会被转发给上游路由器。

在网络数据必须经过多个路由器的情况下，在各个路由器上分析网络流量是非常重要的。举例来说，考虑你很可能会遇到的一个场景，在网络中由几个路由器将几个网络分段连接在一起。在这个网络中，每个网段与上游网段进行通信，来获取和存储数据。

如图 2-14 所示，我们要解决的一个故障是：一个下游子网 D，无法与网络 A 中的任何设备进行通信。

如果你在存在故障的网络 D 中嗅探流量，你可以清楚地看到数据包被传输到了其他网段，但你可能看不到回来的数据包所说的“一会回来”。如果你重新考虑你嗅探器的部署位置，在网络 D 的直接上游网段（网络 B）中开始嗅探，你将会有个关于故障更清晰的视图。

在这个位置上，你可能会发现，来自网络 D 的流量被丢弃了，或是被网络 B 的路由器错误地路由了。

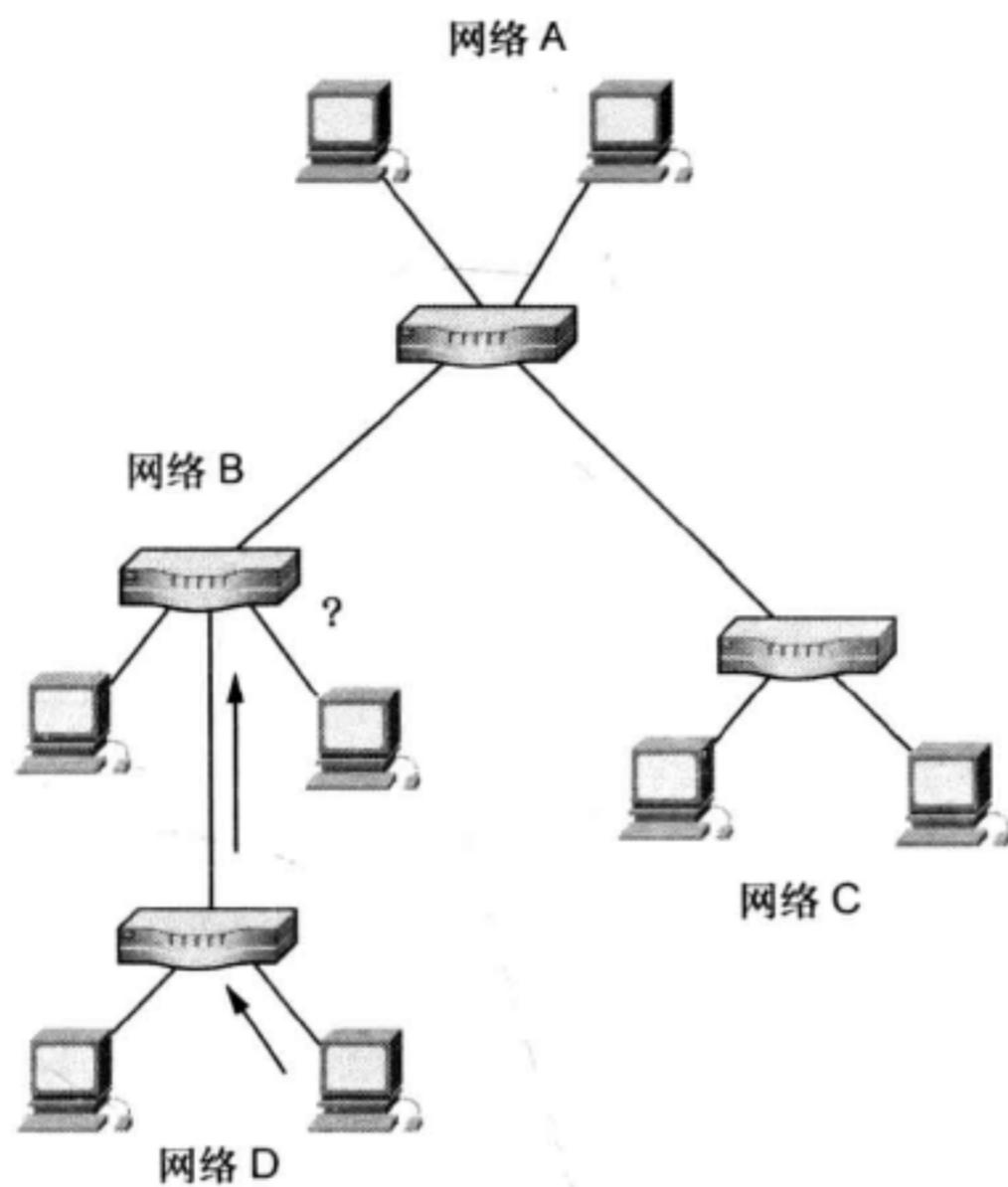


图 2-14 网络 D 中的计算机不能与网络 A 中的计算机进行通信

最终，这会让你了解到这是一个路由器配置问题，在纠正之后，便会解决掉你的大麻烦。虽然这个场景有点宽泛，但其中的精髓是，在处理涉及多个网段与路由器的问题时，你可能需要将你的嗅探器移动到不同的位置上，才能获得一个完整的网络画面。

这是一个基本的例子，说明了为什么往往需要在不同的网段中对多个设备流量进行嗅探，才能很快地诊断出故障的根本原因。

### “网络地图”

在关于网络布局的讨论中，我们已经研究了好几种不同的“网络地图”。“网络地图”或称为网络拓扑图，是一个显示了网络中所有技术资源以及它们之间连接关系的图形表示。

在决定你数据包嗅探器的安放位置时，没有比拿着一张“网络地图”来进行分析更好的办法了。如果你有一张“网络地图”，请把它保留在手边，它在故障排除和分析过程中，会是一份宝贵的财产。建议你对自己的网络画出一份详细的“网络地图”。请记住，在大多数时候，排除故障一半以上的工作都集中在收集正确的网络数据上。

## 2.5 部署嗅探器的实践指南

我们已经介绍了在交换式网络中捕获网络流量的 4 种不同方法。我们可以再增加一种方式，适用于我们仅仅在单个系统上安装嗅探器软件并监听这台系统进出的流量。在某个特定场景中，你可能不太容易确定应该用上述这 5 种方法中的哪种才是最合适。表 2-2 提供了每种部署方法的通用准则。

表 2-2 在交换式网络环境中进行数据包嗅探的指导准则

| 技术       | 指导准则   |
|----------|--|
| 端口镜像     | <ul style="list-style-type: none"><li>通常是首选的，因为它不会留下网络痕迹，也不会因此而产生额外的数据包</li><li>可以在不让客户端脱机下线的情况下进行配置，非常便于镜像路由器或者服务器端口</li></ul>  |
| 集线器输出    | <ul style="list-style-type: none"><li>当你不需要考虑主机暂时下线带来的后果时适用</li><li>当你必须捕获多台主机的流量时是低效率的，因为碰撞和丢包会导致性能低下</li><li>可能会导致现代的 100/1000Mbit/s 主机丢失数据包，因为大多数真正的集线器都只是 10Mbit/s 的</li></ul>         |
| 使用网络分流器  | <ul style="list-style-type: none"><li>当你不需要考虑主机暂时下线带来的后果时适用</li><li>当你需要嗅探光纤通信时，这是唯一选项</li><li>由于网络分流器就是为了网络监听嗅探而设计的，而且能够跟上现代网络速度，因此这种方法比起集线器输出要更优一些</li><li>在预算紧张时，这种方法的成本会过于高昂</li></ul> |
| ARP 缓存中毒 | <ul style="list-style-type: none"><li>这会被认为是非常草率的，因为它涉及在网络上注入数据包，以重路由流经嗅探器的网络流量</li><li>在你需要一个暂时性快速实施的方法，能够将一个设备的网络流量进行捕获，而又不用将其下线，同时端口镜像又不被支持的时候，这种方法会是一个高效的选择</li></ul>                  |
| 直接安装     | <ul style="list-style-type: none"><li>一般不建议，因为如果一台主机存在故障和问题，这个问题可能会导致数据包被丢弃，或是被配置成它们无法被准确地展示的样子</li><li>主机的网卡不需要设置在混杂模式</li><li>在进行环境测试、评估和审查性能，或是检查在其他地方捕获的数据包文件时，这是最佳方案</li></ul>        |

作为分析师，我们需要尽可能地隐蔽。最理想的境界是，我们采集我们所需要的数据，而不留下任何的痕迹。这就像是法医在调查时不想对犯罪现场造成任何破坏一样。我们也不希望破坏我们所捕获的网络流量。

当在后面章节中逐步面对一些实际场景时，我们将会逐个对案例进行详细分析，来讨论捕获数据最好的方式。目前来说，图 2-15 中给出的流程图应该能够帮助你决定用来捕获流量的最佳方法。请记住，这个流程图只是一个简单的

通用参考，并不涵盖所有用来监听网络线路的可能方法。

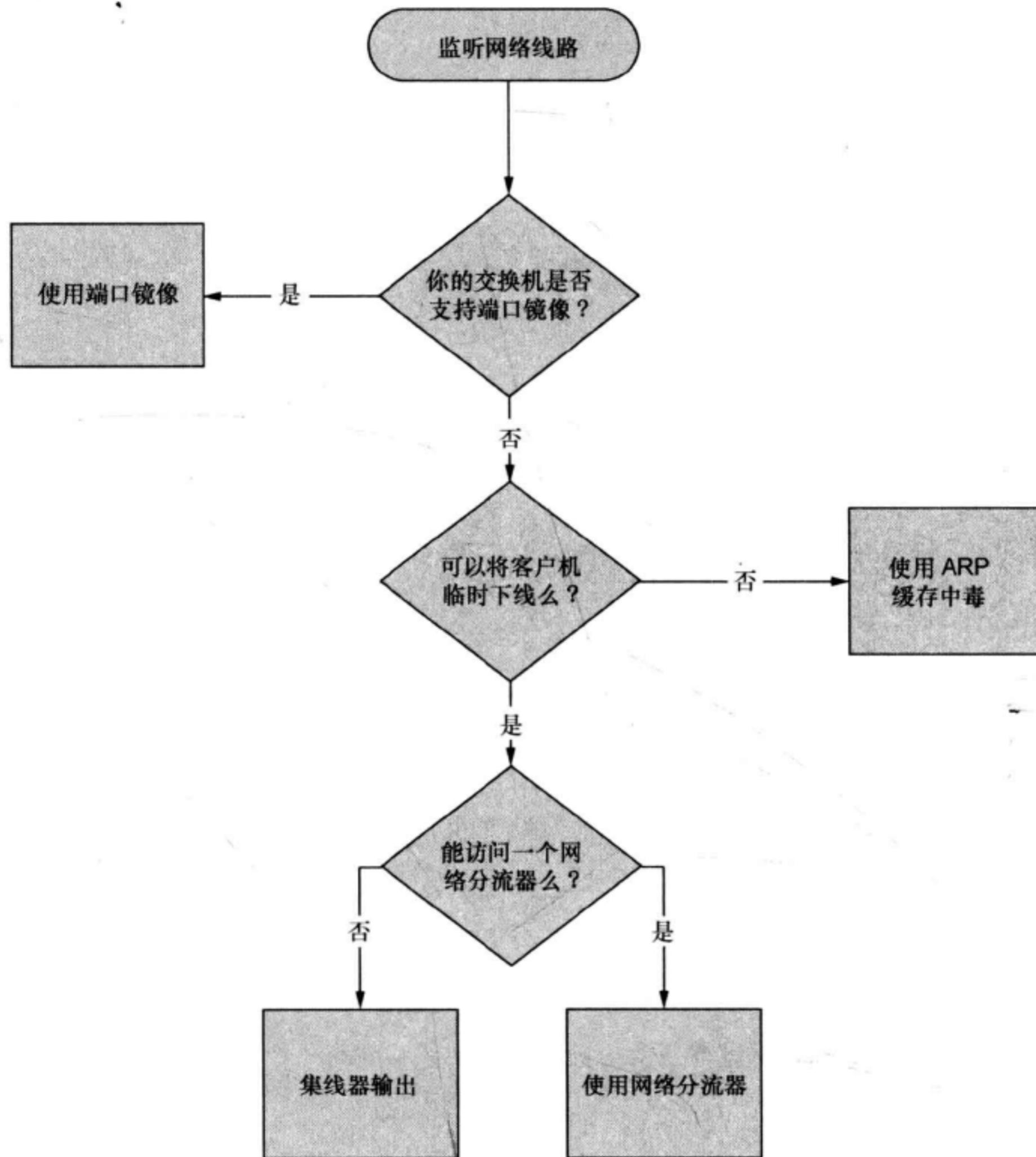


图 2-15 帮助确定哪种是最适合的网络监听方法的流程图



# 第3章

## Wireshark 入门



在第1章中，我们介绍了几种可以进行网络分析的数据包嗅探工具软件，但在本书中我们将只使用 Wireshark，并在此章进行简要的介绍。

### 3.1 Wireshark 简史

Wireshark 的历史相当久远，其最初的版本叫做 Ethereal，由毕业于密苏里大学堪萨斯城分校计算机科学专业的 Gerald Combs 出于项目需要而开发，并于 1998 年以 GNU Public Licence (GPL) 开源许可证发布。

在发布了 Ethereal 8 年之后，Combs 辞职另谋高就，但是在那个时候他的雇主公司掌握着 Ethereal 的商标权，而 Combs 也没能和其雇主就取得 Ethereal 商标达成协议。于是 Combs 和整个开发团队在 2006 年中的时候将这个项目重新命名为 Wireshark。

Wireshark 随后迅速地取得了大众的青睐，而其合作开发团队也壮大到 500 人以上，然而之前的 Ethereal 项目却再没有前进过一步。

## 3.2 Wireshark 的优点

Wireshark 在日常应用中具有许多优点，无论你是初学者还是数据包分析专家，Wireshark 都能通过丰富的功能来满足你的需要。在第 1 章中，我们为挑选数据包嗅探工具提出过一些重要的判断特征，让我们来检查一下 Wireshark 是否具有这些特征。

**支持的协议：**Wireshark 在支持协议的数量方面是出类拔萃的——于本书截稿时 Wireshark 已提供了超过 850 种协议的支持。这些协议包括从最基础的 IP 协议和 DHCP 协议到高级的专用协议比如 AppleTalk 和 BitTorrent 等。由于 Wireshark 在开源模式下进行开发，每次更新都会增加一些对新协议的支持。

---

**注意** 在一些特殊情况下，如果 Wireshark 并不支持你所需要的协议，你还可以通过自己编写代码提供相应的支持，并提供给 Wireshark 的开发者，以便于使之能被包含在之后版本中（当然是在代码被采纳的情况下）。

---

**用户友好度：**Wireshark 的界面是数据包嗅探工具中最容易理解的工具之一。它基于 GUI，并提供了清晰的菜单栏和简明的布局。为了增强实用性，它还提供了不同协议的彩色高亮，以及通过图形展示原始数据细节等不同功能。与 tcpdump 使用复杂命令行的那些数据包嗅探工具相比，Wireshark 的图形化界面对于那些数据包分析的初学者而言，是十分方便的。

**价格：**由于 Wireshark 是开源的，它在价格上面是无以匹敌的。Wireshark 是遵循 GPL 协议发布的自由软件，任何人无论出于私人还是商业目的，都可以下载并且使用 Wireshark。

---

**注意** 尽管 Wireshark 是免费的，但是还是会有一些人不小心去“付费”购买它。如果你在 eBay 搜索“数据包嗅探”，你会惊讶地发现会有如此多的人想以 \$39.95 的跳楼价向你出售 Wireshark 的“专业企业级许可证”。显而易见，这些都是骗人的把

---

---

戏。但是如果你执意想要购买这些所谓的“许可证”，不如给我打个电话，我正有些肯塔基的海边别墅以跳楼价出售（肯塔基州是美国的一个内陆州——译者注）。

---

**程序支持：**一个软件的成败通常取决于其程序支持的好坏。虽然像 Wireshark 这样的自由分发软件很少会有正式的程序支持，而是依赖于开源社区的用户群，但是幸运的是，Wireshark 社区是最活跃的开源项目社区之一。Wireshark 网页上给出了许多种程序支持的相关链接，包括在线文档、支持与开发 wiki、FAQ，并可以注册 Wireshark 开发者都关注的邮件列表。CACE Technologies 通过 SharkNet 项目也对外提供付费支持。

**支持的操作系统：**Wireshark 对主流的操作系统都提供了支持，其中包括 Windows、Mac OS X 以及基于 Linux 的系统。你可以在 Wireshark 的主页上查询所有 Wireshark 支持的操作系统列表。

### 3.3 安装 Wireshark

Wireshark 的安装过程极其简单。但在你安装之前要确保你的机器满足如下要求。

- 400MHz 及以上的处理器
- 128MB 内存
- 至少 75MB 的可用存储空间
- 支持混杂模式的网卡
- WinPcap 驱动

WinPcap 驱动是 Windows 对于 pcap 数据包捕获的通用程序接口（API）的实现，简单来说就是这个驱动能够通过操作系统捕捉原始数据包、应用过滤器，并能够让网卡切入或切出混杂模式。

尽管你也可以单独下载安装 WinPcap (<http://www.winpcap.org>)，但一般最好使用 Wireshark 安装包中的 WinPcap。因为这个版本的 WinPcap 经过测试，能够和 Wireshark 一起工作。

#### 3.3.1 在微软 Windows 系统中安装

在 Windows 中安装 Wireshark 的第一步就是在 Wireshark 的官方网站

<http://www.wireshark.org> 上找到 Download 页面，并选择一个镜像站点下载最新版的安装包。在下载好安装包之后，遵照如下步骤进行安装。

1. 双击.exe 文件开始进行安装，在介绍页面上单击 **Next**。
2. 阅读许可证条款，如果同意接受此条款，单击 **I Agree**。
3. 选择你希望安装的 Wireshark 组件，如图 3-1 所示。在本书中接受默认设置即可，并单击 **Next**。

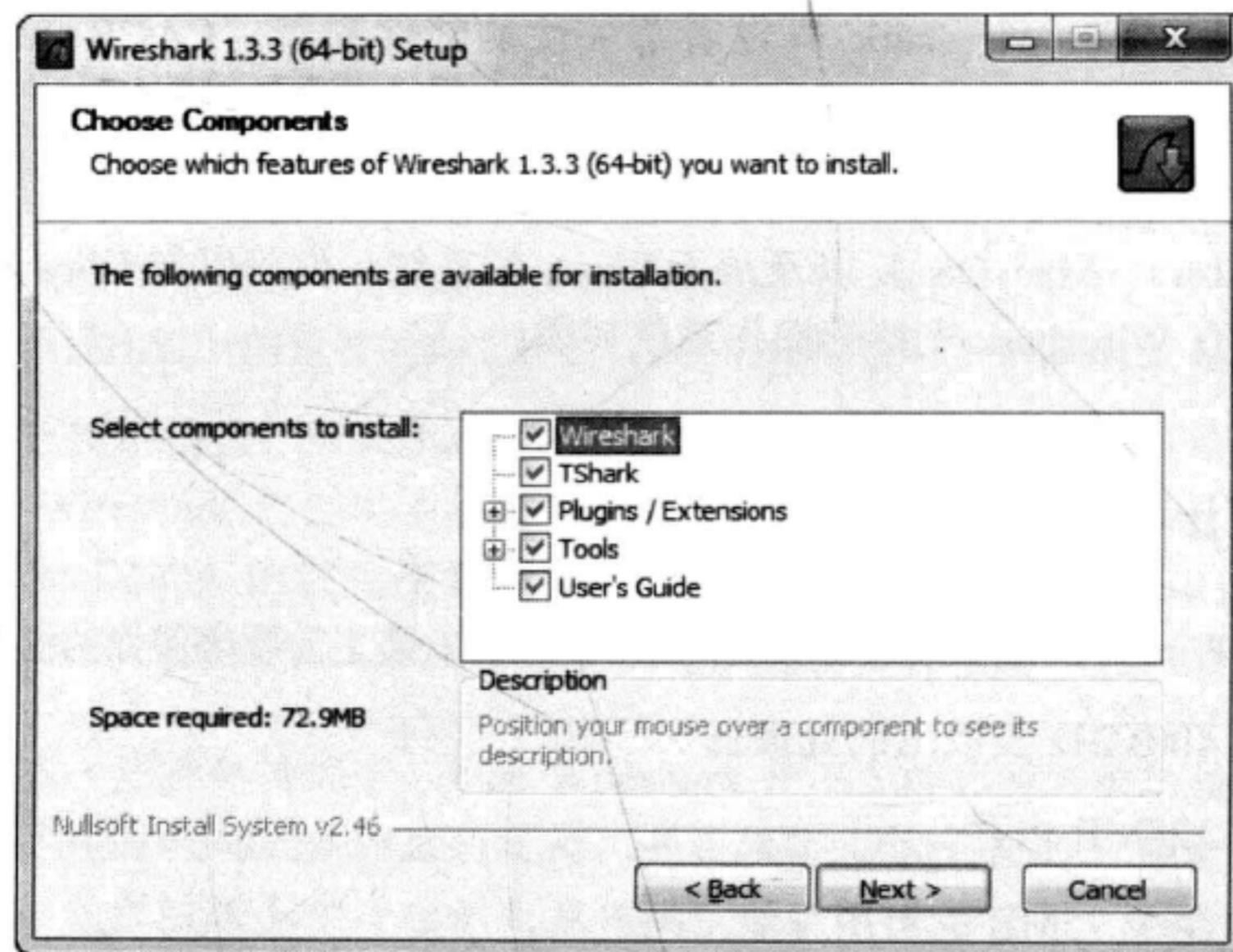


图 3-1 选择你想要安装的 Wireshark 组件

4. 在 Additional Tasks 窗口中单击 **Next**。
5. 选择 Wireshark 的安装位置，并单击 **Next**。
6. 当弹出是否需要安装 WinPcap 的对话框时，务必确保 **Install WinPcap** 选项已被勾选，如图 3-2 所示，然后单击 **Install**。安装过程便会随即开始。
7. Wireshark 的安装过程进行了大约一半的时候，会开始安装 WinPcap。在介绍页面单击 **Next** 之后，请阅读许可协议并单击 **I Agree**。
8. WinPcap 应该已经安装到你的电脑上，在安装完成之后，单击 **Finish**。
9. Wireshark 应该已经安装到你的电脑上，在安装完成之后，单击 **Next**。
10. 在安装完成确认界面中，单击 **Finish**。

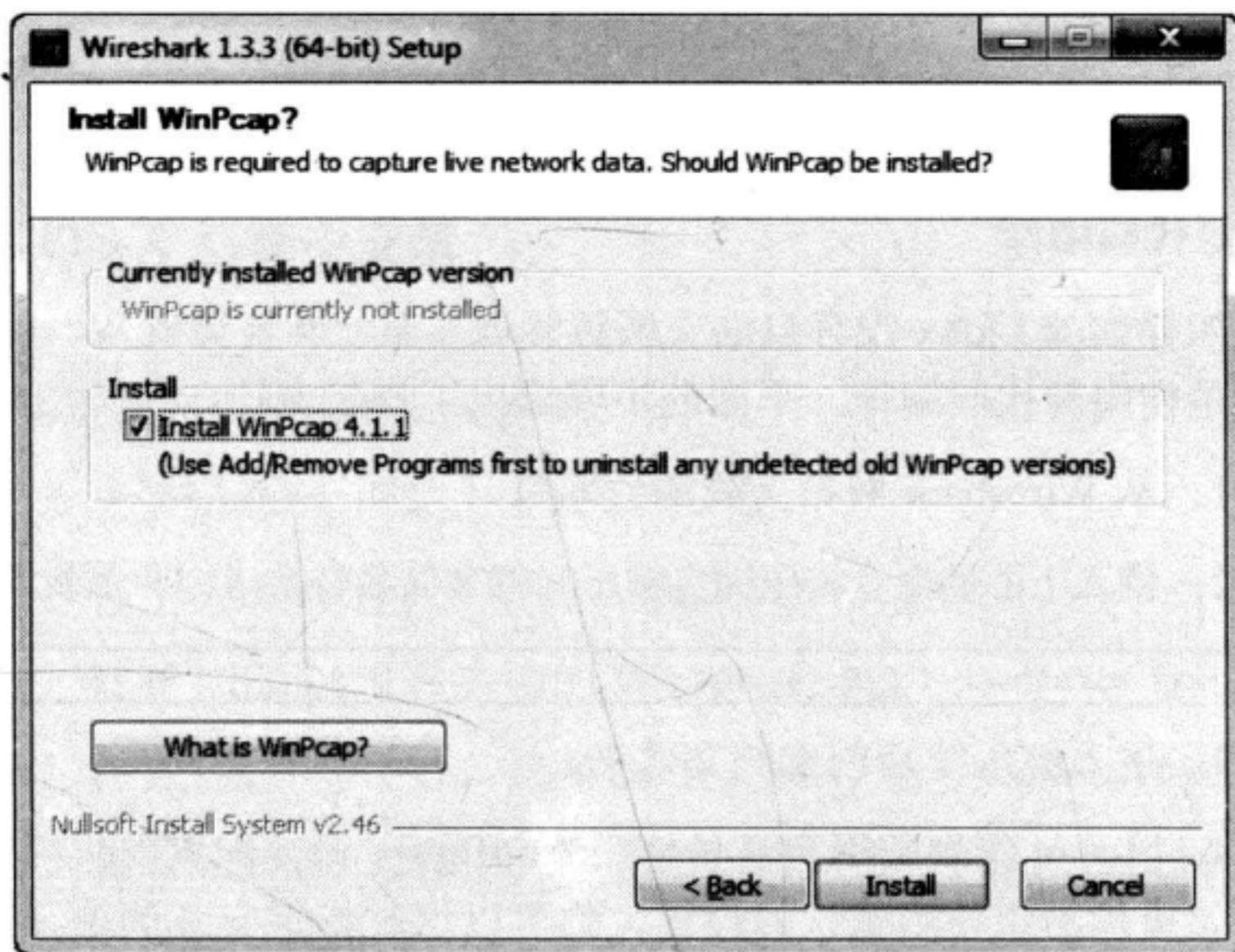


图 3-2 选中安装 WinPcap 驱动的选项

### 3.3.2 在 Linux 系统中安装

在 Linux 系统中安装 Wireshark 的第一步是先下载合适的安装包。由于 Wireshark 并不支持所有的 Linux 版本，所以你可能会发现并没有合适你的 Linux 版本对应的安装包可供下载。

一般来说，如果作为系统软件安装，你需要具有 root 权限。但如果你通过编译源代码安装为本地软件，那么通常就不需要 root 权限。

#### 使用 RPM 的系统

对于类似于红帽 Linux (Red Hat Linux) 等使用 RPM 的 Linux 发行版，在从 Wireshark 网站上下载好合适的安装包之后，打开一个命令行程序并键入如下命令（将文件名替换成你所下载安装包的名称）：

```
rpm -ivh wireshark-0.99.3.i386.rpm
```

如果缺少相关程序支持，在安装好这些之后，再重新安装 Wireshark。

#### 使用 DEB 的系统

对于类似与 Debian 和 Ubuntu 等使用 DEB 的 Linux 版本，你可以从系统源

中安装 Wireshark，打开一个命令行窗口并键入如下命令。

---

```
apt-get install wireshark
```

---

## 使用源代码编译

如果你的 Linux 没有自动安装包管理工具，那么安装 Wireshark 最高效的方法就是使用源代码编译。下面的步骤给出了安装方法。

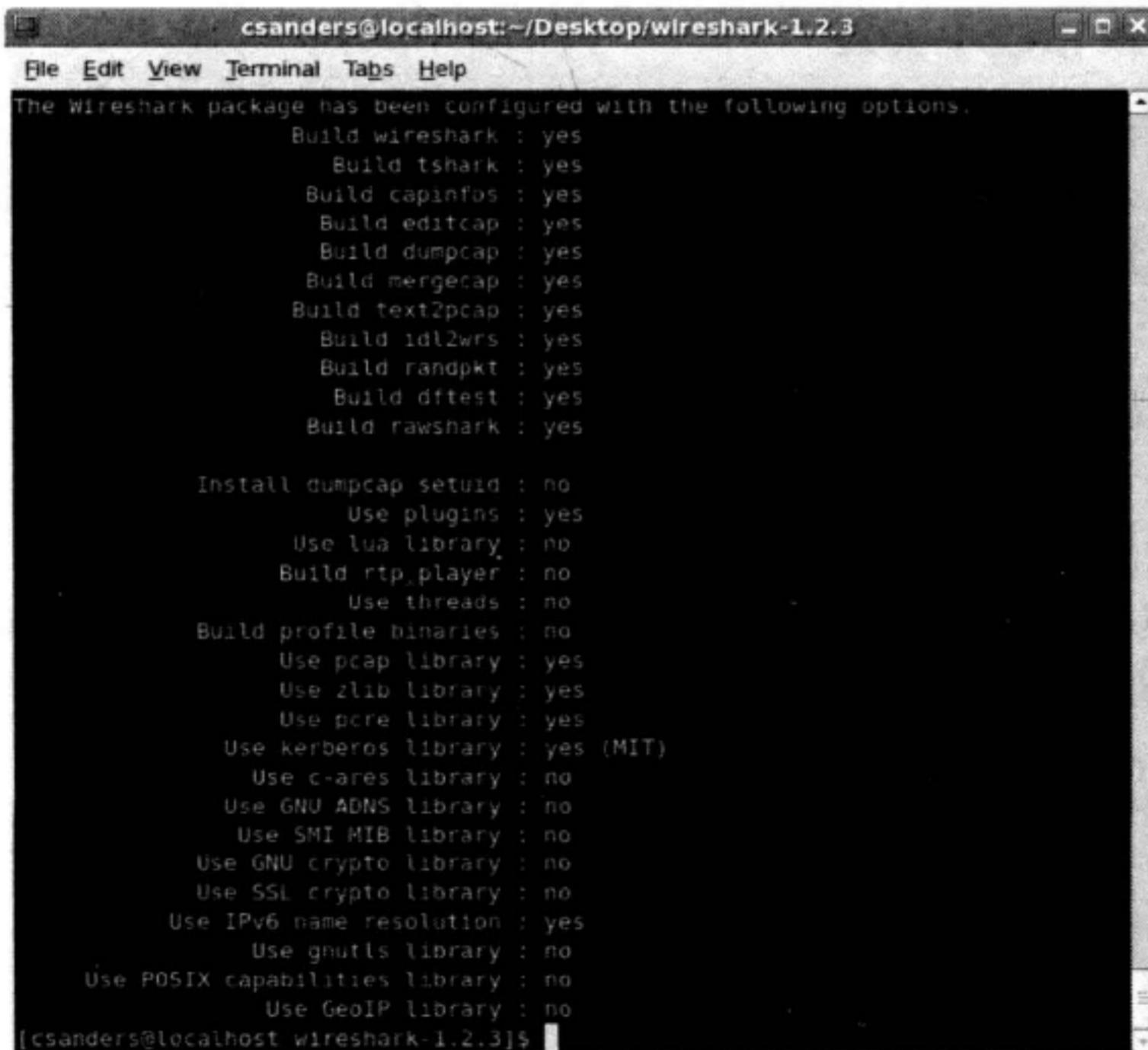
1. 从 Wireshark 网站下载源代码包。
2. 键入下面的命令将压缩包解压（将文件名替换成你所下载源代码包的名称）。

---

```
tar -jxvf wireshark-1.2.2.tar.bz2
```

---

3. 进入解压缩后创建的文件夹。
4. 以 root 级别的用户身份使用`./configure`命令配置源代码以便其能正常编译。如果你不想使用默认的设置，你可以这时指定安装选项。如果缺少相关软件支持，你应该会得到相关错误信息。如果安装成功了，你应该可以得到成功的提示，如图 3-3 所示。



```
csanders@localhost:~/Desktop/wireshark-1.2.3
File Edit View Terminal Tabs Help
The Wireshark package has been configured with the following options.
  Build wireshark : yes
    Build tshark : yes
    Build capinfos : yes
    Build editcap : yes
    Build dumpcap : yes
    Build mergecap : yes
    Build text2pcap : yes
    Build idl2wrs : yes
    Build randpkt : yes
    Build dftest : yes
    Build rawshark : yes

  Install dumpcap setuid : no
    Use plugins : yes
    Use lua library : no
    Build rtp_player : no
    Use threads : no
  Build profile binaries : no
    Use pcap library : yes
    Use zlib library : yes
    Use pcre library : yes
    Use kerberos library : yes (MIT)
    Use c-ares library : no
    Use GNU ADNS library : no
    Use SMI MIB library : no
    Use GNU crypto library : no
    Use SSL crypto library : no
    Use IPv6 name resolution : yes
    Use gnutls library : no
  Use POSIX capabilities library : no
    Use GeoIP library : no
[csanders@localhost wireshark-1.2.3]$
```

图 3-3 由`./configure`命令得到的成功输出

5. 输入 **make** 命令将源代码编译成二进制文件。
6. 使用 **make install** 命令完成最后的安装。

### 3.3.3 在 Mac OS X 系统中安装

在 Mac OS X 雪豹系统中安装 Wireshark 有一些注意事项,但安装并不困难,我在这里罗列了所需的安装步骤。

1. 从 Wireshark 网站上下载 DMG 包。
2. 将 Wireshark.app 复制到 Applications 文件夹。
3. 打开 Utilities 文件夹中的 Wireshark.app。
4. 在 Finder 中单击 Go, 选择 Go To Folder。输入`/usr/local/bin/`打开这个文件夹。
5. 将 Command Line 文件夹中的内容复制到`/usr/local/bin/`, 这时你需要输入你的密码以完成操作。
6. 在 Utilities 文件夹中, 将 ChmodBPF 文件夹复制到 StartupItems 文件夹, 这时你需要再次输入你的密码以完成该操作。安装过程至此宣告结束。

## 3.4 Wireshark 初步入门

当你成功地在你的系统中装好了 Wireshark, 你就可以开始熟悉它了。当你终于打开了这个功能强大的数据包嗅探器, 却会发现你什么都看不见!

好吧, Wireshark 在刚打开的时候确实不太好玩, 只有在拿到一些数据之后事情才会变得有趣起来。

### 3.4.1 第一次捕获数据包

为了能让 Wireshark 得到一些数据包, 你可以开始你的第一次数据包捕获实验了。你可能会想:“当网络什么问题也没有的时候, 怎么能捕获数据包呢?”

首先, 网络总是有问题的。如果你不相信, 那么你去给你网络上所有的用户发一封邮件, 告诉他们一切都工作得非常好。

第二, 做数据包分析并不一定要等到有问题的时候再做。事实上, 大多数的数据包分析员在分析没有问题的网络流量上花的时间要比解决问题的时候

多。为了能高效地解决网络问题，你也同样需要得到一个基准来与之对比。举例来说，如果你想通过分析网络流量来解决关于 DHCP 的问题，你至少需要知道 DHCP 在正常工作时的数据流是什么样子的。

更广泛地讲，为了能够发现日常网络活动的异常，你必须对日常网络活动的情况有所掌握。当你的网络正常运行时，你以此作为基准，就能知道网络流量在正常情况下的样子。

闲言少叙，让我们来捕获一些数据包吧！

1. 打开 Wireshark。
2. 从主下拉菜单中选择 **Capture**，然后是 **Interface**。

这时你应该可以看到一个对话框，里面列出了你可以用来捕获数据包的各种设备，以及它们的 IP 地址。

3. 选择你想要使用的设备，如图 3-4 所示，然后单击 **Start**，或者直接单击欢迎画面中 Interface List 下的某一个设备。随后数据就会在窗口中呈现出来。

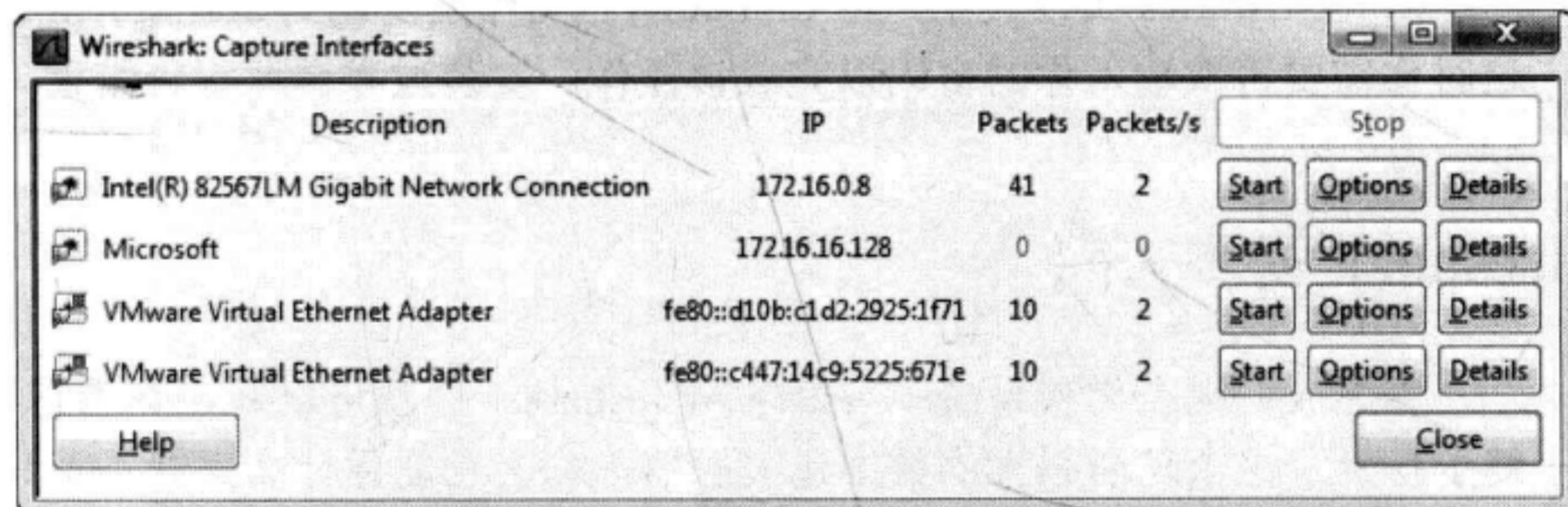


图 3-4 选择你想要进行数据包捕获的端口

4. 等上一分钟左右，当你打算停止捕获并查看你的数据时，在 Capture 的下拉菜单中单击 **Stop** 按钮即可。

当你做完了以上步骤并完成了数据包的捕获，Wireshark 的主窗口中应该已经呈现了相应的数据，但此时你可能已经对那些数据的规模感到头疼，这也就是为什么我们把 Wireshark 一整块的主窗口进行拆分的原因。

### 3.4.2 Wireshark 主窗口

Wireshark 的主窗口是将你所捕获的数据包显示或拆分成更容易使人理解的方式的地方，也将是你花费时间最多的地方。我们使用刚刚捕获的数据包来介

绍一下 Wireshark 的主窗口，如图 3-5 所示。

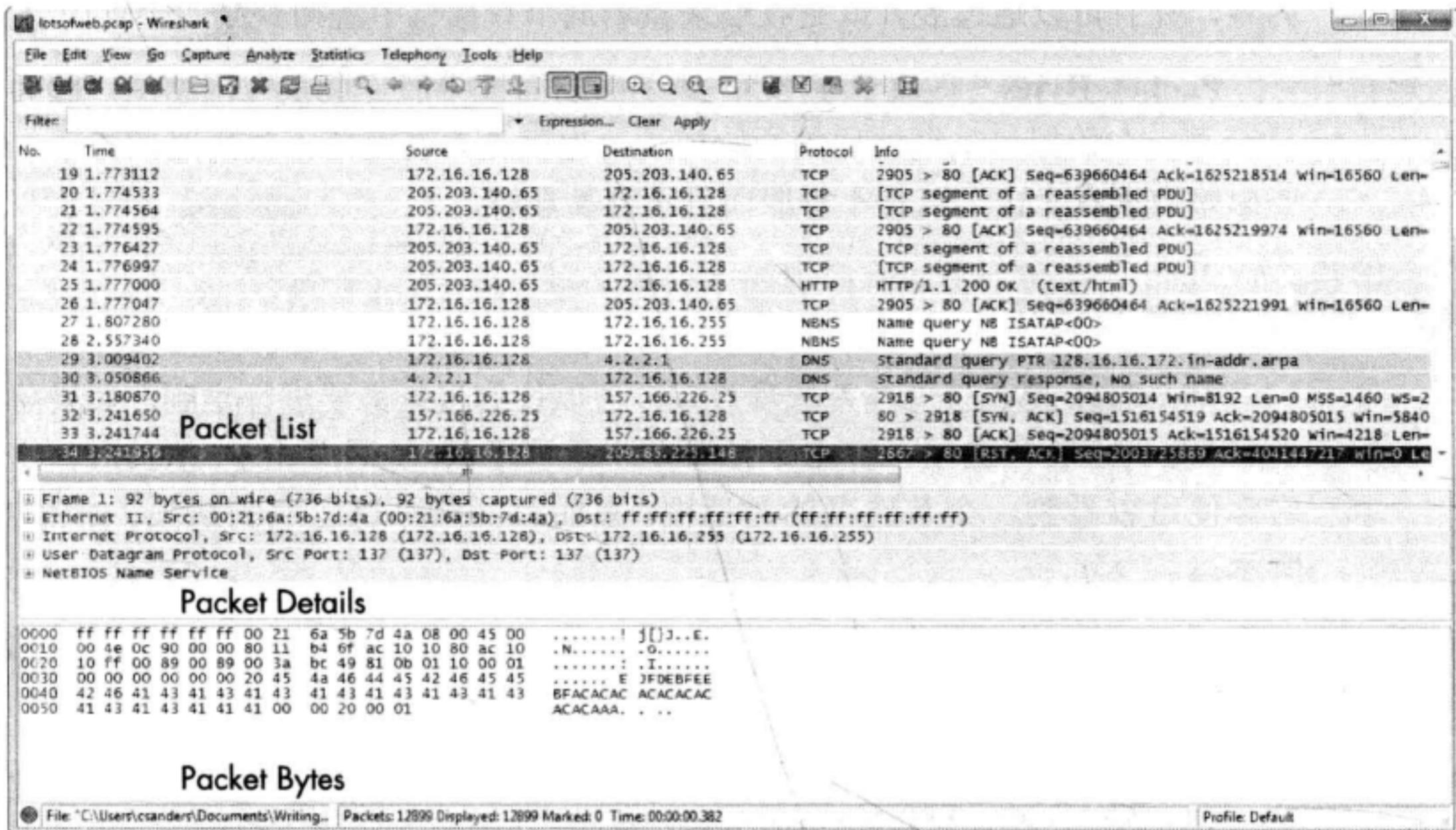


图 3-5 Wireshark 主窗口的设计使用了 3 个面板

主窗口的 3 个面板相互有着联系。如果希望在 **Packet Details** 面板中查看一个单独的数据包的具体内容，你必须现在 **Packet List** 面板中单击选中那个数据包。在你选中了数据包之后，你可以通过在 **Packet Details** 面板中选中数据包的某个字段，从而在 **Packet Bytes** 面板中查看相应字段的字节信息。

#### 注意

在图 3-5 中的 **Packet List** 面板中列出了几种不同的协议，但这里并没有使用不同的层次来对不同的协议进行视觉上的区分，所有的数据包都是按照其在链路上接收到的顺序排列的。

下面介绍了每个面板的内容。

**Packet List (数据包列表)**: 最上面的面板用表格显示了当前捕获文件中的所有数据包，其中包括了数据包序号、数据包被捕获的相对时间、数据包的源地址和目标地址、数据包的协议以及在数据包中找到的概况信息等列。

#### 注意

当文中提到流量的时候，我通常是指 **Packet List** 面板中所有呈现出来的数据包，而当特别提到 DNS 流量时，我指的是 **Packet List** 面板中 DNS 协议的数据包。

**Packet Details** (数据包细节): 中间的面板分层次地显示了一个数据包中的内容，并且可以通过展开或是收缩来显示这个数据包中所捕获到的全部内容。

**Packet Bytes** (数据包字节): 最下面的面板可能是最令人困惑的，因为它显示了一个数据包未经处理的原始样子，也就是其在链路上传播时的样子。这些原始数据看上去一点都不舒服而且不容易理解。

### 3.4.3 Wireshark 首选项

Wireshark 提供了一些首选项设定，可以让你根据需要进行定制。如果需要设定 Wireshark 首选项，在主下拉菜单中选择 **Edit** 然后单击 **Preferences**，然后你便可以看到一个首选项的对话框，里面有一些可以定制的选项，如图 3-6 所示。

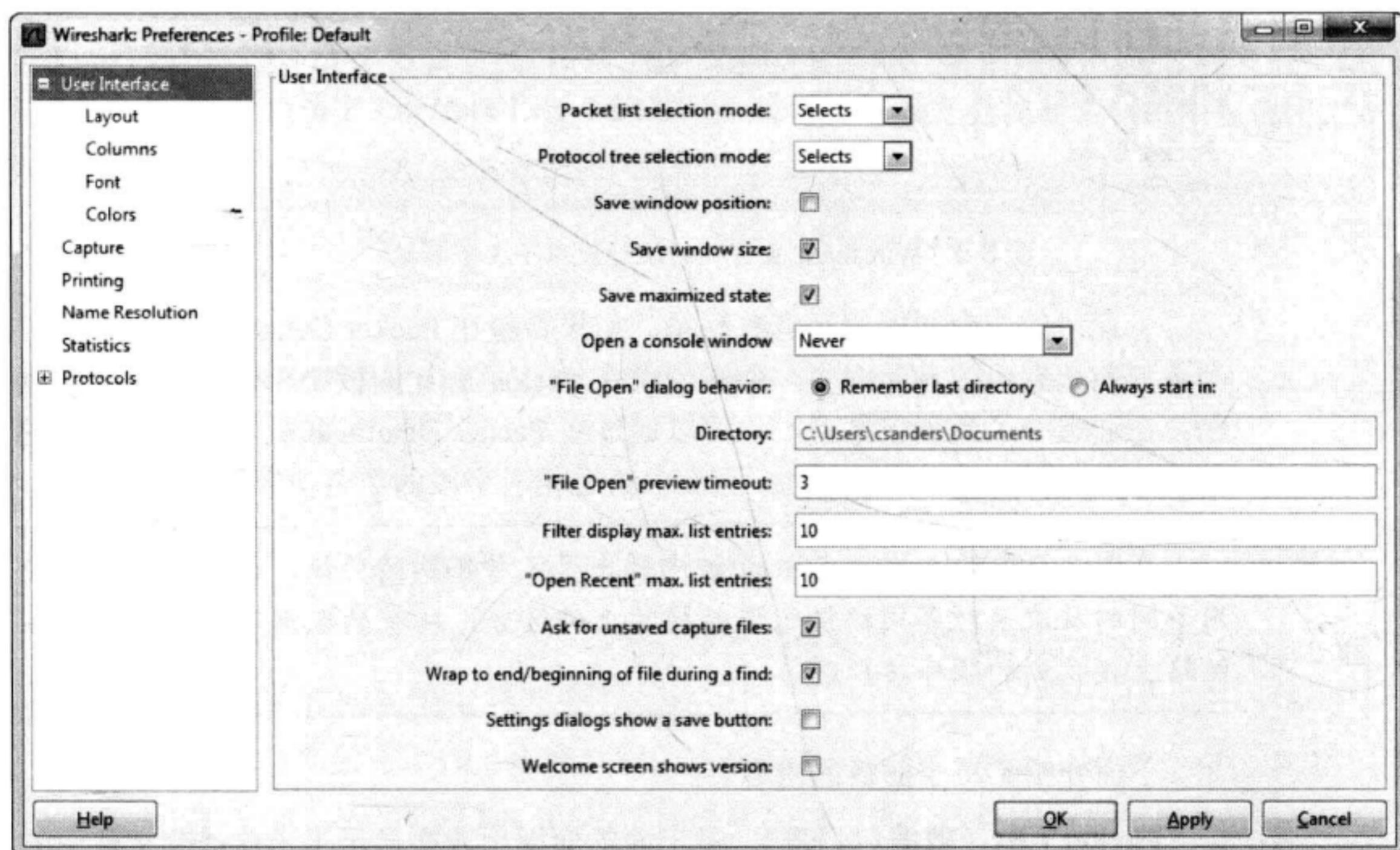


图 3-6 你可以使用 Preferences 对话框中的选项自定义 Wireshark 的配置

Wireshark 首选项分为 6 个主要部分。

**User Interface** (用户接口): 这些选项决定了 Wireshark 将如何显示数据。你可以根据你的个人喜好对大多数选项进行调整，比如是否保存窗口位置、3

个主要窗口的布局、滚动条的摆放、Packet List 面板中列的摆放，以及显示捕获数据的字体、前景色和背景色等。

**Capture**（捕获）：这些选项可以让你对你捕获数据包的方式进行特殊的设定，比如你默认使用的设备、是否默认使用混杂模式、是否实时更新 Packet List 面板等。

**Printing**（打印）：这个部分中的选项可以让你对 Wireshark 如何打印你的数据进行各种特殊的设定。

**Name Resolutions**（名字解析）：通过这些设定，你可以开启 Wireshark 将地址（包括 MAC、网络以及传输名字解析）解析成更加容易分辨的名字这一功能，并且可以设定可以并发处理名字解析请求的最大数目。

**Statistics**（统计）：这一部分提供了一些 Wireshark 中统计功能的设定选项。

**Protocols**（协议）：这个部分中的选项与捕捉和显示各种 Wireshark 能够解码的数据包有关。并不是每一个协议都有配置选项，但是一些协议的某些选项则可以进行更改。除非你有特殊的原因去修改这些选项，否则最好保持它们的默认值。

### 3.4.4 数据包彩色高亮

如果你像我一样喜欢五颜六色的物体，那么你应该会对 Packet List 面板中那些不同的颜色感到兴奋。如图 3-7 所示（尽管图示是黑白的，但你应该可以理解），那些颜色看上去就像是随机分配给每一个数据包的，但其实并不是这样的。

| No. | Time     | Source         | Destination    | Protocol | Info   |
|-----|----------|----------------|----------------|----------|--|
| 28  | 2.557340 | 172.16.16.128  | 172.16.16.255  | NBNS     | Name query NB ISATAP<00>                                       |
| 29  | 3.009402 | 172.16.16.128  | 4.2.2.1        | DNS      | Standard query PTR 128.16.16.172.in-addr.arpa                  |
| 30  | 3.050866 | 4.2.2.1        | 172.16.16.128  | DNS      | Standard query response, no such name                          |
| 31  | 3.180870 | 172.16.16.128  | 157.166.226.25 | TCP      | 2918 > 80 [SYN] Seq=2094805014 Win=8192 Len=0 MSS=1460 WS=2    |
| 32  | 3.241650 | 157.166.226.25 | 172.16.16.128  | TCP      | 80 > 2918 [SYN, ACK] Seq=1516154519 Ack=2094805015 Win=5840    |
| 33  | 3.241744 | 172.16.16.128  | 157.166.226.25 | TCP      | 2918 > 80 [ACK] Seq=2094805015 Ack=1516154520 Win=4218 Len=0   |
| 34  | 3.241956 | 172.16.16.128  | 209.85.225.148 | TCP      | 2867 > 80 [RST, ACK] Seq=2003725889 Ack=4041447217 Win=0 Len=0 |
| 35  | 3.242063 | 172.16.16.128  | 209.85.225.118 | TCP      | 2866 > 80 [RST, ACK] Seq=1479685855 Ack=4097855950 Win=0 Len=0 |

图 3-7 Wireshark 的彩色高亮有助于快速标识协议

每一个数据包的颜色都是有讲究的，这些颜色对应着数据包使用的协议。举例来说，所有的 DNS 流量都是蓝色的，而 HTTP 流量都是绿色的。将数据包进行彩色高亮，可以让你很快地将不同协议的数据包分开，而不需要对每个数据包都查看 Packet List 面板中的协议列。你会发现这样在浏览较大的捕获文件时，可以极大地节省时间。

如图 3-8 所示，Wireshark 通过 Coloring Rules（着色规则）窗口可以很容易地查看每个协议所对应的颜色。如果想要打开这个窗口，在主下拉菜单中选择 View 然后单击 **Coloring Rules**。

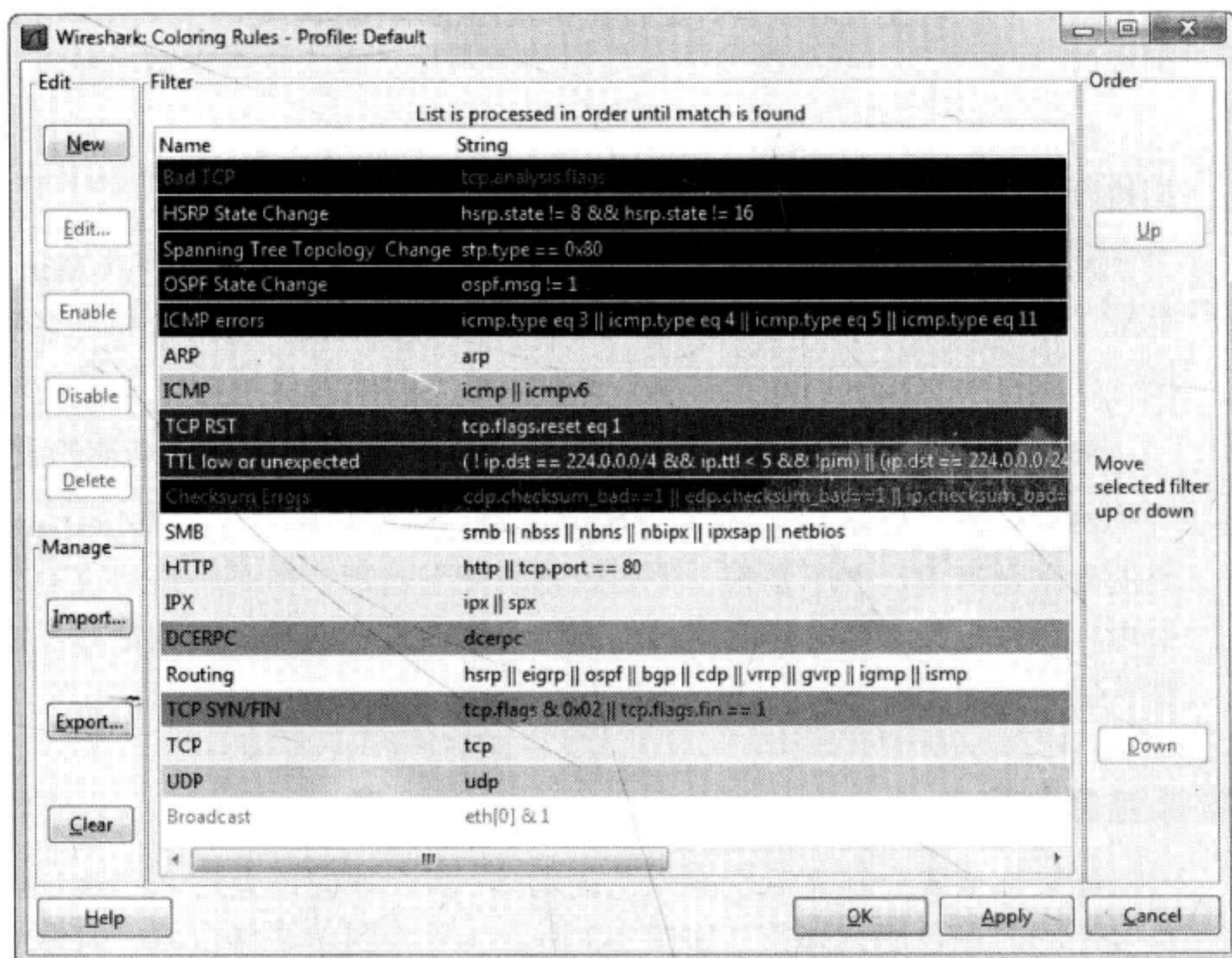


图 3-8 你可以在 Coloring Rules 窗口中查看并更改数据包的着色

你可以创建你自己的着色规则，或者修改已有设置。举例来说，使用下列步骤可以将 HTTP 流量绿色的默认背景改成淡紫色。

1. 打开 Wireshark，并且打开 Coloring Rules 窗口（View>**Coloring Rules**）。
2. 在着色规则的列表中找到 HTTP 着色规则并单击选中。
3. 单击 **Edit** 按钮，你会看到一个 Edit Color Filter 窗口，如图 3-9 所示。
4. 单击 **Background Color** 按钮。
5. 使用颜色滚轮选择一个你希望使用的颜色，然后单击 **OK**。
6. 再次单击 **OK** 来应用改变，并回到主窗口。主窗口此时应该已经重载，并使用了更改过的颜色样式。

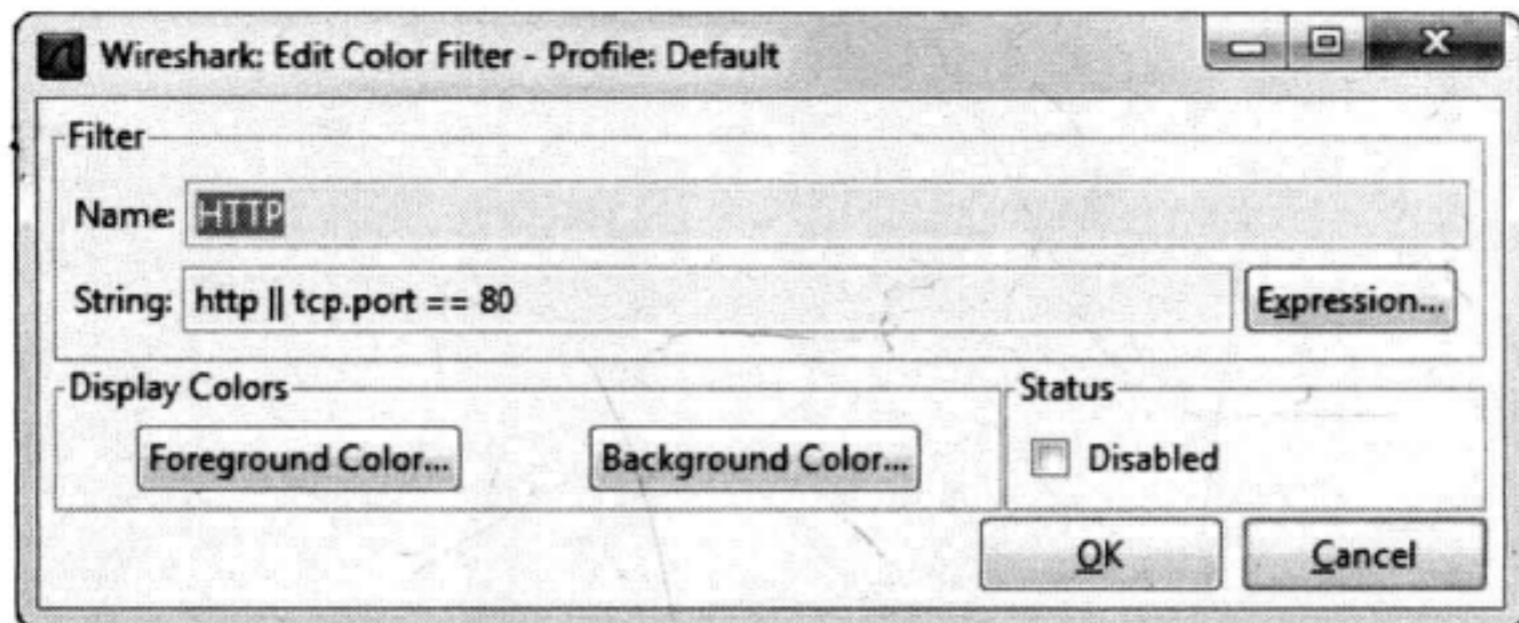


图 3-9 在编辑着色过滤器时，前景色和背景色都可以进行更改

当你在网络上使用 Wireshark 时，你可能会发现你处理某些协议比其他协议要多。这时彩色高亮的数据包就能让你工作得更加方便。举例来说，如果你觉得你的网络上有一个恶意的 DHCP 服务器在分发 IP，你可以简单地修改 DHCP 协议的着色规则，使其呈现明黄色（或者其他易于辨认的颜色）。这可以使你能够更快地找出所有 DHCP 流量，并让你的数据包分析工作更有效率。

你还可以通过基于你自己定制的过滤器创建着色规则，来扩展这些着色规则的用途。

现在你的 Wireshark 应该已经安装好并运行起来了，你已经准备好进行数据包的分析了。在下一章中，我们将详细讲述如何处理你所捕获的数据包。



# 第4章

## 玩转捕获数据包



现在，你已经了解了 Wireshark，并且也准备好进行数据包的捕获和分析了。在这一章中，你将会学习如何使用捕获文件、分析数据包以及时间显示格式。我们也会介绍更多捕获数据包所用到的高级选项，并进入过滤器的世界。

### 4.1 使用捕获文件

当你进行数据包分析的时候，你会发现很大一部分分析工作是在你捕获数据包之后进行的。通常情况下，你会在不同时间进行多次捕获，将结果保存下来，然后一起进行分析，所以 Wireshark 可以让你保存你的捕获文件，以便之后分析，你也可以将多个捕获文件进行合并。

### 4.1.1 保存和导出捕获文件

如果想要保存捕获的数据包，选择 **File->Save As**，之后你应该就能看见 Save File As 对话框，如图 4-1 所示。对话框会询问你想要保存捕获的数据包的位置，以及你希望保存的格式。如果你不选择一个文件格式，那么 Wireshark 会默认使用.pcap 文件格式。

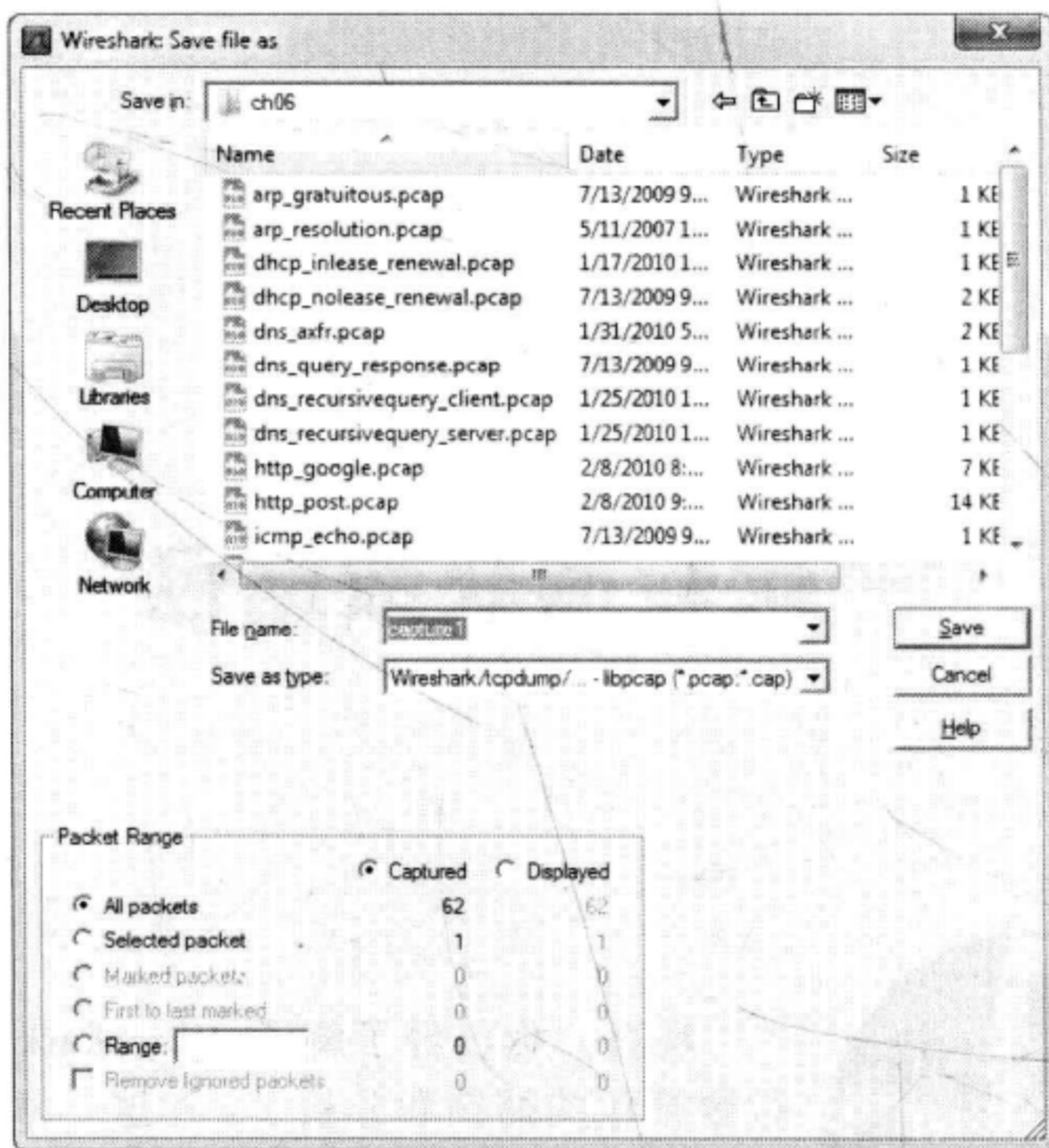


图 4-1 Save File As 对话框可以让你保存数据包捕获

Save File As 对话框的一个更强大的功能是你可以指定需要保存的数据包范围。这是一个让“胖”捕获文件变“瘦”的好方法。你可以选择只保存一定序号范围内的数据包、标记了的数据包，或者经过过滤器筛选后显示出来的数据包等（标记的数据包和过滤器会在这一章后面进行讨论）。

你可以将 Wireshark 捕获的数据导出到几种不同格式的文件中，以便于在其他工具中查看，或是导入到其他的数据包分析工具中。这些格式包括文本文件、PostScript、逗号分隔值（CSV）和 XML。如果想要导出捕获的数据包，选择 **File->Export**，并选择你想要导出的文件格式。你将会看到一个包含着相应文件

格式选项的 Save As 对话框。

### 4.1.2 合并捕获文件

某些类型的分析工作需要能够合并多个捕获文件，一般在比较两个数据流或者合并单独捕获的同一数据流量时比较常见。

如果想要合并捕获文件，先打开一个你想要合并的文件，然后选择 **File->Merge**，这时便会弹出 Merge with Capture File 的对话框，如图 4-2 所示。选择一个你希望合并到当前文件的另一个文件，然后选择你希望进行合并的方式。你可以将所选文件添加到当前文件的前面或者后面，也可以按照它们时间戳的先后进行合并。

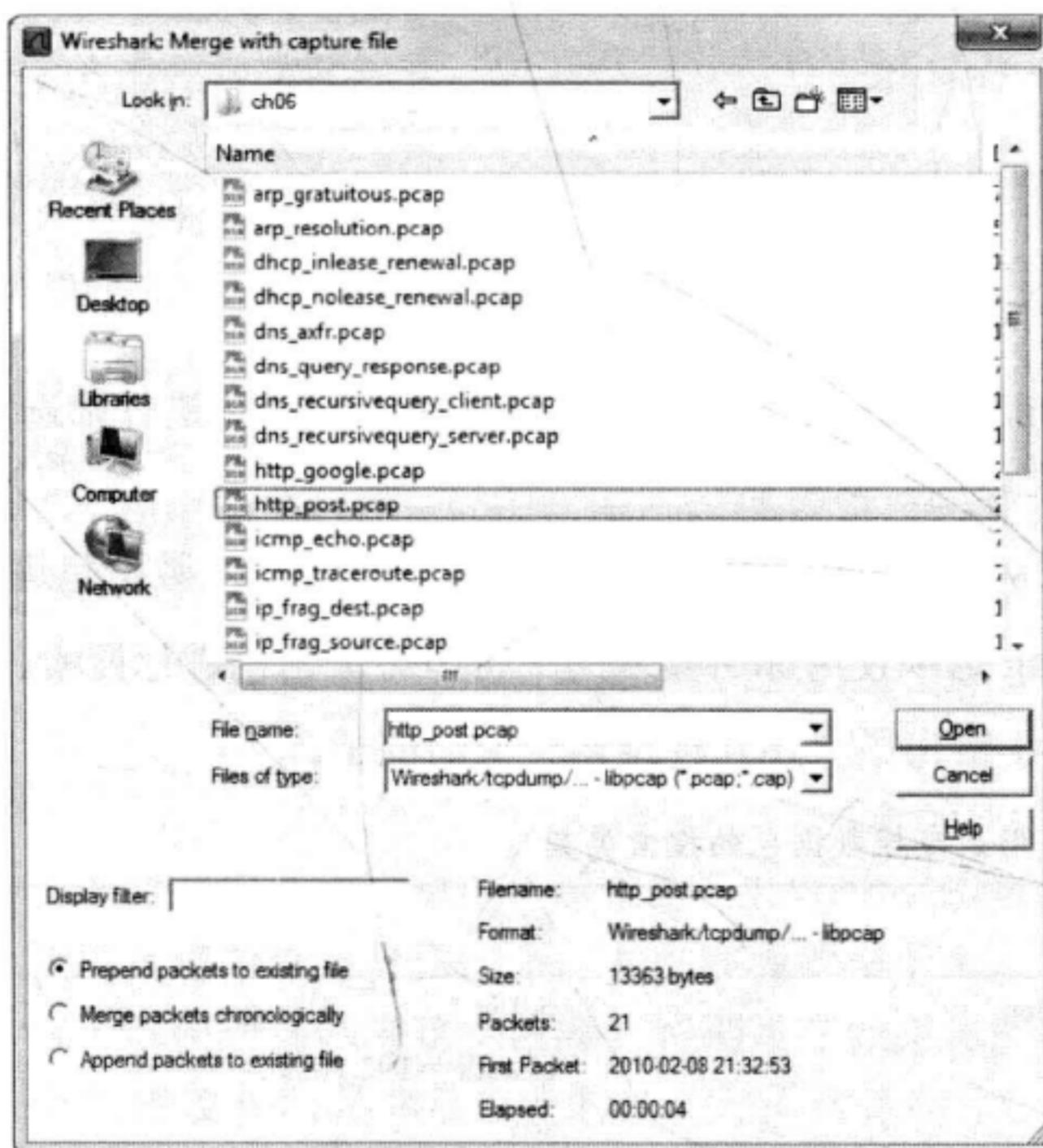


图 4-2 Merge with Capture File 对话框可以让你合并两个捕获文件

## 4.2 分析数据包

你肯定会遇到处理大规模数据包的情形。当这些数据包上千甚至上万时，你需要更高效地在这些数据包中进行查找。出于这个目的，Wireshark 可以让

你将符合一定条件的数据包进行标记，或者为了更容易参考而打印出来。

### 4.2.1 查找数据包

如果想要找到符合特定条件的数据包，按 Ctrl-F 打开 Find Packet 对话框，如图 4-3 所示。

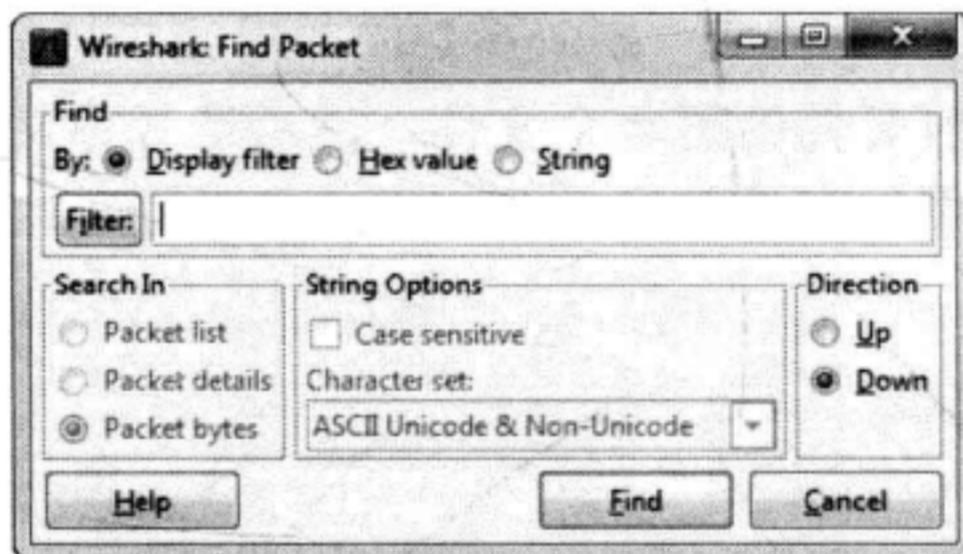


图 4-3 在 Wireshark 中根据条件查找数据包

这个对话框为查找数据包提供了 3 个选项。

- Display filter 选项可以让你通过输入表达式进行筛选，并只找出那些满足该表达式的数据包。
- Hex value 选项使用你所输入的十六进制数对数据包进行搜索。
- String 选项使用你所输入的字符串对数据包进行搜索。

表 4-1 给出了上述几种搜索类型的例子。

表 4-1 用来查找数据包的搜索类型

| 搜索类型           | 例子                                    |
|----------------|---------------------------------------|
| Display filter | not ip<br>ip.addr==192.168.0.1<br>arp |
| Hex value      | 00:ff<br>ff:ff<br>00:AB:B1:f0         |
| String         | Workstation1<br>UserB<br>domain       |

一些其他选项包括选择你所希望进行搜索的窗口、使用的字符集，以及搜索方向。你可以通过指定搜索的面板、设定使用的字符集，以及使搜索对大小写敏感，来对你的字符搜索进行扩展。

当你选好选项并在文本框中输入搜索关键词之后，单击 **Find**，就会找到满足该关键词的第一个数据包。如果想要找到下一个匹配的数据包，按 **Ctrl-N**，想要找到前一个，按 **Ctrl-B**。

## 4.2.2 标记数据包

在找到那些符合搜索条件的数据包之后，你可以根据需要进行标记。举例来说，可能你希望将那些需要分开保存的数据包标记出来，或者根据颜色快速地查找它们。如图 4-4 所示，被标记的数据包会以黑底白字显示（你也可以仅仅将标记了的数据包选择出来，然后作为数据包捕获保存下来）。

| No. | Time     | Source         | Destination    | Protocol | Info  |
|-----|----------|----------------|----------------|----------|---|
| 1   | 0.000000 | 172.16.0.8     | 157.166.224.25 | TCP      | 3426 > 80 [SYN] Seq=1745901259 Win=8192 Len=0 MSS=1460 WS=2                     |
| 2   | 0.024063 | 157.166.224.25 | 172.16.0.8     | TCP      | 80 > 3426 [SYN, ACK] Seq=2324576412 Ack=1745901260 Win=5840 Len=0 MSS=1460 WS=7 |

图 4-4 被标记的数据包将在你的屏幕上以高亮显示。在这个例子中，数据包 1 被标记并且显示为深色

如果你想要标记一个数据包，右击 **Packet List** 面板，并在弹出菜单中选择 **Mark Packet**，或者在 **Packet List** 面板中选中一个数据包，然后按 **Ctrl-M**。如果想取消对一个数据包的标记，再按一次 **Ctrl-M** 就可以将其取消。在一次捕获中，你想标记多少个数据包都可以。如果你想要在标记的数据包间前后切换，分别按 **SHIFT-CTRL-N** 和 **SHIFT-CTRL-B** 即可。

## 4.2.3 打印数据包

尽管大多数分析都会在电脑屏幕前进行，但你仍然可能需要将捕获结果打印出来。我经常将数据包打印出来，并贴在我的桌子上，这样我在做其他分析的时候，就可以快速地参考这些内容。特别是在做报告的时候，能够将数据包打印成一个 PDF 文件将是非常方便的。

如果需要打印捕获的数据包，在主菜单中选择 **File->Print** 打开 Print 对话框。你可以在图 4-5 中看到 Print 对话框的样子。

你可将选中的数据以文本或者 PostScript 的格式打印或者输出到一个文件。与 **Save File As** 对话框相似，你可以按一定范围打印数据包，比如被标记的数据包，或者作为过滤器筛选结果显示出来的数据包。对于每一个数据包，你也可以在 Wireshark 3 个主面板中选择打印对象。在你做好了这些选择之后，

单击 Print。

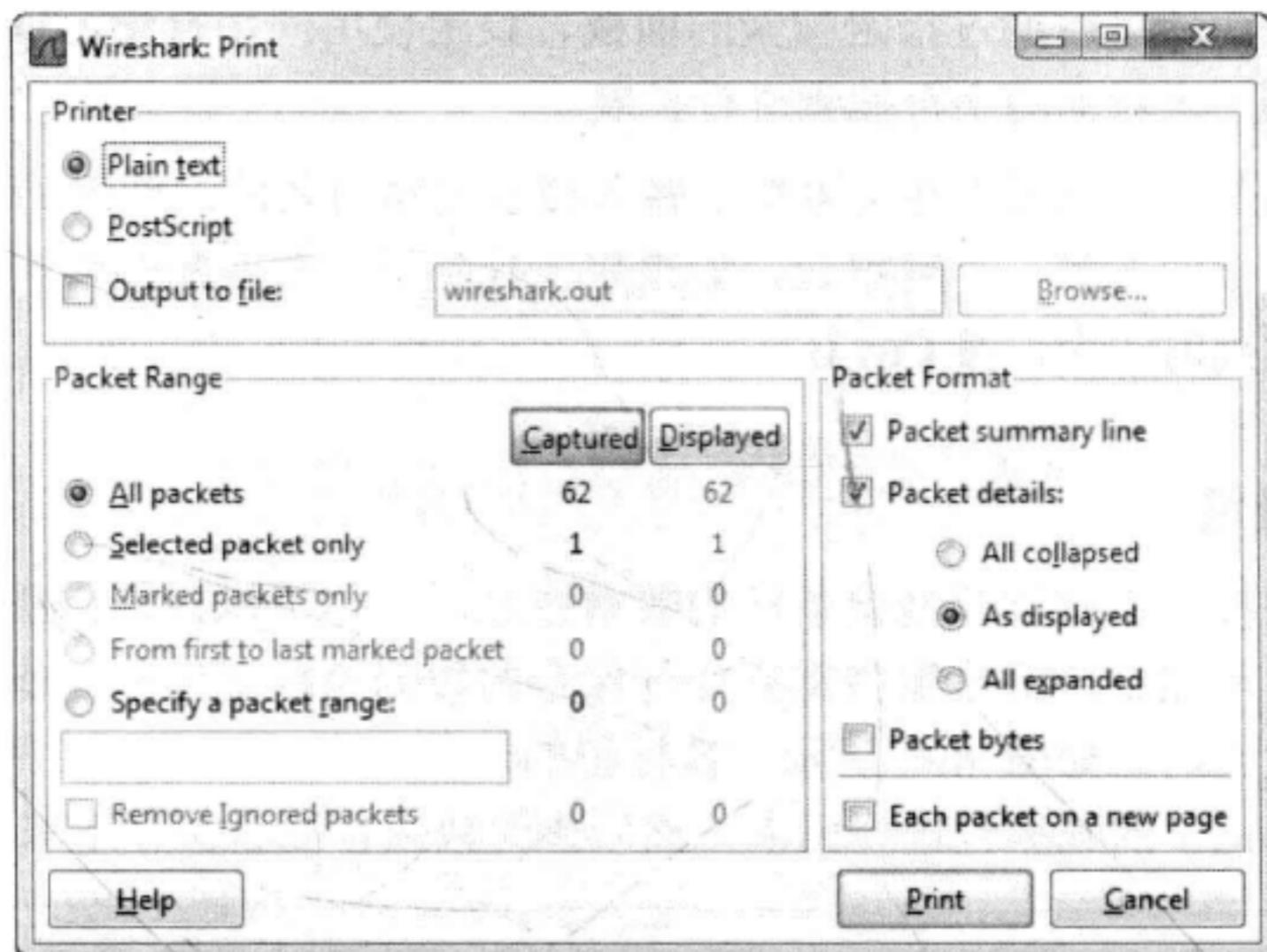


图 4-5 Print 对话框可以让你打印所指定的数据包

## 4.3 设定时间显示格式和相对参考

时间在数据包的分析中格外重要。所有在网络上发生的事情都是与时间息息相关的，并且你几乎需要在每一个捕获文件中检查时间规律以及网络延迟。Wireshark 意识到时间的重要性，并提供了一些相关的选项以供设定。在这一节中，我们将介绍时间的显示格式和相对参考。

### 4.3.1 时间显示格式

Wireshark 所捕获的每一个数据包都会由操作系统给予一个时间戳。Wireshark 可以显示这个数据被捕获时的绝对时间戳，也可以是与上一个被捕获的数据包或是捕获开始及结束的相对时间戳。

与时间显示相关的选项可以在主菜单中的 View 菜单中找到。如图 4-6 所示，Time Display Format 部分可以让你设置时间的显示格式和精度。格式选项可以让你选择不同的格式，而精度选项可以让你将精度设定为自动或者手动，比如秒、毫秒、微秒等。在本书后面，我们将调整这些设置，所以你现在就需要熟悉它。

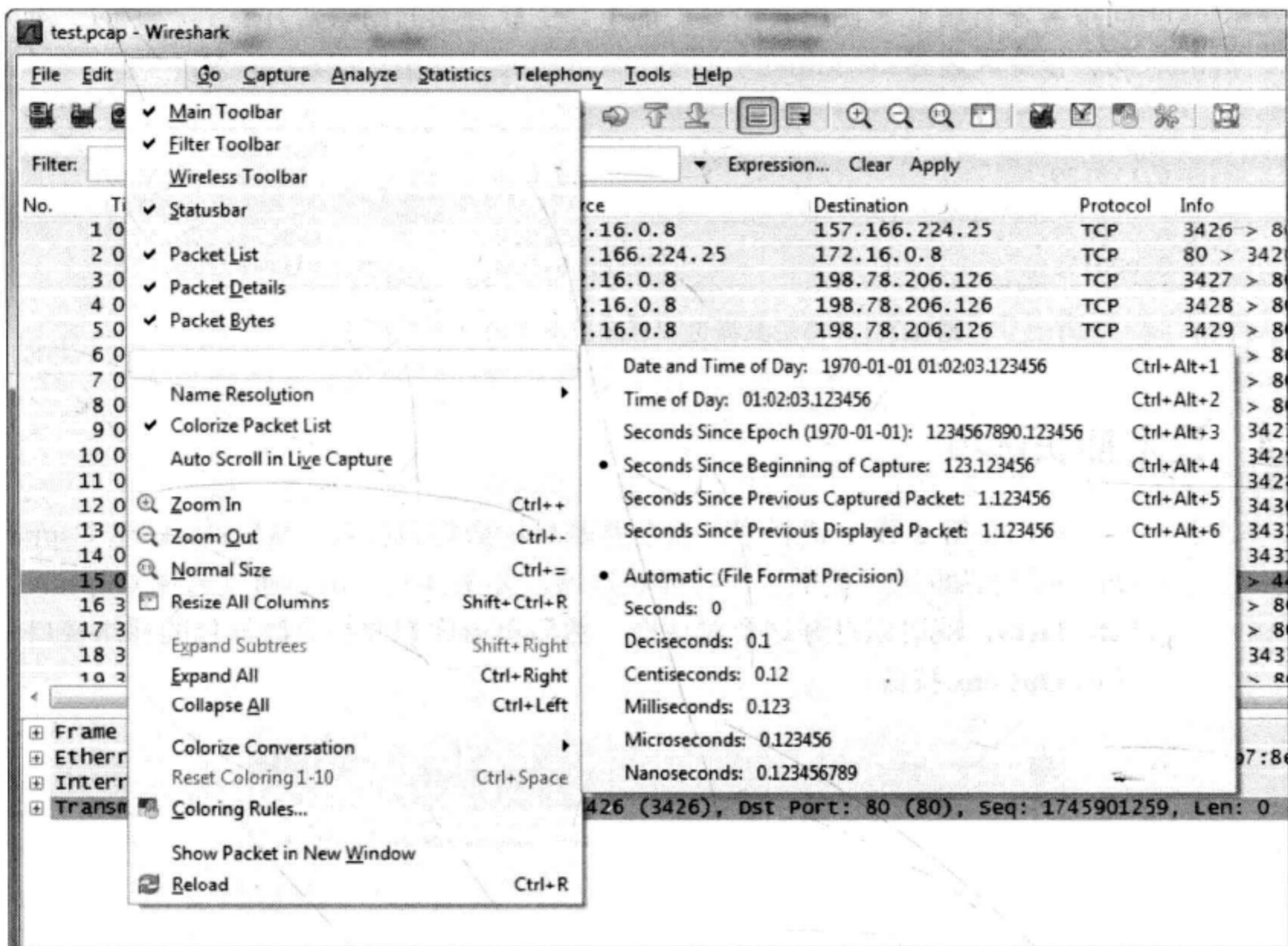


图 4-6 多种可用的时间显示格式

### 4.3.2 数据包的相对时间参考

数据包的相对时间参考，可以让你以一个数据包作为基准，而之后的数据包都以此计算相对时间戳。当一系列的顺序事件不是在捕获开始时被触发，而是在中间某个地方被触发，这个功能会变得非常好用。

如果希望将某一个数据包设定为时间参考，在 Packet List 面板中选择作为相对参考的数据包，然后在主菜单中选择 **Edit->Set Time Reference**。如果希望取消一个数据包的相对时间参考，选择那个数据包，然后将 **Edit->Set Time Reference** 设定关掉。

当你将一个数据包设定为时间参考之后，Packet List 面板中这个数据包的 Time 列就会显示为\*REF\*，如图 4-7 所示。

只有当捕获的时间显示格式设定为相对于捕获开始的时间，设定数据包时

间参考才有用处。使用其他设定都不会生成有用的结果，并且其产生的一堆时间会很令人迷惑。

| No. | Time     | Source     | Destination    |
|-----|----------|------------|----------------|
| 4   | 0.118129 | 172.16.0.8 | 198.78.206.126 |
| 5   | *REF*    | 172.16.0.8 | 198.78.206.126 |
| 6   | 0.000077 | 172.16.0.8 | 198.78.206.126 |
| 7   | 0.000153 | 172.16.0.8 | 198.78.206.126 |

图 4-7 开启了数据包相对时间参考的一个数据包

## 4.4 设定捕获选项

我们在第 3 章中进行了一次非常基础的数据包捕获。Wireshark 在 Capture Options 对话框中，提供了一些捕获选项，如图 4-8 所示。通过选择 **Capture->Interfaces**，就可以打开这个对话框，然后单击你想要捕获数据包的网络接口旁边的 **Options** 按钮。

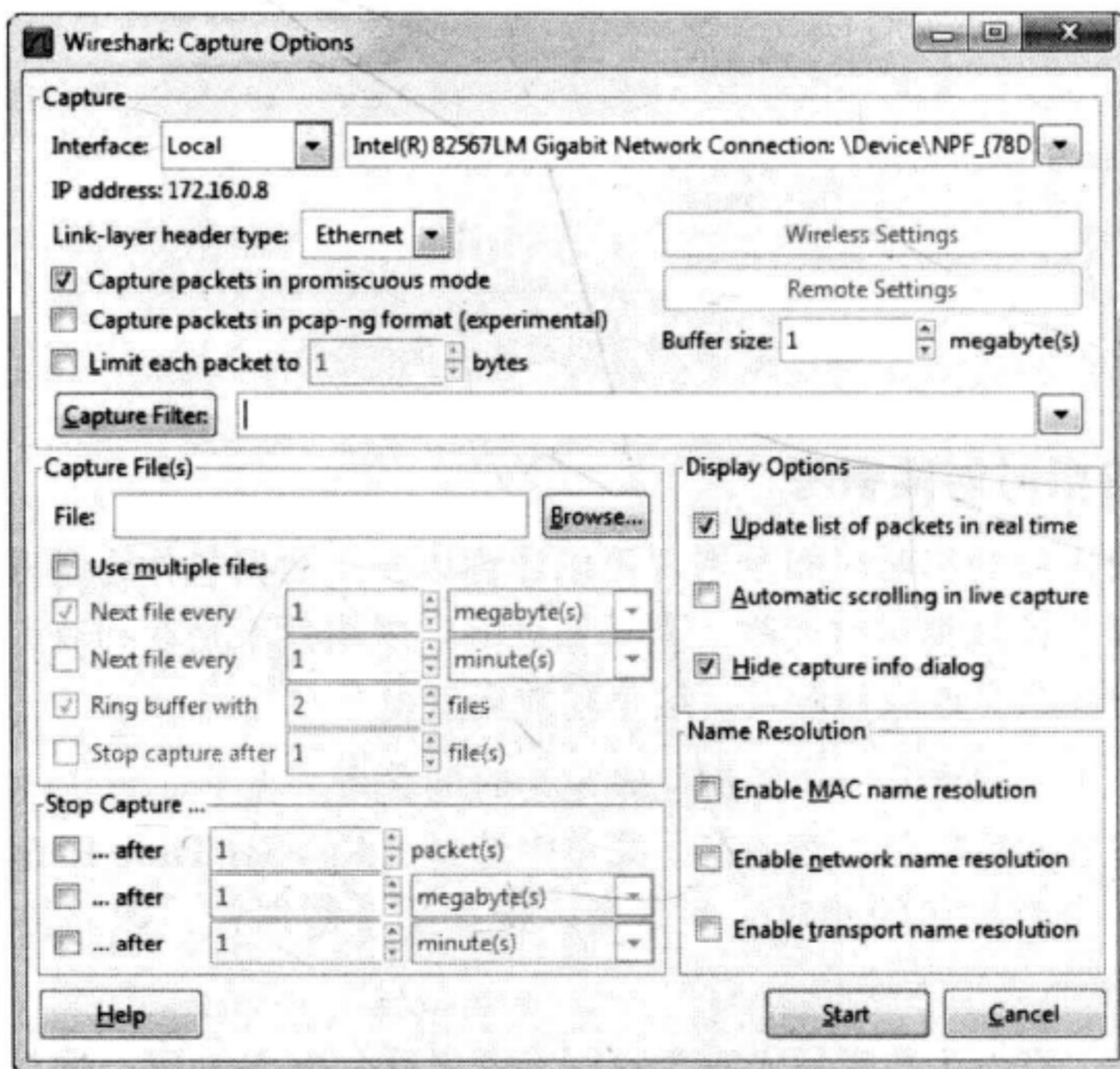


图 4-8 Capture Options 对话框

Capture Options 对话框提供给了你意想不到的各种选项，为的就是能在

进行数据包捕获时给你更多的灵活性。这些选项分为 Capture、Capture Files、Stop Capture、Display Options 和 Name Resolutions 5 个部分，在此将逐个予以说明。

#### 4.4.1 捕获设定

你可以在 Capture 部分中的 Interface 下拉菜单中设定网络接口，其中右边的下拉菜单显示了所有可用的捕获接口，在左边的下拉菜单中，你可以设定接口是本地的还是远程的。你选中接口的 IP 地址会在这个下拉菜单的正下方显示出来。

对话框左侧的 3 个选框可以让你开启或关闭混杂模式（默认为开启）、以实验性的 pcap-ng 格式来捕获数据包，以及按字节数限制每个捕获数据包的大小。

Capture 部分右侧的按钮可以让你进行无线和远程的设定（如果可用）。在这些的下面是缓冲区大小的选项，不过这些选项只在微软 Windows 系统中可用。你可以设定在写入磁盘之前可以存储在内核缓冲区内所捕获数据的大小（这个选项一般只在你注意到很多丢包发生时才会进行修改，而通常情况下是不会动它的）。Capture Filter 选项可以让你指定一个捕获过滤器。

#### 4.4.2 捕获文件设定

Capture File(s)部分可以让你自动将捕获的数据包直接存储到文件中，而不是先捕获再存储。这样你在管理数据包的储存上有了更大的灵活性。你可以将它们存储为单一文件或者一个文件集，甚至还可以使用环状缓冲来控制创建的文件数量。如果你想开启这个功能，你需要在 File 文本框中输入完整的文件路径和文件名。

当你所捕获的流量很大，或者捕获的持续时间很长时，使用文件集进行存储被证明是很好用的。文件集是通过特定条件进行分割的一组文件的集合。如果将数据存到文件集中，在这里需勾选 Multiple Files 选项。

Wireshark 使用基于文件大小或是时间条件的各种触发器，对文件集进行管理。如果想启用这些功能，在 Next File Every 选项旁边打勾（最上面的那个是文件大小的触发器，下面的那个是基于时间的触发器），然后输入触发条件的数值和单位。举例来说，你可以创建一个每捕获 1MB 或者一分钟的流量就创建一个新文件的触发器，如图 4-9 所示。

这些选项也可以进行组合，比如你可以创建两个触发器，每当文件超过 1MB 或者每过 1min（以先满足的条件为准），就创建一个新文件。

| Name                         | Date modified      |
|------------------------------|--------------------|
| Capture_00001_20091115155100 | 11/15/2011 3:51 PM |
| Capture_00002_20091115155200 | 11/15/2011 3:52 PM |
| Capture_00003_20091115155300 | 11/15/2011 3:53 PM |
| Capture_00004_20091115155400 | 11/15/2011 3:54 PM |
| Capture_00005_20091115155500 | 11/15/2011 3:56 PM |
| Capture_00006_20091115155600 | 11/15/2011 3:56 PM |
| Capture_00007_20091115155700 | 11/15/2011 3:57 PM |
| Capture_00008_20091115155800 | 11/15/2011 3:58 PM |
| Capture_00009_20091115155900 | 11/15/2011 3:59 PM |
| Capture_00010_20091115160000 | 11/15/2011 4:00 PM |

图 4-9 Wireshark 以 1min 为间隔所创建的文件集

**Ring Buffer With** 选项可以让你使用环状缓冲创建一个文件集。这是一个应用在 Wireshark 写入多个文件时的一个先进先出 (FIFO) 算法。尽管环状缓冲在信息科学中有着多种意思，但在这里我们指的是当文件集中最后一个文件写完后并且仍有数据需要写入磁盘时，第一个文件会被覆盖。你可以勾选这个选项，然后指定你希望循环使用的文件的最大数目。举例来说，当你在你的捕获中使用多文件存储时，每小时创建一个文件，然后你将环状缓冲设为 6，那么第 6 个文件被创建之后，这个环状缓冲会循环回来覆盖第 1 个文件，而不是创建第 7 个文件。这样就保证了当允许新数据写入时不会有大于 6 个文件（在这个例子中也意味着 6 个小时）的数据会留在你的硬盘中。

**Stop Capture After** 选项可以让你在一定数目的文件被创建之后停止当前捕获。

#### 4.4.3 停止捕获选项

**Stop Capture** 部分可以让你在满足一定的触发条件时停止正在进行的捕获。与多文件集的情形类似，你可以使用文件大小、时间间隔或者数据包的数目作为触发条件。这些选项可以与之前介绍的多文件捕获一起使用。

#### 4.4.4 显示选项

**Display** 选项部分用来控制捕获的数据包如何进行显示。**Update List of Packets in Real Time** 选项的名字就很明白了，并且可以和 **Automatic Scrolling in Live Capture** 一同使用<sup>1</sup>。当这两个选项启用之后，所有捕获的数据包都会显示在屏幕上，并且新捕获的数据包会立刻显示出来。

<sup>1</sup> **Update List of Packets in Real Time**: 实时更新数据包列表；**Automatic Scrolling in Live Capture**: 在当前捕获中进行自动滚动。——译者注

---

### 警告

当 Update List of Packets in Real Time 和 Automatic Scrolling in Live Capture 选项都被选中并且当捕获一定数量的数据包时，这将会对处理器产生相当的负担。除非你一定需要实时查看数据包，否则最好将这两个选项都取消掉。

---

Hide Capture Info Dialog 选项可以让你屏蔽掉用来根据协议显示数据包数量和比率的小窗口。

#### 4.4.5 名字解析选项

Name Resolution 部分可以让你在你的捕获中，启用自动的数据链路层（第 2 层）、网络层（第 3 层）和传输层（第 4 层）的名字解析。我们将在第 5 章深入地讨论 Wireshark 的名字解析以及其不足。

### 4.5 使用过滤器

过滤器可以让你找出你所希望进行分析的数据包。简单来说，一个过滤器就是定义了一定条件，用来包含或者排除数据包的表达式。如果你不希望看到一些数据包，你可以写一个过滤器来屏蔽它们。如果你希望只看到某些数据包，你可以写一个只显示这些数据包的过滤器。

Wireshark 主要提供两种主要的过滤器。

- 捕获过滤器：当进行数据包捕获时，只有那些满足给定的包含/排除表达式的数据包会被捕获。
- 显示过滤器：该过滤器根据指定的表达式用于在一个已捕获的数据包集合中，隐藏不想显示的数据包，或者只显示那些需要的数据包。

我们先看一下捕获过滤器。

#### 4.5.1 捕获过滤器

捕获过滤器用于进行数据包捕获的实际场合，使用它的一个主要原因就是性能。如果你知道你并不需要分析某个类型的流量，你可以简单地使用捕获过滤器过滤掉它，从而节省那些会被用来捕获这些数据包的处理器资源。

当处理大量数据的时候，创建自定义的捕获过滤器是相当好用的。它可以

让你专注于那些与你手头事情有关的数据包，从而加速分析过程。

举个简单例子，你在一台有多种角色的服务器上捕获流量时，很可能需要用捕获过滤器，假设你正在解决一个运行于 262 端口的网络服务问题，如果你正在分析的那台服务器在许多端口运行着各种不同的网络服务，找到并分析只运行于 262 端口的流量本身可能就具有一定的工作量。你可以通过本章前面讨论过的 Capture Options 对话框达到目的，步骤如下所示。

1. 选择 **Capture->Interfaces**，然后单击你想要进行数据包捕获的设备旁的 **Options** 按钮，以打开 Capture Options 对话框。
2. 选择你想进行数据包捕获的设备，然后选择一个捕获过滤器。
3. 你可以在 Filter Button 旁输入一个表达式，并应用捕获过滤器。我们希望我们的过滤器只显示出 262 端口的出站和入站流量，所以如图 4-10 所示输入 **port 262**（我们将在下一区段中仔细地讨论关于过滤表达式的问题）。

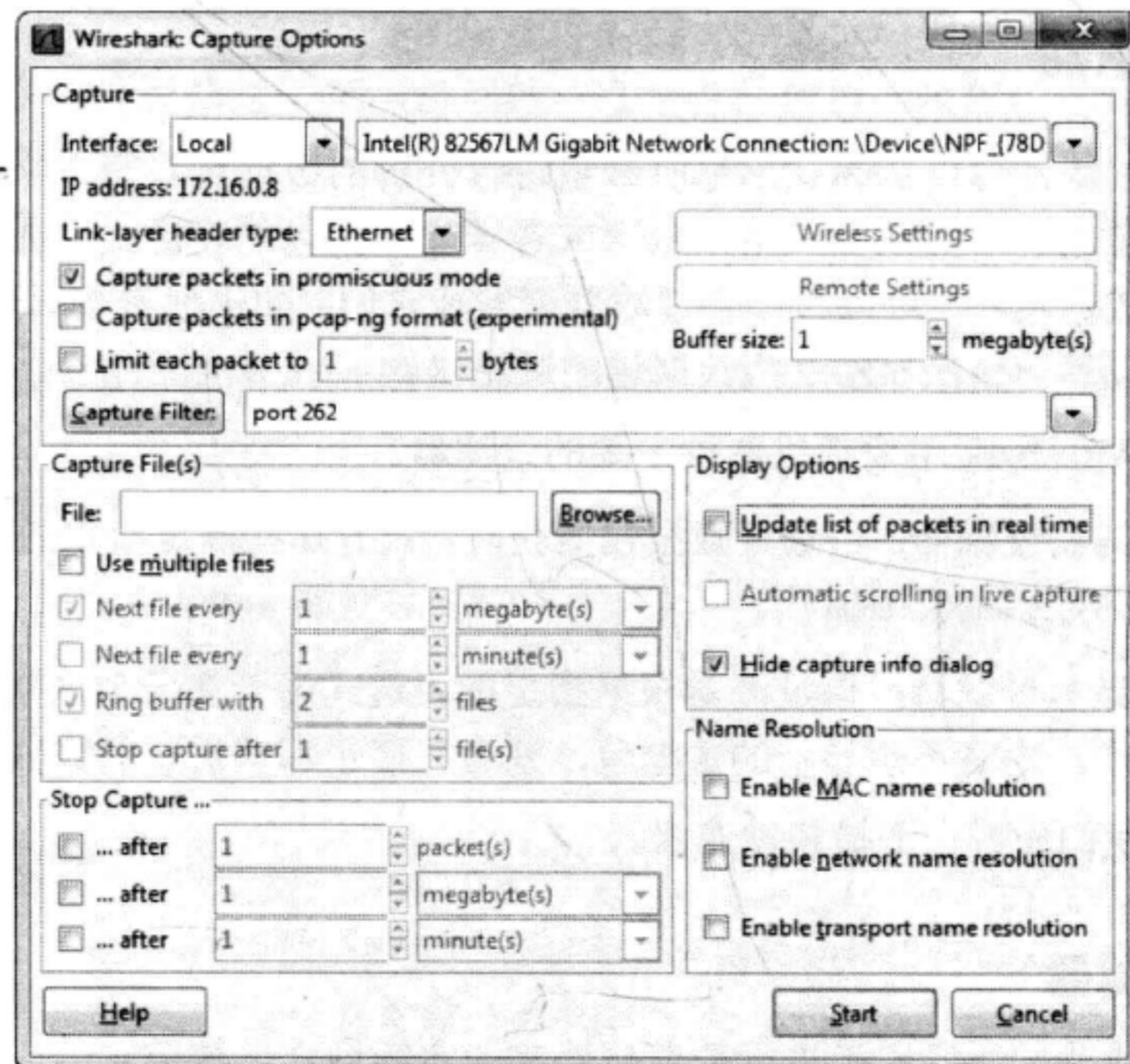


图 4-10 在 Capture Options 对话框中创建一个捕获过滤器

4. 当你设定好你的过滤器之后，单击 **Start** 开始捕获。

当收集到足够多的样本之后，你应该只看见了端口 262 的流量，这样就能

更有效地分析这些数据了。

### 1. 捕获过滤器的 BPF 语法

捕获过滤器应用于 WinPcap，并使用 Berkeley Packet Filter (BPF) 语法。这个语法被广泛用于多种数据包嗅探软件，主要因为大部分数据包嗅探软件都依赖于使用 BPF 的 libpcap/WinPcap 库。掌握 BPF 语法对你在数据包层级更深入地探索网络来说，非常关键。

使用 BPF 语法创建的过滤器被称为表达式，并且每个表达式包含一个或多个原语。每个原语包含一个或多个限定词，然后跟着一个 ID 名字或者数字（如表 4-2 所列），如图 4-11 所示。

表 4-2 BPF 限定词

| 限定词   | 说明                 | 例子                        |
|-------|--------------------|---------------------------|
| Type  | 指出名字或数字所代表的意义      | host、net、port             |
| Dir   | 指明传输方向是前往还是来自名字或数字 | src、dst                   |
| Proto | 限定所要匹配的协议          | ether、ip、tcp、udp、http、ftp |

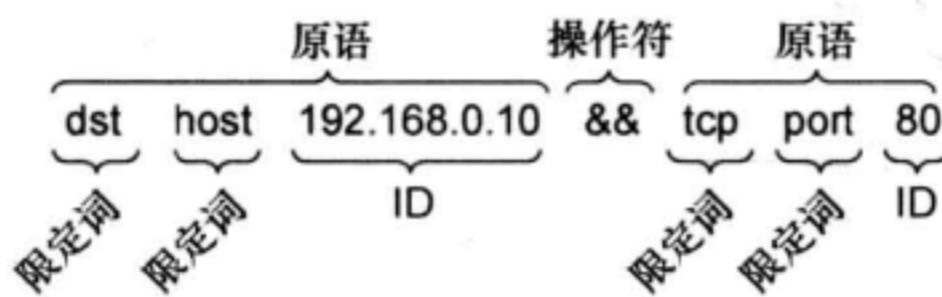


图 4-11 一个捕获过滤器样例

在给定表达式的组成部分中，一个 src 限定词和 192.168.0.10 组成了一个原语。这个原语本身就是表达式，可以用它只捕获那些源 IP 地址是 192.168.0.10 的流量。

你可以使用以下 3 种逻辑运算符，对原语进行组合，从而创建更高级的表达式。

- 连接运算符 与 (&&)
- 选择运算符 或 (||)
- 否定运算符 非 (!)

举例来说，下面的这个表达式只对源地址是 192.168.0.10 和源端口或目标端口是 80 的流量进行捕获。

---

```
src 192.168.0.10 && port 80
```

---

## 2. 主机名和地址过滤器

你所创建的大多数过滤器都会关注于一个或一些特定的网络设备。根据这个情况，可以根据设备的 MAC 地址、IPv4 地址、IPv6 地址或者 DNS 主机名配置过滤规则。

举例来说，假设你对一个正在和你网络中某个服务器进行交互的主机所产生的流量感兴趣，你在这台服务器上可以创建一个使用 host 限定词的过滤器，来捕获所有和那台主机 IPv4 地址相关的流量。

---

```
host 172.16.16.149
```

---

如果你在使用一个 IPv6 网络，你可能需要使用基于 IPv6 地址的 host 限定词进行过滤，如下所示。

---

```
host 2001:db8:85a3:8a2e:370:7334
```

---

你同样可以使用基于一台设备的主机名 host 限定词进行过滤，就像以下这样。

---

```
host testserver2
```

---

或者，如果你考虑到一台主机的 IP 地址可能会变化，你可以通过加入 ether 协议限定词，对它的 MAC 地址进行过滤。

---

```
ether host 00-1a-a0-52-e2-a0
```

---

传输方向限定词通常会和前面例子演示的那些过滤器一起使用，来捕获流向或者流出某台主机的流量。举例来说，如果想捕获来自某台主机的流量，加入 src 限定词。

---

```
src host 172.16.16.149
```

---

如果想只捕获离开 172.16.16.149 服务器，其目的地为有问题的主机的流量，使用 dst 限定词。

---

```
dst host 172.16.16.149
```

---

当你在一个原语中没有指定一种类型限定符（host、net 或者 port）时，host 限定词将作为默认选择。所以上面的那个例子也可以写成没有类型限定符的样子。

---

```
dst 172.16.16.149
```

---

## 3. 端口和协议过滤器

不仅仅可以基于主机过滤，你还可以对基于每个数据包的端口进行过滤。端口过滤通常被用来过滤使用已知端口的服务和应用。举例来说，下面是一个

只对 8080 端口进行流量捕获的简单过滤器的例子。

---

`port .8080`

---

如果想要捕获除 8080 端口外的所有流量，如下所示。

---

`!port 8080`

---

端口过滤器可以和传输方向限定符一起使用。举例来说，如果希望只捕获前往监听标准 HTTP80 端口的 Web 服务器的流量，使用 `dst` 限定符。

---

`dst port 80`

---

#### 4. 协议过滤器

协议过滤器可以让你基于特定协议进行数据包过滤。这通常被用于那些不是应用层的不能简单地使用特定端口进行定义的协议。所以如果你只想看看 ICMP 流量，可以使用下面这个过滤器。

---

`icmp`

---

如果你想看除了 IPv6 之外的所有流量，下面这个过滤器能够满足要求。

---

`!ip6`

---

#### 5. 协议域过滤器

BPF 语法提供给我们的一项强大功能，就是我们可以通过检查协议头中的每一字节来创建基于那些数据的特殊过滤器。在这节中我们将要讨论的这些高级过滤器，可以让你匹配一个数据包中从某一个特定位置开始一定数量的字节。

举例来说，假设我们想要基于 ICMP 过滤器的类型域进行过滤，而类型域位于数据包的最开头也就是偏移量为 0 的位置，那么我们可以通过在协议限定符后输入由方括号括起的字节偏移量，在这个例子中就是 `icmp[0]`，来指定我们想在一个数据包内进行检查的位置。这样将返回一个 1 字节的整型值用于比较。比如只想要得到代表目标不可达（类型 3）信息的 ICMP 数据包，我们在我们的过滤器表达式中令其等于 3，如下所示。

---

`icmp[0]==3`

---

如果只想要检查代表 echo 请求（类型 8）或 echo 回复（类型 0）的 ICMP 数据包，使用带有 OR 运算符的两个原语。

---

`icmp[0]==8||icmp[0]==0`

---

这些过滤器尽管很好用，但是它们只能基于数据包头部的 1 个字节进行过滤。当然，你也可以在方括号中偏移值的后面以冒号分隔加上一个字节长度，来指定你希望返回给过滤器表达式的数据长度。

举例来说，我们想要创建一个捕获所有以类型 3 代码 1 表示的目标不可达、主机不可达的 ICMP 数据包，它们都是彼此相邻的 1 字节字段，位于数据包头部偏移量为 0 的位置。那么我们通过创建一个检查数据包头部偏移量为 0 处 2 个字节数据的过滤器，并与十六进制值 0301（类型 3、代码 1）进行比较，如下所示。

---

`icmp[0:2]==0x0301`

一个常用的场景就是只捕获带有 RST 标志的 TCP 数据包。我们将在第 6 章深入讲述 TCP 的相关内容，而现在你只需要知道 TCP 数据包的标志位在偏移为 13 字节的地方。有趣的是，尽管整个标志位加一起是 1 字节，但是这个字节中每一比特位都是一个标志。在一个 TCP 数据包中，多个标志可以被同时设置，所以多个值可能都代表 RST 位被设置，所以我们不能只通过一个 `tcp[13]` 的值来进行有效过滤。我们必须通过在当前的原语中加入一个单一的 & 符号，来指定我们希望在这个字节中检查的比特位置。在这个字节中 RST 标志所在的比特位代表数字 4，也就是说这个比特位被设置成 4，就代表这个标志被设置了。过滤器看上去是这个样子的。

---

`tcp[13]&4==4`

如果希望看到所有被设置了 PSH 标志（比特位代表数字 8）的数据包，我们的过滤器应该会将其相应位置替换成这样。

---

`tcp[13]&8==8`

## 6. 捕获过滤器表达式样例

你将会发现你分析的成败很多时候取决于你能否编写出恰当的过滤器。表 4-3 给出了一些我经常使用的捕获过滤器。

表 4-3 常用捕获过滤器

| 过滤器   | 说明                 |
|---|--------------------|
| <code>tcp[13]&amp;32==32</code>                           | 设置了 URG 位的 TCP 数据包 |
| <code>tcp[13]&amp;16==16</code>                           | 设置了 ACK 位的 TCP 数据包 |
| <code>tcp[13]&amp;8==8</code>                             | 设置了 PSH 位的 TCP 数据包 |
| <code>tcp[13]&amp;4==4</code>                             | 设置了 RST 位的 TCP 数据包 |
| <code>tcp[13]&amp;2==2</code>                             | 设置了 SYN 位的 TCP 数据包 |
| <code>tcp[13]&amp;1==1</code>                             | 设置了 FIN 位的 TCP 数据包 |
| <code>tcp[13]==18</code>                                  | TCP SYN-ACK 数据包    |
| <code>ether host 00:00:00:00:00:00<br/>(替换为你的 MAC)</code> | 流入或流出你 MAC 地址的流量   |

续表

| 过滤器  | 说明                |
|--|-------------------|
| !ether host 00:00:00:00:00:00<br>(替换为你的 MAC) | 不流入或流出你 MAC 地址的流量 |
| broadcast                                    | 仅广播流量             |
| icmp   | ICMP 流量           |
| icmp[0:2]==0x0301                            | ICMP 目标不可达、主机不可达  |
| ip   | 仅 IPv4 流量         |
| ip6  | 仅 IPv6 流量         |
| udp  | 仅 UDP 流量          |

## 4.5.2 显示过滤器

显示过滤器应用于捕获文件，用来告诉 Wireshark 只显示那些符合过滤条件的数据包。你可以在 Packet List 面板上方的 Filter 文本框中，输入一个显示过滤器。

显示过滤器比捕获过滤器更加常用，是因为它可以让你对数据包进行过滤，却并不省略掉捕获文件中的其他数据。也就是说如果你想回到原先的捕获文件，你仅仅需要清空显示过滤表达式。

你可能会需要使用显示过滤器，来清理过滤文件中不相关的广播流量，比如清理掉 Packet List 面板中与当前的分析问题无关的 ARP 广播，但是那些 ARP 广播之后可能会有用，所以最好是把它们暂时过滤掉，而不是删除它们。

如果想要过滤掉捕获窗口中所有的 ARP 数据包，将你的鼠标放到 Packet List 面板上方的 Filter 文本框中，然后输入!arp，就可以从 Packet List 面板中去掉所有的 ARP 数据包了，如图 4-12 所示。如果想要删除过滤器，单击 Clear 按钮。

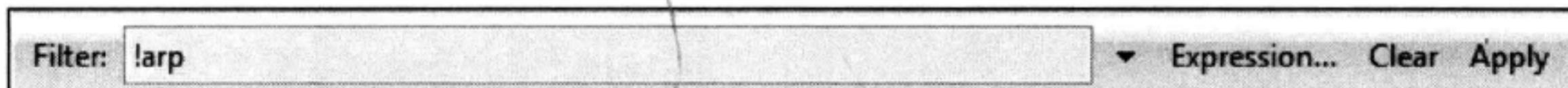


图 4-12 使用 Packet List 面板上方的 Filter 文本框创建一个显示过滤器

### 1. 过滤器表达式对话框（简单方法）

过滤器表达式对话框，如图 4-13 所示，使得 Wireshark 的初学者也能很简地创建捕获和显示过滤器。如果想要打开这个对话框，在 Capture Option 对话框中单击 **Capture Filter** 按钮，然后单击 **Expression** 按钮。

对话框左边列出了所有可用的协议域，这些域指明了所有可能的过滤条件。

如果想创建一个过滤器，按照如下步骤。

1. 单击一个协议旁的加号（+），以展开所有与这个协议相关可做为条件的域，找到你所要在你过滤器中使用的那一项，然后单击选中它。
2. 选择一个你想要在选中条件域和条件值之间建立的关系，比如等于、大于、小于等。
3. 通过输入一个和你选中条件域相关的条件值。你可以自己定义这个值，或者从 Wireshark 预定义的值中选择一个。
4. 当你完成所有上述步骤时，单击 OK 就可以看到你过滤器表达式的文本表示。

Filter Expression 对话框对于初学者来说很好用，但当你熟悉了这一套规则之后，会发现手动输入过滤器表达式更有效率。显示过滤器表达式的语法结构非常简单，但功能十分强大。

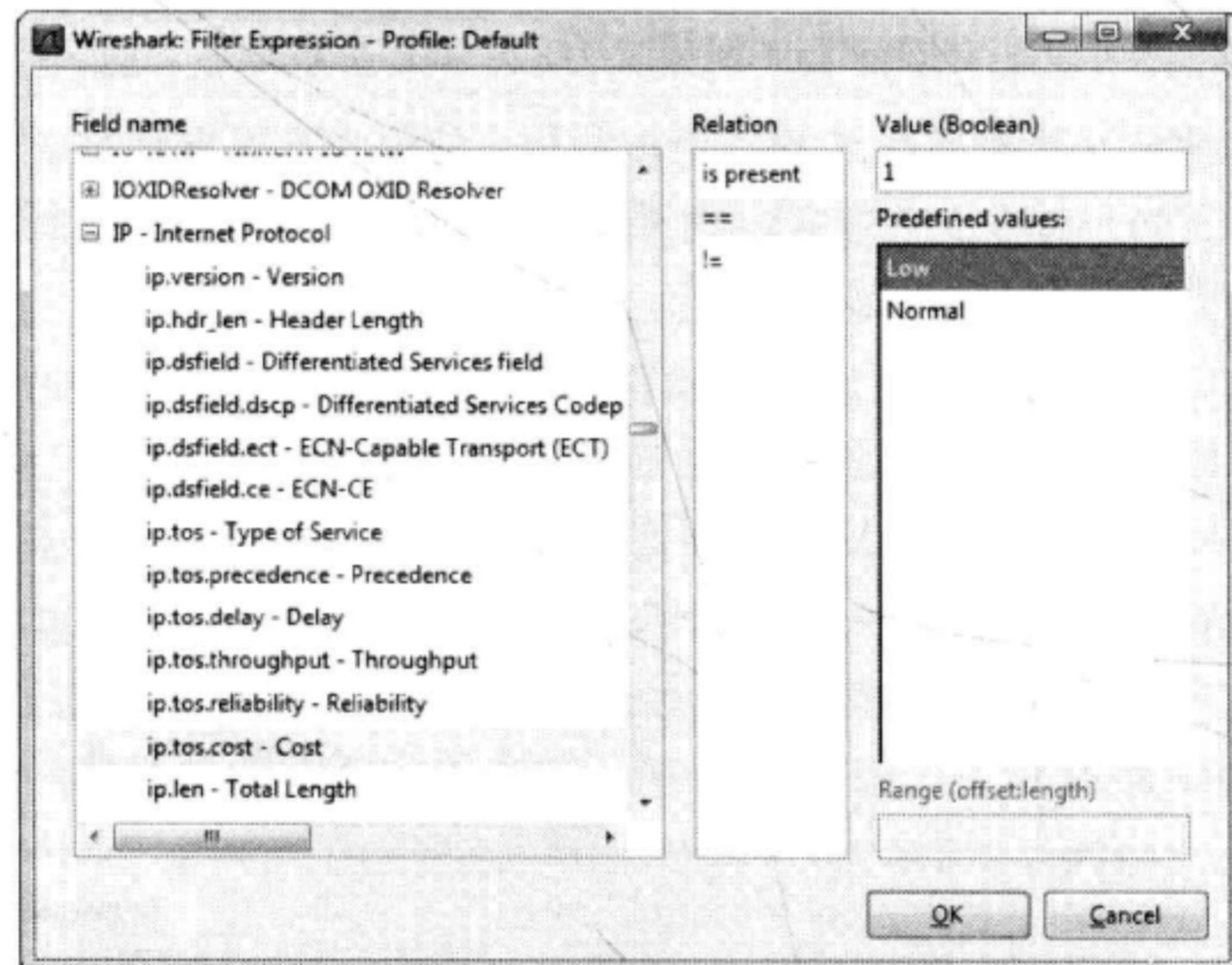


图 4-13 Filter Expression 对话框可以让你很容易地在 Wireshark 中创建过滤器

## 2. 过滤器表达式语法结构（高级方法）

你会经常用到捕获或者显示过滤器来对某一个协议进行过滤。举例来说，如果你在解决一个 TCP 问题，那么你就只希望看到捕获文件中的 TCP 流量。一个简单的 tcp 过滤器就可以解决这个问题。

现在让我们看看另外一些情况。假如为了解决你的 TCP 问题，你使用了很

多 ping 功能，所以也就产生了很多 ICMP 流量。你可以通过!icmp 这个过滤器表达式，将你捕获文件中的 ICMP 流量屏蔽掉。

比较操作符可以让你进行值的比较。举例来说，当你在检查一个 TCP/IP 网络中的问题时，你可能经常需要检查和某一个 IP 地址相关的数据包。等于操作符可以让你创建一个只显示 192.168.0.1 这个 IP 地址相关数据包的过滤器。

ip.addr==192.168.0.1

现在假设你只需要查看那些长度小于 128 字节的数据包，你可以使用“小于或等于”操作符，来完成这个要求，其过滤器表达式如下。

frame.len<=128

表 4-4 给出了 Wireshark 过滤器表达式的比较操作符。

表 4-4 Wireshark 过滤器表达式的比较操作符

| 操作符 | 说明    |
|-----|-------|
| ==  | 等于    |
| !=  | 不等于   |
| >   | 大于    |
| <   | 小于    |
| >=  | 大于或等于 |
| <=  | 小于或等于 |

逻辑运算符可以让你将多个过滤器表达式合并到一个语句中，从而极大提高过滤器的效率。举例来说，如果我们只想显示两个 IP 地址上的数据包，我们可以使用 or 操作符来创建一个表达式，只显示这两个 IP 地址的数据包，如下所示。

ip.addr==192.168.0.1 or ip.addr==192.168.0.2

表 4-5 列出了 Wireshark 的逻辑操作符。

表 4-5 Wireshark 过滤器表达式的逻辑操作符

| 操作符 | 说明          |
|-----|-------------|
| and | 两个条件需同时满足   |
| or  | 其中一个条件被满足   |
| xor | 有且仅有一个条件被满足 |
| not | 没有条件被满足     |

### 3. 显示过滤器表达式实例

尽管在理论上编写过滤器表达式很简单，但你针对不同问题创建过滤器时，仍然需要许多特定的关键词与操作符。表 4-6 给出了我经常使用的显示过滤器，你也可以访问 <http://www.wireshark.org/docs/dref>，来查看显示过滤器相关信息的一个完整列表。

表 4-6 常用显示过滤器

| 过滤器                                      | 说明                           |
|--|------------------------------|
| <code>!tcp.port==3389</code>             | 排除 RDP 流量                    |
| <code>tcp.flags.syn==1</code>            | 具有 SYN 标志位的 TCP 数据包          |
| <code>tcp.flags.rst==1</code>            | 具有 RST 标志位的 TCP 数据包          |
| <code>!arp</code>                        | 排除 ARP 流量                    |
| <code>http</code>                        | 所有 HTTP 流量                   |
| <code>tcp.port==23    tcp.port 21</code> | 文本管理流量（Telnet 或 FTP）         |
| <code>smtp    pop    imap</code>         | 文本 email 流量（SMTP、POP 或 IMAP） |

#### 4.5.3 保存过滤器

当你创建了很多捕获和显示过滤器之后，会发现其中有一些使用得格外频繁。这时你并不需要每次使用它们的时候都重新输入，Wireshark 可以让你把常用的过滤器规则保存下来，供以后使用。如果想要保存一个自定义的捕获过滤器规则，按照如下步骤。

1. 选择 **Capture -> Capture Filters** 打开 Capture Filter 对话框。
2. 在对话框的左边单击 **New** 按钮，创建一个新的过滤器。
3. 在 Filter Name 框中给你的过滤器起一个名字。
4. 在 Filter String 框中输入实际的过滤器表达式。
5. 单击 **Save** 按钮，将你的过滤器表达式保存到列表中。

按照如下步骤保存一个自定义的显示过滤器规则。

1. 选择 **Analyze->Display Filter**，或者在 Packet List 面板上方单击 **Filter** 按钮，打开 Display Filter 按钮，如图 4-14 所示。
2. 单击对话框左侧的 **New** 按钮，创建一个新过滤器。
3. 在 Filter Name 框中给你的过滤器起一个名字。

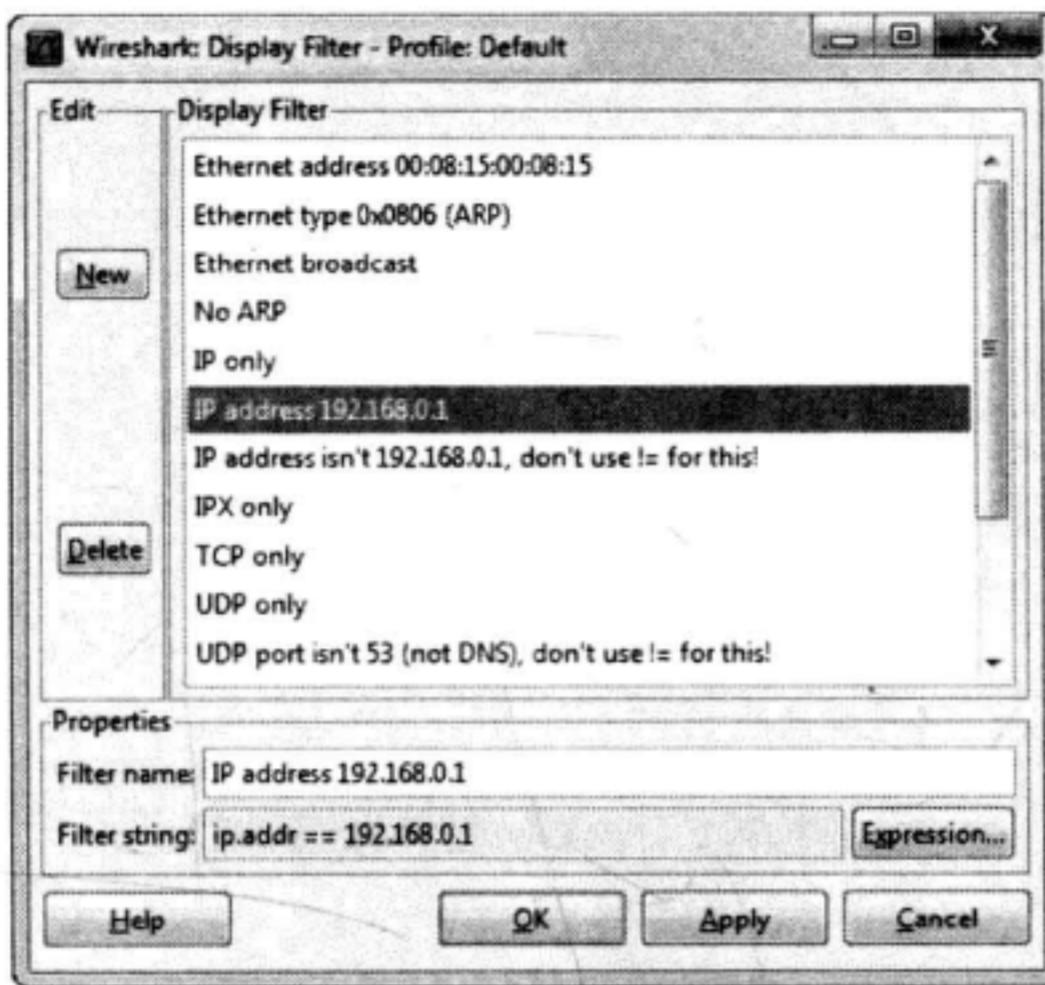


图 4-14 Display Filter 对话框可以让你保存过滤器表达式

4. 在 Filter String 框中输入实际的过滤器表达式。
5. 单击 Save 按钮，将你的过滤器表达式保存到列表中。

Wireshark 内置了许多可以作为规则范例的过滤器。你在创建自己的过滤器时，可能会用到它们（可参考 Wireshark 帮助页面）。我们在这整本书的例子中都会用到过滤器。



# 第5章

## Wireshark 高级特性



在你掌握了 Wireshark 的基础知识之后，下一步便是仔细钻研它的分析和图形化功能了。在这一章中，我们将会从这些强大的功能特性中挑选一些进行介绍，其中包括端点和会话窗口、名字解析的细节、协议解析、数据流跟踪、IO 图形化等。

### 5.1 网络端点和会话

如果想让网络通信正常进行，你必须至少拥有两台设备进行数据流交互。端点（endpoint）就是指网络上能够发送或者接收数据的一台设备。举例来说，在 TCP/IP 的通信中就有两个端点：接收和发送数据系统的 IP 地址，比如 192.168.1.25 和 192.168.1.30。

例如在数据链路层，通信是基于两台物理网卡和它们的 MAC 地址进行的。如果接收和发送数据的地址是 00:ff:ac:ce:0b:de 和 00:ff:ac:e0:dc:0f，那么这些地址就是通信中的端点，如图 5-1 所示。

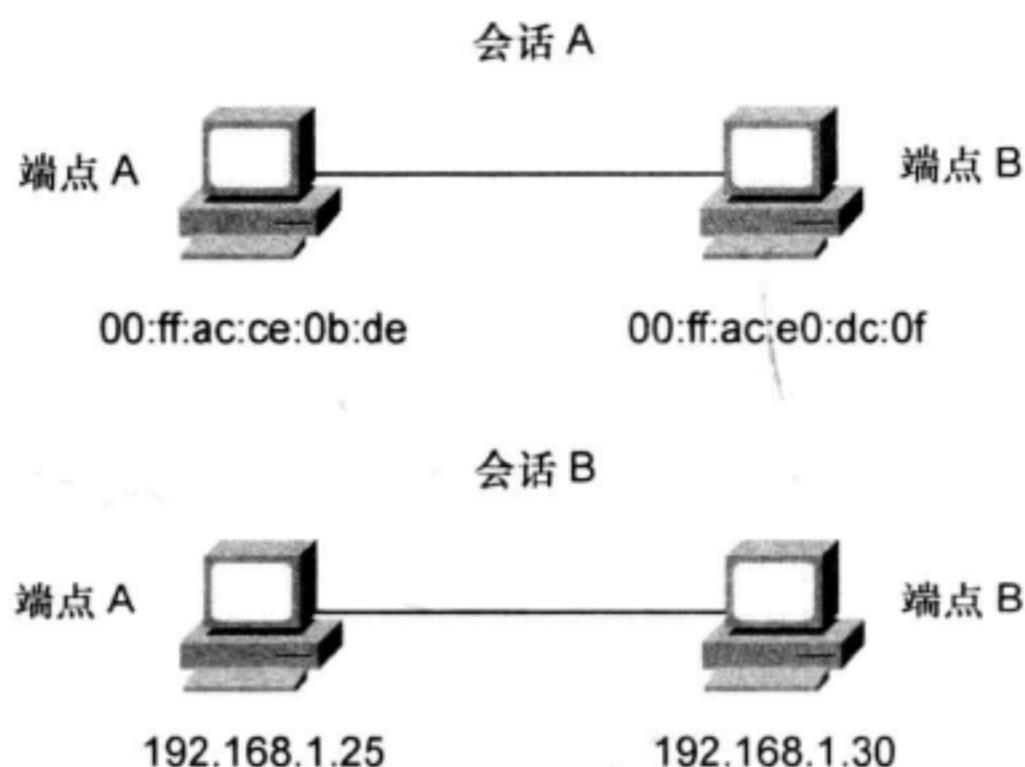


图 5-1 网络上的端点

网络中的一个会话（conversation），就如同两个人之间的会话一样，描述的是两台主机（端点）之间进行的通信。举例来说，Jim 和 Sally 的会话可能是这样子的：“你好吗？”“我很好，你呢？”“非常好！”。192.168.1.5 和 192.168.0.8 之间的一个会话可能就是这样的：“SYN”“SYN/ACK”“ACK”（我们将在第 6 章介绍更多关于 TCP/IP 通信过程的细节）。

### 5.1.1 查看端点

在分析数据流量时，你可能会发现你可以将问题定位到网络中的一个特定端点上去。Wireshark 的 Endpoints 窗口（Statistics->Endpoints）给出了每一端点的许多有用统计数据，包括每个端点的地址、传输发送数据包的数量和字节数。

这个窗口顶部的选项卡给出了当前捕获文件中所有支持和被识别的端点。单击其中一个选项卡，就可以将端点的列表缩小到某一个协议上。勾选 Name resolution 多选框，可以在端点窗口中开启名字解析功能。

你可以使用端点窗口将特定的数据包过滤出来，显示在 Packet List 面板中。右键单击一个特定的端点，可以看到许多选项，包括创建过滤器以显示只与这个端点相关的流量，或者与选定端点无关的所有流量。你也可以直接将端点导出到一个着色规则中（着色规则在第 3 章中已有介绍）。

Endpoints: lotsofweb.pcap

| Ethernet: 12    | Fibre Channel | FDDI      | IPv4: 95   | IPv6: 5  | IPX        | JXTA      | NCP      | RSVP      | SCTP | TCP: 358 | Token Ring | UDP: 106 | USB | WLAN |
|-----------------|---------------|-----------|------------|----------|------------|-----------|----------|-----------|------|----------|------------|----------|-----|------|
| IPv4 Endpoints  |               |           |            |          |            |           |          |           |      |          |            |          |     |      |
| Address         | Packets       | Bytes     | Tx Packets | Tx Bytes | Rx Packets | Rx Bytes  | Latitude | Longitude |      |          |            |          |     |      |
| 172.16.16.128   | 8 324         | 7 387 292 | 2 790      | 507 866  | 5 534      | 6 879 426 | -        | -         |      |          |            |          |     |      |
| 172.16.16.255   | 43            | 3 956     | 0          | 0        | 43         | 3 956     | -        | -         |      |          |            |          |     |      |
| 239.255.255.250 | 23            | 8 577     | 0          | 0        | 23         | 8 577     | -        | -         |      |          |            |          |     |      |
| 172.16.16.2     | 23            | 9 045     | 23         | 9 045    | 0          | 0         | -        | -         |      |          |            |          |     |      |
| 224.0.1.60      | 1             | 86        | 0          | 0        | 1          | 86        | -        | -         |      |          |            |          |     |      |
| 224.0.0.252     | 28            | 1 848     | 0          | 0        | 28         | 1 848     | -        | -         |      |          |            |          |     |      |
| 205.203.140.65  | 363           | 251 133   | 235        | 179 061  | 128        | 72 072    | -        | -         |      |          |            |          |     |      |
| 4.2.2.1         | 103           | 11 426    | 51         | 7 275    | 52         | 4 151     | -        | -         |      |          |            |          |     |      |
| 157.166.226.25  | 48            | 29 167    | 26         | 24 277   | 22         | 4 890     | -        | -         |      |          |            |          |     |      |
| 209.85.225.148  | 67            | 26 631    | 31         | 15 961   | 36         | 10 670    | -        | -         |      |          |            |          |     |      |
| 209.85.225.118  | 164           | 87 053    | 94         | 74 046   | 70         | 13 007    | -        | -         |      |          |            |          |     |      |
| 209.85.225.133  | 101           | 50 738    | 56         | 43 181   | 45         | 7 557     | -        | -         |      |          |            |          |     |      |

Name resolution     Limit to display filter

Help Copy Map Close

图 5-2 端点窗口可以让你查看一个捕获文件里的每个端点

## 5.1.2 查看网络会话

Wireshark 的会话窗口 (Statistics->Conversations)，如图 5-3 所示，以地址 A (Address A) 和地址 B (Address B) 显示了会话中端点的地址，以及每个设备发送或收到的数据包和字节数。

Conversations: lotsofweb.pcap

| Ethernet: 13       | Fibre Channel   | FDDI    | IPv4: 103 | IPv6: 4      | IPX        | JXTA         | NCP        | RSVP      | SCTP | TCP: 279 | Token Ring | UDP: 93 | USB | WLAN |
|--------------------|-----------------|---------|-----------|--------------|------------|--------------|------------|-----------|------|----------|------------|---------|-----|------|
| IPv4 Conversations |                 |         |           |              |            |              |            |           |      |          |            |         |     |      |
| Address A          | Address B       | Packets | Bytes     | Packets A->B | Bytes A->B | Packets A<-B | Bytes A<-B | Rel Start |      |          |            |         |     |      |
| 172.16.16.128      | 172.16.16.255   | 43      | 3 956     | 43           | 3 956      | 0            | 0          | 0.000000  |      |          |            |         |     |      |
| 172.16.16.128      | 239.255.255.250 | 2       | 350       | 2            | 350        | 0            | 0          | 0.2930930 |      |          |            |         |     |      |
| 172.16.16.2        | 172.16.16.128   | 2       | 818       | 2            | 818        | 0            | 0          | 0.2956330 |      |          |            |         |     |      |
| 172.16.16.128      | 224.0.1.60      | 1       | 86        | 1            | 86         | 0            | 0          | 0.6015650 |      |          |            |         |     |      |
| 172.16.16.128      | 224.0.0.252     | 28      | 1 848     | 28           | 1 848      | 0            | 0          | 0.7541010 |      |          |            |         |     |      |
| 172.16.16.128      | 205.203.140.65  | 363     | 251 133   | 128          | 72 072     | 235          | 179 061    | 1.7092310 |      |          |            |         |     |      |
| 4.2.2.1            | 172.16.16.128   | 16      | 2 101     | 8            | 1 433      | 8            | 668        | 3.0094020 |      |          |            |         |     |      |
| 157.166.226.25     | 172.16.16.128   | 48      | 29 167    | 26           | 24 277     | 22           | 4 890      | 3.1808700 |      |          |            |         |     |      |
| 172.16.16.128      | 209.85.225.148  | 25      | 8 920     | 12           | 4 767      | 13           | 4 153      | 3.2419560 |      |          |            |         |     |      |
| 172.16.16.128      | 209.85.225.118  | 164     | 87 053    | 70           | 13 007     | 94           | 74 046     | 3.2420630 |      |          |            |         |     |      |
| 172.16.16.128      | 209.85.225.133  | 101     | 50 738    | 45           | 7 557      | 56           | 43 181     | 3.2422230 |      |          |            |         |     |      |
| 74.125.166.28      | 172.16.16.128   | 553     | 532 821   | 382          | 519 254    | 171          | 13 567     | 3.2428500 |      |          |            |         |     |      |

Name resolution     Limit to display filter

Help Copy Close

图 5-3 会话窗口可以让你与捕获文件中的每个会话进行交互

这个窗口中列出的会话以不同的协议分开，并可以通过窗口顶部的选项卡进行选择。右键单击一个特定的会话，可以让你创建一些有用的过滤器，比如显示由设备 A 发出的所有流量，设备 B 收到的所有流量，或者设备 A 和设备 B 之间所有的通信流量。

### 5.1.3 使用端点和会话窗口进行问题定位

端点和会话窗口在网络问题定位中十分重要，特别是当你试图找到网络中大规模流量的源头，或者找到哪台服务器最活跃。

举例来说，当你打开了 **lotsofweb.pcap** 文件之后，你可以看到多个客户端浏览互联网时产生的大量 HTTP 流量。如果你使用端点窗口进行查看，你可以立刻对你所查看的流量得出一些结论。

查看 IPv4 选项卡（如图 5-4 所示），你可以看到以字节数排序后的第一个地址是本地的 172.16.16.128 地址，也就是说你网络中的这个设备是数据集中最活跃的信息源（进行了最多通信的主机）。第二个地址 74.125.103.163 不是本地地址，也就意味着你可以假设你有一个客户端和这个地址进行了很多交互，或者多个客户端和它进行了一些交互。一个简便的 WHOIS (<http://whois.arin.net/ui/>) 告诉你这个地址属于 Google，仔细地分析了数据包之后可以发现这是 YouTube 的流量。

The screenshot shows the 'Endpoints' window in Wireshark. The title bar says 'Endpoints: lotsofweb.pcap'. Below it is a tab bar with 'Ethernet: 12' selected, followed by 'Fibre Channel', 'FDDI', 'IPv4: 95', 'IPv6: 5', 'IPX', 'JXTA', 'NCP', 'RSVP', 'SCTP', 'TCP: 358', 'Token Ring', 'UDP: 106', 'USB', and 'WLAN'. The main area is titled 'IPv4 Endpoints' and contains a table with the following data:

| Address         | Packets | Bytes     | Tx Packets | Tx Bytes  | Rx Packets | Rx Bytes  | Latitude | Longitude |
|-----------------|---------|-----------|------------|-----------|------------|-----------|----------|-----------|
| 172.16.16.128   | 8 324   | 7 387 292 | 2 790      | 507 866   | 5 534      | 6 879 426 | -        | -         |
| 74.125.103.163  | 3 927   | 4 232 435 | 2 882      | 4 173 482 | 1 045      | 58 953    | -        | -         |
| 172.16.16.136   | 2 349   | 1 455 670 | 1 137      | 213 891   | 1 212      | 1 241 779 | -        | -         |
| 172.16.16.197   | 2 157   | 1 073 399 | 1 107      | 221 885   | 1 050      | 851 514   | -        | -         |
| 66.35.45.201    | 1 106   | 807 006   | 596        | 702 314   | 510        | 104 692   | -        | -         |
| 74.125.103.147  | 608     | 633 494   | 435        | 620 562   | 173        | 12 932    | -        | -         |
| 74.125.166.28   | 553     | 532 621   | 382        | 519 254   | 171        | 13 567    | -        | -         |
| 64.208.21.43    | 551     | 357 373   | 309        | 280 314   | 242        | 77 059    | -        | -         |
| 74.125.95.149   | 543     | 409 144   | 336        | 365 266   | 207        | 43 878    | -        | -         |
| 65.173.218.96   | 473     | 331 336   | 263        | 305 759   | 210        | 25 577    | -        | -         |
| 4.23.40.126     | 451     | 318 740   | 234        | 291 841   | 217        | 26 899    | -        | -         |
| 204.160.126.126 | 449     | 185 482   | 206        | 118 591   | 243        | 66 891    | -        | -         |
| 72.32.92.4      | 387     | 130 428   | 190        | 97 845    | 197        | 32 583    | -        | -         |

At the bottom, there are checkboxes for 'Name resolution' (checked) and 'Limit to display filter' (unchecked), and buttons for 'Help', 'Copy', 'Map', and 'Close'.

图 5-4 端点窗口显示了哪个主机最活跃

## 注意

IP 地址的分配是根据其地址信息由多个实体进行管理的。在我们的例子中，我们使用了负责美国(及周边地区)IP 地址分配的美国互联网号码注册中心(American Registry for Internet Numbers, ARIN)的数据。一般来说，如果你想对某一个 IP 进行 WHOIS 查询，你在负责这个 IP 组织的网站上操作即可。如果你并不知道地理信息，并且在错误的注册中心网站上进行了查询，这个网站也会告诉你正确的查询位置。类似的地址注册中心有 AfriNIC(非洲)、RIPE(欧洲)和 APNIC(亚太地区)。

知道了这些信息，你最活跃的通信端点一定包含了流量最大的会话么？如果你这时打开了会话窗口，并选中 IPv4 选项卡，你就可以使用字节数对列表排序来验证这一点。在这个例子中，你可以看到这个流量应该是连续的视频下载流量，因为从地址 A(74.125.103.163)发出的数据包比从地址 B(172.16.16.128)发出的要大得多(如图 5-5 所示)。

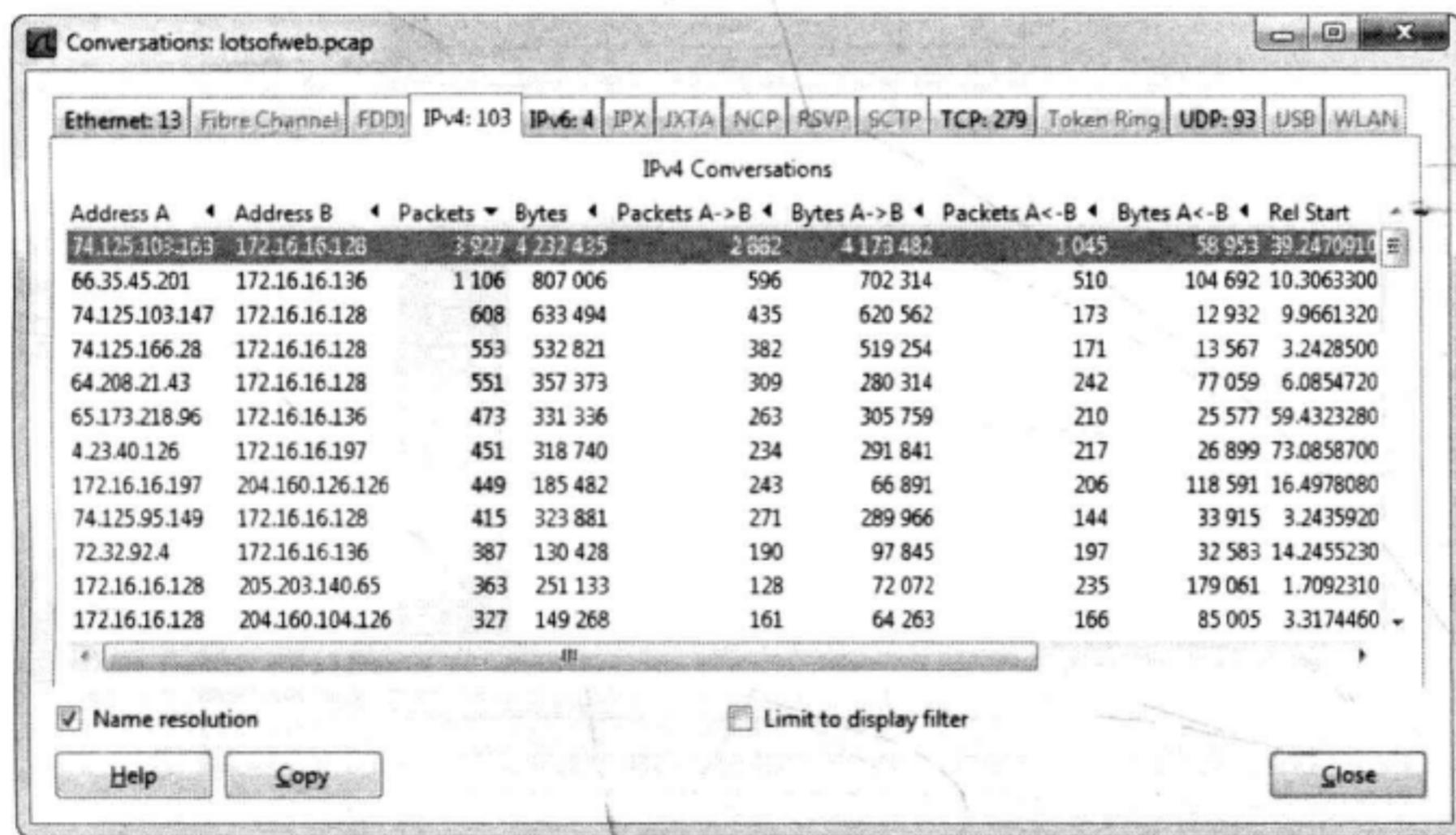


图 5-5 会话窗口中确定了最活跃的两个信息源是在相互通信

在此书后面的实战场景中，你还会看到如何使用端点和会话窗口。

## 5.2 基于协议分层结构的统计数据

当在与特别大的捕获文件打交道时，你有时会需要知道文件中协议的分布情况，也就是捕获中 TCP、IP、DHCP 等所占的百分比是多少。除了数每一个数据包并计算总和外，使用 Wireshark 的 Protocol Hierarchy Statistics(协议分层

统计) 窗口也是一个对网络进行基准分析的好方法。举例来说, 如果你知道网络流量中的 ARP 流量通常占百分之十, 如果有一天你的捕获中发现了百分之五十的 ARP 流量, 那么你就知道一定有地方出了问题。

保持 lotsofweb.pcap 文件打开, 选择 **Statistics->Protocol Hierarchy** 打开协议分层统计窗口 (如图 5-6 所示)。你应该会注意到所有比例加到一起并不是正好等于百分之百, 那是因为很多数据包都包含着不同分层的多个协议, 从而按照每个协议来计算相对于按照每个数据包来计算要有一些差异。无论如何, 你还是得到了捕获文件中协议分布的准确情况。

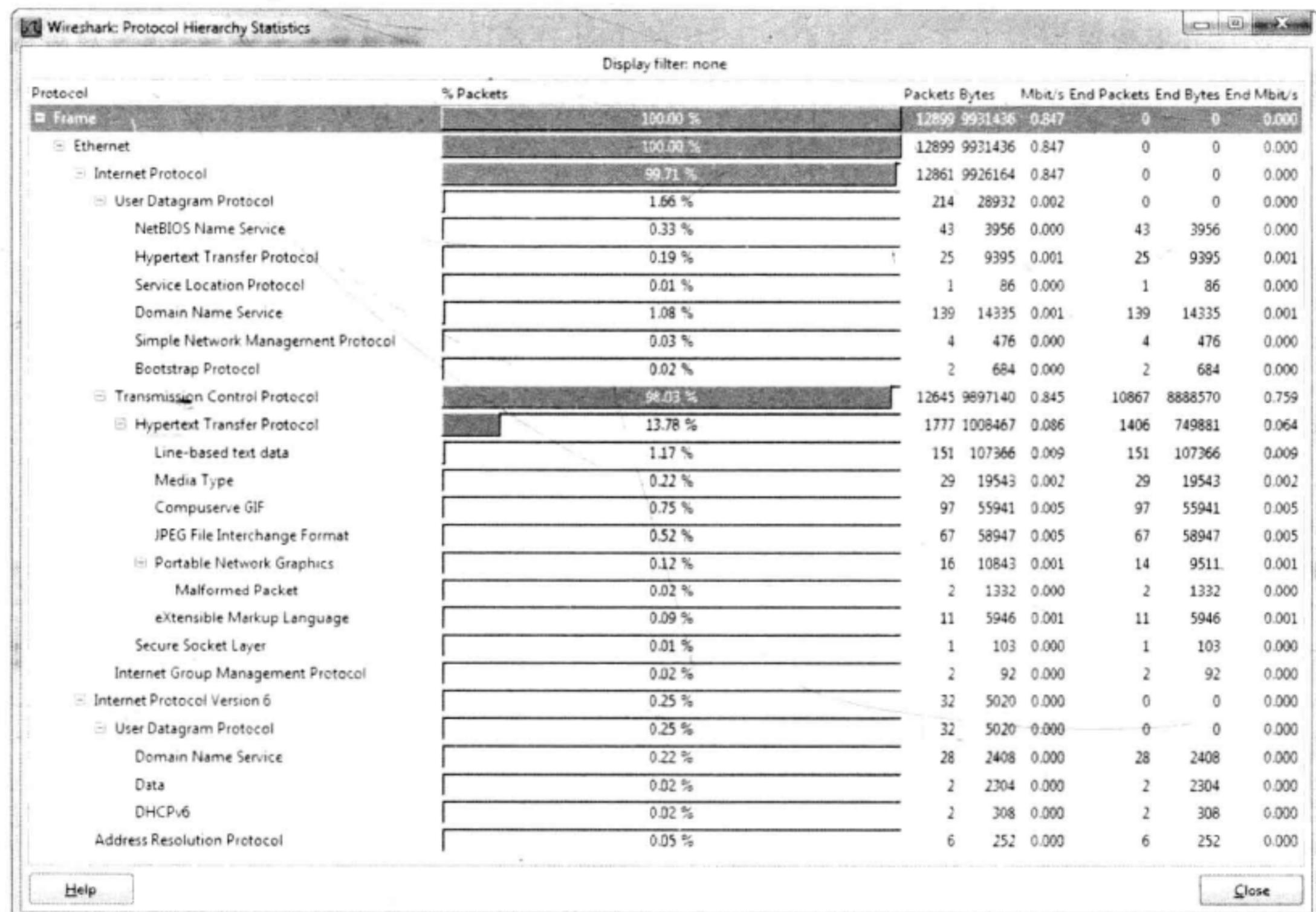


图 5-6 协议分层统计窗口给出了各种协议的分布统计情况

协议分层统计窗口通常是你在检查流量时最先打开的窗口之一。它确实能够提供给你一个网络中活动类型的直观快照。当你开始检查更多的流量时后, 你最终将通过查看正在使用协议的分布情况, 来得到网络中用户和设备的情况。只需要简单地查看网段中的流量, 就可以立即分辨这个网段属于哪个部门。IT 部门网段的流量中通常有管理协议, 例如 ICMP 或者 SNMP 的数据, 订单管理部门通常有着大量的 SMTP 流量, 甚至我还可以在一些烦人实习生的网络区段内找到他们玩魔兽世界的流量。

## 5.3 名字解析

网络数据通过使用各种各样字母数字组成的寻址地址系统进行传输，但这些地址系统通常都因为太长或者太复杂而不容易记住，比如物理硬件地址 00:16:CE:6E:8B:24。名字解析（也称为名字查询）就是一个协议用来将一个独特的地址转换到另一个的过程。举例来说，假如一个计算机有着物理的 MAC 地址 00:16:CE:6E:8B:24，DNS 和 ARP 协议可以让我们将其名字记作 Marketing-2.domain.com。将易读的名字对应到这些费解的地址，我们就可以很容易地记住并分辨出来。

### 5.3.1 开启名字解析

如果想要开启名字解析，选择 **Capture->Options**，打开 Capture Options 对话框。如图 5-7 所示，Wireshark 中有 3 种类型的名字解析可用。

**MAC 地址解析（MAC name resolution）：**这种类型的名字解析使用 ARP 协议，试图将数据链路层 MAC 地址，例如 00:09:5B:01:02:03，转换到网络层地址，例如 10.100.12.1。如果这种转换尝试失败，Wireshark 会使用程序目录中的 ethers 文件尝试进行转换。Wireshark 最后的手段便是将 MAC 地址的前 3 个字节转换到设备的 IEEE 指定制造商名称，例如 Netgear\_01:02:03。

**网络名字解析（Network name resolution）：**这种类型的名字解析试图将一个网络层地址，例如 192.168.1.50 这个 IP 地址，转换到一个易读的 DNS 名称，例如 MarketingPC1.domain.com。

**传输名字解析（Transport name resolution）：**这种类型的名字解析尝试将一个端口号转换成一个与其相关的名字。例是将端口 80 显示为 http。

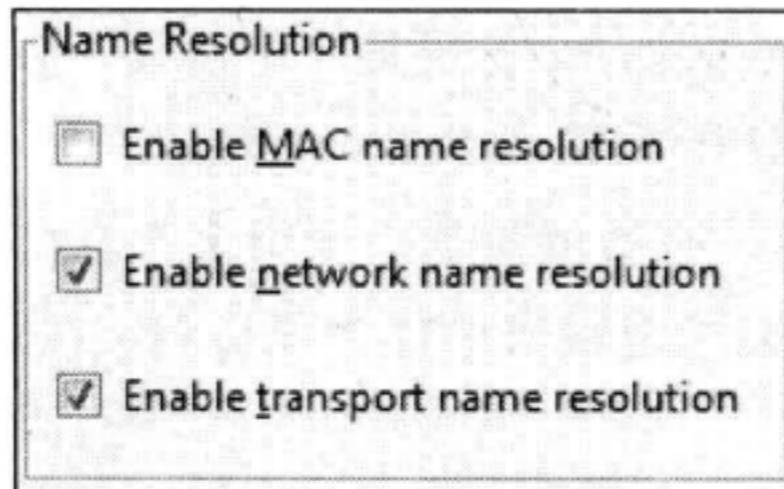


图 5-7 在 Capture Options 对话框中开启名字解析

你可以利用各种名字解析工具使你的捕获文件变得更加可读，从而在一些情况下节省大量时间。举例来说，你可以使用 DNS 名字解析，使你能够更容易地分辨你试图定位数据包来源的计算机名称。

### 5.3.2 名字解析的潜在弊端

名字解析有着很多优点，使用名字解析看上去很容易，但是也存在着一些潜在的弊端，包括如下几点。

- 名字解析可能会失败，尤其是当你所查询的名字服务器不知道这个名字的时候。
- 名字解析在你每次打开一个捕获文件的时候都要重新进行一次，因为这些信息并不会保存在文件之中。这也就意味着当一个文件名字解析所使用服务器不可用的时候，名字解析就会失败。
- 对 DNS 名字解析的依赖，会产生额外的数据包，也就是说你的捕获文件中可能会被解析那些基于 DNS 地址的流量所占据。而在分析其他问题的时候，避免看到自己的流量是一个典型规则。
- 名字解析会带来额外的处理开销。如果你正在处理一个非常大的捕获文件，而内存很少的时候，你可能需要关闭名字解析来节约系统资源。

## 5.4 协议解析

Wireshark 中的协议解析器可以让你将数据包拆分成多个协议区段以便分析。举例来说，Wireshark 的 ICMP 协议解析器可能将捕获网络上的原始数据，并以 ICMP 数据包格式显示出来。

你可以将一个解析器看作是一个网络原始数据流和 Wireshark 程序之间的翻译器。如果需要 Wireshark 支持某一个协议，那么它就必须拥有一个内置的解析器（或者你可以使用 C 或者 Python 自己写一个）。

Wireshark 对每一个数据包都会使用多个解析器一起进行协议解析，也可以通过使用它内部的编写逻辑来进行合理猜测，决定使用哪一种协议解析器。

### 5.4.1 更换解析器

Wireshark 在给一个数据包选择解析器时也并不是每次都能选对的，尤其是

当网络上的一个协议使用了不同于标准的配置，比如一个非默认端口（网络管理员通常会出于安全考虑，或者员工想要避开访问控制时进行设置）。这时我们可以更改 Wireshark 使用特定解析器的方式。

举例来说，打开 wrongdissector.pcap 这个捕获文件，可以注意到这个文件中包含了大量两台计算机之间的 SSL 通信。SSL 是安全套接层协议（Secure Socket Layer protocol），用来在主机之间进行安全加密的传输。由于其保密性，所以大多数的正常情况下，在 Wireshark 中查看 SSL 流量不会产生有用的信息，但这里一定存在着一些问题。如果你单击其中的几个数据包，然后在 Packet Bytes 面板中仔细查看这几个数据包的内容，你会很快发现一些明文流量。事实上，如果你看第 4 个数据包，你会发现其中提到了 FileZilla FTP 服务器程序（FileZilla FTP server application），并且之后的几个数据包清晰地显示了对用户名和密码的请求与响应。

如果这真是 SSL 流量，你应该不会读到数据包中的任何数据，并且你也不会看到以明文传输的所有用户名和密码（如图 5-8 所示）。根据这些信息，我们可以推测出这应该是一个 FTP 流量而不是 SSL 流量，而导致错误选择解析器的原因应该是这个 FTP 流量使用了原本用作 HTTPS（基于 SSL 的 HTTP）标准端口的 443 端口。

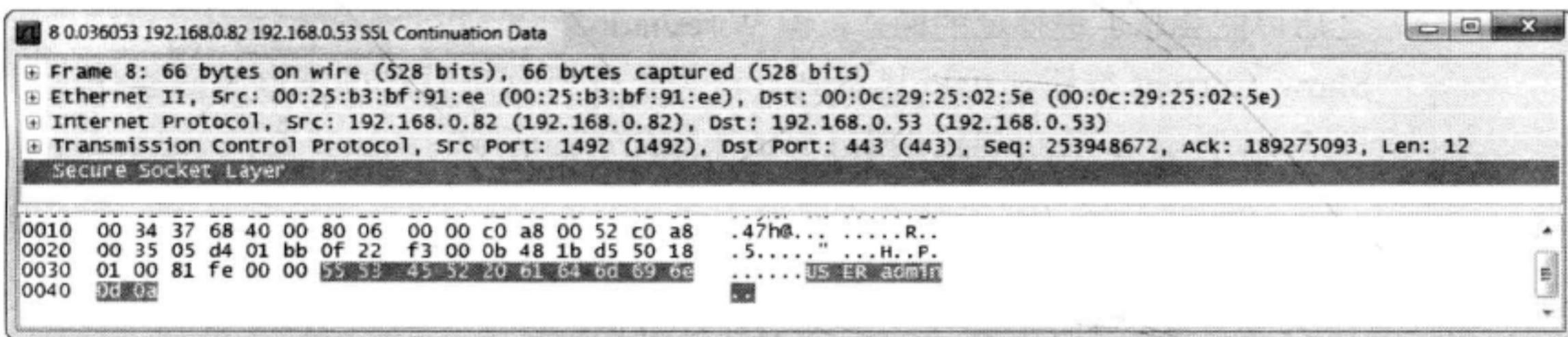


图 5-8 明文用户名和密码？这更像是 FTP 而不是 SSL！

为了解决这个问题，你可以强制 Wireshark 对这个数据包使用 **FTP** 协议解析器。这个过程被称为强制解码，需要按如下步骤操作。

1. 右键单击其中一个 SSL 数据包，并选择 **Decode As**。这时会弹出一个对话框，你便可以从中选择你想要使用的解析器。
2. 在下拉菜单中选择 **destination (443)**，并在 Transport 选项卡中选择 **FTP**，以便让 Wireshark 使用 FTP 解析器对所有口号为 443 的 TCP 流量进行解码（如图 5-9 所示）。

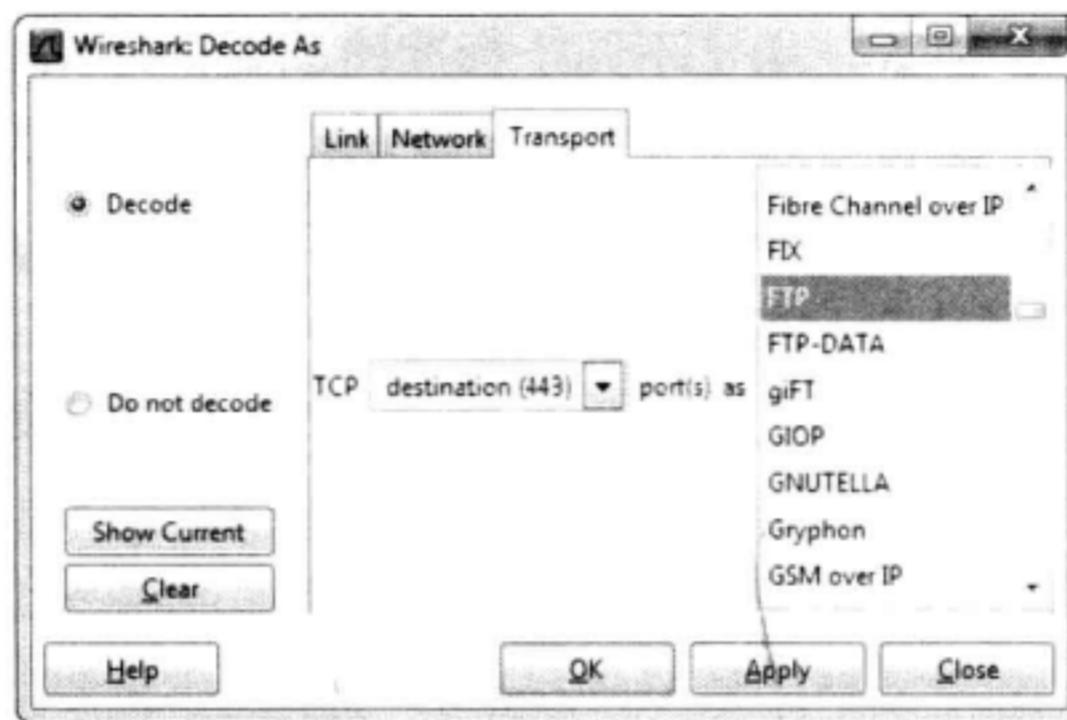


图 5-9 Decode As 对话框可以让你创建强制解码器

3. 在你选好之后单击 **OK**, 就可以立刻将所做修改应用到捕获文件中去。

你应该可以看到数据已经被很好地解码, 这时你就可以从 Packet List 面板中对它进行分析, 而不是对每一个字节下工夫。

#### 警告

你进行强制解码时产生的更改, 并不会在你保存捕获文件并关闭 Wireshark 后保存。在你每次打开捕获文件时, 你都要重新进行强制解码的设置。

你可以在同一个捕获文件中多次使用强制解码功能。当你进行了多次解码之后可能会忘了进行过的操作, 但 Wireshark 不会。在 Decode As 对话框中单击 Show Current 按钮, 可以显示到目前为止你所有进行过的强制解码操作 (如图 5-10 所示)。你可通过单击 Clear 按钮把它们清除。

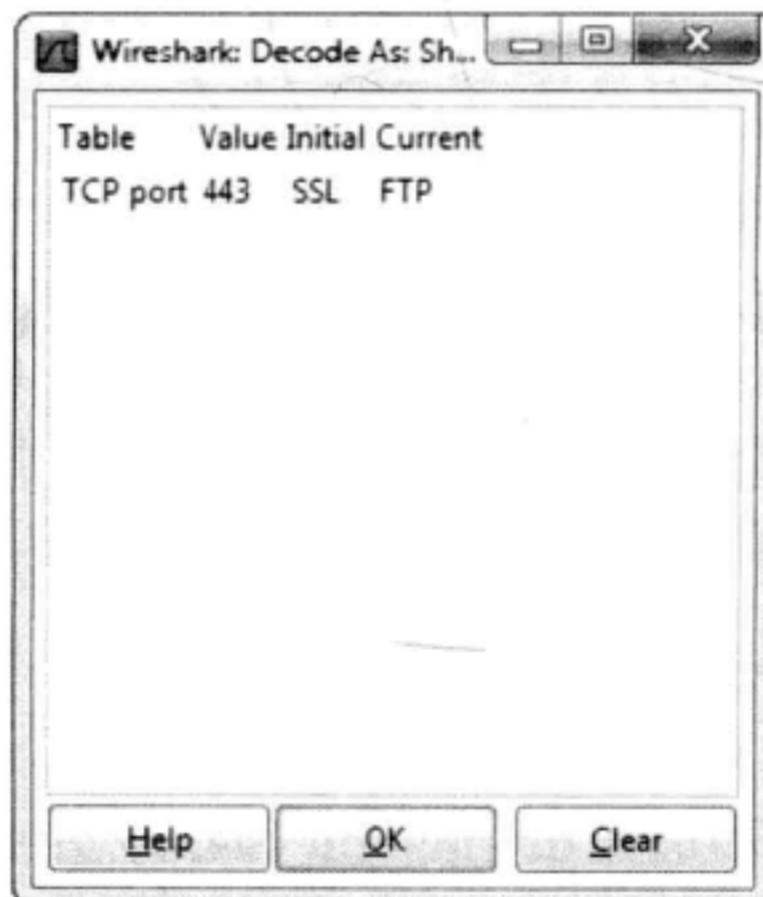


图 5-10 单击 Show Current 按钮将会显示你为一个捕获文件设置的所有强制解码操作

#### 5.4.2 查看解析器源代码

开源软件的美妙之处就在于：如果你对某些事情为什么会产生感到困惑，你可以直接查看源代码来找到具体原因。当你想查明为什么一个特定的协议没有被正确解析的时候，开源软件就派上用场了。

在 Wireshark 网站上的 Develop 链接中，单击 Browse the Code，就可以直接查看协议解析器的源代码。这个链接直接指向 Wireshark 的代码库，里面有 Wireshark 的最新及之前的版本。单击 releases 文件夹，便能看见所有官方的 Wireshark（甚至包括 Ethereal）版本，其中最新版本将在最下面显示出来。在你选择了你想要查看的版本之后，在 epan/dissectors 文件夹下可以找到协议解析器。每一个解析器都以 packets-protocolname.c（数据包-协议名称.c）的形式命名。

这些文件可能会很复杂，但你应该可以发现它们遵循着同一个标准模板并有着很好的注释。你并不需要成为一个 C 语言专家就可以理解每一个解析器的基本功能。如果你想深入理解你在 Wireshark 中所看到的东西，我强烈建议你至少看一些简单协议的解析器。

### 5.5 跟踪 TCP 流

Wireshark 分析功能中最令人满意的一点就是它能够将 TCP 流重组成容易阅读的格式。跟踪 TCP 流这个功能可以将从客户端发往服务器的数据排好顺序使之容易查看，而不需要一小块一小块地看。这在查看 HTTP、FTP 等纯文本应用层协议时非常好用（我们将在下一章中详细讲述这些常见协议是如何工作的）。

我们以一个简单的 HTTP 交互举例来说，打开 http\_google.pcap，并在文件中单击任何一个 TCP 或者 HTTP 数据包，右键单击这个文件并选择 **Follow TCP Stream**。这时 TCP 流就会在一个单独的窗口中显示出来（如图 5-11 所示）。

我们注意到这个窗口中的文字以两个颜色显示，其中红色用来标明从源地址前往目标地址的流量，而蓝色用来区分出相反方向也就是从目标地址到源地址的流量。这里颜色的标记以哪方先开始通信为准，在我们的例子中，客户端最先建立了到服务器的连接，所以显示为红色。

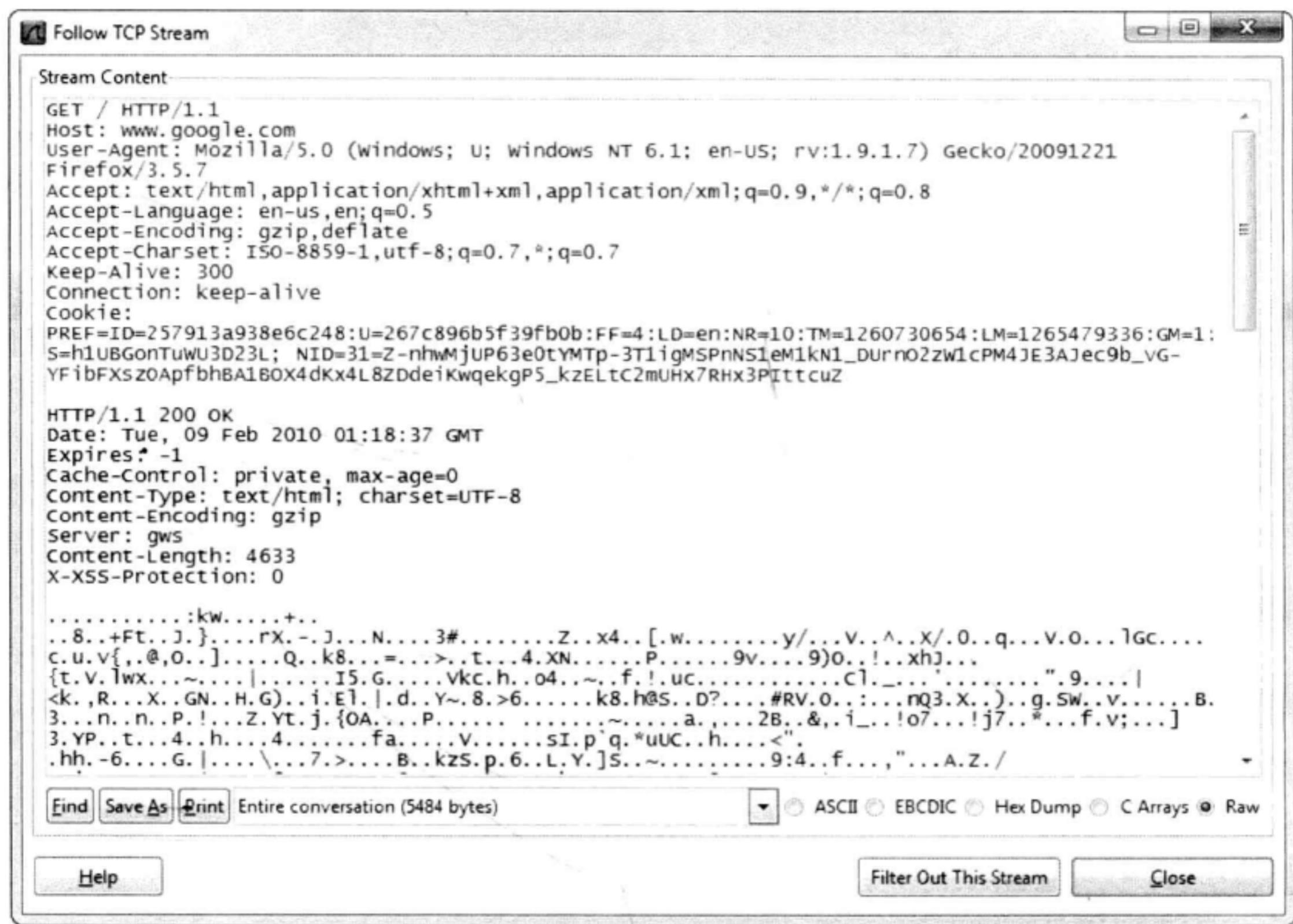


图 5-11 跟踪 TCP 流窗口将通信内容以更简单可读的方式进行了重新组织

在这个 TCP 流中，你可以清晰地看到这两台主机之间进行的绝大多数通信。在这些通信开始的时候，最初是一个对 Web 根目录的 GET 请求，然后是来自服务器的一个用 HTTP/1.1 200 OK 表示请求成功的响应。在每一次客户端请求另一个文件以及服务器给予响应的时候，这个简单模式都会重复出现。你可以看到一个用户正在浏览谷歌首页，而事实上你和这个用户看到的别无二致，只不过是以更深入的形式去看。

在这个窗口中除了能够看到这些原始数据，你还可以在文本间进行搜索，将其保存成一个文件、打印，或者以 ASCII 码、EBCDIC、十六进制或者 C 数组的格式去查看。这些选项都可以在跟踪 TCP 流窗口的下面找到。

跟踪 TCP 流在你和一些协议打交道的时候，绝对是你的好帮手。

## 5.6 数据包长度

一个或一组数据包的大小可以告诉你很多情况。在正常情况下，一个以太

网上的帧最大长度为 1518 字节，除去以太网、IP 以及 TCP 头，还剩下 1460 字节以供应用层协议的头或者数据使用。基于此，我们可以通过一个捕获文件中数据包长度的分布情况，做一些对流量合理的猜测。

文件 download-slow.pcap 就是一个很好的例子。打开文件后，选择 **Statistics->Packet Lengths**，然后单击 **Create Stat**，就会出现一个如图 5-12 所示的结果窗口。

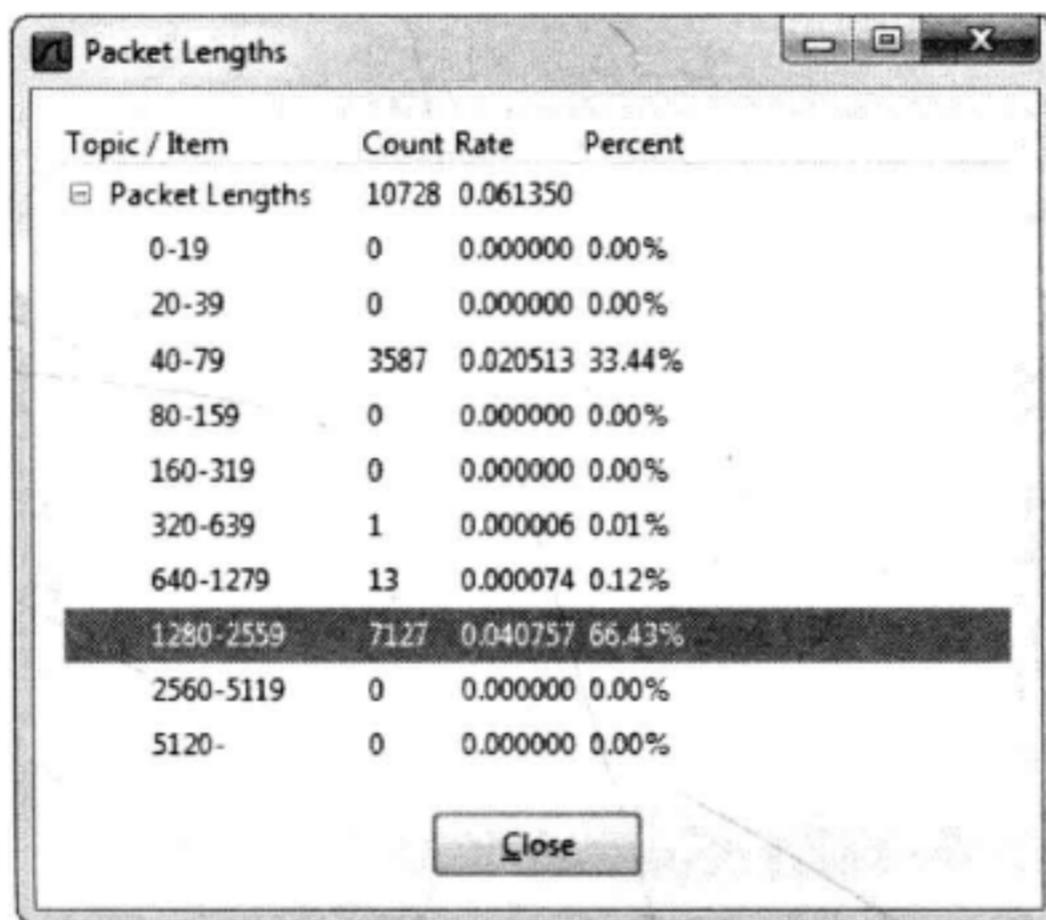


图 5-12 数据包长度窗口帮助你对捕获文件中的流量进行合理的猜测

我在大小为 1280~2559 字节的数据包统计数据区域标了高亮。这些较大的数据包通常用于传输数据，而较小的数据包则是协议控制序列。在这个例子中，我们看到较大的数据包占了相当大的比重（66.43%）。即使不看这个文件中的数据包，我们仍然可以知道捕获中包含了一个或多个数据传输流量。这可能是 HTTP 下载、FTP 上传，或者其他类型在主机之间进行数据传输的网络通信。

大多数剩下的数据包（33.44%）都是在 40~79 字节之间，而处于这个范围的数据包通常不包含数据的 TCP 控制数据包。我们可以想一下协议头一般的大。以太网头是 14 字节（包含 4 字节 CRC），IP 头最小 20 字节，没有数据以及选项的 TCP 数据包也是 20 字节，也就意味着典型的 TCP 控制数据包——例如 TCP、ACK、RST 和 FIN 数据包——大约是 54 字节并落入了这个区域。当然 IP 或 TCP 的额外选项会增加这个大小。

查看数据包长度是一个概览捕获文件的好方法。如果存在着很多较大的数据包，那么很大的可能便是进行了数据传输。如果绝大多数的数据包都很小，我们便可以假设这个捕获中存在协议控制命令，而没有传输大规模的数据。尽管这不是一个必需的操作，但在深入分析前做一些类似的假设，有时还是很保险的。

## 5.7 图形展示

图形是分析工作中必不可少的一部分，并且也是得到一个数据集概览的最好方法。Wireshark 中有一些不同的图形展示功能帮助你更好地了解捕获数据，首先介绍 IO 图形化功能。

### 5.7.1 查看 IO 图

Wireshark 的 IO 图窗口可以让你对网络上的吞吐量进行绘图。你可以利用这些图，找到数据吞吐的峰值，找出不同协议中的性能时滞，以及用来比较实时数据流。

打开 download-fast.pcap，单击任意一个 TCP 数据包将其高亮显示，然后选择 **Statistics->IO Graphs** 就可以看到一台电脑从互联网下载文件时 IO 图的例子。

这个 IO 图窗口显示了捕获文件过程中数据流的一个图形化视图。在图 5-13 的例子中，IO 图显示了下载量，每个周期大约有 500 个数据包，在其过程中基本上保持不变，并在最后逐渐减少。

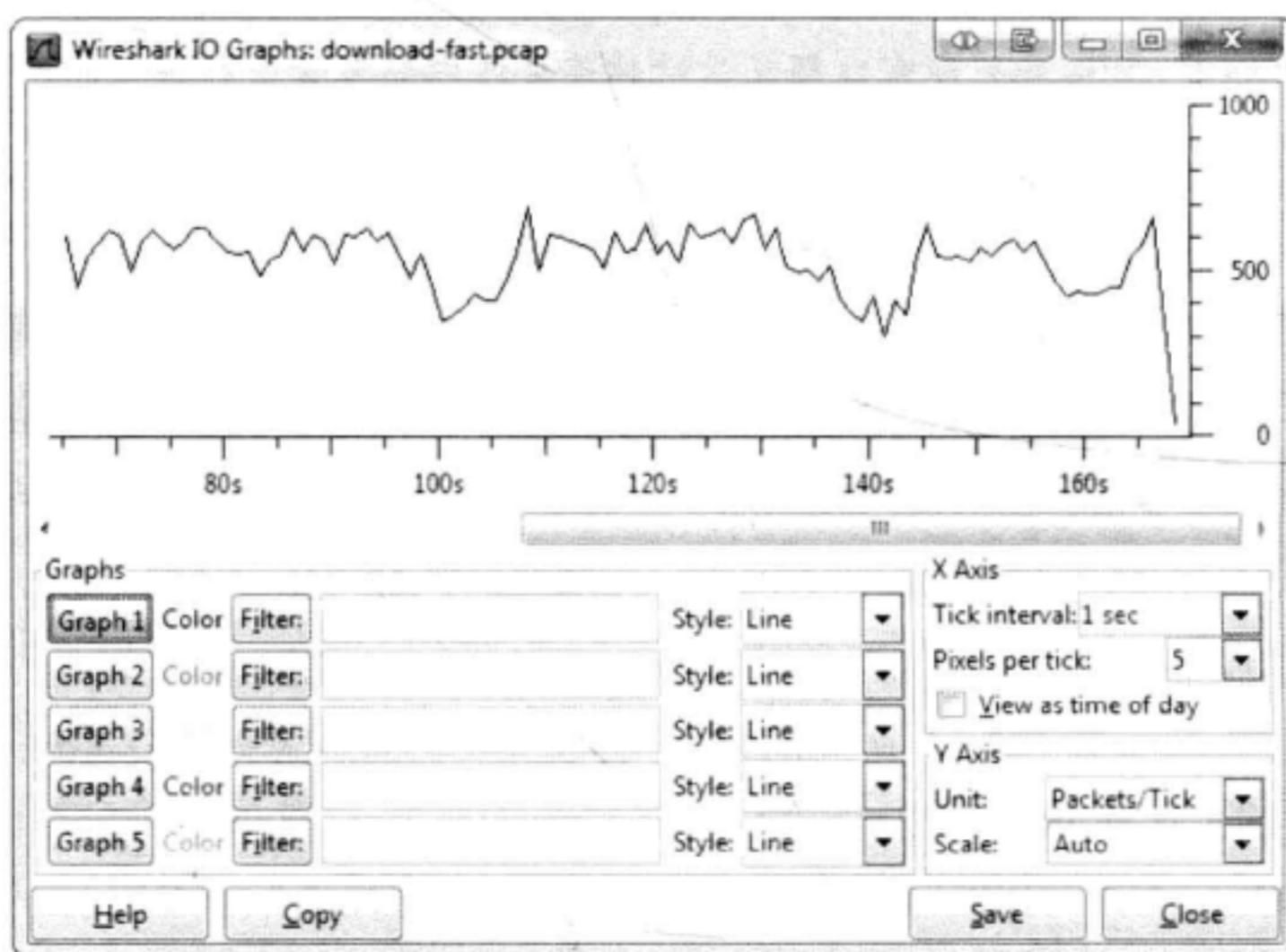


图 5-13 快速下载的 IO 图基本上是稳定的

我们可以将它与一个较慢的下载过程做一个比较。不要关闭当前这个文件，然后另外再启动一个 Wireshark 并打开 download-slow.pcap。打开这个下载过程的 IO 图，如图 5-14 所示，便可以看到其与之前的 IO 图大为不同。

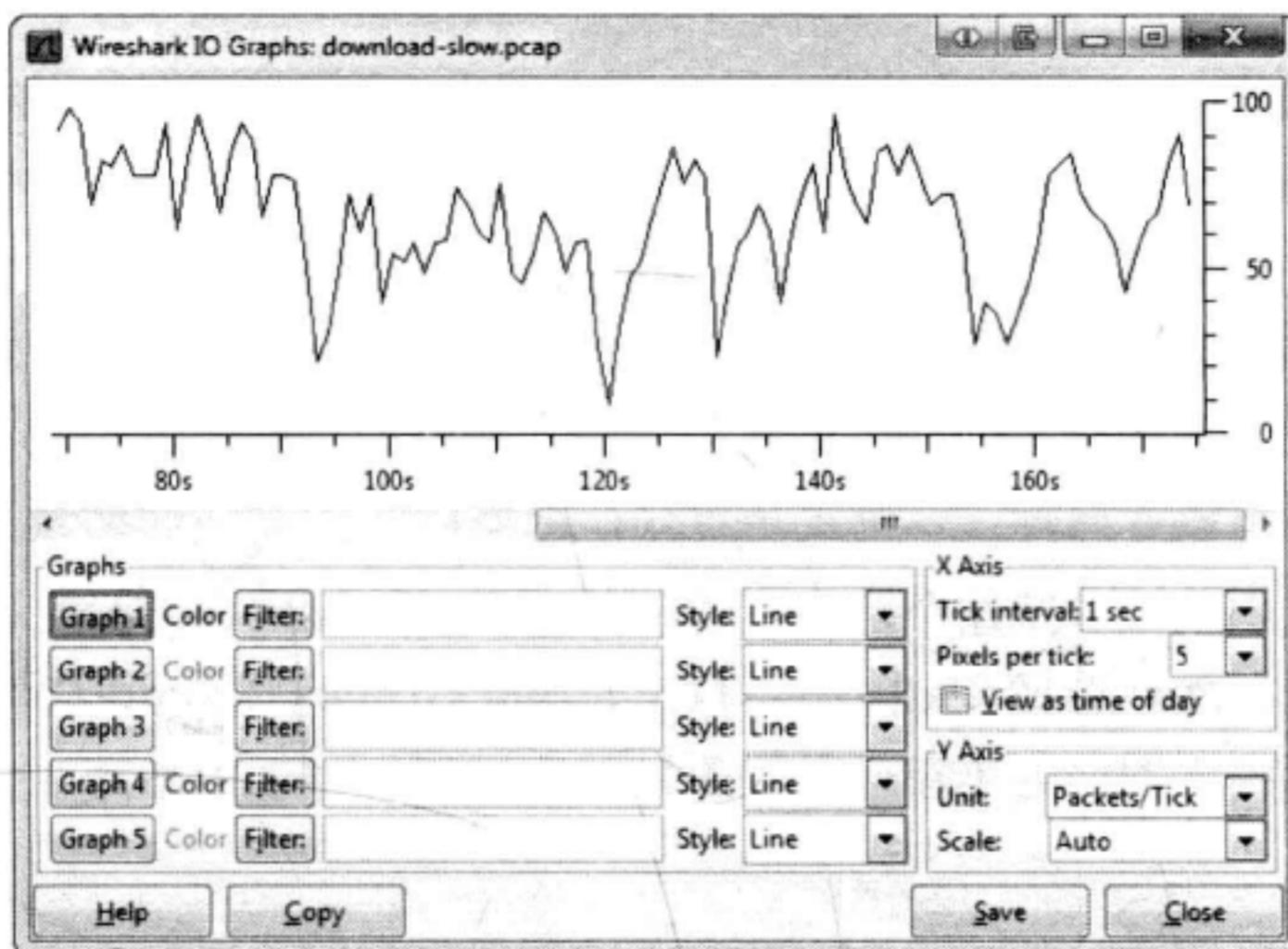


图 5-14 慢速下载的 IO 图一点都不稳定

这个下载过程每秒传输的数据包在 0~100 个数据包之间，并且波动很大，其中甚至暂时接近每秒 0 个数据包。如果你将这两个捕获文件的 IO 并排放置，你就能更清楚地看到这些波动（如图 5-15 所示）。

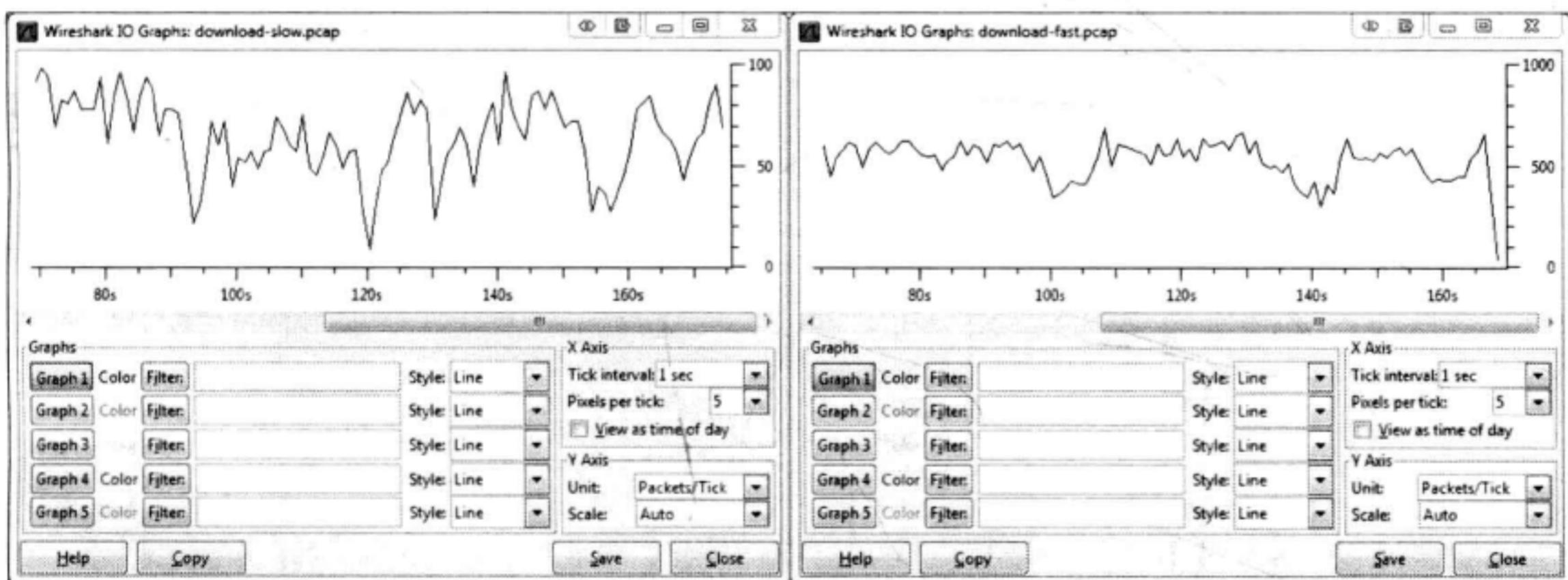


图 5-15 并排查看多个 IO 图有助于发现它们之间的差异

你应该可以注意到这个窗口的下面有一些配置选项。你可以创建 5 个不同的过滤器（使用与将在第 6 章和第 7 章介绍的显示或者捕获过滤器相同的语法），并为这些过滤器指定显示的颜色。比如你可以创建只显示 ARP 和 DHCP 流量的过滤器，并让这些线在图上分别以红色和蓝色显示。这样你可以更加容易分辨出这两种类型的吞吐趋势。

## 5.7.2 双向时间图

Wireshark 中的另一个绘图功能就是对给定捕获文件进行双向时间绘图。双向时间（round-trip time, RTT）就是确认一个数据包已被成功接收所需的时间。解释得更清楚一点就是，双向时间就是你的数据包抵达目的地和这个数据包抵达所发送的确认返回到你的时间之和。对双向时间的分析通常被用来找到通信中的慢点或者瓶颈，以确定是否存在延迟。

我们来试一试这个功能吧。打开 download-fast.pcap 这个文件，选择一个 TCP 数据包，然后选择 **Statistics->TCP Stream Graph->Round Trip Time Graph**，查看双向时间图。这个双向时间图如图 5-16 所示。

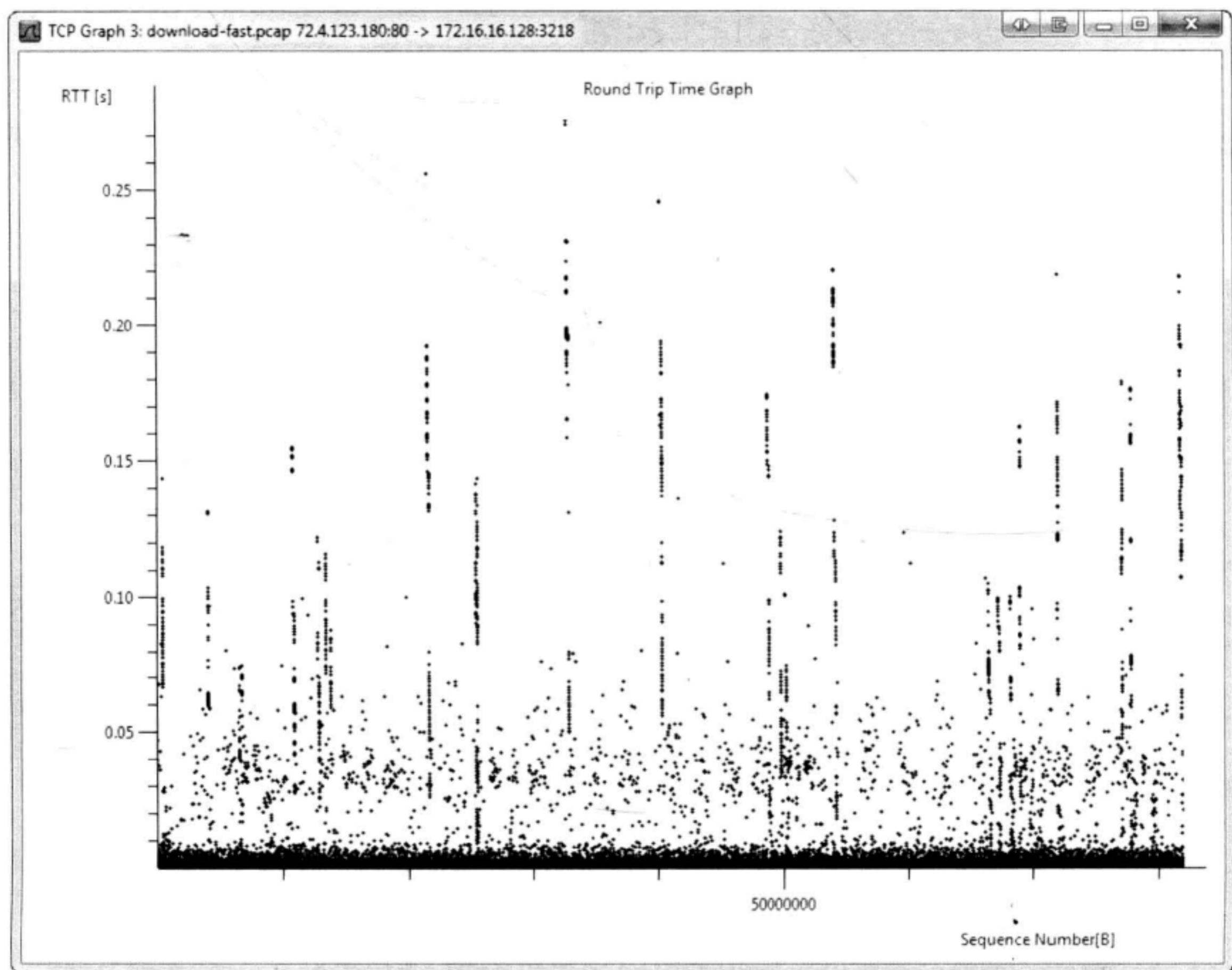


图 5-16 这个下载的 RTT 图除了一些偏离值之外大体上还是保持稳定的

这个图中的每一个点都代表了一个数据包的双向时间。在默认情况下，这些值按照其序列号排序。你可以单击这个图中的任何一个点，并将在 Packet List 面板中看到相应的数据包。

看上去这个快速下载过程双向时间图中的双向时间大多都在 0.05s 以下，并有一些较慢的点位于 0.10~0.25s 之间。尽管存在少量的值超出了可以接受的范围，但大多数的双向时间还是可以的，所以对于文件下载来说，这个双向时间是可以接受的。

### 5.7.3 数据流图

数据流绘图功能对于将连接可视化，以及将一段时间中的数据流显示出来，非常有用。数据流图一般以列的方式将主机之间的连接显示出来，并将流量组织到一起，以便于你更直观地解读。

打开 http\_google.pcap，并选择 **Statistics->Flow Graph**，便可以创建一个数据流图。你应该可以看到一个小对话框，其中有一些简单的选项对应所要处理的数据包以及数据流的类型。在这个例子中，使用默认设置即可，单击 OK，便可以创建一个数据流图（如图 5-17 所示）。

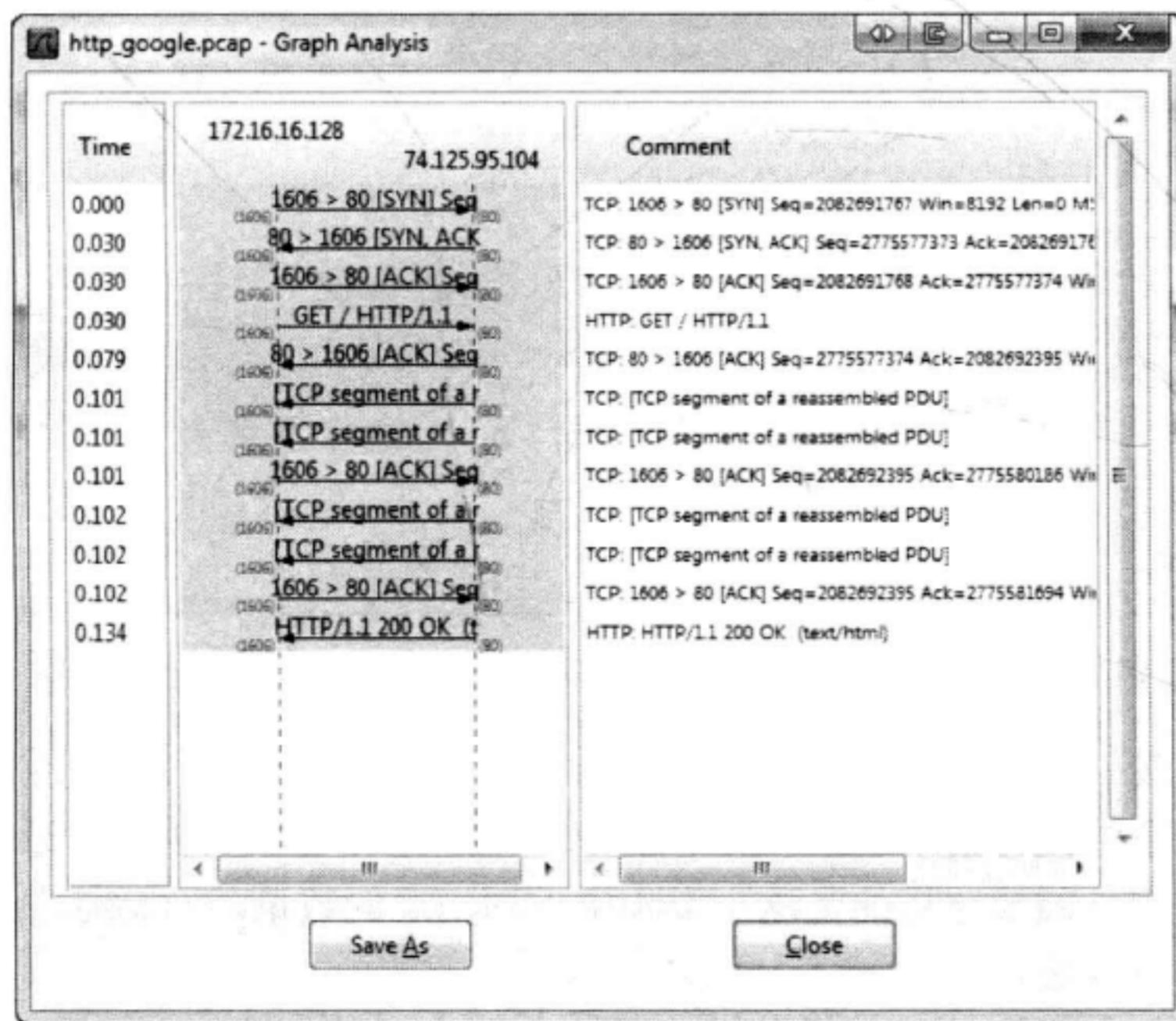


图 5-17 TCP 流图可以让我们更好地看到整个连接

## 5.8 专家信息

Wireshark 中每个协议的解析器都有一些专家信息，可以让你得到使用这个协议的数据包中一些特定状态的警告。这些状态可以分为 4 类。

**对话：**关于通信的基本信息。

**注意：**正常通信中的异常数据包。

**警告：**不是正常通信中的异常数据包。

**错误：**数据包中的错误，或者解析器解析时的错误。

举例来说，打开 download-slow.pcap 这个文件，然后单击 **Analyze**，并选择 **Expert Info Composite**，便可以打开这个捕获文件的专家信息窗口（如图 5-18 所示）。

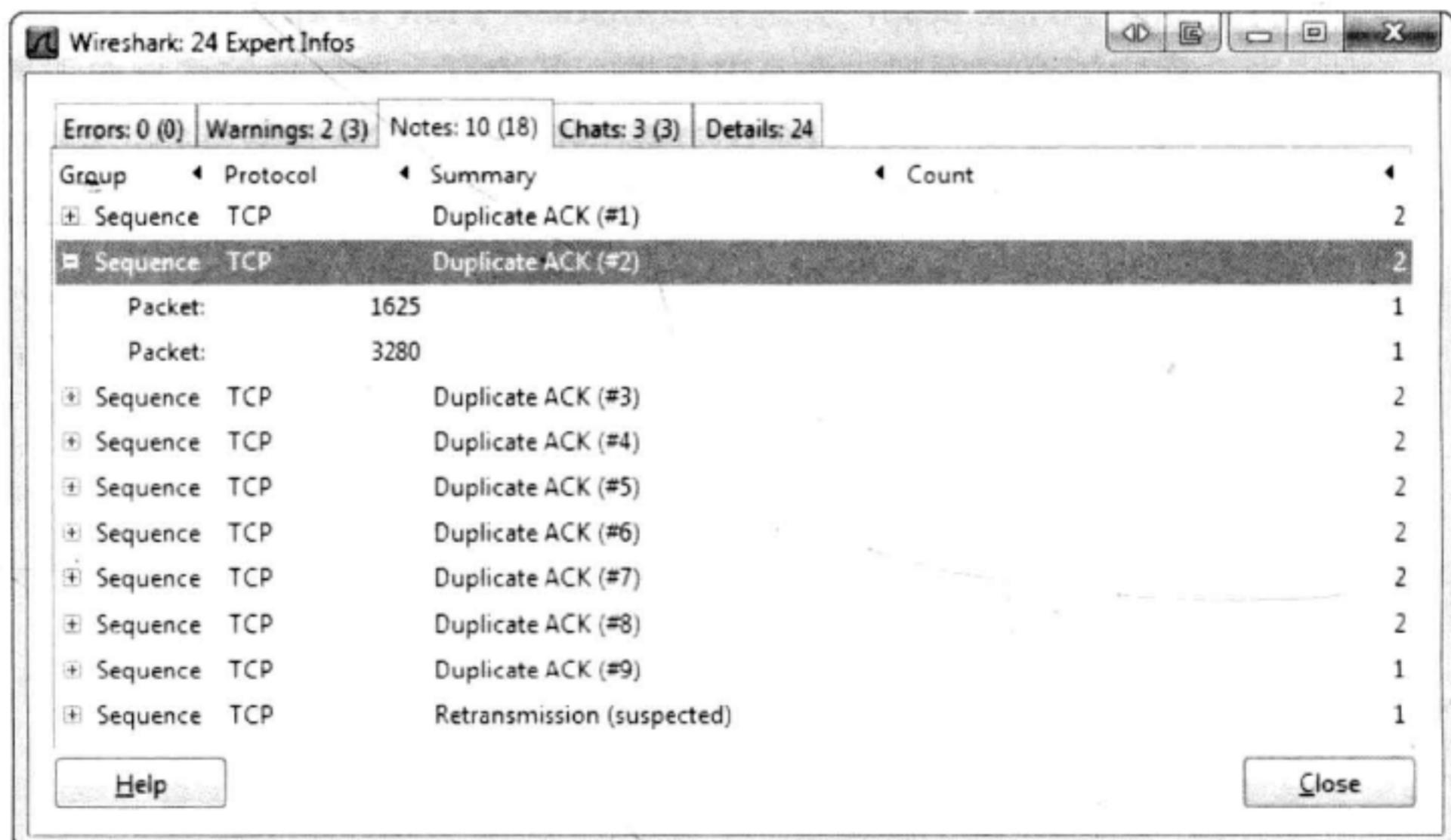


图 5-18 专家信息窗口给出了协议解析器中内置专家系统的信息

我们应该注意到这个窗口中每种类型的信息都有一个对应的选项卡，在这个例子中没有错误消息，有 3 个警告、18 个注意以及 3 个对话。在选项卡上，括号之外的数字指的是这个类别中不同消息的数量，而括号中的数字是消息出现的总次数。

这个捕获文件中所有的信息都是与 TCP 有关的，因为至本书截稿时，专家

信息系统还没有在其他的协议中实现。目前，总共为 TCP 配置了 14 种专家信息消息，并且这些消息在解决捕获文件的问题时非常有用。这些信息可以在满足如下条件的时候对数据包进行标记。

### 对话消息

**窗口更新** 由接收者发送，用来通知发送者 TCP 接收窗口的大小已被改变。

### 注意消息

**TCP 重传输** 数据包丢失的结果。发生在收到重复的 ACK，或者数据包的重传输计时器超时的时候。

**重复 ACK** 当一台主机没有收到下一个期望序列号的数据包时，它会生成最近收到一次数据的重复 ACK。

**零窗口探查** 在一个零窗口包被发送之后，用来监视 TCP 接收窗口的状态（将在第 9 章中介绍）。

**保活 ACK** 用来响应保活数据包。

**零窗口探查 ACK** 用来响应零窗口探查数据包。

**窗口已满** 用来通知传输主机其接收者的 TCP 接收窗口已满。

### 警告信息

**上一段丢失** 指明数据包丢失。发生在当数据流中一个期望的序列号被跳过时。

**收到丢失数据包的 ACK** 发生在当一个数据包已经确认丢失但收到了其 ACK 数据包时。

**保活** 当一个连接的保活数据包出现时触发。

**零窗口** 当接收方已经达到 TCP 接收窗口大小时，发出一个零窗口通知，要求发送方停止传输数据。

**乱序** 当数据包被乱序接收时，会利用序列号进行检测。

**快速重传输** 一次重传会在收到一个重复 ACK 的 20 毫秒内进行。

## 错误消息

### 无错误消息

关于这些消息的意义，我们会在第 6 章中学习 TCP 以及第 9 章检测慢速网络问题时了解到。

尽管这章中介绍的一些功能看上去只有在一些冷僻的情况下用到，你可能会发现它们比你想象中使用得要多。你需要熟悉这些窗口和选项，因为我会在之后的几个章节中频繁地提到它们。

# 第6章

## 通用底层网络协议



无论是处理延迟问题，还是甄别存在错误的应用，抑或对安全威胁进行聚焦检查，都是为了发现异常的流量，而你必须首先了解正常的流量。在下面的几章中，你将会学到正常的网络流量在数据包级别是如何工作的。

我们将介绍最常见的几种协议，包括最基础的 TCP、UDP 和 IP，以及如 HTTP、DHCP、DNS 等最常用的应用层协议。在每个协议的相关部分，都会至少有一个捕获文件供你下载，并可以让你直接上手分析。在这一章中，我们将着重关注在 OSI 分层模型中从第 1 层到第 4 层的底层协议。

这应该是这本书中最重要的几章。如果跳过了这些内容，你会感到如同周末晚餐中没有牛角面包那样不完整，即使你对每个协议是如何工作的都了然于胸，那也请至少快速浏览一遍，以便复习每个协议数据包的结构。

## 6.1 地址解析协议

网络上的通信会使用到逻辑地址和物理地址。逻辑地址可以使得不同网络以及没有直接相连的设备之间能够进行通信。物理地址则用来在单一网段中交换机直接连接的设备之间进行通信。在大多数情况下，正常通信需要这两种地址协同工作。

我们假设这样一个场景：你需要和你网络中的一个设备进行通信，这个设备可能是某种服务器，或者只是你想与之共享文件的另一个工作站。你所用来创建这个通信的应用已经得到了这个远程主机的 IP 地址（通过 DNS 服务，这将在第 7 章中介绍），也意味着系统已经拥有了用来构建它想要在第 3 层到第 7 层中传递的数据包所有需要的信息。这时它所需要的唯一信息就是第 2 层包含目标主机 MAC 地址的数据链路层数据。

之所以需要 MAC 地址，是因为网络中用于连接各个设备的交换机使用了内容寻址寄存器（CAM）。这个表列出了它在每一个端口的所有连接设备的 MAC 地址。当交换机收到了一个指向特定 MAC 地址的流量，它会使用这个表，来确定应该使用哪一个端口发送流量。如果目标的 MAC 地址是未知的，这个传输设备会首先在它的缓存中查找这个地址，如果没有找到，那么这个地址就需要在网络上额外的通信来进行解析了。

TCP/IP 网络（基于 IPv4）中用来将 IP 地址解析为 MAC 地址的过程称为地址解析协议（Address Resolution Protocol, ARP）。这个协议在 RFC826 中进行了定义，它的解析过程只使用两种数据包：一个 ARP 请求与一个 ARP 响应，如图 6-1 所示。

---

注

RFC（Request for Comments）是定义协议实现标准的官方文档。你可以在 RFC Editor 的首页上搜索 RFC 文档，<http://www.rfc-editor.org/>。

---

这个传输计算机会发出一个 ARP 请求，基本上就是问“大家好，我的 IP 地址是 XX.XX.XX.XX，MAC 地址是 XX:XX:XX:XX:XX:XX。我需要向那个 IP 地址是 XX.XX.XX.XX 的家伙发些东西，但我不知道它的硬件地址，你们谁有这个 IP 地址的，可否请回复给我你的 MAC 地址？”

这个数据包将被广播给网段中的所有设备。不是这个 IP 地址的设备将简单

地丢弃这个数据包，而拥有这个 IP 地址的设备将发送一个 ARP 响应，就像是说：“你好，传输设备，我就是你所找的那个拥有 IP 地址为 XX.XX.XX.XX 的，我的 MAC 地址是 XX:XX:XX:XX:XX:XX。”

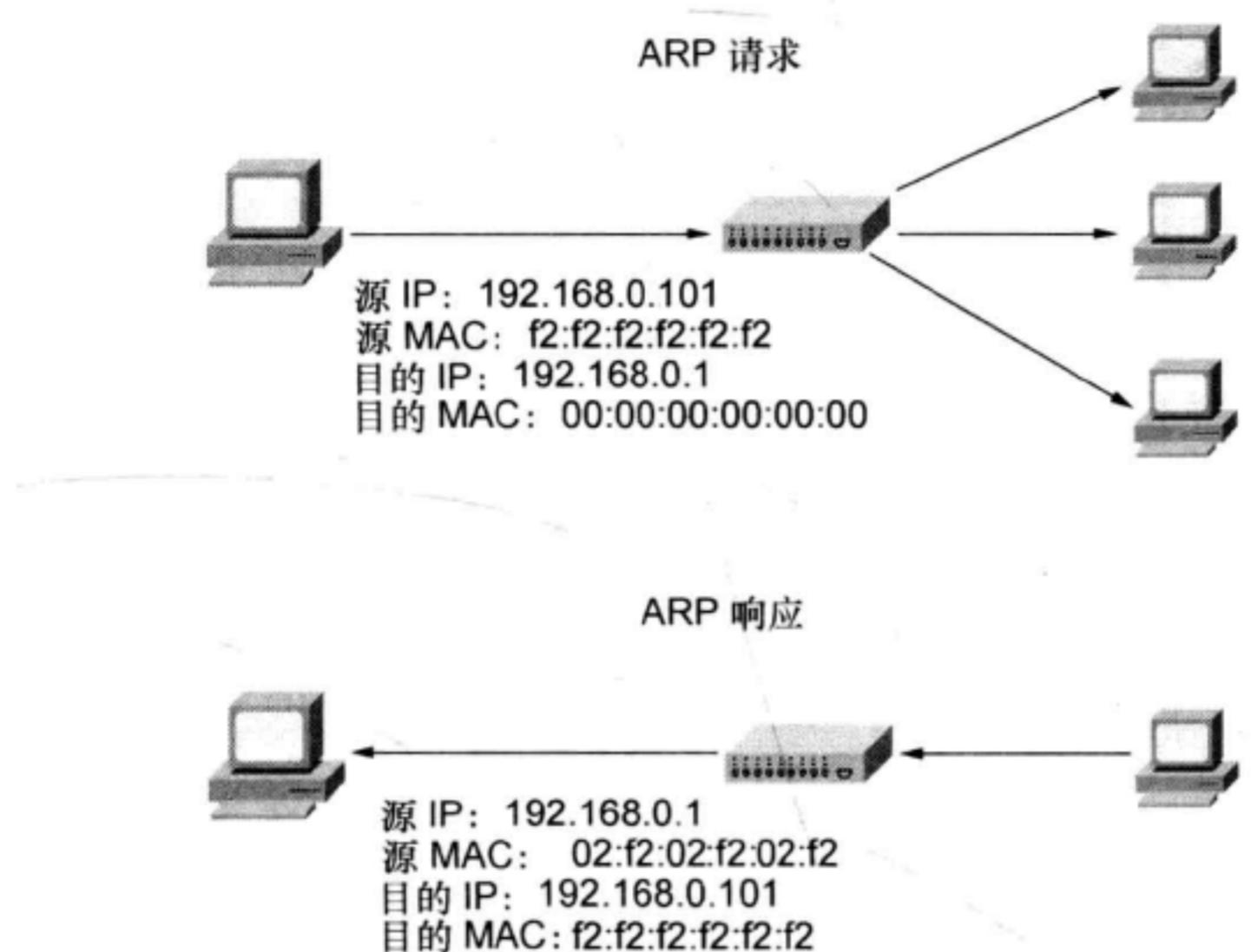


图 6-1 ARP 解析过程

一旦这个解析过程完成了，传输设备就会对这个设备 MAC 和 IP 对应关系的缓存进行更新，并且开始传输数据。

注

在 Windows 主机中，你可以通过在命令行中键入 arp -a 来查看 ARP 表。

通过实际情况来看地址解析的这个过程，有助于你更好地理解它究竟是怎么工作的。但是在看一些例子之前，我们先介绍一下 ARP 数据包头。

### 6.1.1 ARP 头

如图 6-2 所示，ARP 头包含下列的几个域。

**硬件类型：**数据链路层使用的类型数据。在大多数情况下，这个类型都是以太网（类型 1）。

**协议类型：**ARP 请求正在使用的高层协议。

**硬件地址长度：**正在使用的硬件地址的长度（八位组/字节）。

| 地址解析协议 |                 |        |
|--------|-----------------|--------|
| 偏移位    | 0~7             | 8~15   |
| 0      | 硬件类型            |        |
| 16     | 协议类型            |        |
| 32     | 硬件地址长度          | 协议地址长度 |
| 48     | 操作              |        |
| 64     | 发送方硬件地址(第1个16位) |        |
| 80     | 发送方硬件地址(第2个16位) |        |
| 96     | 发送方硬件地址(第3个16位) |        |
| 112    | 发送方协议地址(第1个16位) |        |
| 128    | 发送方协议地址(第2个16位) |        |
| 144    | 目标硬件地址(第1个16位)  |        |
| 160    | 目标硬件地址(第2个16位)  |        |
| 176    | 目标硬件地址(第3个16位)  |        |
| 192    | 目标协议地址(第1个16位)  |        |
| 208    | 目标协议地址(第2个16位)  |        |

图 6-2 ARP 数据包结构

**协议地址长度：**对于指定协议类型所使用的逻辑地址的长度（八位组/字节）。

**操作：**ARP 数据包的功能：1 表示请求，2 表示响应。

**发送方硬件地址：**发送者的硬件地址。

**发送方协议地址：**发送者的高层协议地址。

**目标硬件地址：**目标接收方的硬件地址（ARP 请求中为 0）。

**目标协议地址：**目标接受方的高层协议地址。

现在打开 arp\_resolution.pcap 这个文件，就可以看到实际的解析过程。我们将对这个过程中的每个数据包单独进行分析。

## 6.1.2 数据包 1：ARP 请求

如图 6-3 所示，第 1 个数据包是一个 ARP 请求。我们可以通过在 Wireshark 的 Packet Details 面板中，检查以太网头，来确定这个数据包是否是一个真的广播数据包。这个数据包的目的地址是 ff:ff:ff:ff:ff:ff❶。这是一个以太网的广播地址，所有发送到这个地址的数据包都会被广播到当前网段中的所有设备。这个数据包中以太网头的源地址就是我们的 MAC 地址❷。

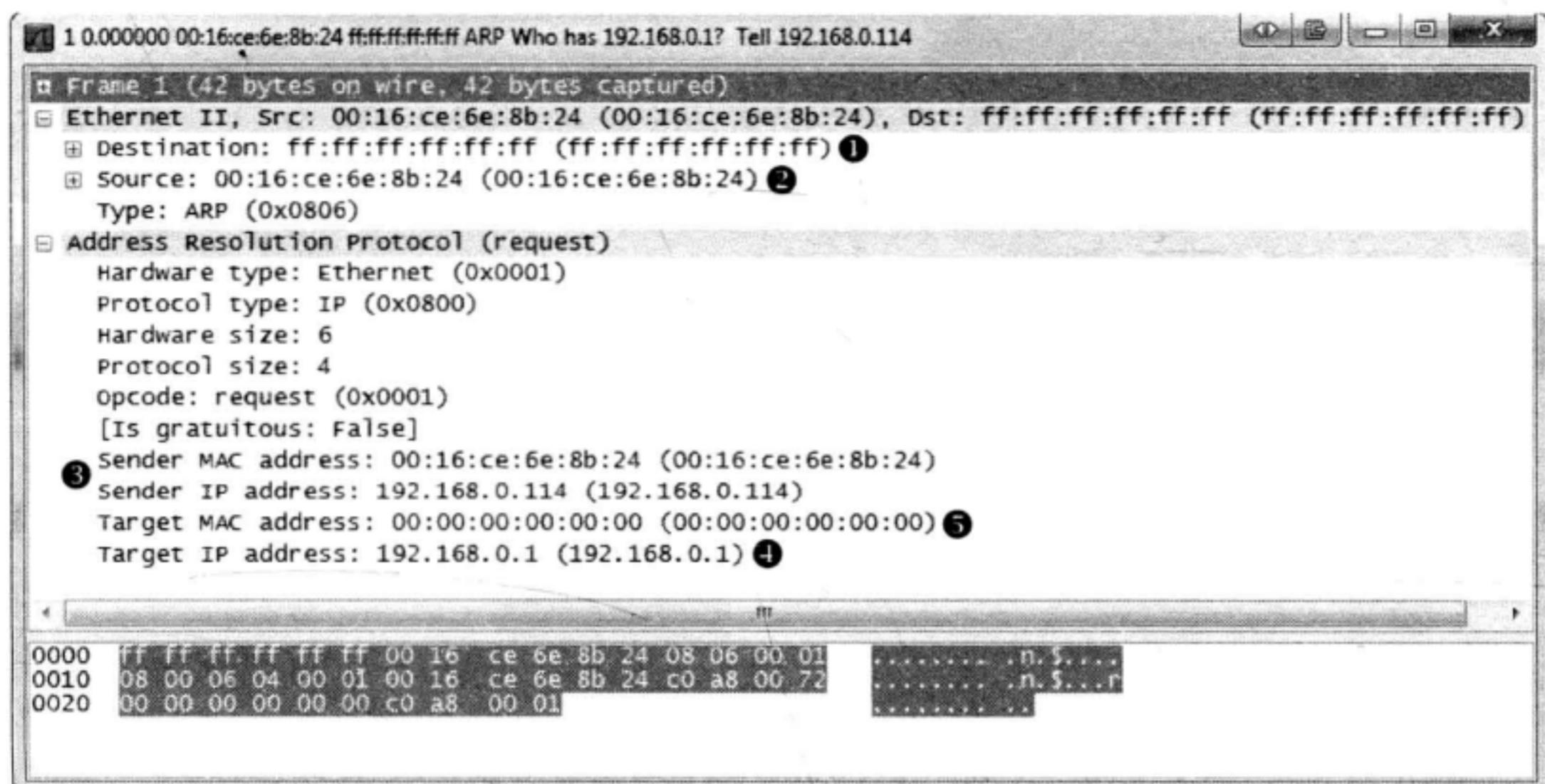


图 6-3 一个 ARP 请求数据包

在这个给定的结构中，我们可以确定这的确是一个在以太网上使用 IP 的 ARP 请求。这个 ARP 头列出了发送方的 IP (192.168.0.114) 和 MAC 地址 (00:16:ce:6e:8b:24) ③，以及接收方的 IP 地址 192.168.0.1④。我们想要得到的目标 MAC 地址，还是未知的，所以这里的目的 MAC 地址填写为 00:00:00:00:00:00。

### 6.1.3 数据包 2：ARP 响应

在我们对最初请求的响应中（如图 6-4 所示），第一个数据包中的源 MAC 地址成为了这个以太网头中的目的地址。这个 ARP 响应和之前的 ARP 请求看上去很像，除了以下几点。

- 数据包的操作码 (opcode) 现在是 0x0002，用来表示这是一个响应而不是请求。
- 地址信息进行了颠倒——发送方的 MAC 地址和 IP 地址现在变成了目的 MAC 地址和 IP 地址。
- 最重要的是，现在数据包中所有的信息都是可用的，也就是说我们现在有了 192.168.0.1 主机的 MAC 地址 (00:13:46:0b:22:ba)。

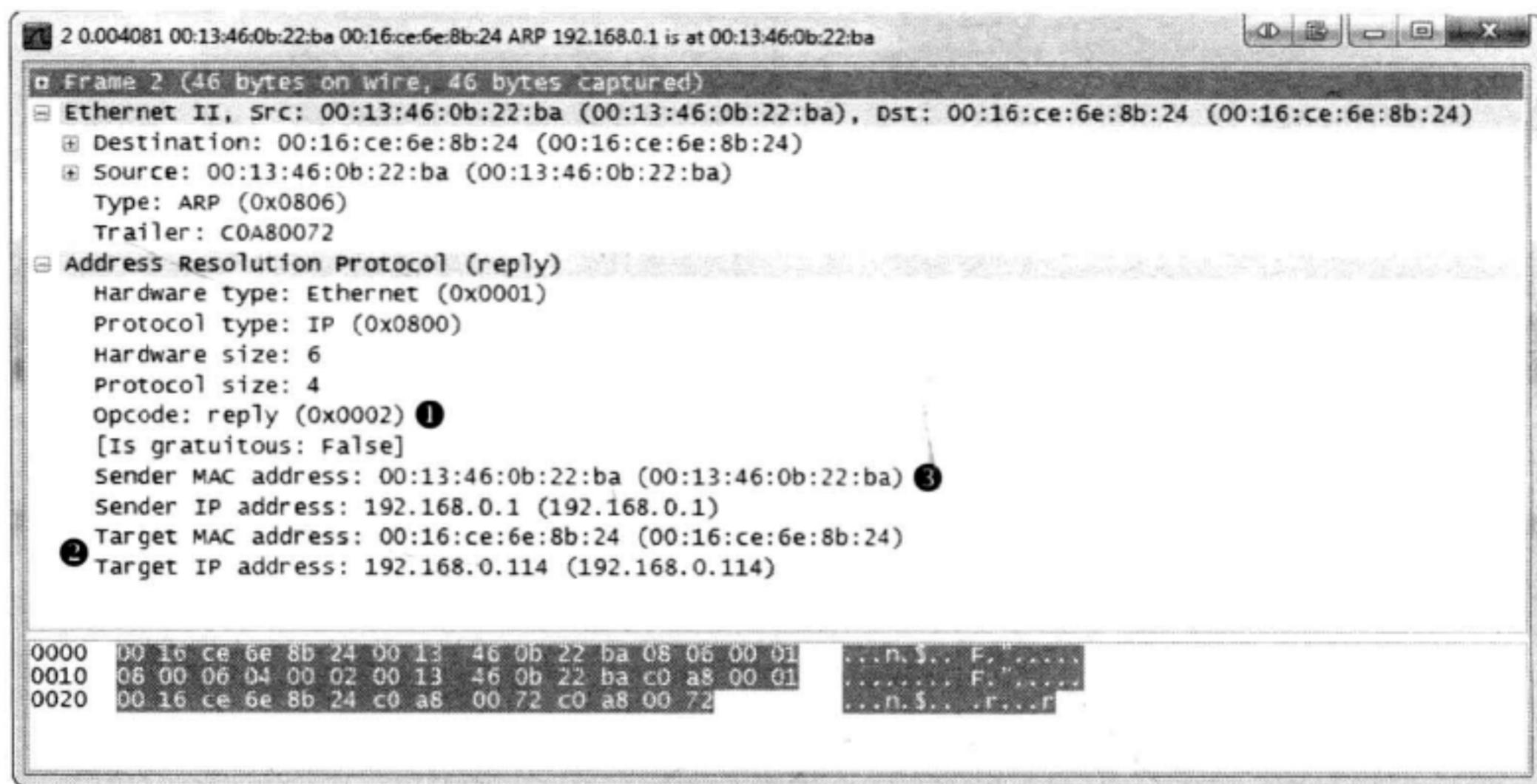


图 6-4 一个 ARP 回复数据包

#### 6.1.4 无偿的 ARP

在我的家乡，当一些事是所谓的“无偿”的时候，那通常没有什么好的含义。但无偿发送的 ARP 却是一个好东西。

在多数情况下，一个设备的 IP 地址是可以改变的。当这样的改变发生后，网络主机中缓存的 IP 和 MAC 地址映射就不再有效了。为了防止造成通信错误，无偿的 ARP 请求会被发送到网络中，强制所有收到它的设备去用新的 IP 和 MAC 地址映射更新缓存（如图 6-5 所示）。

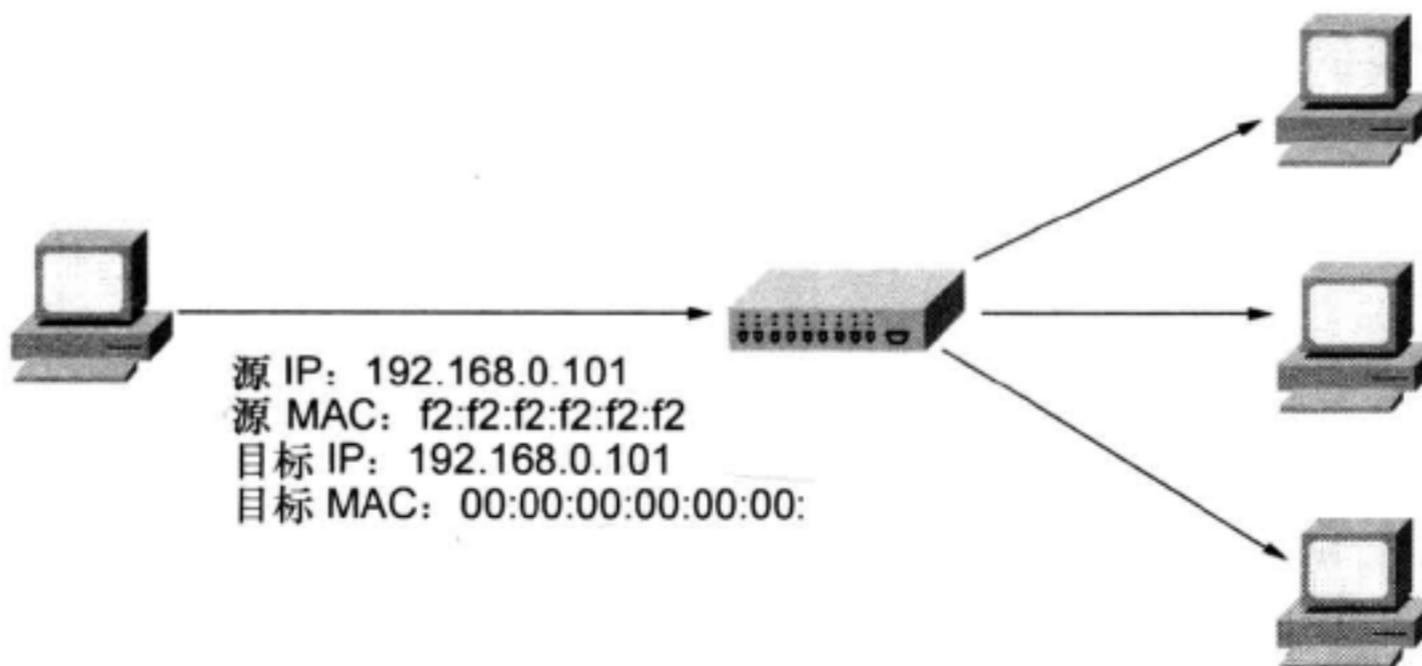


图 6-5 无偿 ARP 工作过程

几个不同的情形都会产生无偿 ARP 数据包，其中一个最常见的就是 IP 地

址的改变。打开 `arp_gratuitous.pcap` 这个捕获文件，你就会看到一个实际例子。这个文件只包含一个数据包（如图 6-6 所示），因为这就是无偿数据包的全部了。

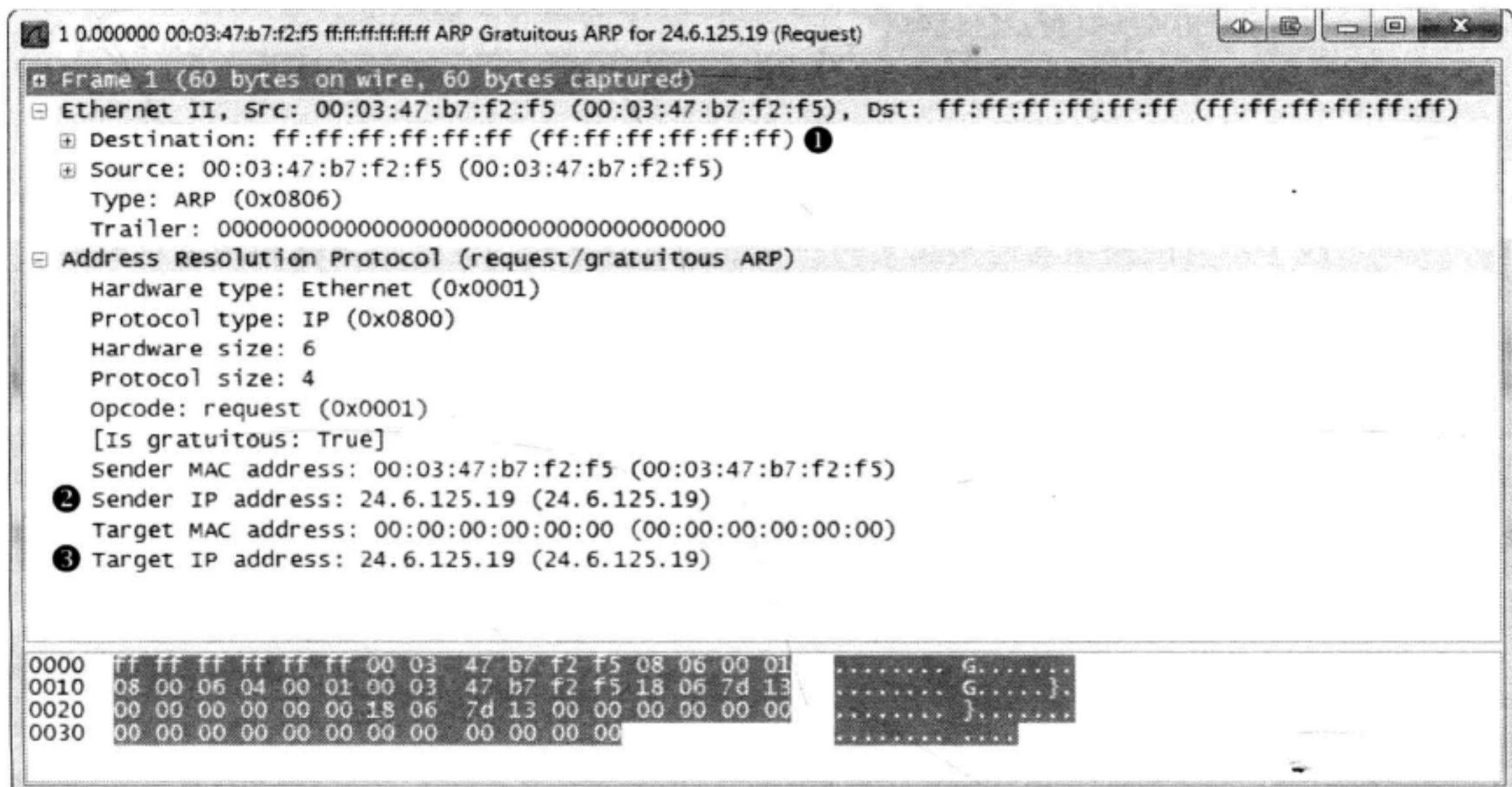


图 6-6 一个无偿 ARP 数据包

检查这个以太网头，你会看见这个数据包是以广播的形式发送，以便网络上的所有主机能够接收到它❶。这个 ARP 头看上去和 ARP 请求很像，除了发送方的 IP 地址❷和目标 IP 地址❸是相同的。当这个数据包被网络中的其他主机接收到之后，它会让这些主机使用新的 IP 和 MAC 地址关系更新它们的 ARP 表。由于这个 ARP 数据包是未经请求的，却导致客户端更新 ARP 缓存，所以会称之为无偿。

你会在一些不同的情形下注意到无偿 ARP 数据包的存在。如上所示，设备 IP 地址的改变会生成它，并且一些操作系统也会在启动时进行无偿 ARP 的发送。此外，你可能会注意到一些系统使用无偿 ARP 数据包对流入流量进行负载均衡。

## 6.2 互联网协议

位于 OSI 模型中第 3 层的协议的主要目的就是使得网络间能够互联互通。正如你所知道的，MAC 地址被用来在第 2 层处理单一网络中的通信。与其类似，

第 3 层则负责跨网络通信的地址。在这层上工作的不止一个协议，但最普遍的还是互联网协议（IP）。在此，我们将介绍在 RFC 791 中所定义的 IP 协议版本 4（IPv4）。

如果网络中的所有设备仅使用集线器或者交换机进行连接，那么这个网络称为局域网。如果你想将两个局域网连接起来，你可以使用路由器办到这一点。在复杂的网络中，可能会包含成千上万的局域网，而这些局域网是由世界各地成千上万的路由器连接起来的。互联网本身就是由无数局域网和路由器所组成的一个集合。

## 6.2.1 IP 地址

IP 地址是一个 32 位的地址，用来唯一标识连接到网络的设备。由于让人记住一串 32 位长的 01 字符确实比较困难，所以 IP 地址采用点分四组的表示法。

在点分四组表示法中，以 A.B.C.D 的形式，构成 IP 地址的四组 1 和 0 分别转换为十进制 0 到 255 之间的数（如图 6-7 所示）。我们拿这样一个 IP 地址 11000000 10101000 00000000 00000001 举例来说，这个值显然不容易记忆或者表示，但如果采用点分四组的表示法，我们就可以将其表示为 192.168.0.1。

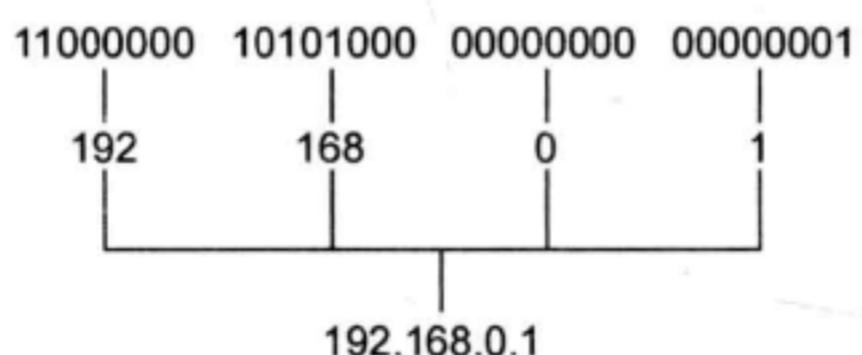


图 6-7 IPv4 地址的点分四组表示法

IP 地址之所以会被分成 4 个单独的部分，是因为每个 IP 地址都包含着两个部分：网络地址和主机地址。网络地址用来标识设备所连接到的局域网，而主机地址则标识这个网络中的设备本身。用来决定究竟 IP 地址哪部分属于网络或者主机的划分通常并不唯一。这实际上是由另一组名为网络掩码（netmask, network mask）的地址信息所决定的，有时它也会被称为子网掩码（subnet mask）。

网络掩码用来标识 IP 地址中究竟哪一部分属于网络地址而哪一部分属于主机地址。网络掩码也是 32 位长，并且被设为 1 的每一位都标识着 IP 地址的对应部分是属于网络地址的，而剩下设为 0 的部分则标识着主机地址。

我们以 IP 地址 10.10.1.22 为例，其二进制形式为 00001010 00001010 00000001 00010110。为了能够区分出 IP 地址的每一个部分，我们将使用网络掩码。在这个例子中，我们的网络掩码是 11111111 11111111 00000000 00000000。这意味着 IP 地址的前一半（10.10 或者 00001010 00001010）是网络地址，而后一半（1.22 或者 00000001 00010110）标识着这个网络上的主机，如图 6-8 所示。

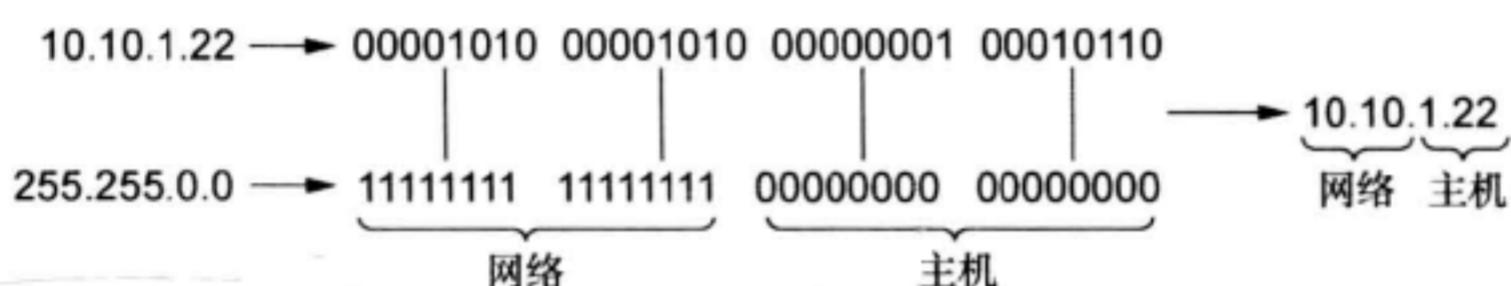


图 6-8 网络掩码决定了 IP 地址中比特位的分配

网络掩码也可以写成点分四组的形式。比如网络掩码 11111111 11111111 00000000 00000000 可以被写成 255.255.0.0。

IP 地址和网络掩码为简便起见，通常会被写成无类型域间选路（Classless Inter-Domain Routing, CIDR）的形式。在这个形式下，一个完整的 IP 地址后面会有一个左斜杠 (/)，以及一个用来表示 IP 地址中网络部分位数的数字。举例来说，IP 地址 10.10.1.22 和网络掩码 255.255.0.0，在 CIDR 表示法下就会被写成 10.10.1.22/16 的形式。

## 6.2.2 IPv4 头

源 IP 地址和目的 IP 地址都是 IPv4 数据包头中的重要组成部分，但它们并不是你在数据包中能够找到的 IP 信息的全部。IP 头比起我们刚刚介绍过的 ARP 数据包复杂得多。这其中包含很多额外的信息，以便 IP 完成其工作。

如图 6-9 所示，IPv4 头有着下列几个域。

**版本号 (Version)**: IP 所使用的版本。

**首部长度 (Header Length)**: IP 头的长度。

**服务类型 (Type of Service)**: 优先级标志位和服务类型标志位，被路由器用来进行流量的优先排序。

**总长度 (Total Length)**: IP 头与数据包中数据的长度。

**标识符 (Identification)**: 一个唯一的标识数字，用来识别一个数据包或者

被分片数据包的次序。

| IP 协议          |          |      |      |       |       |  |  |
|----------------|----------|------|------|-------|-------|--|--|
| 偏移位            | 0~3      | 4~7  | 8~15 | 16~18 | 19~31 |  |  |
| 0              | 版本       | 首部长度 | 服务类型 | 总长度   |       |  |  |
| 32             | 标识符      |      |      | 标记    | 分段偏移  |  |  |
| 64             | 存活时间     |      | 协议   | 首部校验和 |       |  |  |
| 96             | 源 IP 地址  |      |      |       |       |  |  |
| 128            | 目的 IP 地址 |      |      |       |       |  |  |
| 160            | 选项       |      |      |       |       |  |  |
| 160 or<br>192+ | 数据       |      |      |       |       |  |  |

图 6-9 IPv4 数据包结构

**标记 (Flags)**: 用来标记一个数据包是否是一组分片数据包的一部分。

**分片偏移 (Fragment Offset)**: 一个数据包是一个分片，这个域中的值就会被用来将数据包以正确的顺序重新组装。

**存活时间 (Time to Live)**: 用来定义数据包的生存周期，以经过路由器的跳数/秒数进行描述。

**协议 (Protocol)**: 用来识别在数据包序列中上层协议数据包的类型。

**首部校验和 (Header Checksum)**: 一个错误检测机制，用来确认 IP 头的内容没有被损坏或者篡改。

**源 IP 地址 (Source IP Address)**: 发出数据包的主机的 IP 地址。

**目的 IP 地址 (Destination IP Address)**: 数据包目的地的 IP 地址。

**选项 (Options)**: 保留作额外的 IP 选项。它包含着源站选路和时间戳的一些选项。

**数据 (Data)**: 使用 IP 传递的实际数据。

### 6.2.3 存活时间

存活时间 (TTL) 值定义了在该数据包被丢弃之前，所能经历的时间，或者能够经过的最大路由数目。TTL 在数据包被创建时就会被定义，而且通常在每次被发往一个路由器的时候减 1。举例来说，如果一个数据包的存活时间是 2，

那么当它到达第一个路由器的时候，其 TTL 会被减为 1，并会被发向第二个路由。这个路由器接着会将 TTL 减为 0，这时如果这个数据包的最终目的地不在这个网络中，那么这个数据包就会被丢弃，如图 6-10 所示。由于 TTL 的值在技术上还是基于时间的，一个非常繁忙的路由器可能会将 TTL 的值减去不止 1，但通常情况下，我们还是可以认为一个路由设备在多数情况下只会将 TTL 的值减去 1。

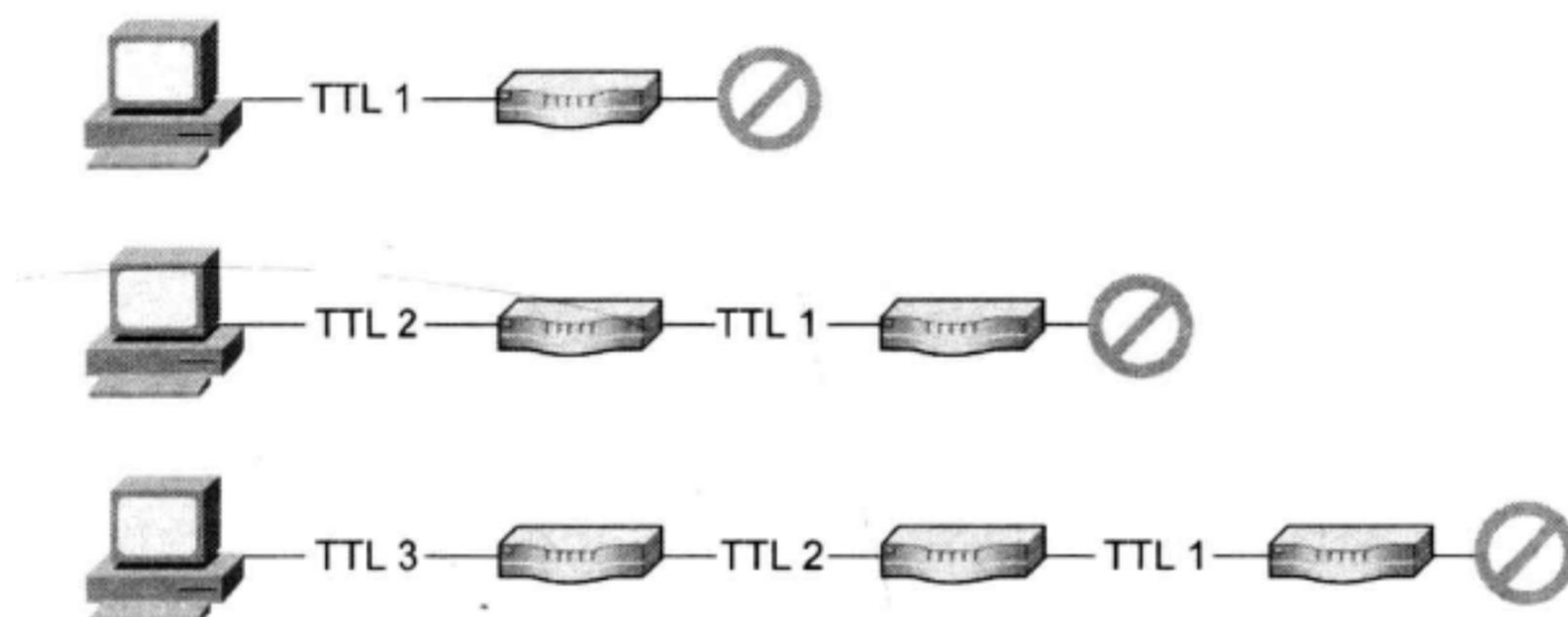


图 6-10 数据包的 TTL 在每次经过一个路由器的时候减少

为什么 TTL 的值会这样重要？我们通常所关心的一个数据包的生存周期，只是其从源前往目的地所花去的时间。但是考虑到一个数据包想要通过互联网发往一台主机需要经过数十个路由器。在这个数据包的路径上，它可能会碰到被错误配置的路由器，而失去其到达最终目的地的路径。在这种情况下，这个路由器可能会做很多事情，其中一件就是将数据包发向一个网络，而产生一个死循环。

如果你有任何编程背景，那么你就会知道死循环会导致各种问题，一般来说会导致一个程序或者整个操作系统的崩溃。理论上，同样的事情也会以数据包的形式发生在网络上。数据包可能会在路由器之间持续循环。随着循环数据包的增多，网络中可用的带宽就会减少，直至拒绝服务（DoS）的情况出现。IP 头中的 TTL 域就为了防止出现这个潜在的问题。

让我们看一下 Wireshark 中的示例。文件 ip\_ttl\_source.pcap 包含着两个 ICMP 数据包。ICMP（我们会在这章之后介绍到）利用 IP 传递数据包，我们可以通过在 Packet Details 面板中展开 IP 头区段看到。

你可以看到 IP 的版本号为 4①，IP 头的长度是 20 字节②，首部和载荷的总长度是 60 字节③并且 TTL 域的值是 128④。

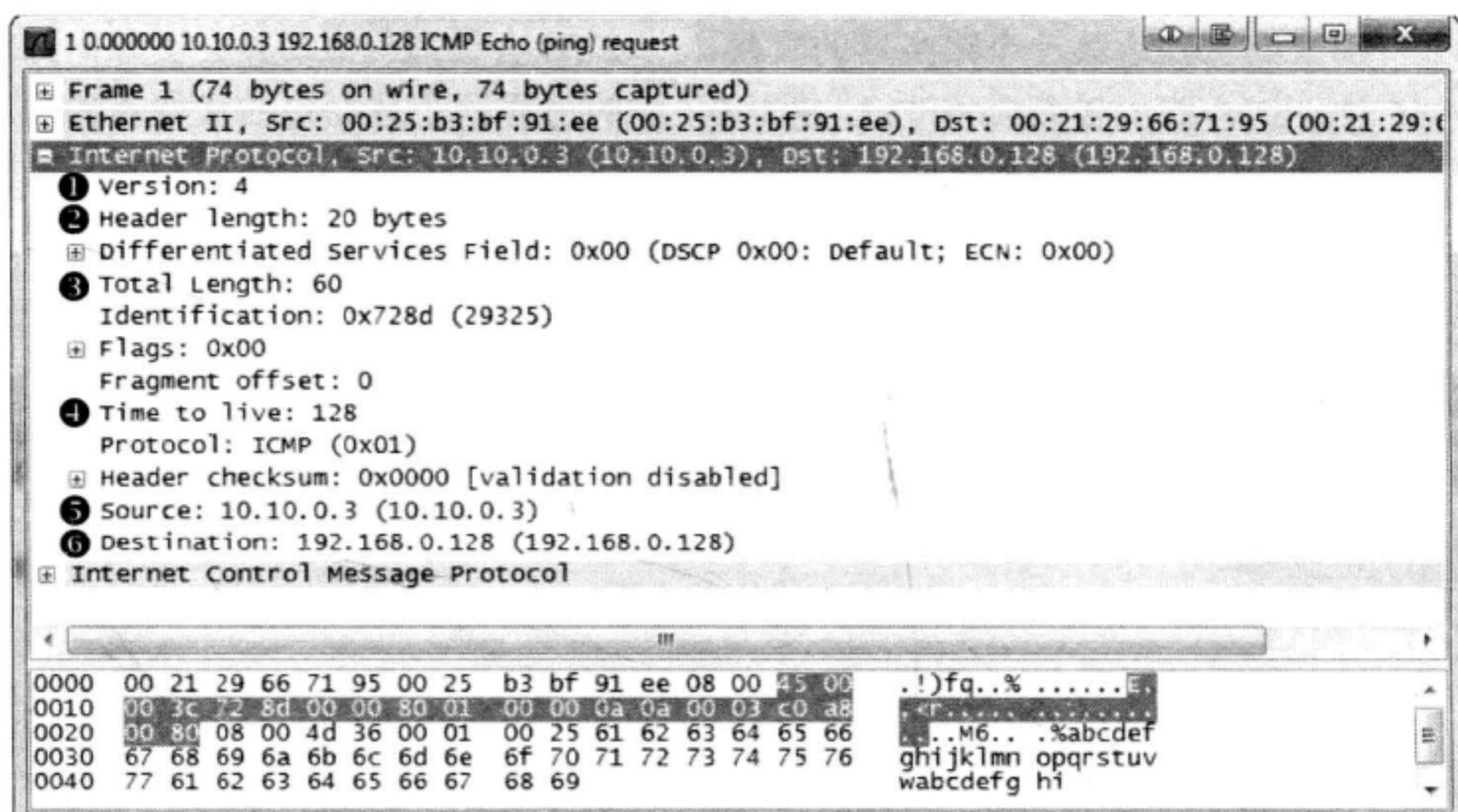


图 6-11 源数据包的 IP 头

ICMP ping 的主要目的就是测试设备之间的通信。数据从一台主机发往另一台作为请求，而后接收主机会将那个数据发回作为响应。这个文件中，我们一台 IP 地址为 10.10.0.3⑤的设备将一个 ICMP 请求发向了地址为 192.168.0.128⑥的设备。这个原始的捕获文件是在源主机 10.10.0.3 上被创建的。

现在打开文件 ip\_ttl\_dest.pcap。在这个文件中，数据在目的主机 192.168.0.128 处被捕获。展开这个捕获中第一个数据包的 IP 头，来检查它的 TTL 值（如图 6-12 所示）。

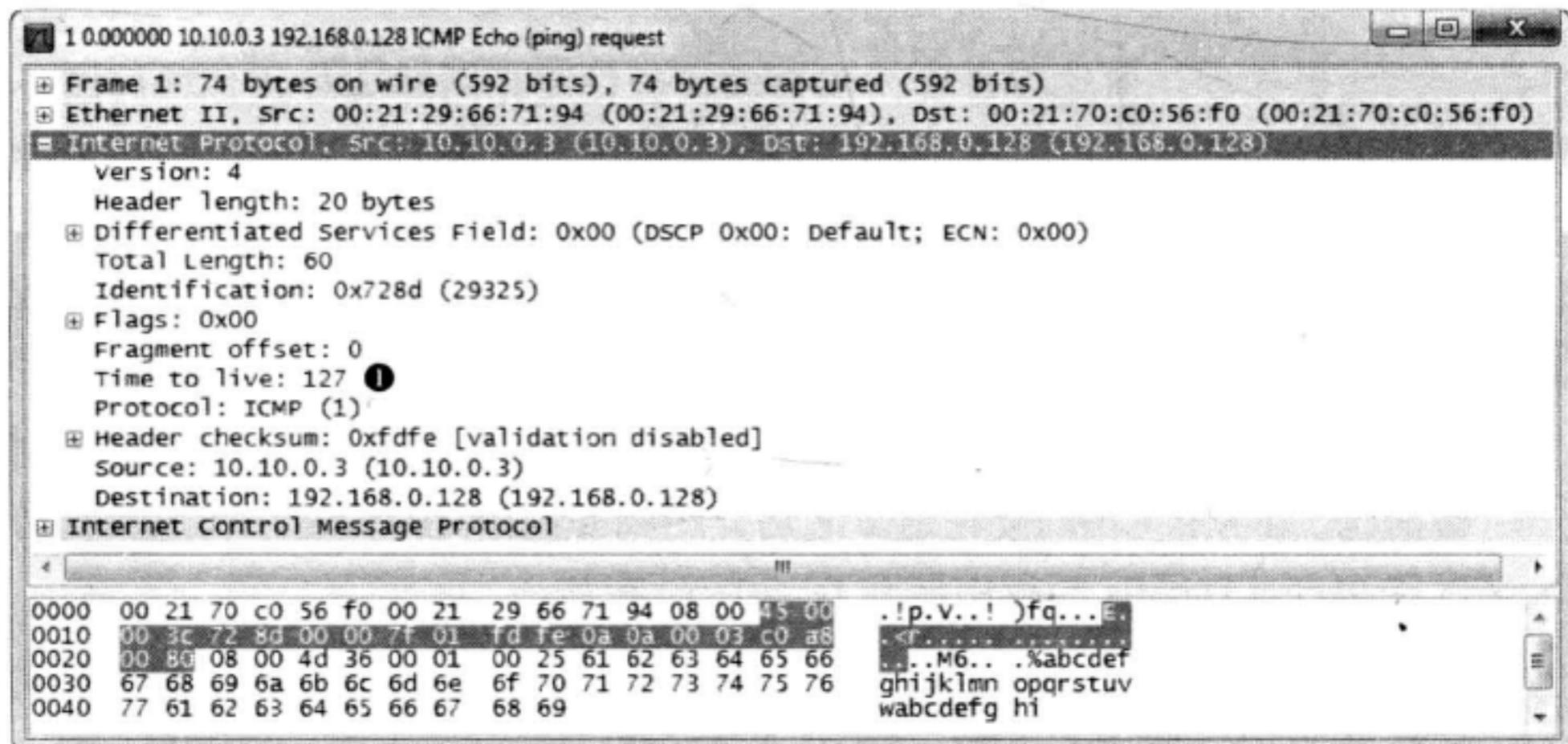


图 6-12 IP 头告诉我们 TTL 已经被减 1 了

你可以立刻注意到 TTL 的值变为 127 了，比原先的 TTL 减少了 1。即使不知道网络的结构，我们也可以知道这两台设备是由一台路由器隔开，并且经过这台路由器的路径会将 TTL 值减 1。

## 6.2.4 IP 分片

数据包分片是将一个数据流分为更小的片段，是 IP 用于解决跨越不同类型网络时可靠传输的一个特性。

一个数据包的分片主要基于第 2 层数据链路协议所使用的最大传输单元 (Maximum Transmission Unit, MTU) 的大小，以及使用这些第 2 层协议的设备配置情况。在多数情况下，第 2 层所使用的数据链路协议是以太网。以太网的默认 MTU 是 1500，也就是说，以太网的网络上所能传输的最大数据包大小是 1500 字节（并不包括 14 字节的以太网头本身）。

注

尽管存在着标准的 MTU 设定，但是一个设备的 MTU 通常可以手工设定。MTU 是基于接口进行设定，其可以在 Windows 或者 Linux 系统上修改，或者在管理路由器的界面上修改。

当一个设备准备传输一个 IP 数据包时，它将会比较这个数据包的大小，以及将要把这个数据包传送给出去的网络接口 MTU，用于决定是否需要将这个数据包分片。如果数据包的大小大于 MTU，那么这个数据包就会被分片。将一个数据包分片包括下列的步骤。

1. 设备将数据分为若干个可成功进行传输的数据包。
2. 每个 IP 头的总长度域会被设置为每个分片的片段长度。
3. 更多分片标志将会在数据流的所有数据包中设置为 1，除了最后一个数据包。
4. IP 头中分片部分的分片偏移将会被设置。
5. 数据包被发送出去。

文件 ip\_frag\_source.pcap 从地址为 10.10.0.3 的计算机上捕获而来。它向一个地址为 192.168.0.128 的设备发送 ping 请求。注意在 ICMP (ping) 请求之后，Packet List 面板的 Info 列中列出了两个被分段的 IP 数据包。

先检查数据包 1 的 IP 头（如图 6-13 所示）。

根据更多分片和分片偏移域，你可以断定这个数据包是分片数据包的一部分。被

分片的数据包要么有一个大于 0 的分片偏移，要么设定了更多分片的标志位。在第一个数据包中，更多分片标志位被设定❶，意味着接收设备应该等待接收序列中的另一个数据包。分片偏移被设为 0❷，意味着这个数据包是这一系列分片中的第一个。

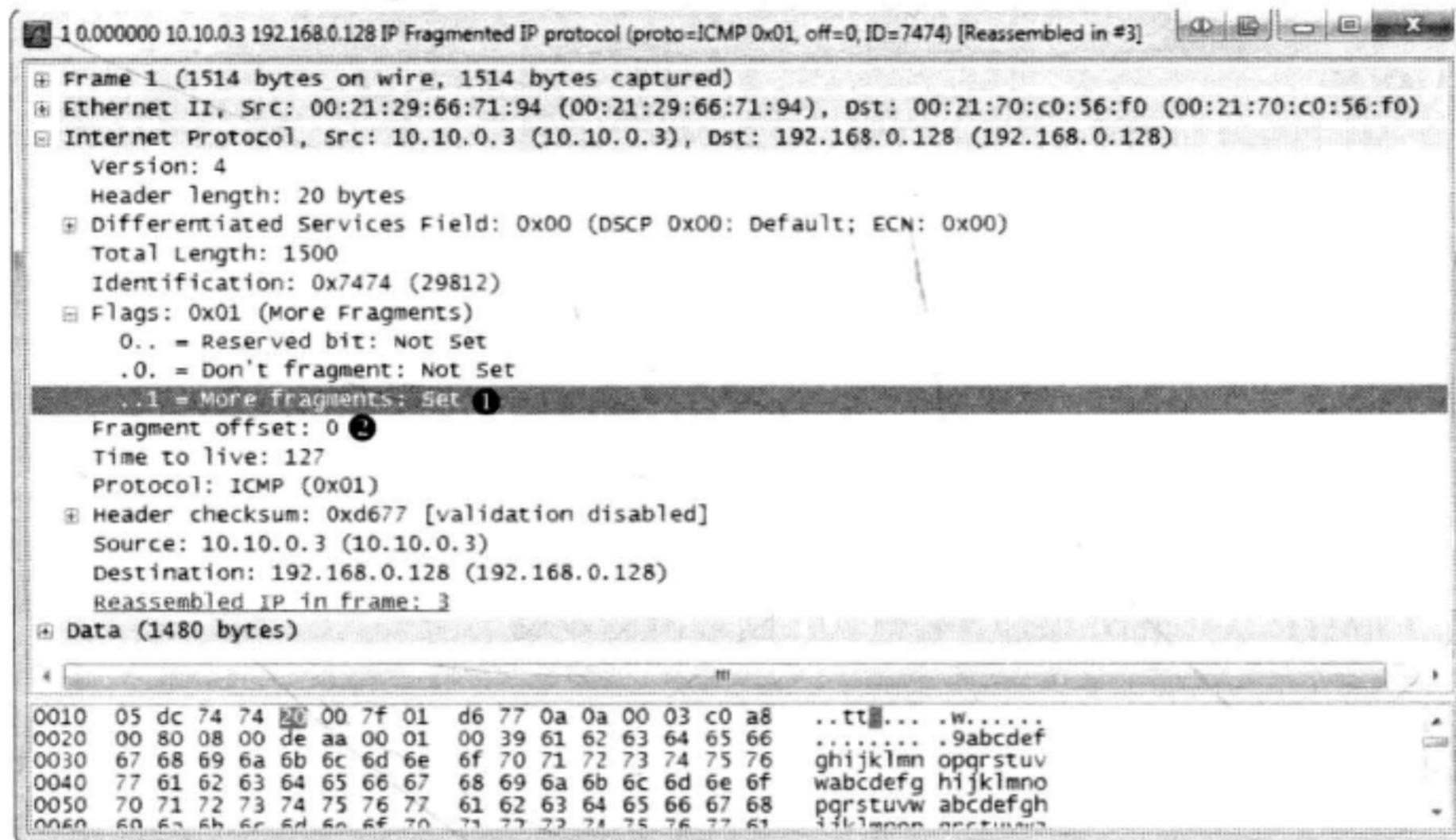


图 6-13 更多分片和分片偏移值可以用来识别分片数据包

第二个数据包的 IP 头（如图 6-14 所示），同样被设定了更多分片的标志位，但在这里分片偏移的值是 1480。这里明显意味着 1500 字节的 MTU，减去 IP 头的 20 字节。

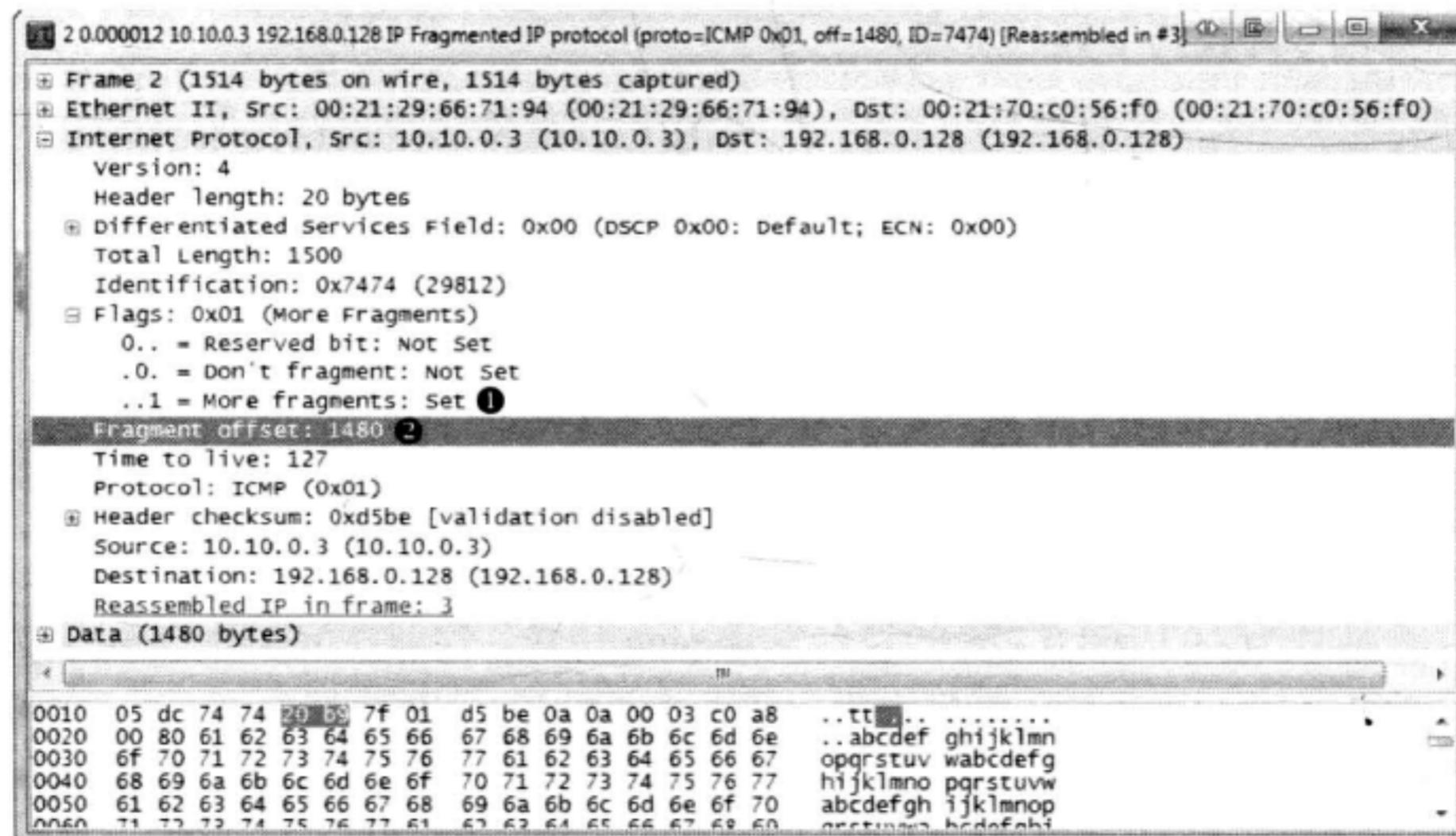


图 6-14 分片偏移值会根据数据包的大小而增大

第三个数据包（如图 6-15 所示），并没有设定更多分片标志位①，也就标志着整个数据流中的最后一个分片。并且其分片偏移被设定为 2960②，也就是  $1480 + (1500 - 20)$  的结果。这些分片可以被认为是同一个数据序列的一部分，是因为它们 IP 头中的标志位域拥有相同的值。

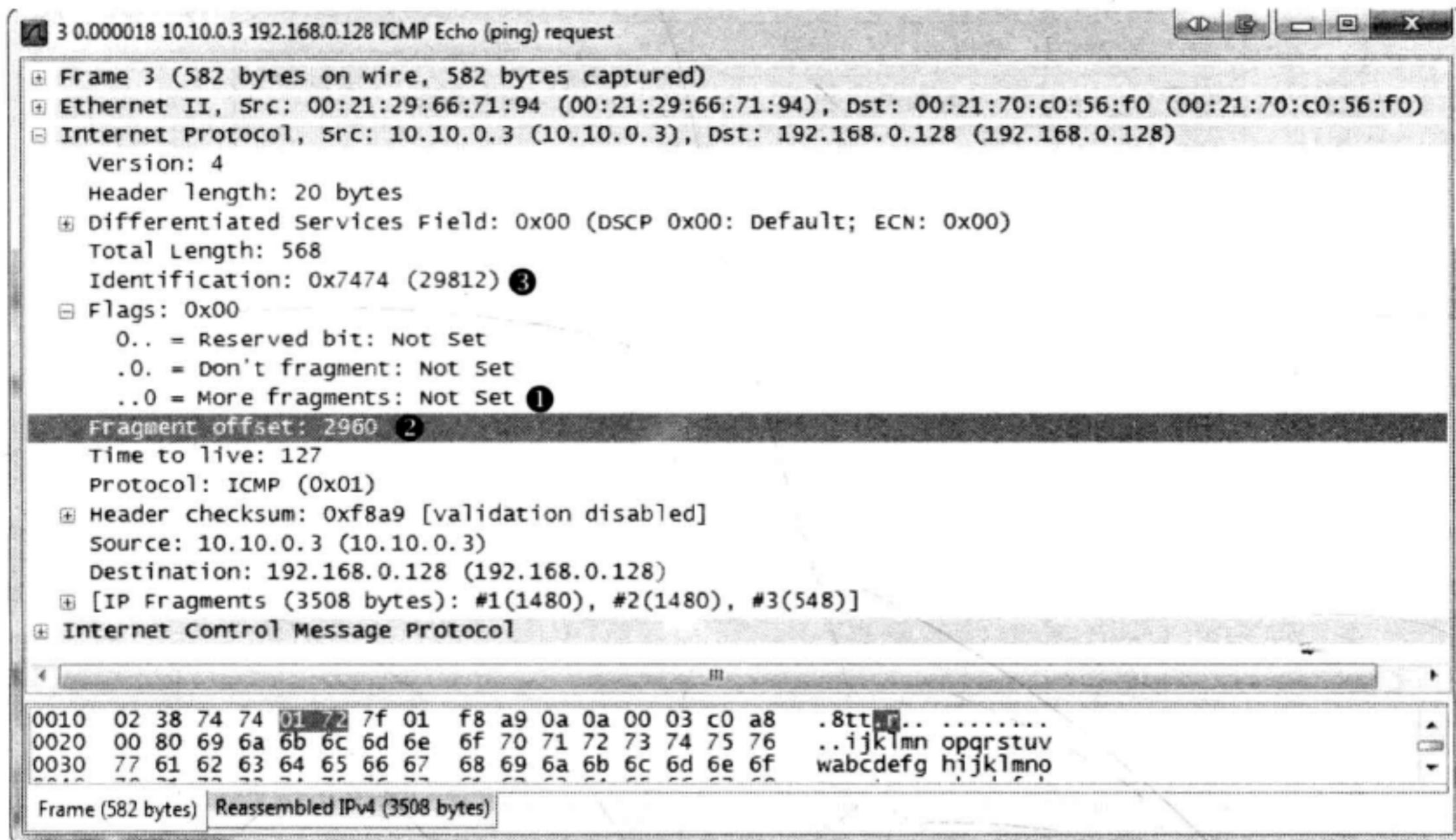


图 6-15 没有设置更多分片标志位意味着这是最后一个分片

## 6.3 传输控制协议

传输控制协议（Transmission Control Protocol, TCP）的最终目的是为数据提供可靠的端到端传输。TCP 在 RFC793 中定义，在 OSI 模型中的第 4 层工作。它能够处理数据的顺序和错误恢复，并且最终保证数据能够到达其应到达的地方。很多普遍使用的应用层协议都依赖于 TCP 和 IP 将数据包传输到其最终目的地。

### 6.3.1 TCP 头

TCP 提供了许多功能，并且反映在了其头部的复杂性上面。如图 6-16 所示，以下是 TCP 头的域。

| 传输控制协议 |      |     |      |       |      |
|--------|------|-----|------|-------|------|
| 偏移位    | 0~3  | 4~7 | 8~15 | 16~31 |      |
| 0      | 源端口  |     |      |       | 目的端口 |
| 32     | 序号   |     |      |       |      |
| 64     | 确认号  |     |      |       |      |
| 96     | 数据偏移 | 保留  | 标记   | 窗口大小  |      |
| 128    | 校验和  |     |      | 紧急指针  |      |
| 160    | 选项   |     |      |       |      |

图 6-16 TCP 头

**源端口 (Source Port)**: 用来传输数据包的端口。

**目的端口 (Destination Port)**: 数据包将要被发送到的端口。

**序号 (Sequence Number)**: 这个数字用来表示一个 TCP 片段。这个域用来保证数据流中的部分没有缺失。

**确认号 (Acknowledgment Number)**: 这个数字是通信中希望从另一个设备得到的下一个数据包的序号。

**校验和 (Checksum)**: 用来保证 TCP 头和数据的内容在抵达目的地时的完整性。

**窗口大小 (Window Size)**: TCP 接收者缓冲的字节大小。

**紧急指针 (Urgent Pointer)**: 如果设置了 URG 位, 这个域将被检查作为额外的指令, 告诉 CPU 从数据包的哪里开始读取数据。

**选项 (Options)**: 各种可选的域, 可以在 TCP 数据包中进行指定。

### 6.3.2 TCP 端口

所有 TCP 通信都会使用源端口和目的端口, 而这些可以在每个 TCP 头中找到。端口就像是老式电话总机上的插口。一个总机操作员会监视着一个面板上的指示灯和插头, 当指示灯亮起的时候, 他就会连接这个呼叫者, 问她想要和谁通话, 然后插一根电缆将她和她的目的位置连接起来。每次呼叫都需要有一个源端口 (呼叫者) 和一个目的端口 (接收者)。TCP 端口大概

就是这样工作的。

为了能够将数据传输到远程服务器或设备的特定应用中去，TCP 数据包必须知道远程服务所监听的端口。如果你想要试着连接一个不同于所设置的端口，那么这个通信就会失败。

这个序列中的源端口并不十分重要，所以可以随机选择。远程服务器也可以很简单地从发送过来的原始数据包中得到这个端口（如图 6-17 所示）。

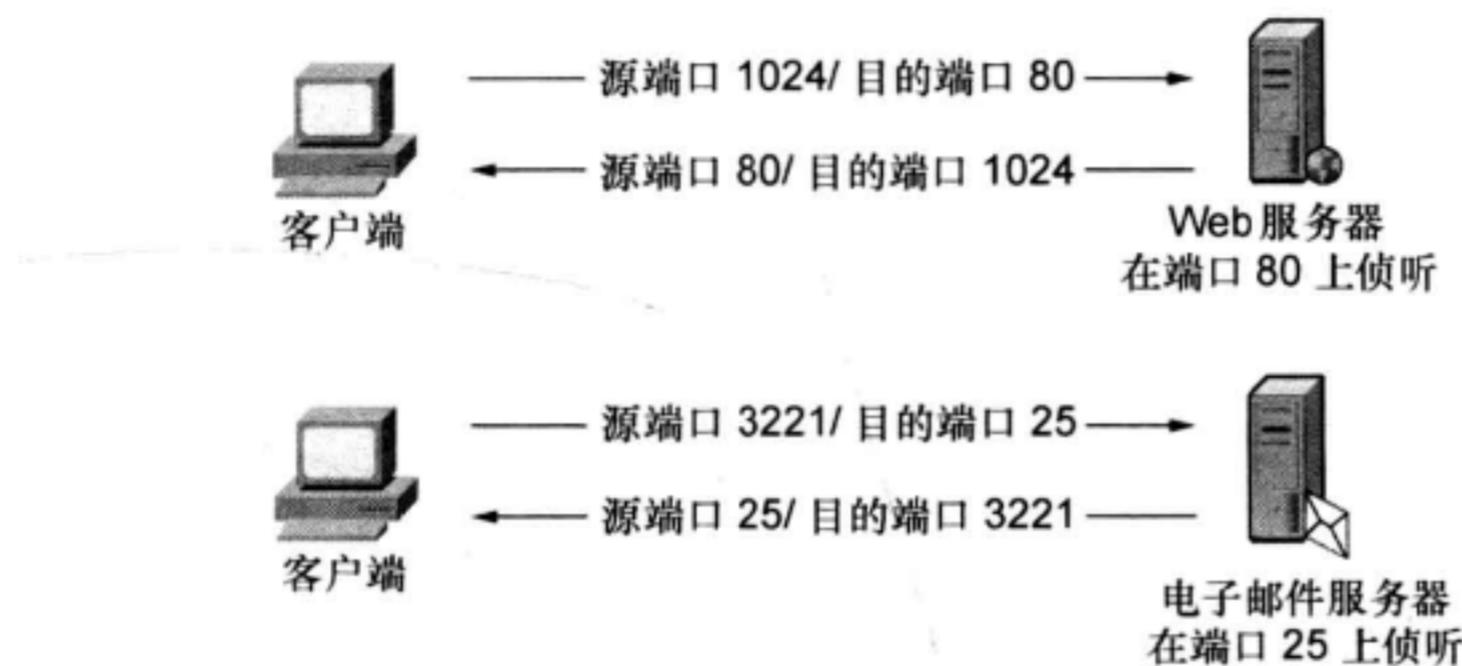


图 6-17 TCP 使用端口传输数据

在使用 TCP 进行通信的时候，我们有 65535 个端口可供使用，并通常将这些端口分成两个部分。

- 1~1023 是标准端口组（忽略掉被预留的 0），特定服务会用到这些通常位于标准端口分组中的标准端口。
- 1024~65535 是临时端口组（尽管一些操作系统对此有着不同的定义），当一个服务想在任意时间使用端口进行通信的时候，现代操作系统都会随机地选择一个源端口，让这个通信使用唯一源端口。这些源端口通常就位于临时端口组。

让我们打开文件 `tcp_ports.pcap`，看一些不同的 TCP 数据包，并识别出它们所使用的端口号。在这个文件中，我们会看到一个客户端在浏览两个网站时产生的 HTTP 通信。正如前面所提到的 HTTP 使用 TCP 进行通信，所以这将是一个非常典型的 TCP 流量案例。

在这个文件中的第一个数据包中（如图 6-18 所示），一开始的两个值代表着这个数据包的源端口和目的端口。这个数据包从 172.16.16.128 发往 212.58.226.142，它的源端口是属于临时端口组的 2826❶（需要记住的是源端口是由操作系统

随机选取的，尽管它们可能在随机选择的过程中会选择递增策略）。目的端口是一个标准端口，80 端口。这个标准端口正是提供给使用 HTTP 的 Web 服务器的。

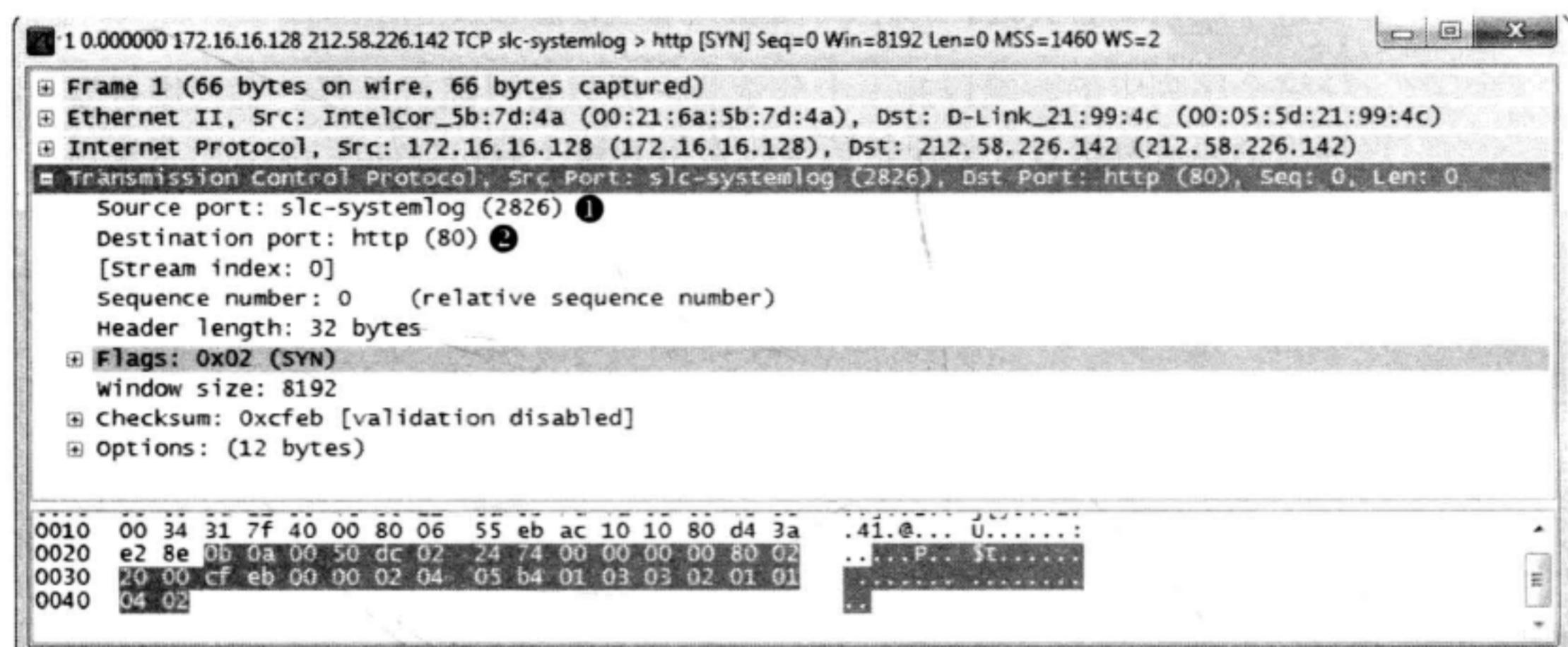


图 6-18 在 TCP 头中可以找到源端口和目标端口

你可能会注意到 Wireshark 将这些端口打上了 slc-systemlog (2826) 和 http (80) 的标签。Wireshark 会维护一个端口的列表，并记录它们最普遍的应用。尽管列表还是以标准端口为主，但很多临时端口也关联着常用的服务。这些端口的标签可能会让人迷惑，所以一般来说最好通过关闭传输名字解析来禁用它。选择 Edit -> Preference -> Name Resolution，然后取消勾选 Enable Transport Name Resolution 就可以将其禁用了。如果你希望保留这个功能但希望改变 Wireshark 对每一个端口的识别，你可以通过改变 Wireshark 程序目录下的 Services 文件。这个文件是根据互联网数字分配机构（Internet Assigned Numbers Authority, IANA）的通用端口列表编写的。

第二个数据包是由 212.58.226.142 发往 172.16.16.128 的（如图 6-19 所示）。除了 IP 地址之外，源端口和目的端口也同样有所改变。

所有基于 TCP 的通信都以相同的方式工作：选择一个随机的源端口与一个已知的目的端口进行通信。在发出初始数据包之后，远程设备就会与源设备使用建立起的端口进行通信。

在这个捕获文件中还有另外一个通信流，你可以试着找出它通信时使用的端口。

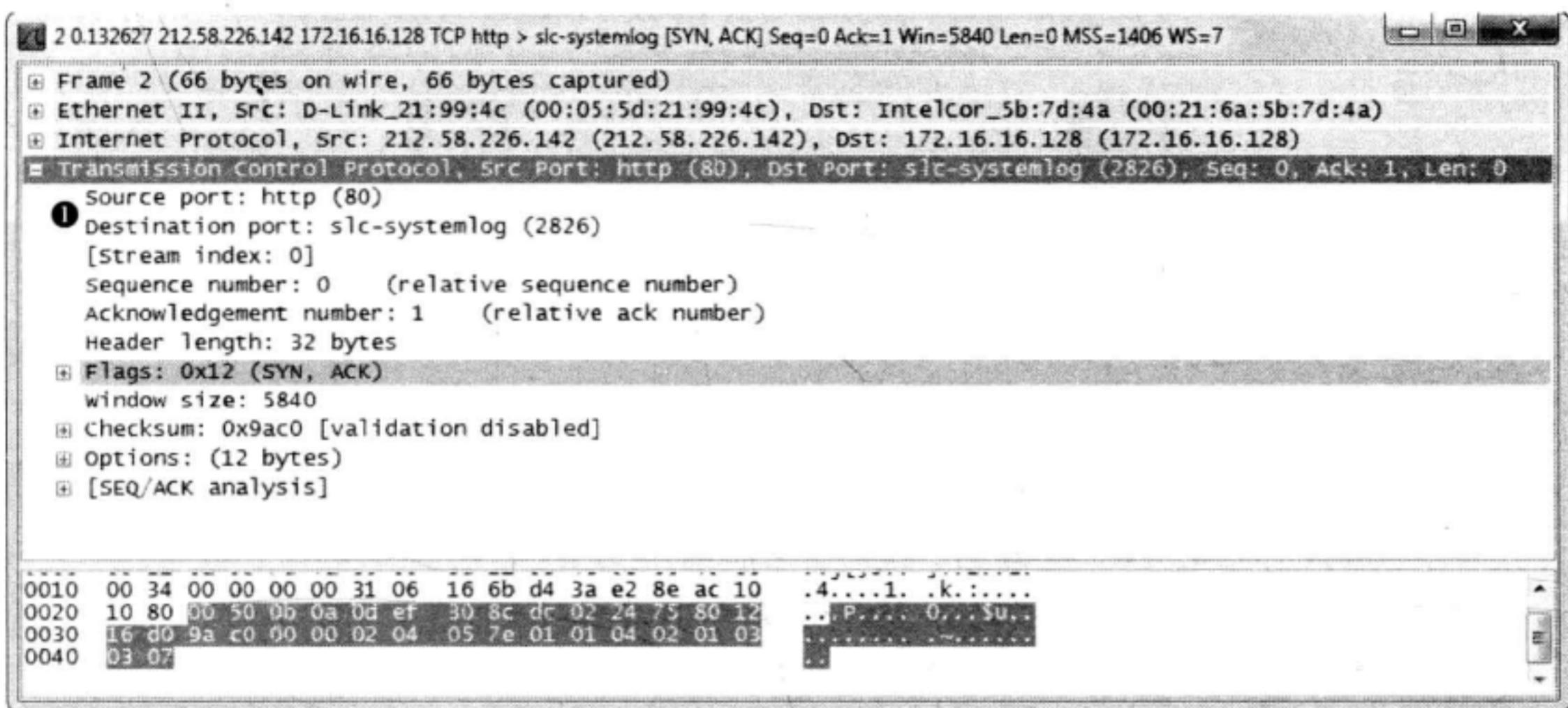


图 6-19 在反向通信中，源端口和目的端口进行了互换

注 随着这本书的深入，你将会知道更多与通用协议和端口相关联的端口，并且最终可以通过端口来识别使用它们的服务和设备。如果希望查阅详细的通用端口列表，可以访问 <http://www.iana.org/assignments/port-numbers/>。

### 6.3.3 TCP 的三次握手

所有基于 TCP 的通信都需要以两台主机的握手开始。这个握手过程主要希望能达到以下不同的目的。

- 保证源主机确定目的主机在线，并且可以进行通信。
- 让源主机检查它是否正在监听试图去连接的端口。
- 允许源主机向接收者发送它的起始序列号，使得两台主机可以将数据包流保持有序。

TCP 握手分为 3 个步骤，如图 6-20 所示。在第一步中，主动发起通信的设备（主机 A）向目标（主机 B）发送了一个 TCP 数据包。这个初始数据包除了底层协议头之外不包含任何数据。这个数据包的 TCP 头设置了 SYN 标志，并包含了在通信过程中会用到的初始序列号和最大分段大小（MSS）。主机 B 对于这个数据包回复了一个类似于设置了 SYN 和 ACK 标志以及包含了它初始序列号的数据包。最后，主机 A 向主机 B 发送最后一个仅设置了 ACK 标志的数据包。在这个过程完成之后，双方设备应该已经具有了开始正常通信所需的信息。

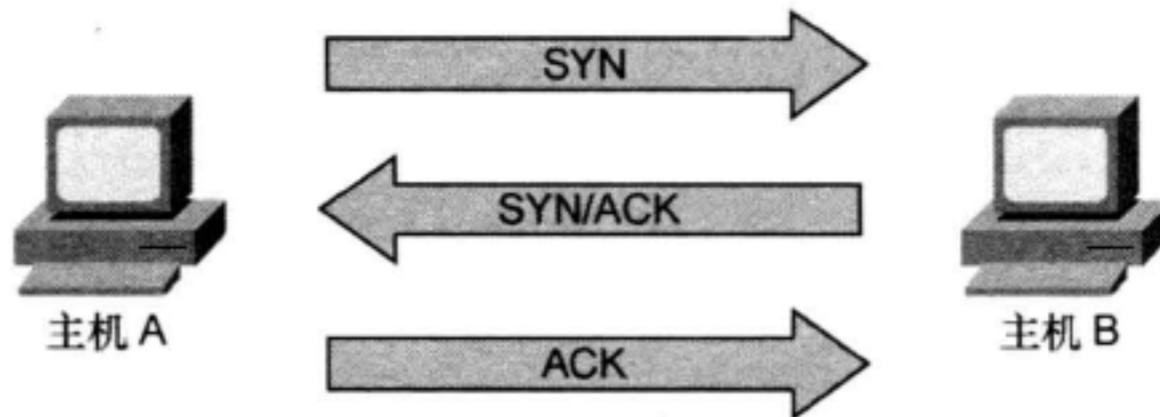


图 6-20 TCP 三次握手

注

TCP 数据包在称呼上通常会被其所设置的标志所代表。比如，对于设置了 SYN 标志的 TCP 数据包，我们将会简称其为 SYN 包。所以 TCP 握手过程中使用的数据包会被称为 SYN 包、SYN/ACK 包和 ACK 包。

打开 `tcp_handshake.pcap`，可以更实际地看到这个过程。Wireshark 为了分析将简便，引入了一个特性，可以将 TCP 数据包的序列号替换为相对值。但在这里，我们将这个功能关闭，以便于能看到实际的序列号值。选择 **Edit -> Preferences**，展开 **Protocols** 并选择 **TCP**，然后取消勾选 **Relative Sequence Numbers and Window Scaling** 框，并单击 **OK** 就可以禁用了。

这个捕获中的第一个数据包是我们的初始 **SYN** 数据包（如图 6-21 所示）。这个数据包从 172.16.16.128 的 2826 端口发往 212.58.226.142 的 80 端口。我们可以看到这里传输的序列号是 3691127924❶。

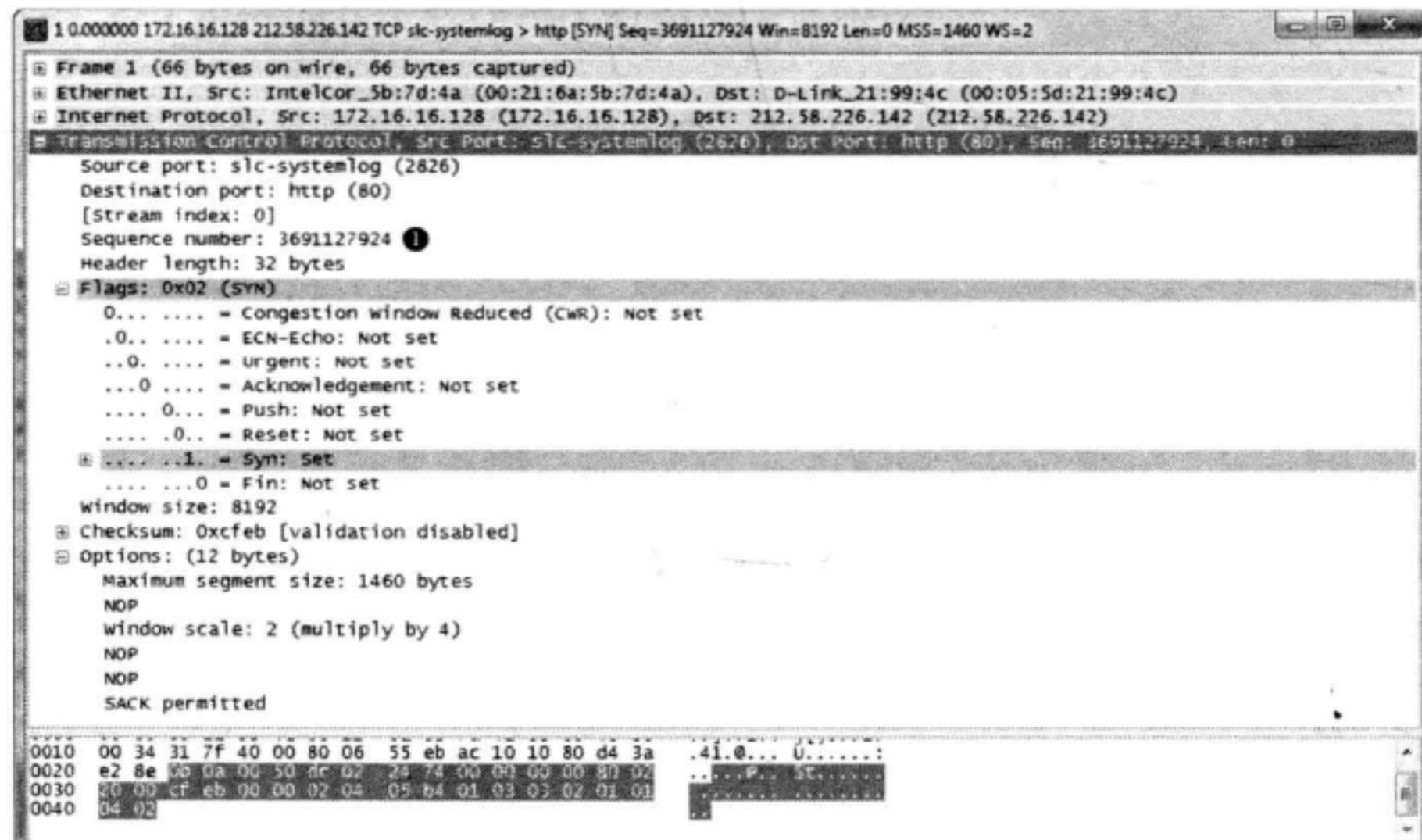


图 6-21 初始 SYN 数据包

握手中第二个数据包是从 212.58.226.142 发出的 SYN/ACK 响应（如图 6-22 所示）。这个数据包也包含着这台主机的初始序列号（233779340）①，以及一个确认号（2691127925）②。这个确认号比之前的那个数据包序列号大 1，是因为这个域是用来表示主机所期望得到的下一个序列号的值。

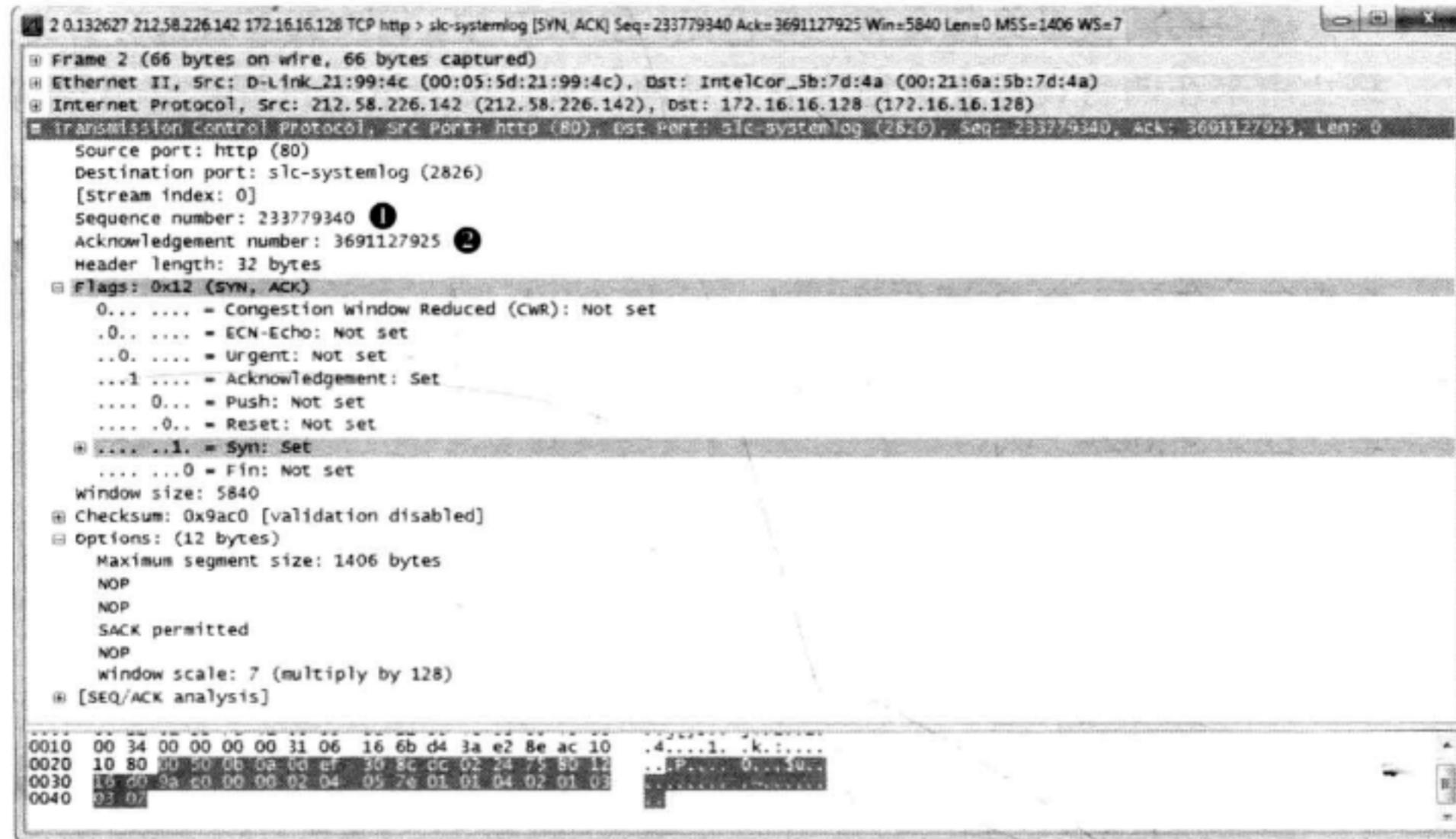


图 6-22 SYN/ACK 响应

最后的数据包是从 172.16.16.128（如图 6-23 所示）发出的 ACK 数据包。这个数据包正如所期望的那样，包含着之前数据包确认号域所定义的序列号 3691127925 ①。



图 6-23 最后的 ACK 包

握手伴随着每次 TCP 的通信序列。当在一个繁忙的捕获文件中搜索通信序列的开始时，SYN、SYN/ACK、ACK 的序列是一个很好的标志。

### 6.3.4 TCP 终止

所有的问候最终都会有一句再见，在 TCP 中，每次握手后也会有终止。TCP 终止用来在两台设备完成通信后正常地结束连接。这个过程包含 4 个数据包，并且用一个 FIN 标志来表明连接的终结。

在一个终止序列中，主机 A 通过发送一个设置了 FIN 和 ACK 标志的 TCP 数据包，告诉主机 B 通信的完成。主机 B 以一个 ACK 数据包响应，并传输自己的 FIN/ACK 数据包。主机 A 响应一个 ACK 数据包，然后结束通信过程。这个过程如图 6-24 所示。

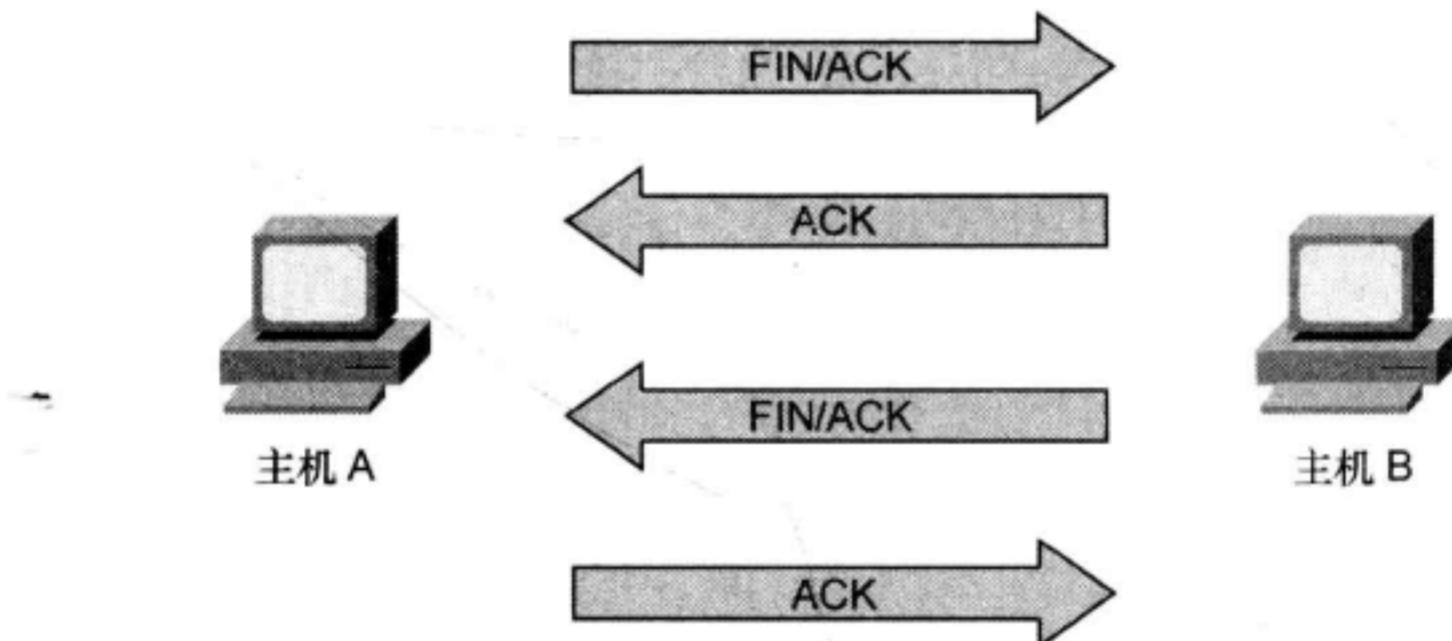


图 6-24 TCP 终止过程

打开文件 `tcp_teardown.pcap` 可以在 Wireshark 中看到这个过程。在序列的第一个数据包（如图 6-25 所示），你可以看到位于 67.228.110.120 的设备通过发送有着 FIN 和 ACK 标志的数据包①来开始终止过程。

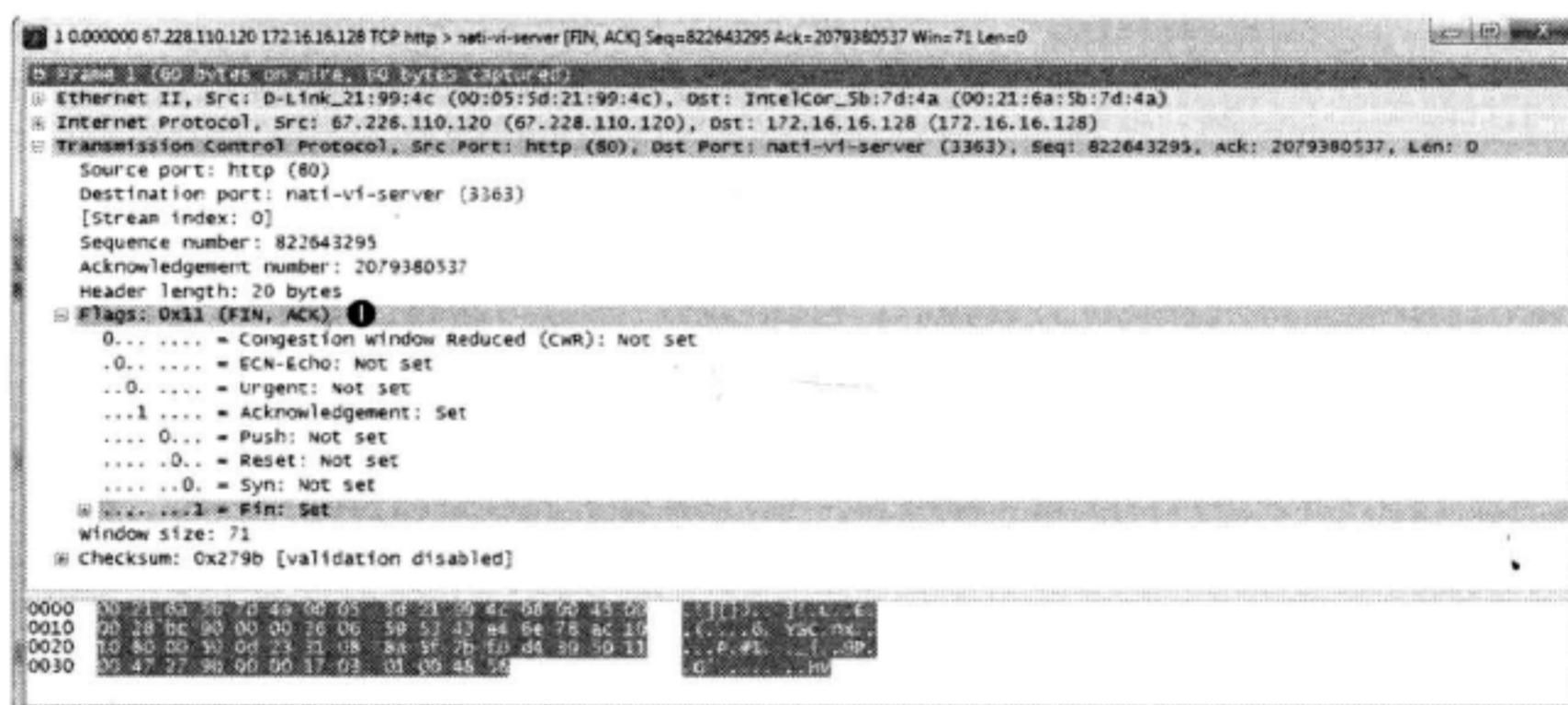


图 6-25 FIN/ACK 包作为终止过程的开始

在这个数据包被发出去之后，172.16.16.128 使用了一个 ACK 数据包进行响应来确认第一个数据包的接收，然后发送了一个 FIN/ACK 数据包。整个过程在 67.228.110.120 发送了最终的 ACK 之后结束。这时，这两个设备的通信便已经结束，如果想要再次开始通信就必须完成新的 TCP 握手。

### 6.3.5 TCP 重置

在理想情况下，每一个连接都会以 TCP 终止来正常地结束。但在现实中，连接经常会突然断掉。举例来说，这可能由于一个潜在的攻击者正在进行端口扫描，或者仅仅是主机配置错误。在这些情况下，就需要使用设置了 RST 标志的 TCP 数据包。RST 标志用来指出连接被异常中止或拒绝连接请求。

文件 `tcp_refuseconnection.pcap` 给出了一个包含有 RST 数据包网络流量的例子。这个文件中的第一个数据包发自 192.168.100.138，其尝试与 192.168.100.1 的 80 端口进行通信。这个主机并不知道 192.168.100.1 并没有在监听 80 端口，因为那是一个思科路由器，并且没有配置 Web 接口，也就是说，并没有服务监听 80 端口的连接。为了响应这个连接请求，192.168.100.1 向 192.168.100.138 发送了一个数据包，告诉它其对 80 端口的通信无效。图 6-26 中展示了在第二个数据包的 TCP 头中这个连接尝试突然终止的情况。RST 数据包除了包含 RST 和 ACK 标志❶外，没有任何其他的东西，之后也并没有额外的通信。

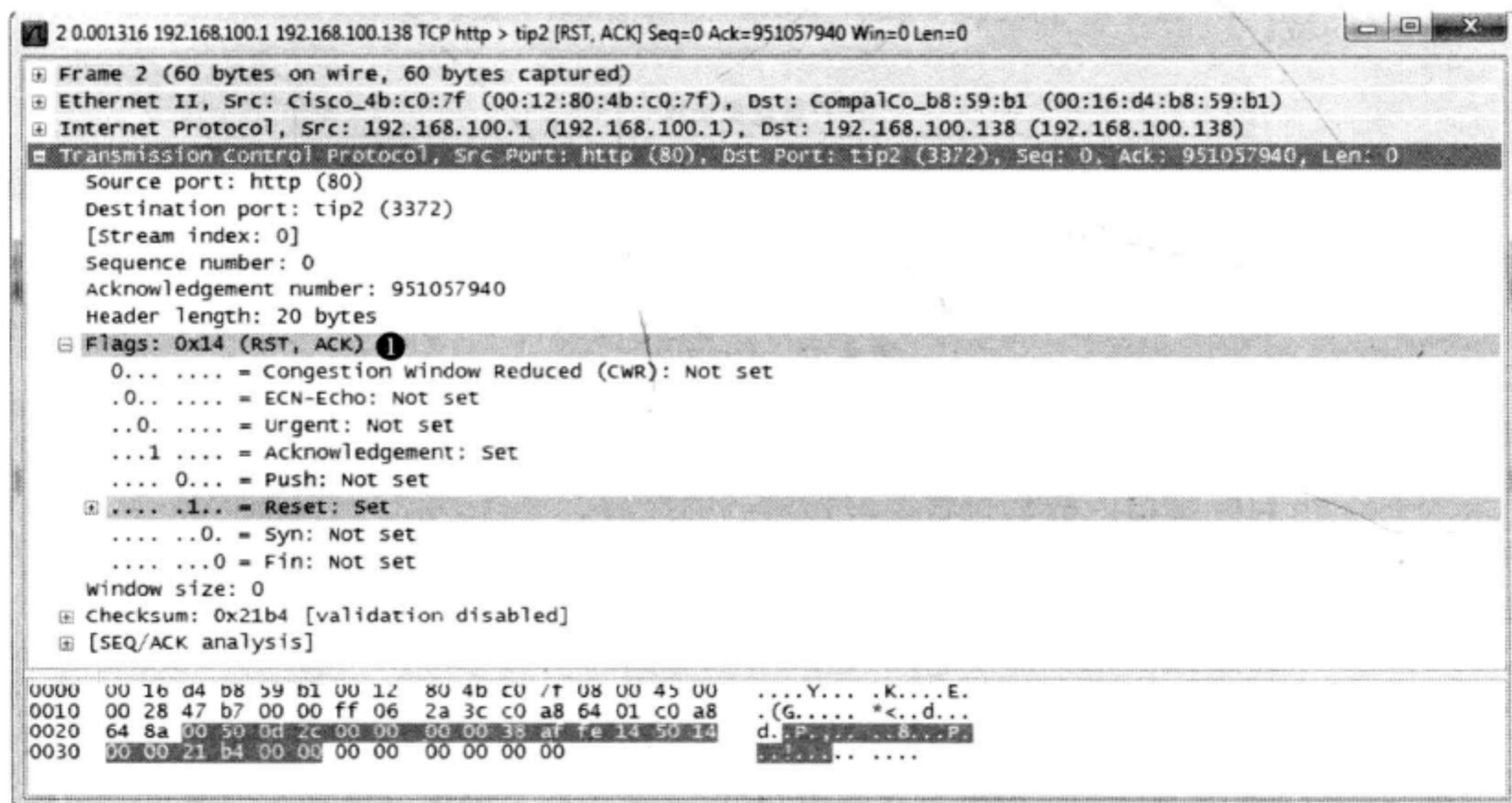


图 6-26 RST 和 ACK 标志代表着通信的结束

如本例所示, RST 数据包可以在通信序列的开始或者在主机通信的过程中, 终止通信。

## 6.4 用户数据报协议

用户数据报协议 (User Datagram Protocol, UDP) 是在现代网络中最常使用的另外一种第 4 层协议。如果说 TCP 是为了满足带有内在错误检测的可靠数据传输, 那么 UDP 主要是为了提供高速的传输。出于这个原因, UDP 是一种尽力服务, 通常会被称为无连接协议。一个无连接协议并不会正式地建立和结束主机之间的连接, 也不会像 TCP 那样存在握手和终止过程。

无连接协议意味着不可靠服务, 这将使得 UDP 流量一点都不稳定。事实上正是如此, 但依赖于 UDP 的协议通常都会有其内置的可靠性服务, 或者使用 ICMP 的一些功能来保证连接更可靠一些。举例来说, 应用层协议 DNS 和 DHCP 需要高度依赖数据包在网络上传输的速度, 其使用 UDP 作为它们的传输层协议, 但是它们自己进行错误检查以及重传计时。

### 6.4.1 UDP 头

UDP 头比 TCP 头要小得多, 也简单得多。如图 6-27 所示, 以下是 UDP 头的域。

| 用户数据报协议 |       |       |
|---------|-------|-------|
| 偏移位     | 0~15  | 16~31 |
| 0       | 源端口   | 目的端口  |
| 32      | 数据包长度 | 校验和   |

图 6-27 UDP 头

**源端口:** 用来传输数据包的端口。

**目标端口:** 数据包将要被传输到的端口。

**数据包长度:** 数据包的字节长度。

**校验和:** 用来确保 UDP 头和数据到达时的完整性。

文件 `udp_dnsrequest.pcap` 中包含有一个数据包, 这个数据包是一个使用 UDP 的 DNS 请求。当你展开这个数据包的 UDP 头时, 你可以看到 4 个域 (如图 6-28 所示)。

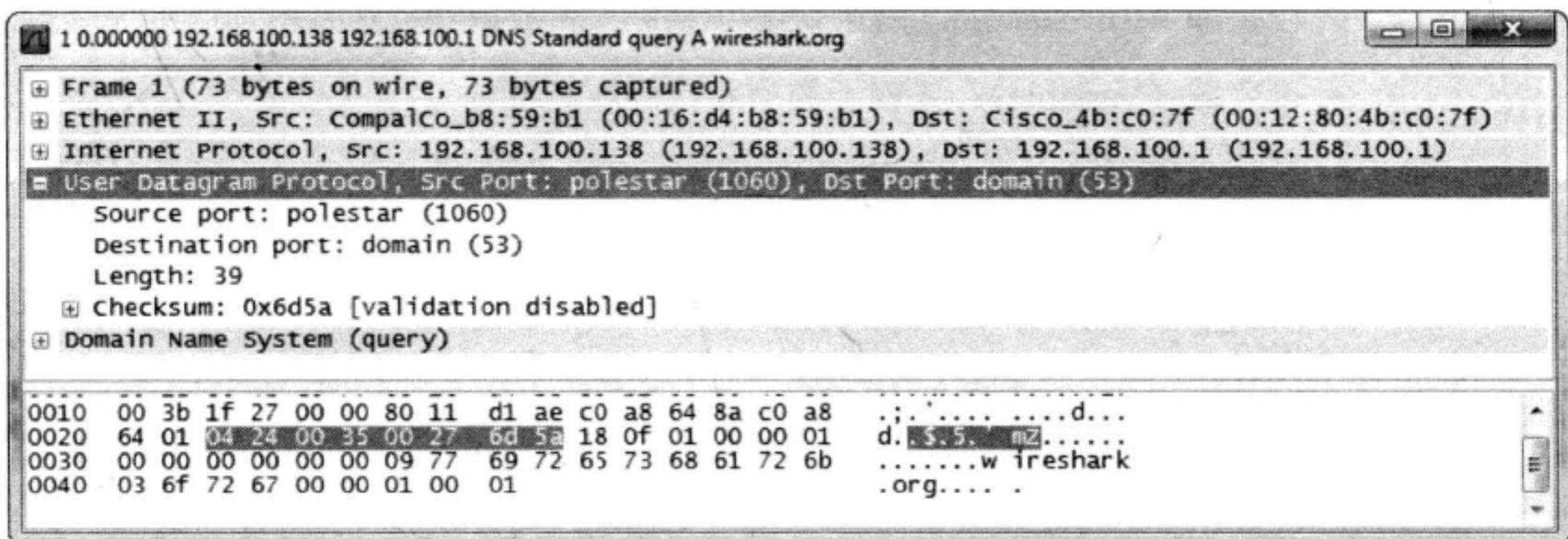


图 6-28 UDP 数据包的内容非常简单

需要记住的是，UDP 并不关心传输的可靠性，所以任何使用 UDP 的应用在必要的时候都需要采取特殊的步骤，以保证可靠的传输。

## 6.5 互联网控制消息协议

互联网控制消息协议（Internet Control Message Protocol, ICMP）是 TCP/IP 协议族中的一个效用协议，负责提供在 TCP/IP 网络上设备、服务以及路由器可用性的信息。大多数网络检修技巧和工具都是基于常用的 ICMP 消息类型。ICMP 在 RFC792 中定义。

### 6.5.1 ICMP 头

ICMP 是 IP 的一部分并依赖 IP 来传递消息。ICMP 头相对较小并根据用途而改变。如图 6-29 所示，ICMP 头包含了以下几个域。

| 控制消息协议 |      |    |       |
|--------|------|----|-------|
| 偏移位    | 0~15 |    | 16~31 |
| 0      | 类型   | 代码 | 校验和   |
| 32     | 可变域  |    |       |

图 6-29 ICMP 头

**类型 (Type):** ICMP 消息基于 RFC 规范的类型或分类。

**代码 (Code):** ICMP 消息基于 RFC 规范的子类型。

校验和 (Checksum): 用来保证 ICMP 头和数据在抵达目的地时的完整性。

可变域 (Variable): 依赖于类型和代码域的部分。

## 6.5.2 ICMP 类型和消息

正如刚才所说, ICMP 数据包的结构取决于它由 *Type* 和 *Code* 域中的值所定义的用途。

你可以将 ICMP 的类型域作为数据包的分类, 而 *Code* 域作为它的子类。举例来说, *Type* 域的值为 3 时意味着“目标不可达”。但只有这个信息可能不足以发现问题, 当如果数据包在 *Code* 域中指明值为 3, 也就是“端口不可达”时, 你就知道这应该是你试图进行通信的端口的问题。

注

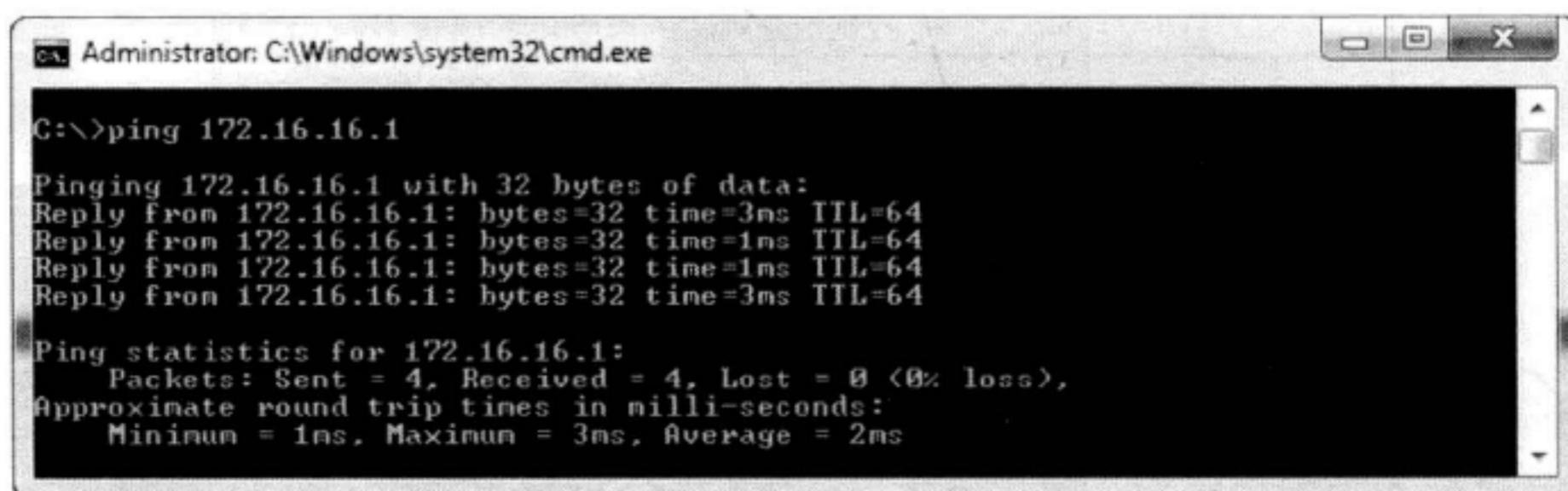
如果想要浏览所有可用的 ICMP 类型和代码, 可以访问 <http://www.iana.org/assignments/icmp-parameters>。

## 6.5.3 Echo 请求与响应

ICMP 因为其 ping 功能而声名大噪。ping 用来检测一个设备的可连接性。大多数信息技术专家都会对 ping 很熟悉。

在命令行中输入 ping <ip 地址>, 其中将<ip 地址>替换为你网络上的一个实际 IP 地址, 就可以使用 ping 了。如果目标设备在线, 你的计算机有到达目标的通路, 并且没有防火墙隔离通信的话, 你将能够看到对 ping 命令的响应。

在图 6-30 所示的例子中, 给出了 4 个成功显示了大小、RTT 和 TTL 的响应。Windows 还会提供一个总结信息, 告诉你有多少数据包被发送、接收或者丢失。如果通信失败, 你会看到一条告诉你原因的信息。



```
C:\>ping 172.16.16.1

Pinging 172.16.16.1 with 32 bytes of data:
Reply from 172.16.16.1: bytes=32 time=3ms TTL=64
Reply from 172.16.16.1: bytes=32 time=1ms TTL=64
Reply from 172.16.16.1: bytes=32 time=1ms TTL=64
Reply from 172.16.16.1: bytes=32 time=3ms TTL=64

Ping statistics for 172.16.16.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 1ms, Maximum = 3ms, Average = 2ms
```

图 6-30 使用 ping 命令测试可连接性

基本上来说，ping 命令每次向一个设备发送一个数据包，并等待回复，以确定设备是否可连接，如图 6-31 所示。

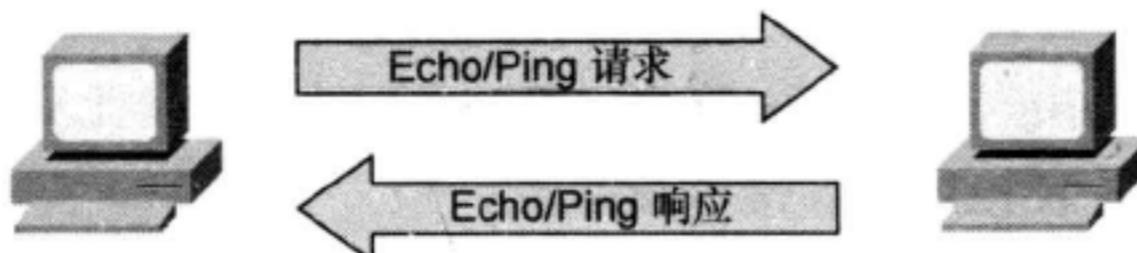


图 6-31 ping 命令只包含两步

**注** 尽管 ping 对于 IT 业必不可少，但当部署了基于主机的防火墙时，它的结果就可能具有欺骗性了。现在很多的防火墙都限制了设备去响应 ICMP 数据包。这样对于安全性是有帮助的，因为潜在的攻击者可能会使用 ping 来判断主机是否可达，从而放弃进一步的行动。但这样查找问题时也变得困难起来——当你知道你可以和一台设备通信时，使用 ping 检测连接却收不到任何响应会让你很抓狂。

ping 功能在实际中是很好的简单 ICMP 通信的例子。文件 *icmp\_echo.pcap* 中的数据包会告诉你在运行 ping 时都发生了什么。

第一个数据包（如图 6-32 所示）显示了主机 192.168.100.138 在给 192.168.100.1 发送数据包①。当你展开这个数据包的 ICMP 区段时，你可以通过查看类型和代码域判断 ICMP 数据包的类型。在这个例子中，数据包的类型是 8②，代码是 0③，意味着这是一个 echo 请求（Wireshark 会告诉你所显示的类型/代码究竟是什么意思）。这个 echo（ping）请求是整个过程的前一半。这是一个简单的 ICMP 数据

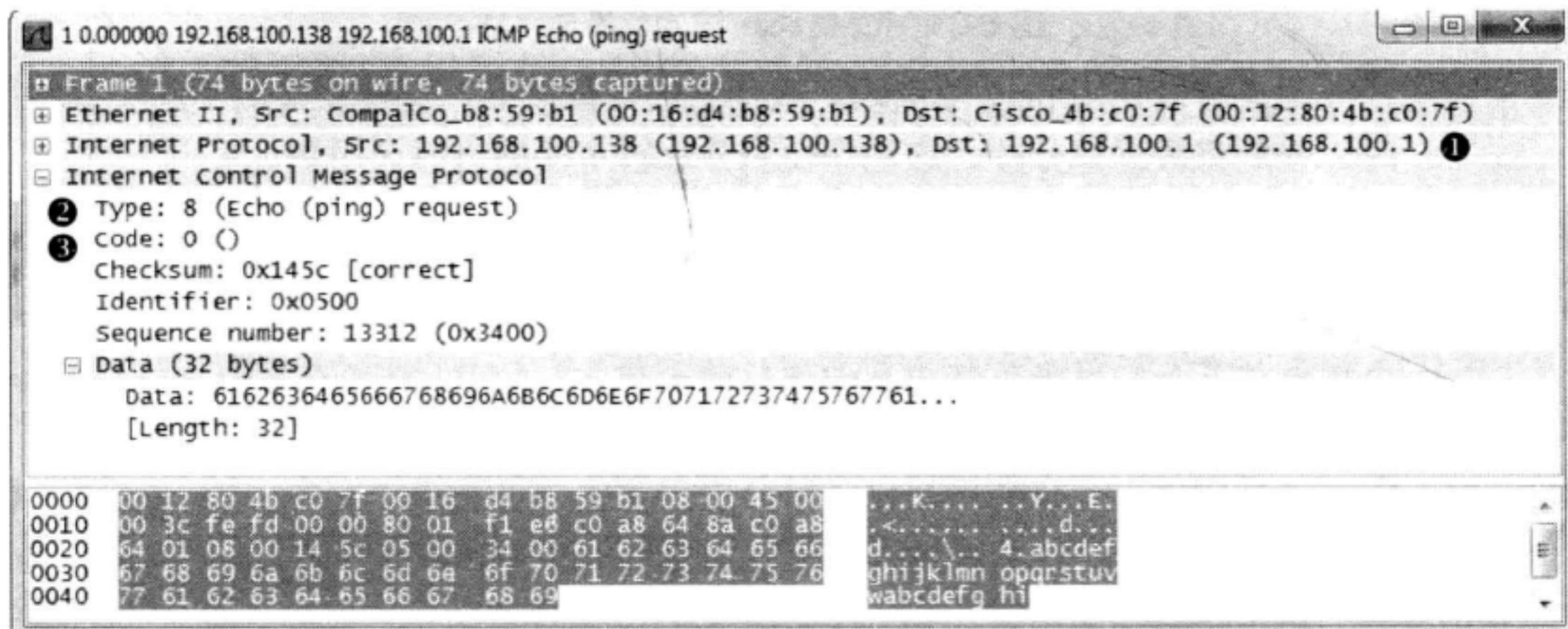


图 6-32 ICMP echo 请求数据包

包，使用 IP 发送，包含了很多的数据。除了指定的类型、代码以及校验和，我们还会有一个序列号用来匹配请求和响应，还有在 ICMP 数据包可变域中的一串随机文本字符。

**注** echo 和 ping 经常会被混淆，但记住 ping 实际上是一个工具的名字。ping 工具用于发送 ICMP echo 请求数据包。

这个序列的第二个数据包是对我们请求的响应（如图 6-33 所示）。这个数据包的 ICMP 区段类型是 ①，代码是 ②，表示这是一个 echo 响应。由于第二个数据包的序列号与第一个的相匹配③，我们就知道这个 echo 响应与之前数据包的 echo 请求匹配。这个响应数据包中有着和初始请求中传输的 32 位字符串一样的内容④。在这第二个数据包被 192.168.100.138 成功接收到之后，ping 就会报告成功（如图 6-34 所示）。

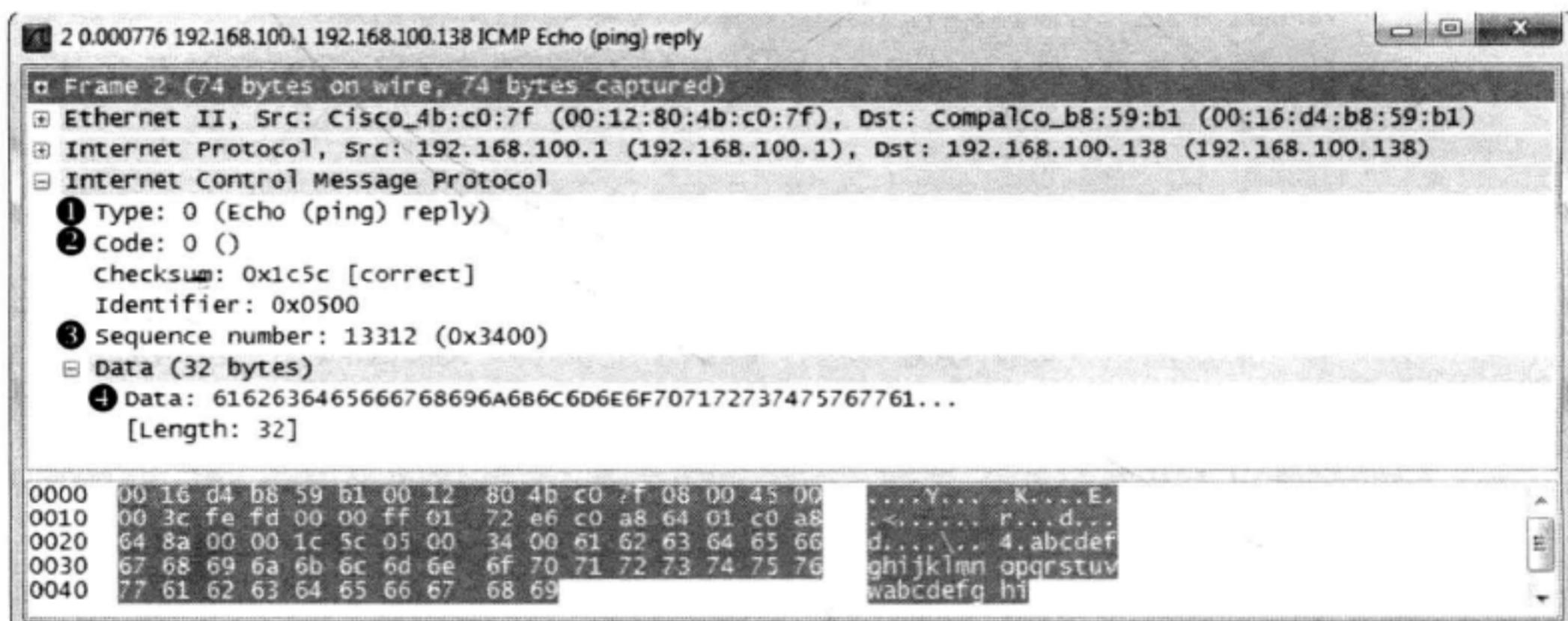


图 6-33 ICMP echo 回复数据包

你还可以使用 ping 的选项，增加它的数据填充，这样在检测不同类型网络时，强制将数据包分片。这在你检测分片较小的网络时会用到。

**注** ICMP 的 echo 请求使用的随机文本可能会引起潜在攻击者的兴趣。攻击者可能会用这段填充的内容来推测设备所使用的操作系统。并且攻击者可能在这个域中放置一些数据位作为反向连接的手段。

#### 6.5.4 路由跟踪

路由跟踪功能用来识别一个设备到另一个设备的网络路径。在一个简单的网络上，这个网络路径可能只经过一个路由器，甚至一个都不经过。但在复杂

的网络中，数据包可能会经过数十个路由器才会到达最终目的地。这就是为什么确定数据包从一个目的地到另一个的实际路径对于通信检修十分重要。

通过使用 ICMP（在 IP 的一些帮助下），路由跟踪可以画出数据包的路径。举例来说，文件 `icmp_traceroute.pcap` 中的第一个数据包，和我们在上一节中看到的 echo 请求（如图 6-34 所示）很类似。

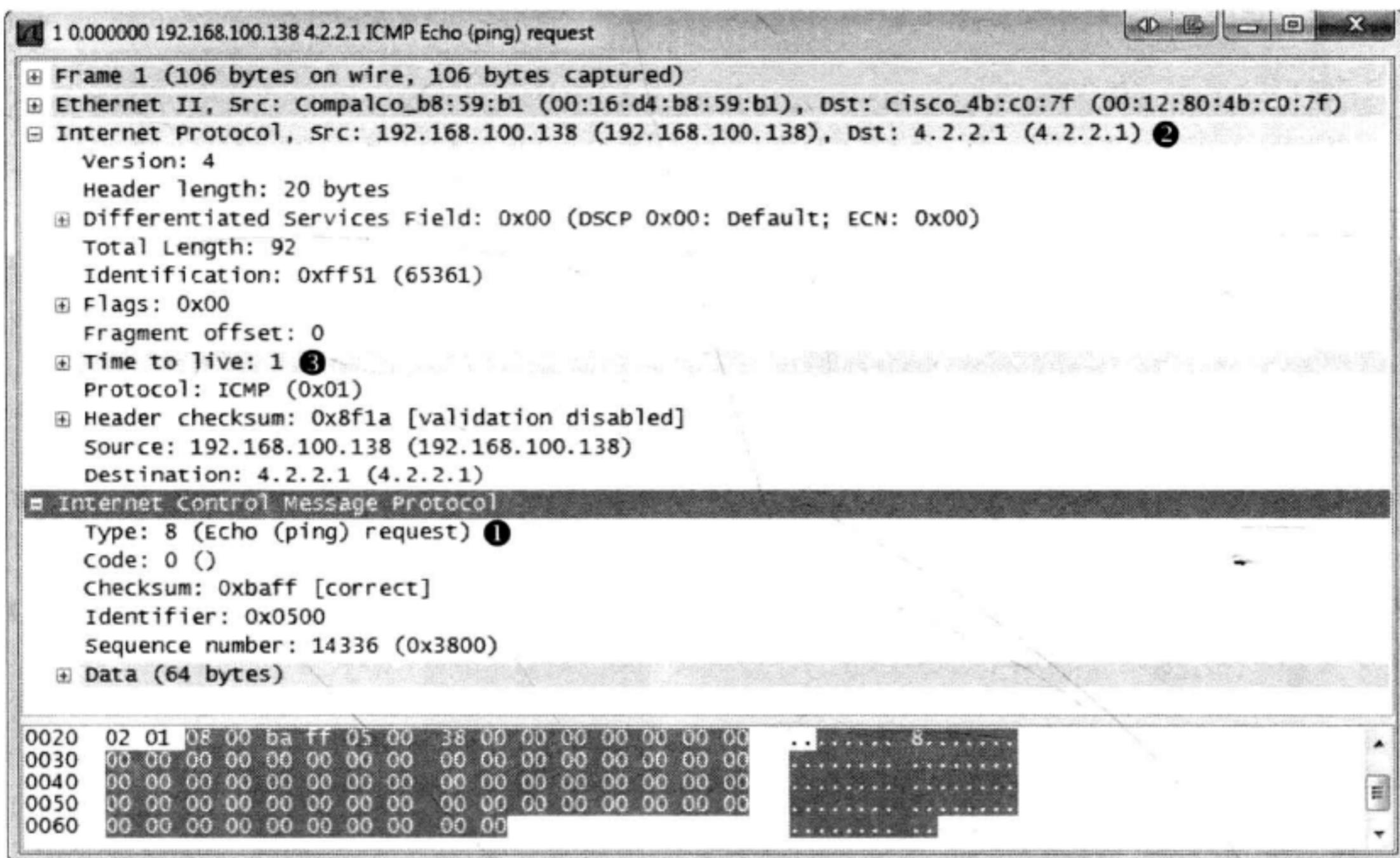


图 6-34 一个 TTL 值为 1 的 ICMP echo 请求数据包

乍看起来，这个数据包就是一个从 192.168.100.138 到 4.2.2.1 ② 的简单 echo 请求 ①，并且 ICMP 中的每一部分都与 echo 请求数据包相同。但是当你展开这个数据包的 IP 头时，你可以注意到一个奇怪的数字。这个数据包的 TTL 被设为了 1 ③，也就意味着这个数据包会在它遇到的第一个路由器处被丢掉。因为目标地址 4.2.2.1 是一个互联网地址，我们就会知道源设备和目的设备之前至少会有一个路由器，所以这个数据包不会到达目的地。这对我们来说是个好事，因为路由跟踪正是需要这个数据包只到达它传输的第一个路由器。

第二个数据包正如所期望的那样，是前往目的地路径上第一个路由器发回的响应（如图 6-35 所示）。这个数据包到达 192.168.100.1 这个设备后，它的 TTL 减为 0，所以它不能继续传输，于是路由器回复了一个 ICMP 响应。这个数据包

的类型是 11①，代码是 0②，也就是告诉我们由于数据包的 TTL 在传输过程中超时，所以目的不可达。

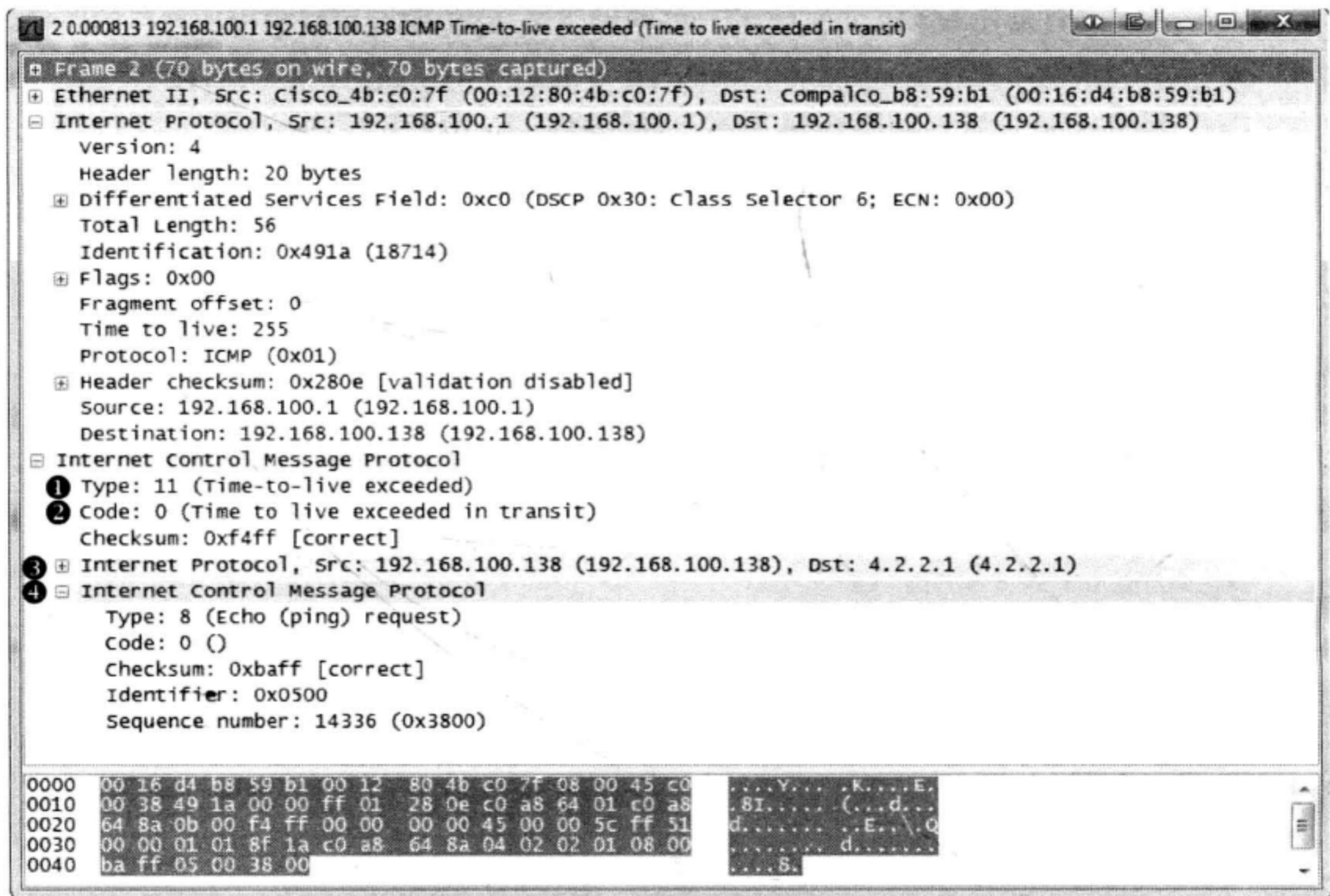


图 6-35 来自路径上第一个路由器的 ICMP 响应

这个 ICMP 数据包有时候被叫做双头包，因为这个 ICMP 的结尾部分包含了原先 echo 请求的 IP 头③和 ICMP 数据④的拷贝。这个信息被证明在检修的时候非常有用。

在第 7 个数据包前，这样发送 TTL 自增数据包的过程又出现了两次。这次的过程与你在第 1 个数据包中看到的过程不同，IP 头的 TTL 值被设为了 2，从而保证这个数据包会在被丢弃前到达第二跳路由。和我们所期望的一样，我们从下一跳的路由 12.180.241.1 接到了一个有着同样 ICMP 目的不可达和 TTL 超时消息的响应。这个将 TTL 自加 1 的过程一直持续直到到达目的地 4.2.2.1。

总体来说，路由跟踪要与路径上的每一个路由器进行通信，从而画出前往目的地的路由图，如图 6-36 所示。

```
Administrator: C:\Windows\system32\cmd.exe
C:\>tracert 4.2.2.1
Tracing route to unsc-pri.sys.gtei.net [4.2.2.1]
over a maximum of 30 hops:
1     1 ms      1 ms      1 ms  172.16.16.1
2     9 ms      9 ms      9 ms  74-137-8-1.dhcp.insightbb.com [74.137.8.1]
3    12 ms      9 ms      9 ms  74-137-0-225.dhcp.insightbb.com [74.137.0.225]
4    11 ms     11 ms     10 ms  74.128.9.205
5    17 ms     19 ms     18 ms  cer-edge-13.inet.quest.net [65.123.102.153]
6    17 ms     18 ms     18 ms  chp-brdr-03.inet.quest.net [67.14.8.194]
7    17 ms     18 ms     17 ms  63.146.27.18
8    18 ms     22 ms     18 ms  ae-11-55.car1.Level3.net [4.68.101.130]
9    17 ms     18 ms     18 ms  unsc-pri.sys.gtei.net [4.2.2.1]

Trace complete.
```

图 6-36 路由跟踪功能的样例输出

注

我们这里所讨论的路由跟踪主要是基于 Windows，因为只有它在使用 ICMP。Linux 上的路由跟踪更复杂一些，并使用了其他的协议来进行路径的跟踪。

你会在整本书中看到 ICMP 有着不同的功能。在我们分析更多场景的时候，我们会更频繁地使用 ICMP。

这一章主要向你介绍了数据包分析过程中会经常看到的最重要的几种协议。IP、TCP、UDP 和 ICMP 是所有网络通信的基础，并且对于你每天要进行的工作至关重要。在下一章中，我们将会学习一些常用的应用层协议。



# 第 7 章

## 常见高层网络协议



在这一章中，我们将继续介绍一些协议的功能，以及如何在 Wireshark 中查看它们。我们将介绍 3 个最常见的高层协议（第 7 层）：DHCP、DNS 和 HTTP。

### 7.1 动态主机配置协议 DHCP

在网络发展的早些时候，当一台设备想要在网络上通信，它需要被手动分配一个地址。随着网络的发展，这样的手动过程很快就变得烦琐起来。为了解决这个问题，BOOTP 协议（Bootstrap Protocol）被创建出来给连接到网络的设备自动分配地址。BOOTP 后来被更加复杂的动态主机配置协议 DHCP（Dynamic

Host Configuration Protocol) 所取代。

DHCP 是一个应用层协议，能够让设备自动获取 IP 地址（以及其他重要网络资源，比如 DNS 服务器和路由网关的地址）。大多数的 DHCP 服务器都向客户端提供一些其他的参数，比如网络上的默认网关和 DNS 服务器的地址。

### 7.1.1 DHCP 头结构

DHCP 数据包会为客户端带来很多信息。如图 7-1 所示，下列的域都会在 DHCP 数据包中出现。

| 动态主机配置协议 |                |      |       |    |  |  |
|----------|----------------|------|-------|----|--|--|
| 偏移位      | 0~15           |      | 16~31 |    |  |  |
| 0        | 操作代码           | 硬件类型 | 硬件长度  | 跳数 |  |  |
| 32       | 事务ID           |      |       |    |  |  |
| 64       | 消耗时间           |      | 标记    |    |  |  |
| 96       | 客户端 IP 地址      |      |       |    |  |  |
| 128      | 你的 IP 地址       |      |       |    |  |  |
| 160      | 服务器 IP 地址      |      |       |    |  |  |
| 196      | 网关 IP 地址       |      |       |    |  |  |
| 228+     | 客户端硬件地址(16 字节) |      |       |    |  |  |
|          | 服务器主机地址(64 字节) |      |       |    |  |  |
|          | 启动文件(128 字节)   |      |       |    |  |  |
|          | 选项             |      |       |    |  |  |

图 7-1 DHCP 数据包结构

**操作代码 (OpCode):** 用来指出这个数据包是 DHCP 请求还是 DHCP 回复。

**硬件类型 (Hardware Type):** 硬件地址类型 (10MB 以太网、IEEE802、ATM 以及其他)。

**硬件长度 (Hardware Length):** 硬件地址长度。

**跳数 (Hops):** 中继代理用来帮助寻找 DHCP 服务器。

**事务 ID (Transaction ID):** 用来匹配请求和响应的一个随机数。

**消耗时间 (Seconds Elapsed):** 客户端第一次向 DHCP 服务器发出地址请求后的时间。

**标记 (Flags):** DHCP 客户端能够接受的流量类型 (单播、广播以及其他)。

**客户端 IP 地址 (Client IP Address):** 客户端的 IP 地址 (由“你的”IP 地址域派生)。

**“你的”IP 地址 (Your IP Address):** DHCP 服务器提供的 IP 地址 (最终成为客户端 IP 地址域的值)。

**服务器 IP 地址 (Server IP Address):** DHCP 服务器的 IP 地址。

**网关 IP 地址 (Gateway IP Address):** 网络默认网关的 IP 地址。

**客户端硬件地址 (Client Hardware Address):** 客户端的 MAC 地址。

**服务器主机名 (Server Host Name):** 服务器的主机名 (可选)。

**启动文件 (Boot File):** DHCP 所使用的启动文件 (可选)。

**选项 (Options):** 用来对 DHCP 数据包进行扩展, 以提供更多功能。

## 7.1.2 DHCP 续租过程

DHCP 最主要的任务就是在续租过程中向客户端分配 IP 地址。续租过程在一个客户端和 DHCP 服务器之间进行, 如文件 `dhcp_nolease_renewal.pcap` 中所示。DHCP 的续租过程通常被称为 DORA 过程, 因为它使用了 4 种类型的 DHCP 数据包: 发现 (Discover)、提供 (Offer)、请求 (Request) 和确认 (Acknowledgement), 如图 7-2 所示。在这里, 我们将对 DORA 数据包的每种类型进行逐一介绍。

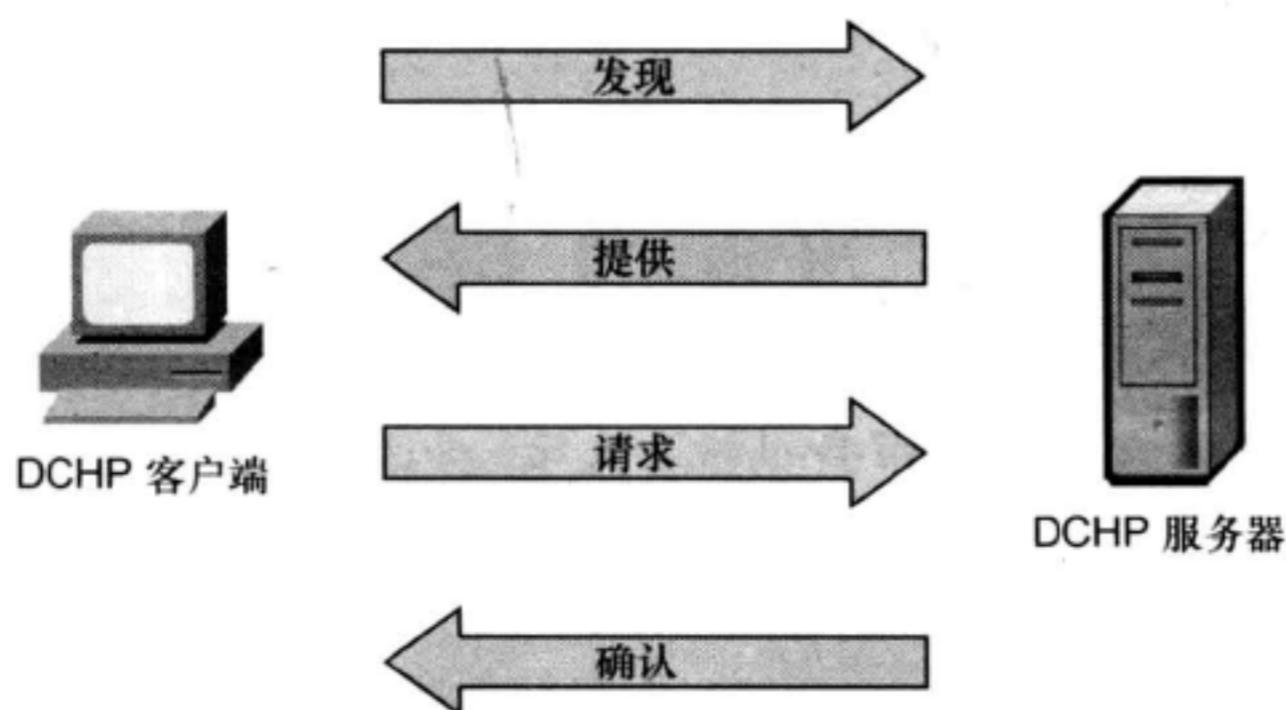


图 7-2 DHCP 的 DORA 过程

## 1. 发现数据包

在你所看到的捕获文件中，第一个数据包是从 0.0.0.0 的 68 端口发向 255.255.255.255 的 67 端口。客户端使用 0.0.0.0，是因为它目前还没有 IP 地址。数据包被发往 255.255.255.255，是因为这是一个独立于网络的广播地址，从而能确保这个数据包会被发往网络上的每台设备。因为这台设备并不知道 DHCP 服务器的地址，所以它的第一个数据包是为了寻找正在监听的 DHCP 服务器。

在 Packet Details 面板中，我们第一眼就可以看到 DHCP 是基于 UDP 作为其传输层协议的。DHCP 客户端对请求响应速度有很高要求。由于 DHCP 有其内置的保证可靠性的方法，也就意味着 UDP 是最适合的协议。你可以在第一个数据包的 Packet Details 面板的 DHCP 部分，查看发现过程的细节，如图 7-3 所示。

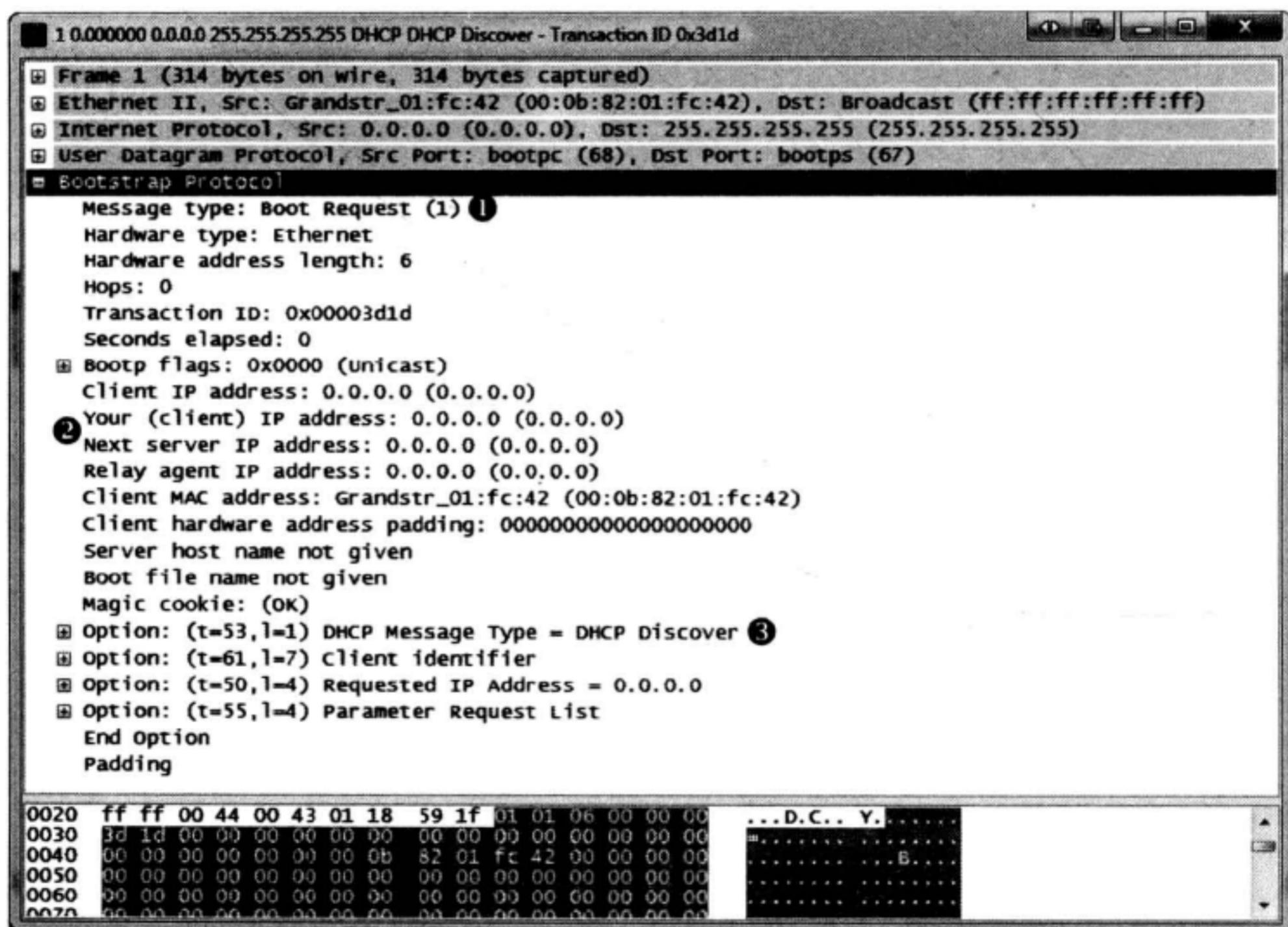


图 7-3 DHCP 发现数据包

注

由于 Wireshark 在处理 DHCP 时，仍然会引用 BOOTP，所以你会在 Packets Detail 面板中看到 Bootstrap Protocol，而不是 DHCP。但无论如何，我在这本书中仍会将其称为数据包的 DHCP 部分。

这是一个请求数据包，因为在消息类型域中标识为 1①。发现数据包中的大多数域要么为空（就像是 IP 地址域②），要么根据上一节所列出的 DHCP 域就能不言自明了。这个数据包的主要内容是以下 4 个域。

**DHCP 消息类型** 这里的选项是一个长度和值均为 1 的 53 类型 ( $t=53$ ) ③。这些值表明这是一个 DHCP 发现数据包。

**客户端标识符** 这里提供了客户端请求 IP 地址的额外信息。

**所请求 IP 地址** 这里提供了客户端希望得到的 IP 地址（通常是之前用过的 IP 地址）。

**请求参数列表** 这里列出了客户端希望从 DHCP 服务器接收到的不同配置项（其他重要网络设备的 IP 地址）。

## 2. 提供数据包

这个文件中的第二个数据包在 IP 头中列出了可用的 IP 地址，显示这个数据包是从 192.168.0.1 前往 192.168.0.10，如图 7-4 所示。客户端实际上还没有 192.168.0.10

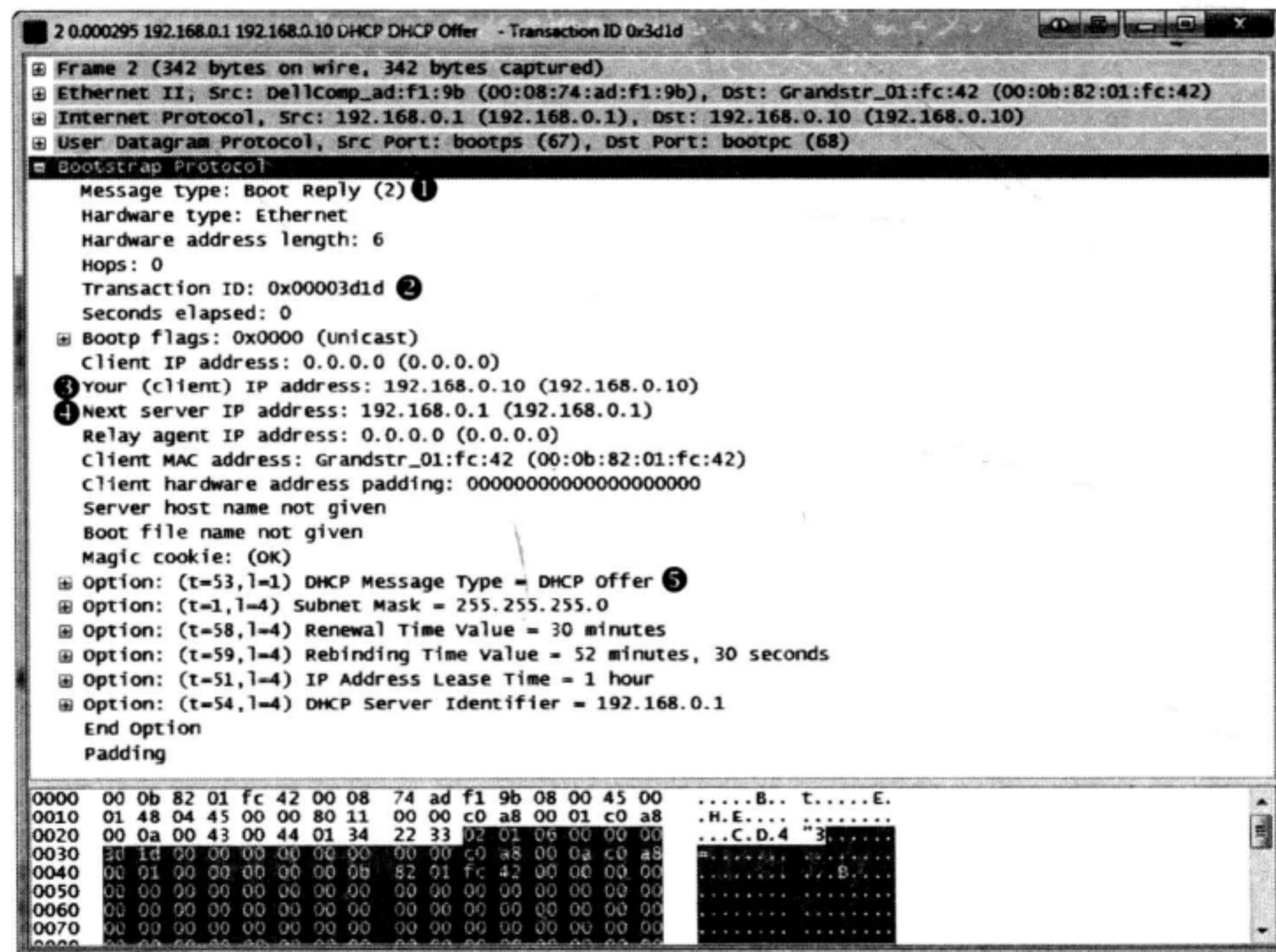


图 7-4 DHCP 提供数据包

这个地址，所以服务器会首先尝试使用由 ARP 提供的客户端硬件地址与之通信。如果通信失败，那么它将会直接将提供（Offer）广播出去，以进行通信。

第二个数据包的 DHCP 部分，称为提供数据包，表明这是一个响应的消息类型❶。这个数据包包含了和前一个数据包相同的事务 ID❷，该 ID 告诉我们这个响应与我们原先的请求相对应。

该提供数据包由 DHCP 服务器发出，用于向客户端提供服务。它提供了关于其自己的信息，以及它先要给客户端提供的地址。在图 7-4 中，在“你的”（客户端）IP 地址域中的 IP 地址 192.168.0.10 就是要提供给客户端的❸。下一个服务器 IP 地址（Next Server IP Address）域❹中的值 192.168.0.1 表明我们的 DHCP 服务器与默认网关共享一个 IP 地址。

列出的第一个选项指明这个数据包是一个 DHCP 提供❺。服务器所提供的以下选项和客户端的 IP 地址，一起给出了它所能提供的额外信息。你可以看到它给出了如下信息。

- 子网掩码是 255.255.255.0。
- 续租时间是 30min。
- 重新绑定时间的值是 52min30s。
- IP 地址的租期是 1h。
- DHCP 服务器的标识符是 192.168.0.1。

### 3. 请求数据包

在客户端接到 DHCP 服务器的提供数据包之后，它将以一个 DHCP 请求数据包作为接收确认，如图 7-5 所示。

这个捕获文件中的第三个数据包仍然从 IP 地址 0.0.0.0 处发出，因为我们还没有完成获取 IP 地址的过程❶。但数据包现在知道了它所要通信的 DHCP 服务器。

消息类型域显示这是一个请求数据包❷。尽管这个捕获文件上的每个数据包都属于同一个续租过程，但因为这是一个新的请求/响应过程❸，所以它有了一个新的事务 ID。这个数据包与发现数据包相似，其所有 IP 地址信息都是空的。

在最后的选项域中❹，我们看到这是一个 DHCP 请求。值得注意的是所请求的 IP 地址不再是空的，并且 DHCP 服务器标识符域也包含 IP 地址。

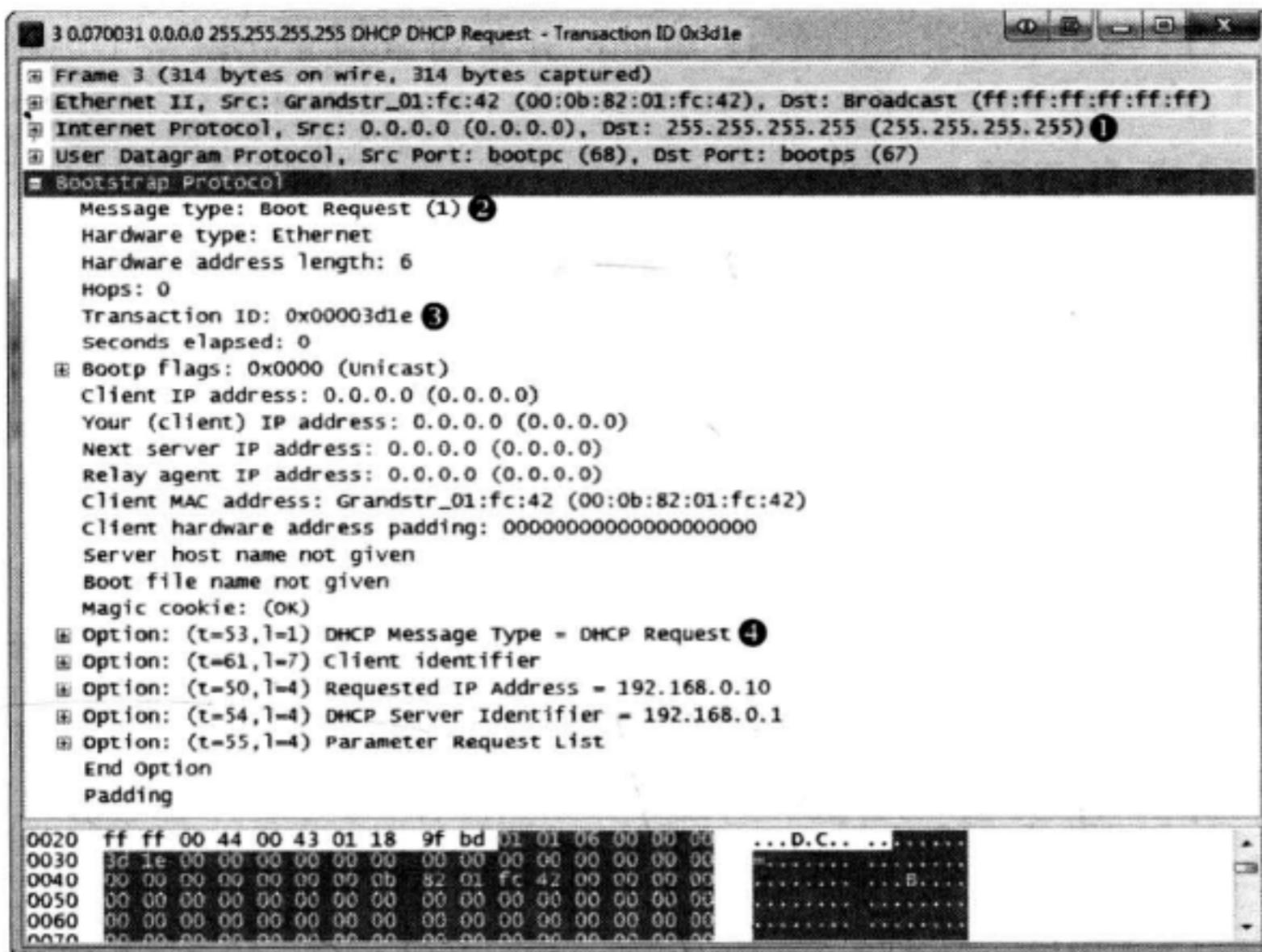


图 7-5 DHCP 请求数据包

#### 4. 确认数据包

这个过程的最后一步就是 DHCP 在确认数据包中给客户端发送其所请求的 IP 地址，并在其数据库中记录相关信息，如图 7-6 所示。



图 7-6 DHCP 确认数据包

这时客户端就有了一个 IP 地址，并且可以用它在网络上通信。

### 7.1.3 DHCP 租约内续租

当 DHCP 给一台设备分配了一个 IP 地址时，它同时给客户端定下了一个租约。也就是说，客户端只能在有限的时间内使用这个 IP 地址，否则就必须续租。我们刚刚介绍的 DORA 过程是出现在客户端第一次获取 IP 地址或者其租约时间已经过期的情况下。在这两种情况中，这台设备都会被视为租约过期。

当一个拥有 IP 地址的客户端在租约内重新启动，它必须进行一次精简版的 DORA 过程来重新认领它的 IP 地址。这个过程被称为租约内续租。

在租约内续租时，发现和提供数据包就变得没有必要了。考虑到其与租约过期时的 DORA 过程类似，可以发现在租约内续租并不需要那么做，而只是完成请求和确认两个步骤就可以了。你可以在文件 `dhcp_inlease_renewal.pcap` 中看到租约内续租的一个捕获样例。

### 7.1.4 DHCP 选项和消息类型

DHCP 依赖于其可选项来提供真正的灵活性。正如你所看到的那样，数据包的 DHCP 选项在大小和内容上都可以变化。数据包的整体大小则取决于其所使用的选项。你可以在 <http://www.iana.org/assignments/bootp-dhcp-parameter> 中看到众多 DHCP 选项的完整列表。

所有 DHCP 数据包都需要的唯一选项就是消息类型选项（选项 53）。这个选项标识着 DHCP 客户端或者服务器将如何处理数据包中的信息。在表 7-1 中给出了所定义的 8 种消息类型。

表 7-1 DHCP 消息类型

| 类型号 | 消息类型 | 描述                          |
|-----|------|-----------------------------|
| 1   | 发现   | 客户端用来定位可用的 DHCP 服务器         |
| 2   | 提供   | 服务器用来给客户端发送发现数据包的响应         |
| 3   | 请求   | 客户端用来请求服务器所提供的参数            |
| 4   | 拒绝   | 客户端向服务器指明数据包的无效参数           |
| 5   | ACK  | 服务器向客户端发送所请求的配置参数           |
| 6   | NAK  | 客户端向服务器拒绝其配置参数的请求           |
| 7   | 释放   | 客户端向服务器通过取消配置参数来取消租约        |
| 8   | 通知   | 当客户端已经有 IP 地址时客户端向服务器请求配置参数 |

## 7.2 域名系统

域名系统（Domain Name System, DNS）是最重要的互联网协议之一，因为它是众所周知的粘合剂。DNS 将例如 `www.google.com` 的名字和例如 `74.125.159.99` 的 IP 地址捆绑起来。当我们想要和一台网络设备通信却不知道它的 IP 地址时，我们就可以使用它的 DNS 名字来进行访问。

DNS 服务器存储了一个有着 IP 地址和 DNS 名字映射资源记录的数据库，并将其和客户端与其他 DNS 服务器共享。

注

由于 DNS 服务器的结构很复杂，我们只关注于通常类型的 DNS 流量。你可以通过 <http://www.isc.org/community/reference/RFCs/DNS> 来查看 DNS 相关的 RFC 文档。

### 7.2.1 DNS 数据包结构

如图 7-7 所示，DNS 数据包和我们之前所看到的数据包类型结构有所不同。DNS 数据包中会出现下面的一些域。

**DNS ID 号 (DNS ID Number)**: 用来对应 DNS 查询和 DNS 响应。

**查询/响应 (Query/Response, QR)**: 用来指明这个数据包是 DNS 查询还是响应。

**操作代码 (OpCode)**: 用来定义消息中请求的类型。

**权威应答 (Authoritative Answer, AA)**: 如果响应数据包中设定了这个值，则说明这个响应是由域内权威域名服务器发出的。

**截断 (Truncation, TC)**: 用来指明这个响应由于太长，无法装入数据包而被截断。

**期望递归 (Recursion Desired, RD)**: 当请求中设定了这个值，则说明 DNS 客户端在目标域名服务器不含有所请求信息的情况下，要求进行递归查询。

**可用递归 (Recursion Available, RA)**: 当响应中设定了这个值，说明域名服务器支持递归查询。

**保留 (Z)**: 在 RFC1035 的规定中被全设为 0，但有时会被用来作为 RCode

域的扩展。

**响应代码 (Response Code):** 在 DNS 响应中用来指明错误。

**问题计数 (Question Count):** 在问题区段中的条目数。

**回答计数 (Answer Count):** 在回答区段中的条目数。

**域名服务器计数 (Name Server Count):** 在权威区段的域名资源记录数。

**额外记录计数 (Additional Records Count):** 在额外信息区段中其他资源记录数。

**问题区段 (Question section):** 大小可变，包含有被发送到 DNS 服务器的一条或多条的信息查询的部分。

**回答区段 (Answer section):** 大小可变，含有用来回答查询的一条或多条资源记录。

**权威区段 (Authority section):** 大小可变，包含指向权威域名服务器的资源记录，用以继续解析过程。

**额外信息区段 (Additional Information section):** 包含资源记录且大小可变的区段，这些资源记录用来存储完全没有必要回答的查询相关的额外信息。

| 域名系统 |          |        |      |                       |                       |           |
|------|----------|--------|------|-----------------------|-----------------------|-----------|
| 偏移位  | 0~15     | 16~31  |      |                       |                       |           |
| 0    | DNS ID 号 | Q<br>R | 操作代码 | A<br>A<br>C<br>D<br>A | T<br>R<br>R<br>D<br>A | Z<br>响应代码 |
| 32   | 问题计数     | 回答区段   |      |                       |                       |           |
| 64   | 域名服务器计数  | 额外记录计数 |      |                       |                       |           |
| 96   | 问题区段     | 回答区段   |      |                       |                       |           |
| 128  | 权威区段     | 额外信息区段 |      |                       |                       |           |

图 7-7 DNS 数据包结构

## 7.2.2 一次简单的 DNS 查询过程

DNS 以查询/响应的模式工作。当一个客户端想要将一个 DNS 名字解析成 IP 地址，它向 DNS 服务器发送一个查询，然后服务器会在响应中提供所请求的信息。在最简单的情形下，这个过程包含着两个数据包，正如在捕获文件 dns\_query\_response.pcap 中所看到的那样。

第一个数据包如图 7-8 所示，是由 192.168.0.114 的客户端通过 DNS 的标准端口 53 发向 205.152.37.23 的服务器的 DNS 查询。

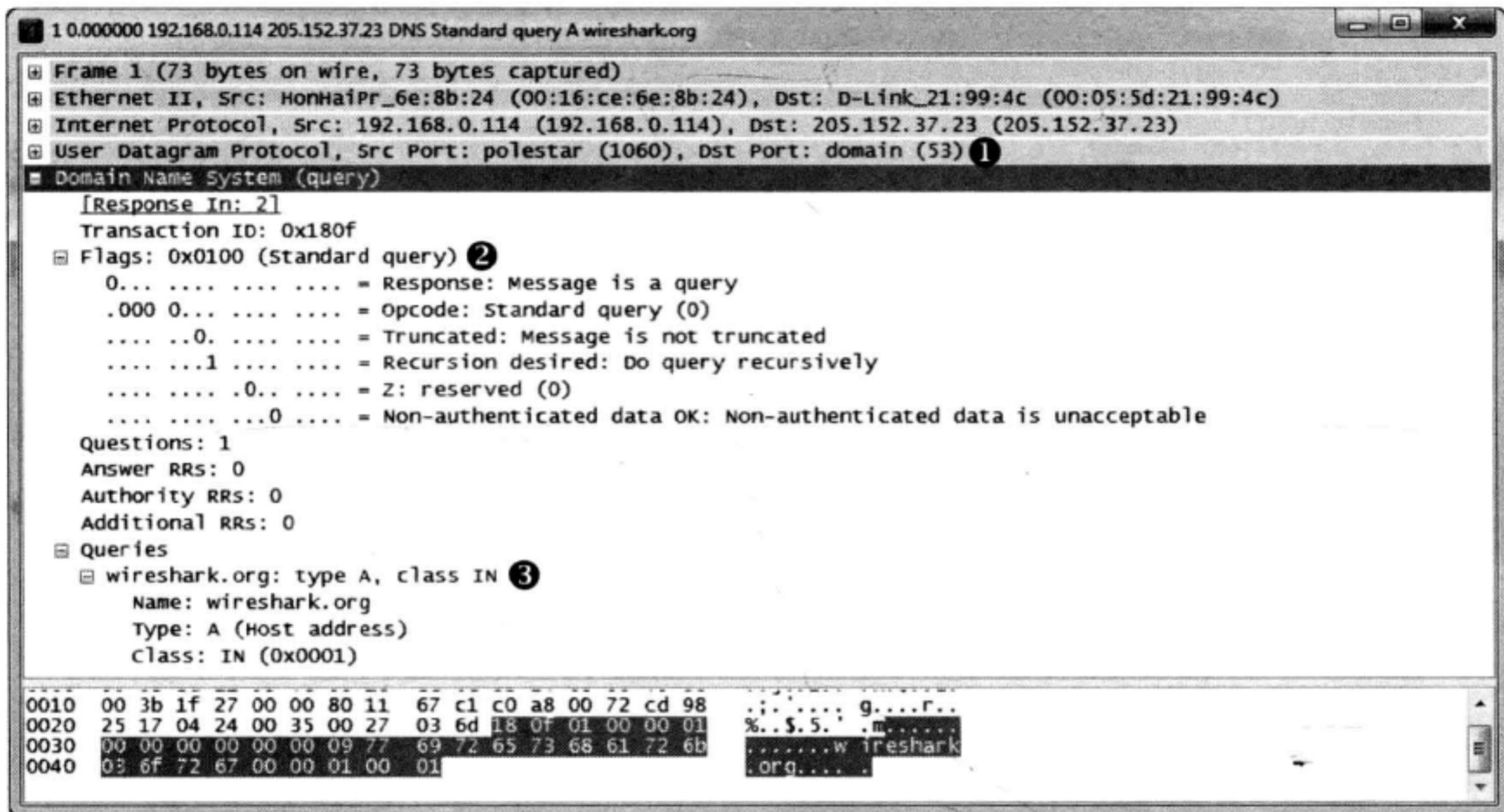


图 7-8 DNS 查询数据包

当你检查这个数据包的头部时，你会发现 DNS 也基于 UDP 协议①。

在数据包的 DNS 区段，你可以看到数据包开头的一些较小域都被 Wireshark 合并成一个标志区段（Flags section）。展开这个区段，你会看到这个消息是一个典型的请求②：没有被截断、期望递归查询（我们将随后介绍递归查询）。在展开查询区段时，仅有的一个问题是查询名字为 wireshark.org 的主机类型（type A）互联网（IN）地址③。这个数据包基本就是在问：“哪个 IP 地址对应着 wireshark.org 域？”

数据包 2 响应了这个请求，如图 7-9 所示。因为这个数据包拥有唯一的标识码①，所以我们知道这里包含着对原始查询的正确响应。

标志区段可以确保这是一个响应并且允许必要的递归②。这个数据包仅包含一个问题和一个资源记录③，因为它将原问题和回答连接了起来。展开回答区段可以看到对于查询的回答：wireshark.org 的地址是 128.121.50.122④。有了这个信息，客户端就可以开始构建 IP 数据包，并与 wireshark.org 进行通信了。

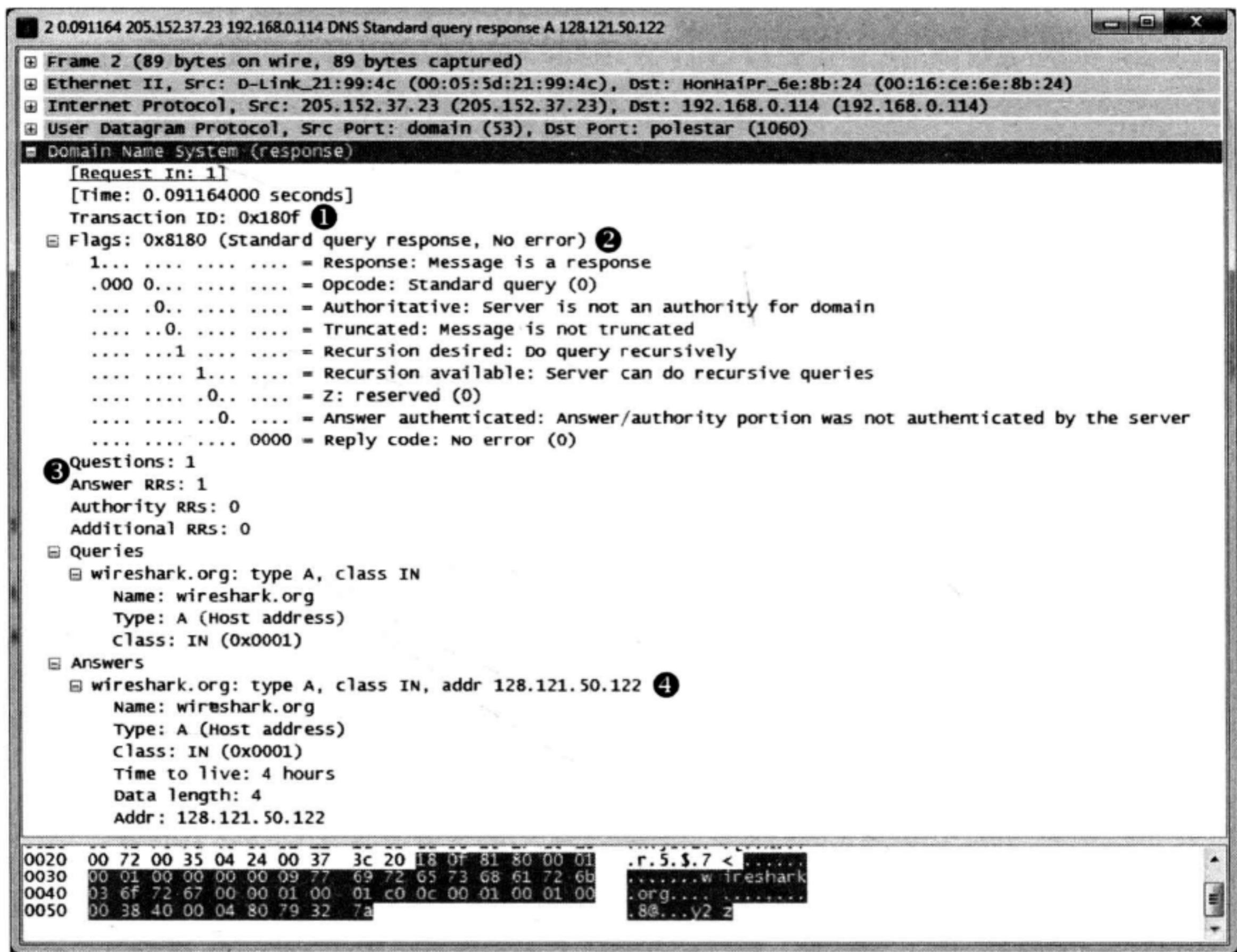


图 7-9 DNS 响应数据包

### 7.2.3 DNS 问题类型

DNS 查询和响应中所使用的类型域，指明了这个查询或者响应的资源记录类型。表 7-2 中列出了一些常用的消息/资源记录类型。

表 7-2 常用 DNS 资源记录类型

| 值 | 类型    | 描述        |
|---|-------|-----------|
| 1 | A     | IPv4 主机地址 |
| 2 | NS    | 权威域名服务器   |
| 5 | CNAME | 规范别名      |

续表

| 值   | 类型   | 描述        |
|-----|------|-----------|
| 15  | MX   | 邮件交换      |
| 16  | TXT  | 文本字符串     |
| 28  | AAAA | IPv6 主机地址 |
| 251 | IXFR | 增量区域传送    |
| 252 | AXFR | 完整区域传送    |

表 7-2 中所列简略且并不详尽，如果希望查看完整的 DNS 资源记录类型，请访问 [http://www.iana.org/assignments/dns-parameters/。](http://www.iana.org/assignments/dns-parameters/)

## 7.2.4 DNS 递归

由于互联网的 DNS 结构是层级式的，为了能够回答客户端提交的查询，DNS 服务器必须能够彼此通信。我们的内部 DNS 服务器知道我们本地局域网服务器的名字和 IP 地址的映射，但不太可能知道谷歌或者戴尔的 IP 地址。

当 DNS 服务器需要查找一个 IP 地址时，它会代表发出请求的客户端向另一个 DNS 服务器查询。实际上，这个 DNS 服务器与客户端的行为相同。这个过程叫做递归查询。

打开文件 `dns_recursivequery_client.pcap`，可以分别看到 DNS 客户端和服务端视角的递归查询过程。这个文件包含了从客户端捕获的两个 DNS 数据包。第一个数据包是从 DNS 客户端 172.16.0.8 发往 DNS 服务器 172.16.0.102 的初始查询，如图 7-10 所示。

当你展开这个数据包的 DNS 区段，你可以看到这是一个用于查找 DNS 名称 `www.nostarch.com`①的 A 类型记录的标准查询。展开标志区段，可以了解更多关于这个数据包的信息，你可以看到期望递归的标志②。

第二个数据包是我们所希望看到的对于初始数据包的响应，如图 7-11 所示。

这个数据包的事务 ID 和我们的查询相匹配①，也没有列出错误，所以我们得到了 `www.nostarch.com` 所对应的 A 类型资源记录②。

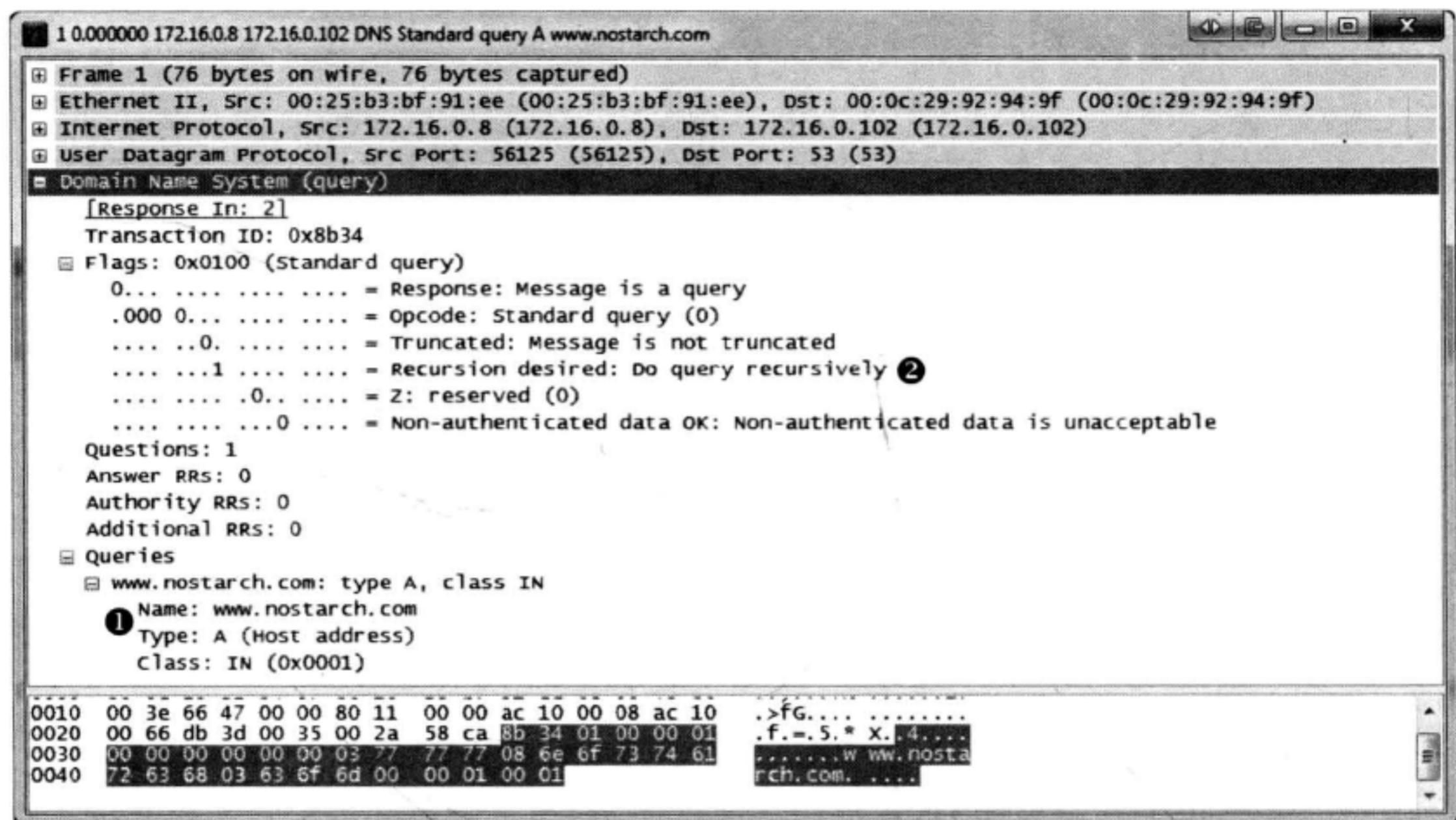


图 7-10 设置有期望递归位的 DNS 查询

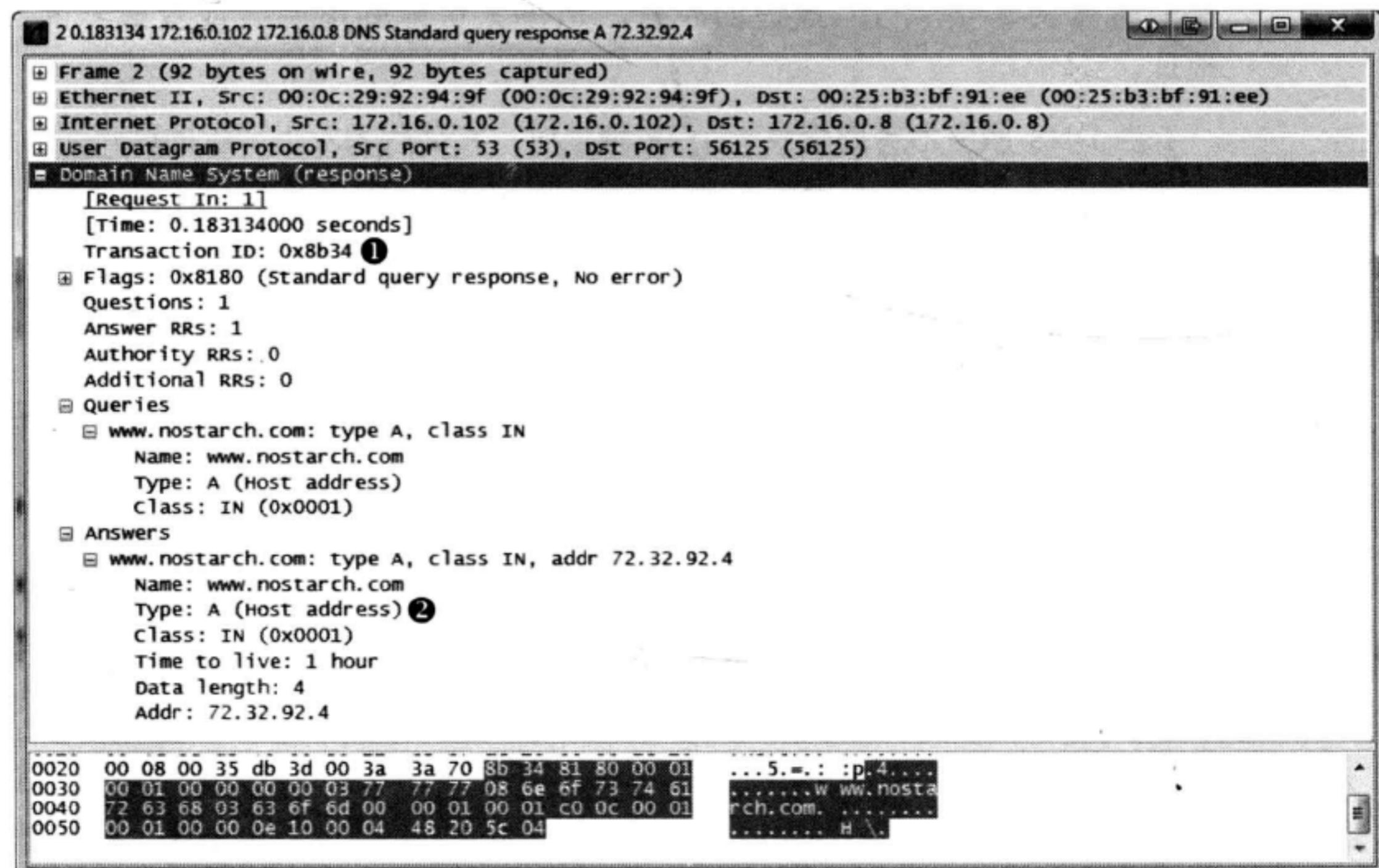


图 7-11 DNS 查询响应

如果我们想要知道查询是否被递归应答，唯一的方法就是当进行递归查询时监听 DNS 服务器的流量，正如文件 dns\_recursivequery\_server.pcap 中所示的那样。这个文件显示了查询开始时在本地 DNS 服务器上捕获的流量。第一个数据包和我们之前捕获文件中的初始查询相同。这时，DNS 服务器接收到了这个查询，检索本地数据库后，发现它并不知道关于 DNS 域名（nostarch.com）所对应 IP 地址这个问题的答案。由于这个数据包发送时设置了期望递归，你就会在第二个数据包中看到这个 DNS 服务器为了得到答案就可以向其他 DNS 服务器询问这个问题。

在第二个数据包中，位于 172.16.0.102 的 DNS 服务器向位于 4.2.2.1，也就是其所设定的要转发上行请求的服务器，发送了一个新的查询，如图 7-12 所示。这个请求是原始请求的镜像，并将 DNS 服务器变成一个客户端。

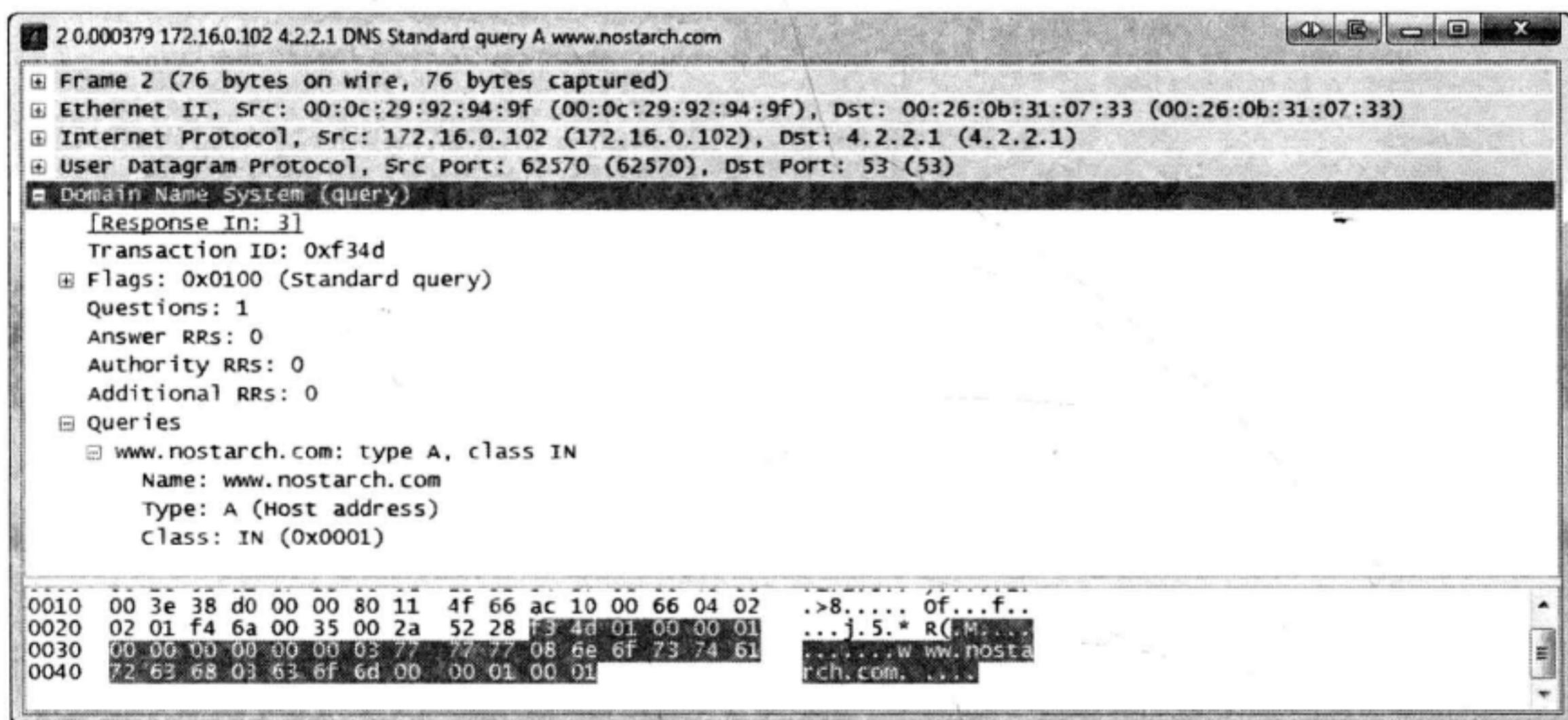


图 7-12 递归 DNS 查询

由于这个事务 ID 与之前捕获文件中的事务 ID 不同，所以我们可以将这个 DNS 查询作为一个新的查询。在这个数据包被服务器 4.2.2.1 接收到之后，本地 DNS 服务器就接收到了响应，如图 7-13 所示。

接到了这个响应后，本地 DNS 服务器就可以将第 4 个也就是最后一个带有请求信息的数据包传递给 DNS 客户端。

尽管这个例子只展示了一层的递归，但对一个 DNS 请求来说递归查询

可能会发生很多次。这里我们接到了来自 DNS 服务器 4.2.2.1 的回答，但那个服务器可能为了寻找答案也向其他服务器进行了递归查询。一个简单查询在其得到最终响应之前可能遍历了全世界。图 7-14 展示了递归 DNS 查询的过程。

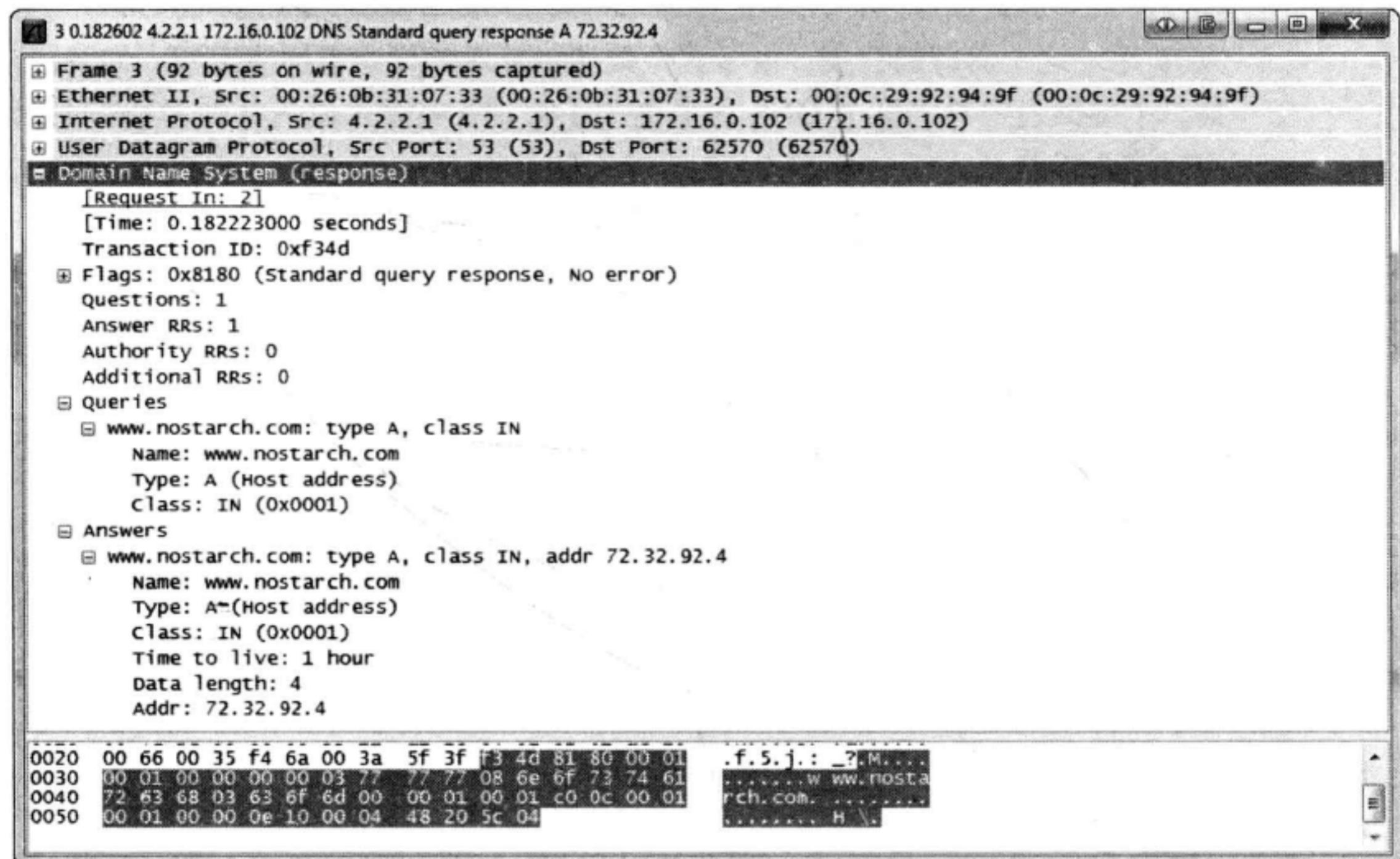


图 7-13 对递归 DNS 查询的响应



图 7-14 递归 DNS 查询

## 7.2.5 DNS 区域传送

DNS 区域是一个 DNS 服务器所授权管理的名字空间（或是一组 DNS 名称）。举例来说，Emma’s Diner 这个网站可能由一个 DNS 服务器对 emmasdiner.com 负责。这样，无论是 Emma’s Diner 内部还是外部的设备，如果希望将 emmasdiner.com

解析成 IP 地址，都需要和这个区域的权威，也就是这个 DNS 服务器联系。如果 Emma's Diner 发展壮大了，它可能会增加一个 DNS 服务器，专门用来处理其名字空间的 email 部分，比如 mail.emmasdiner.com，那么这个服务器，就成为这个邮件子区域的权威。如果必要的话，还可以为子域名添加更多的 DNS 服务器，如图 7-15 所示。

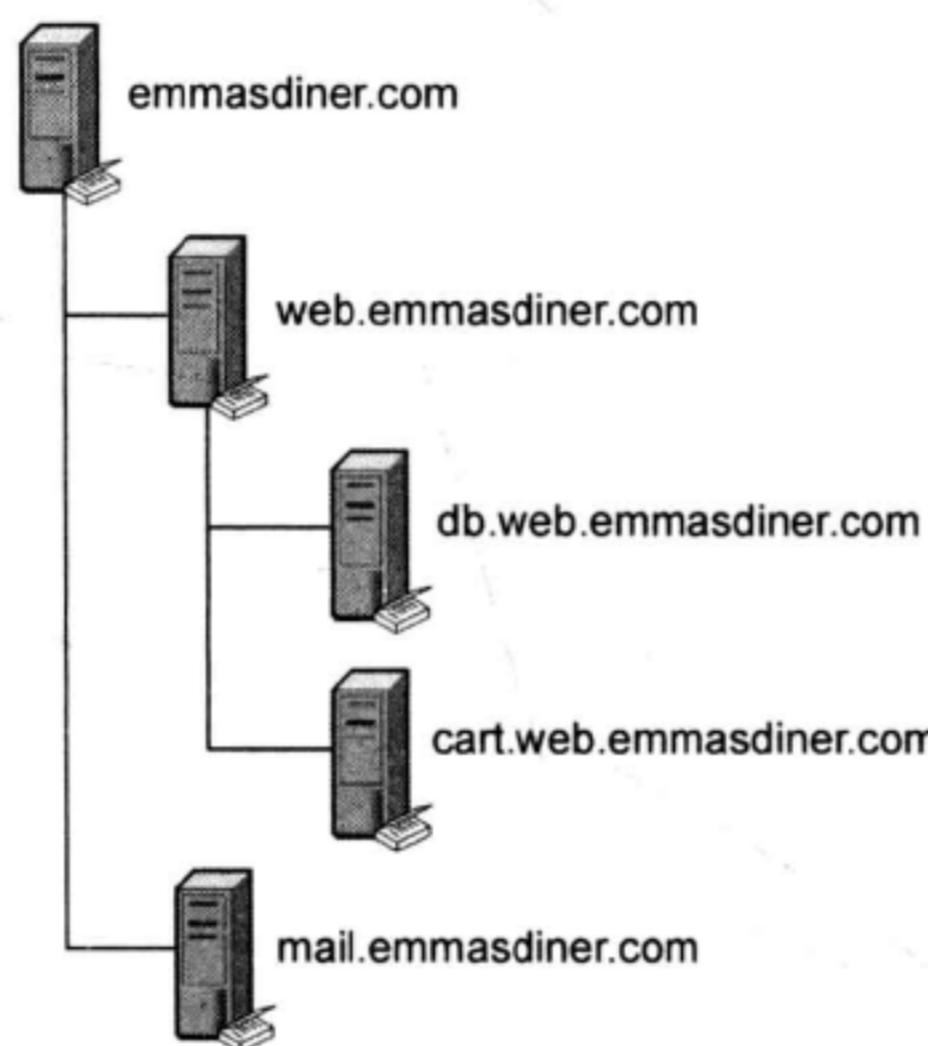


图 7-15 DNS 区域为名字空间划分了责任

区域传送指出于冗余备份的需要，在两台设备之间传送区域数据。举例来说，在拥有多个 DNS 服务器的组织中，管理员通常都会配置一台备用 DNS 服务器，用来维护一份主服务器 DNS 信息的拷贝，以防止主 DNS 服务器不可用。主要存在两种区域传送。

**完整区域传送（AXFR）：**这个类型的传送将整个区域在设备间进行传送。

**增量区域传送（IXFR）：**这个类型的传送仅传送区域信息的一部分。

文件 dns\_axfr.pcap 包含了一个主机 172.16.16.164 和 172.16.16.139 之间完整区域传送的例子。

当你第一眼看这个文件时，你可能会怀疑是否打开了正确的文件，因为你所见到的是 TCP 数据包而不是 UDP 数据包。尽管 DNS 基于 UDP 协议，它在比如区域传送的一些任务中仍会使用 TCP 协议，因为 TCP 对于规

模数据的传输更加可靠。这个捕获文件中的前 3 个数据包是 TCP 的三次握手。

第 4 个数据包开始在 172.16.16.164 和 172.16.16.139 之间进行实际的区域传送。这个数据包并不包含任何 DNS 信息。由于区域传送请求数据包中的数据是由多个数据包发送，所以这个数据包被标记为“重组装 PDU 的 TCP 分片”。数据包 4 和 6 包含了数据包的数据。数据包 5 是对于数据包 4 被成功接收的确认。这些数据包以这种方式显示出来是因为 Wireshark 为了容易阅读，而将 TCP 数据包以这种方式解析并呈现。我们可以将数据包 6 作为完整的 DNS 区域传送请求的参考，如图 7-16 所示。

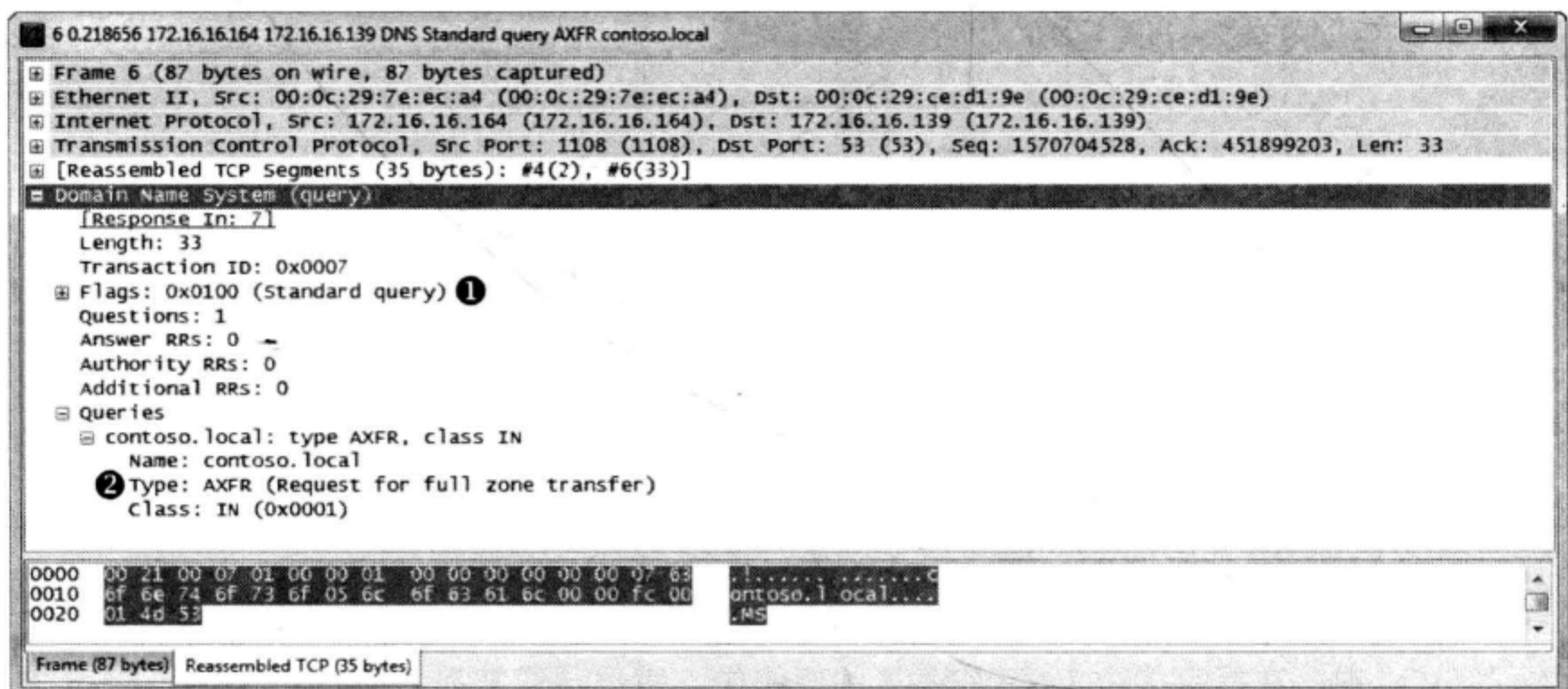


图 7-16 DNS 完整区域传送请求

区域传送请求是典型的查询①，但它请求的是 AXFR 类型②而不是单一记录类型，就意味着它希望从服务器接收全部 DNS 区域。服务器在数据包 7 中回复了区域记录，如图 7-17 所示。正如你所见到的那样，区域传送包含了相当多的数据，并且这还是一个很简单的例子！在区域传送完成之后，捕获文件以 TCP 连接的终止过程作为结束。

#### 警告

区域传送的数据如果落入他人手中可能会很危险。举例来说，通过枚举一个 DNS 服务器，你可以绘出整个网络的基础结构。

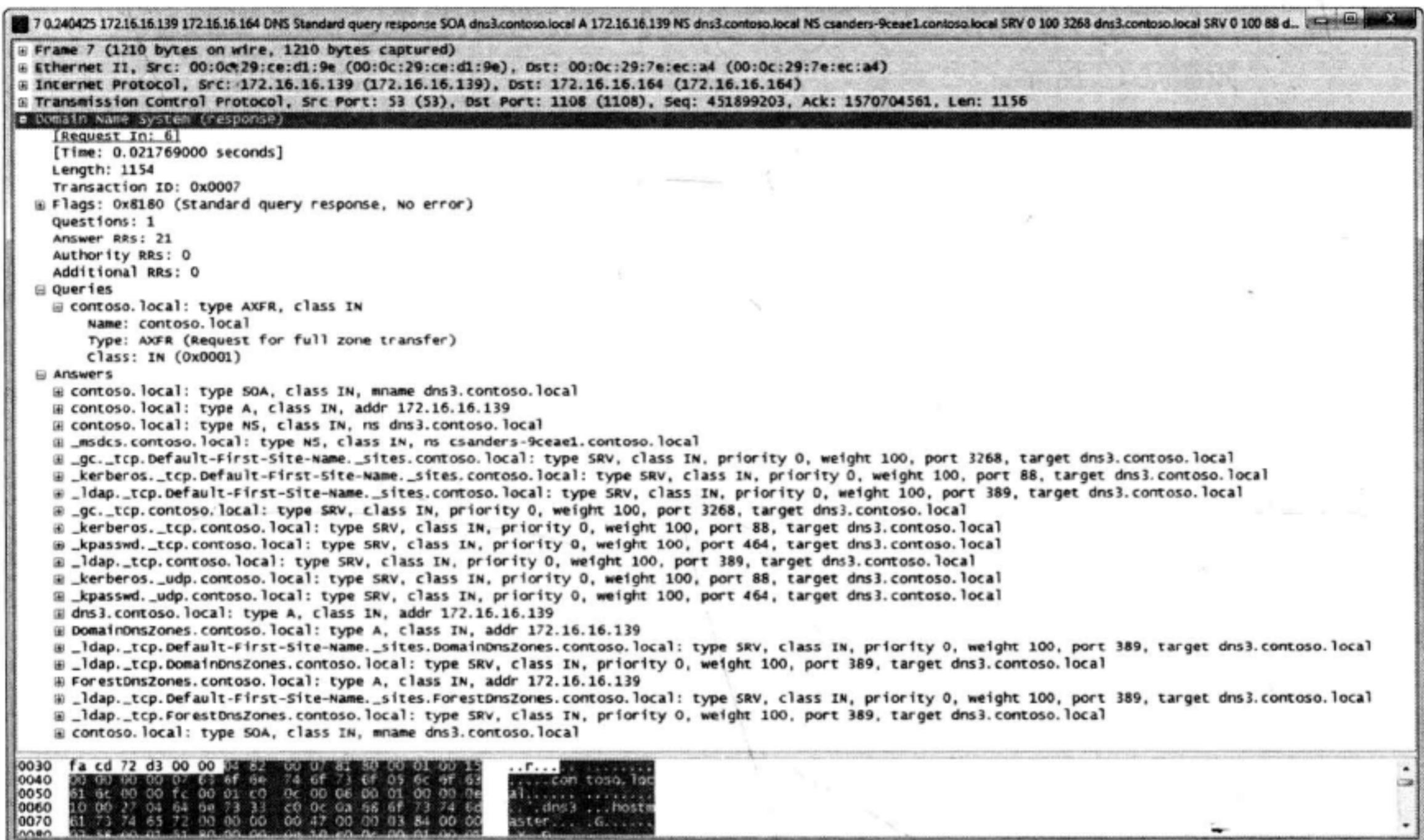


图 7-17 正在进行的 DNS 完整区域传送

## 7.3 超文本传输协议

超文本传输协议 (Hypertext Transfer Protocol, HTTP) 是万维网 (World Wide Web) 的传输机制，允许浏览器通过连接 Web 服务器浏览网页。目前在大多数组织中，HTTP 流量在网络中所占的比率是最高的。每一次使用谷歌搜索，连接 Twitter 发一条微博，或者在 ESPN.com 上查看肯塔基大学的篮球比分，你都会用到 HTTP。

我们不会去看 HTTP 传输的数据包结构，因为不同目的数据包的内容差别会很大。这个任务就留给你了。这里，我们来看 HTTP 的实际应用。

### 7.3.1 使用 HTTP 浏览

HTTP 最常被用来使用浏览器浏览 Web 服务器上的网页。捕获文件 http\_google.pcap 就给出了这样一个使用 TCP 作为传输层协议的 HTTP 传输例子。通信以客户端 172.16.16.128 和谷歌 Web 服务器 74.125.95.104 的三次握手开始。

在建立了连接之后，第一个被标为 HTTP 的数据包是从客户端发向服务器，如图 7-18 所示。

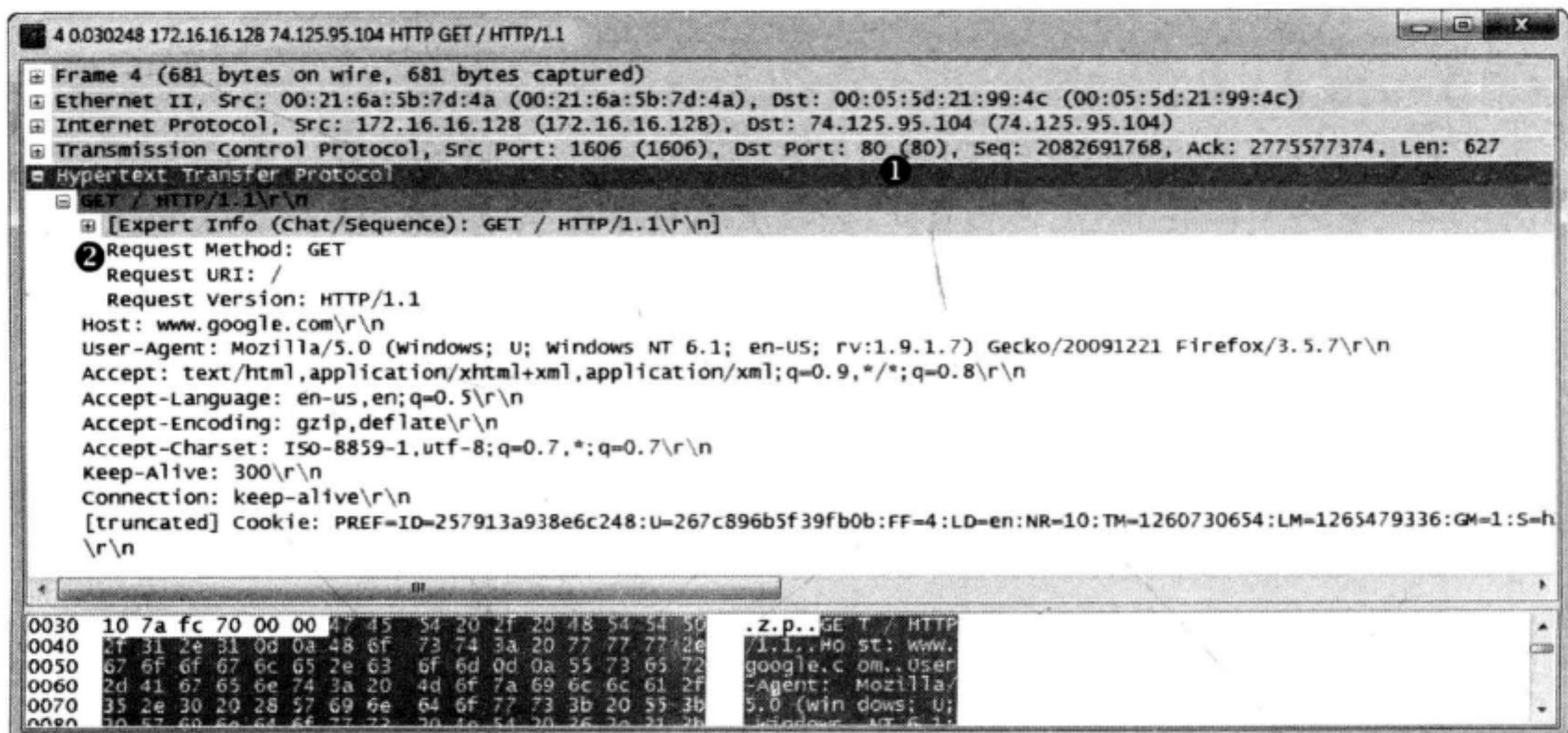


图 7-18 初始 HTTP GET 请求数据包

HTTP 数据包通过 TCP 被传输到服务器的 80 端口❶，也就是 HTTP 通信的标准端口（8080 端口也常被使用）。

HTTP 数据包会被确定为 8 种不同请求方法中的一种（根据 HTTP 规范版本 1.1 的定义）。这些请求方法指明了数据包发送者想要对接收者采取的动作。如图 7-18 所示，这个数据包的方法是 GET，它的请求通用资源标识符（Uniform Resource Indicator）是 /，并且请求版本是 HTTP/1.1❷。这些信息告诉我们这个客户端请求使用 HTTP 的 1.1 版本，下载 Web 服务器的根目录（/）。

接下来，主机向 Web 服务器发送关于自己的信息。这些信息包含了类似于使用的用户代理（浏览器）、浏览器接受的语言（Accept Languages）和 Cookie 信息（位于捕获的底部）。为保证兼容性，服务器可以利用这些信息决定返回给客户端的数据。

当服务器接收到了数据包 4 中的 HTTP 请求，它响应了一个 TCP ACK，用于确认数据包，并在数据包 6 到 11 中传输所请求的数据。HTTP 只被用来发布客户端和服务器的应用层命令。当进行数据传输时，除了在数据流的开始和结束部分，是看不到应用层的控制信息的。

服务器将数据在数据包 6 和 7 中发送，数据包 8 是来自客户端的确认，数

据包 9 和 10 是另外两个数据数据包，数据包 11 是另外一个确认，如图 7-19 所示。尽管 HTTP 仍然负责这些传输，但所有这些数据包在 Wireshark 中都被显示为 TCP 分片而不是 HTTP 数据包。

| No. | Time     | Source        | Destination   | Protocol | Info   |
|-----|----------|---------------|---------------|----------|--|
| 6   | 0.101202 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]                           |
| 7   | 0.101465 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]                           |
| 8   | 0.101495 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [ACK] Seq=2082692395 Ack=2775580186 Win=4218 Len=0 |
| 9   | 0.102282 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]                           |
| 10  | 0.102350 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]                           |
| 11  | 0.102364 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [ACK] Seq=2082692395 Ack=2775581694 Win=4218 Len=0 |

图 7-19 客户端浏览器和 Web 服务器之间使用 TCP 传输数据

在数据传输结束后，数据的重组装流就已经被发送完了，如图 7-20 所示。

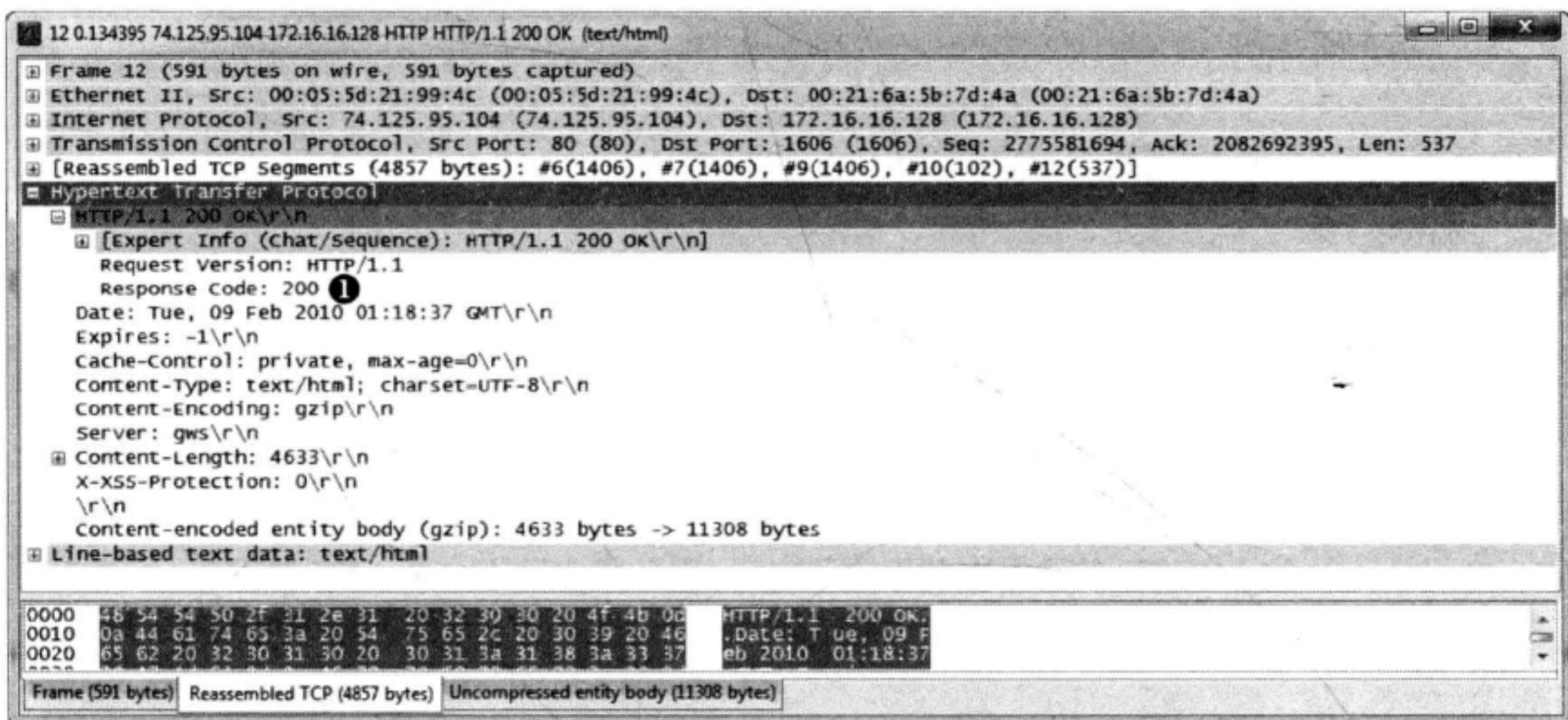


图 7-20 最后有着响应码 200 的 HTTP 数据包

HTTP 使用了一些预定义的响应码来表示请求方法的结果。在这个例子中，我们看到一个带有 200 响应码的数据包①，表示一次成功的请求方法。这个数据包同样包含一个时间戳，以及一些关于 Web 服务器内容编码和配置参数的额外信息。当客户端接收到这个数据包后，这次处理便完成了。

### 7.3.2 使用 HTTP 传送数据

我们看过了从 Web 服务器下载数据的过程，现在来关注上传数据。文件 http\_post.pcap 包含了一个非常简单的上传例子：一个用户向一个网站发表评论。在初始的三次握手之后，客户端（172.16.16.128）向 Web 服务器（69.163.176.56）发送了一个 HTTP 数据包，如图 7-21 所示。

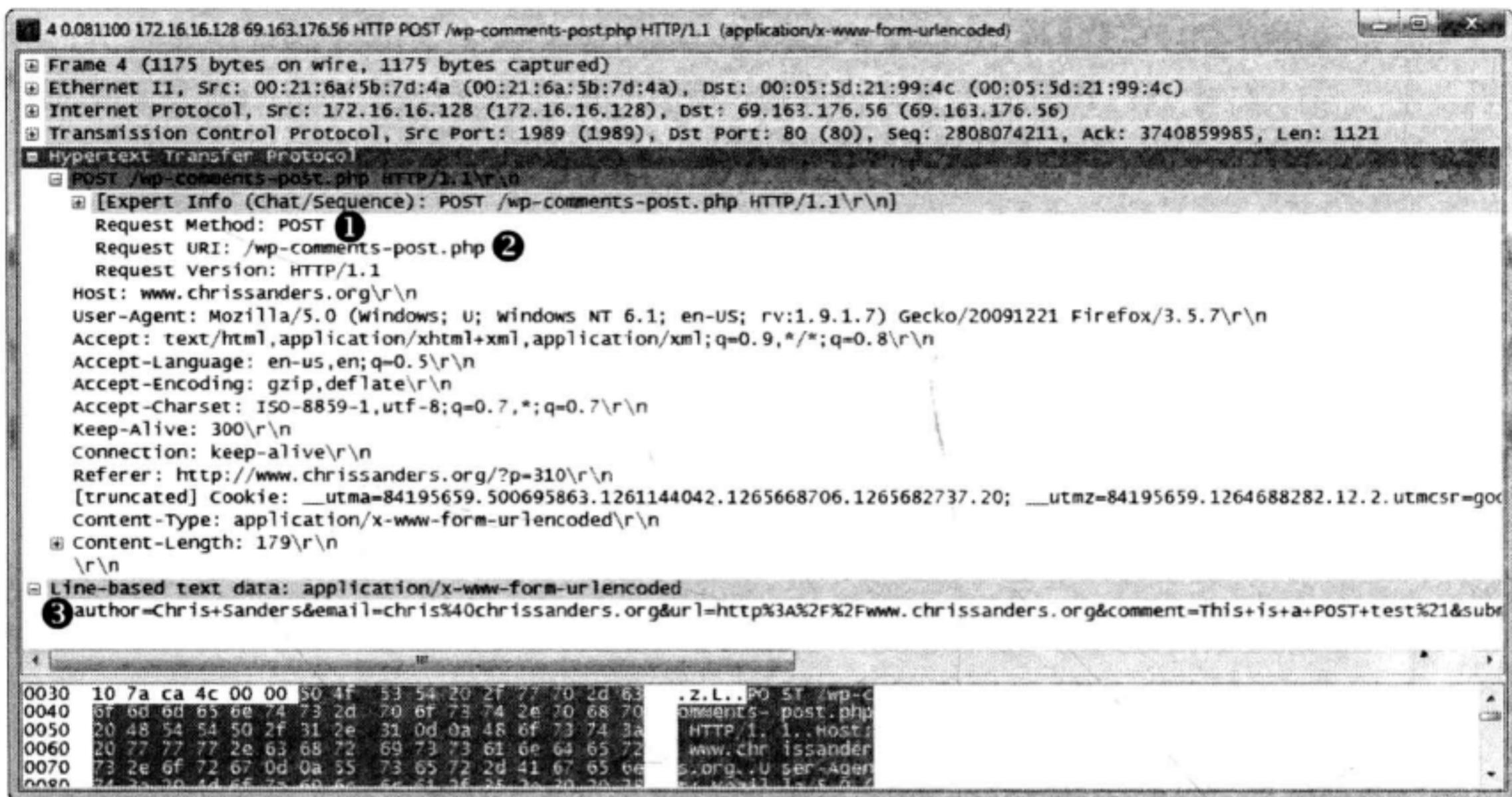


图 7-21 HTTP POST 数据包

这个数据包使用 POST 方法①来向 Web 服务器上传数据以供处理。这里使用的 POST 方法指明了 URI /wp-comments-post.php②，以及 HTTP 1.1 请求版本。如果想看上传数据的内容，可以展开这个数据包的 Line-based Text Data 部分③。

这个 POST 的数据在传输完了之后，服务器会发送一个 ACK 数据包。如图 7-22 所示，服务器在数据包 6 中传输了一个响应码 302（代表“找到”）①作为响应。

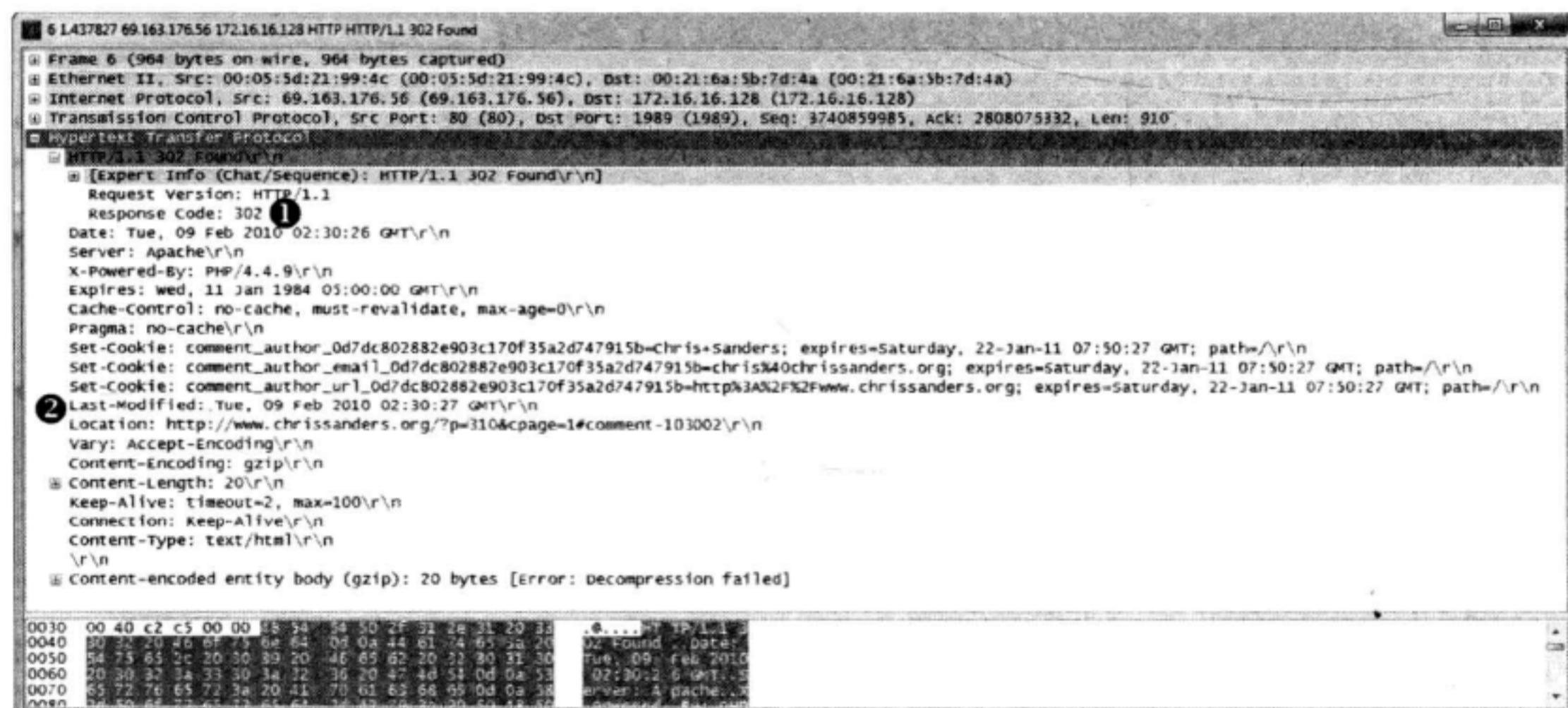


图 7-22 HTTP 响应码 302 用来进行重定向

302 响应码是 HTTP 世界的一个常用的重定向手段。这个数据包的 Location 域指明了客户端被重定向的位置❷。在这种情况下，这个地方便是评论所发表的原先网页。最后，服务器传送一个状态码 200，并且这个页面的内容会在接下来的一些数据包中进行发送，从而完成传输。

## 7.4 小结

这一章介绍了你在检查应用层流量时遇到的最常见的协议。在下面的一章中，我们将通过探索大量实战场景来介绍一些新的协议，以及我们这里所介绍协议的额外功能。

如果希望学习更多关于协议的知识，可以阅读它们相关的 RFC 文档，或者看一看 Charles Kozeriok 的 *The TCP/IP Guide* (No Starch Press, 2005)。同样也可以看一下本书附录中列出的资源。



# 第8章

## 基础的现实世界场景



从这章开始，我们将深入了解数据包分析的内涵，使用 Wireshark 分析现实世界中的网络问题。在本章第一部分，我们将分析网络工程师、服务台技术人员、应用开发者在日常工作中会遇到的场景——全都来自于我与同事们的实际经验。我们将使用 Wireshark 查看来自 Twitter、Facebook 和 ESPN.com 的流量，观察这些常用的服务是如何工作的。

本章第二部分将介绍一系列实际问题。针对每一个具体问题，我描述了它们的情况，并把当时可用的信息提供给分析者。在这个基础上，转到分析数据包的过程，描述捕获适当数据包的方法以及分析过程的每个步骤。分析完成后，我将提供一个完整的问题解决方案，或是帮你指出可能的解决方法，并概述从中吸取的经验教训。

自始至终，要记住分析是一个非常动态的过程。并且，我用来分析每一个场景的方法可能跟你用的不一样。每个人可以用不同的方法来分析。但最重要的是，分析的最终结果能够解决一个问题，或提供一次学习经验。另外，本章讨论的大部分问题，即使不用数据包嗅探器也可以解决。当我初次学习分析数据包时，我发现反常规地使用数据包分析技术来查看典型问题有很多好处，这也是我给你介绍这些场景的原因。

## 8.1 数据包层面的社交网络

首先，我们来看两个流行的社交网站的流量：Twitter 和 Facebook。我们将查看每个服务的身份认证过程，看看这两个非常相似的功能如何使用不同的方法来执行相同的任务。我们还将着眼于每个服务的主要功能如何工作，以便更好地理解我们日常活动中产生的流量<sup>1</sup>。

### 8.1.1 捕获 Twitter 流量

不管你是使用 Twitter 来了解科技领域最新信息，还是只用 Twitter 来抱怨你的女朋友，它毫无疑问是因特网上最常用的服务之一。在这个场景中，你会从 `twitter_login.pcap` 文件里找到 Twitter 流量的捕获记录。

#### 注意

网站经常改变它们的代码。因此，如果你试图在接下来的几节中重建捕获记录，也许会发现结果跟这里有所差异。

#### 1. Twitter 的登录过程

每次讲授数据包分析技术时，我让学生做的第一件事就是登录他们正常使用的网站并捕获登录过程中的流量。这样做有两个目的：它通常能使学生接触到更多数据包，而且允许学生寻找网络线路上传送的明文密码，从而发现日常活动中的不安全因素。

幸好 Twitter 的身份认证过程并非完全不安全。正如你在图 8-1 中看到的，起

<sup>1</sup> 虽然在中国大陆地区无法直接访问到 Twitter 和 Facebook，但是本章案例都提供了捕获文件，因此读者仍然可以重现和学习本章的所有案例。并建议读者能够举一反三，分析下新浪微博、腾讯微博、人人网等国内主流社交网站登录过程是否安全，你应该会有非常有趣的发现。——译者注

始的 3 个数据包构成了本地设备 (172.16.16.128) ①与远程服务器 (168.143.162.68) ②之间的 TCP 握手。远程服务器在 443 端口 ③监听我们的连接，这一般采用 SSL over HTTP 协议，通常称作 HTTPS 协议，是一种安全的数据传输方式。仅基于这些信息，我们就可以假设这是 SSL 流量。

| No. | Time     | Source         | ① Destination  | ② Protocol | Info   | ③ |
|-----|----------|----------------|----------------|------------|--|---|
| 1   | 0.000000 | 172.16.16.128  | 168.143.162.68 | TCP        | 4669 > 443 [SYN] Seq=4164864060 Win=8192 Len=0 MSS=1460                      |   |
| 2   | 0.072728 | 168.143.162.68 | 172.16.16.128  | TCP        | 443 > 4669 [SYN, ACK] Seq=1150193371 Ack=4164864061 Win=16200 Len=0 MSS=1460 |   |
| 3   | 0.000101 | 172.16.16.128  | 168.143.162.68 | TCP        | 4669 > 443 [ACK] Seq=4164864061 Ack=1150193372 Win=16872 Len=0               |   |

图 8-1 连接到 443 端口的握手

紧随握手之后的数据包是 SSL 加密握手的一部分。SSL 依赖于密钥——用来加解密双方通信的字符串。这次握手实现了主机间密钥的正式传输，以及连接和加密特性的协商过程。一旦握手完成，安全的数据传输就开始了。

为了找到处理数据交换的加密数据包，在 Packet Details 面板的 Info 列寻找被识别成 Application Data 的数据包。如图 8-2 所示，展开其中任意一个数据包的 SSL 部分，将会显示 Encrypted Application Data 域①，包含了不可读的加密数据，如图 8-2 所示，展示了登录时用户名和密码的传输过程。

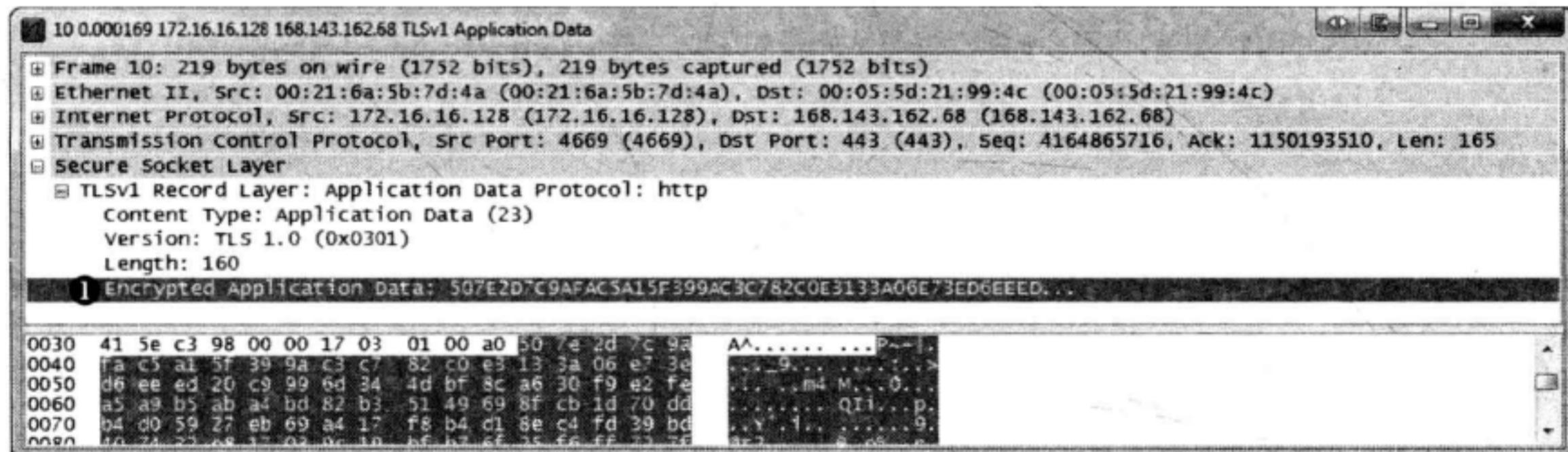


图 8-2 传输的加密凭证

身份认证过程将一直持续，直到第 16 个数据包用 FIN/ACK 标志拆除连接时才宣告结束。身份认证之后，我们猜测浏览器将被重定向到 Twitter 主页，实际上的确如此。如图 8-3 所示，数据包 19、21 和 22 是向同一个远程服务器 (168.143.162.68) 建立新连接的握手过程，但这次是在 80 端口而不是 443 端口①。握手完成之后，我们在数据包 23 中看到了指向 Web 服务器根目录 (/) 的 HTTP GET 请求②。服务器在数据包 24 中确认了这个请求③，然后开始在接下来的几个数据包中传输数据。数据包 41 的内容标记了与此次 GET 请求有关的数据传输结束。

| No. | Time     | Source         | Destination    | Protocol | Info  |
|-----|----------|----------------|----------------|----------|---|
| 19  | 0.000117 | 172.16.16.128  | 168.143.162.68 | TCP      | 4670 > 80 [SYN] Seq=3871493748 Win=8192 Len=0 MSS=1460                      |
| 21  | 0.000063 | 168.143.162.68 | 172.16.16.128  | TCP      | 80 > 4670 [SYN, ACK] Seq=2866679388 Ack=3871493749 Win=18200 Len=0 MSS=1406 |
| 22  | 0.000063 | 172.16.16.128  | 168.143.162.68 | TCP      | 4670 > 80 [ACK] Seq=3871493749 Ack=2866679389 Win=16872 Len=0               |
| 23  | 0.000171 | 172.16.16.128  | 168.143.162.68 | HTTP     | GET / HTTP/1.1  |
| 24  | 0.080775 | 168.143.162.68 | 172.16.16.128  | TCP      | 80 > 4670 [ACK] Seq=2866679389 Ack=3871495149 Win=8400 Len=0                |

图 8-3 身份认证完成之后，指向 Twitter 主页根目录（/）的 GET 请求

捕获记录文件的剩余部分中还有几个 GET 请求，用以检查链接到主页的图像和其他文件。

## 2. 用 Tweet 发送数据

登录之后，下一步是告诉这个世界你在想什么。因为我正在撰写本书，我写道，“This is a tweet for Practical Packet Analysis, second edition”，然后将发表这条 tweet 的流量捕获到 twitter\_tweet.pcap 文件里。

这个文件在我提交 tweet 的那一刻起开始捕获记录。它从我们的本地工作站 172.16.16.134 与远程地址 168.143.162.100 间的握手开始。捕获记录的第 4 个和第 5 个数据包里包含了从客户端发送到服务器的一个 HTTP 数据包。Wireshark 已经合并了这两个数据包的数据，并将它放到数据包 5 的 Packet Details 面板中以便查看。

如图 8-4 所示，展开第 5 个数据包的 Packet Details 面板中的 HTTP 部分，以查看 HTTP 头部信息。你会看见浏览器对 URL/status/update 使用了 POST 方法❶。我们知道这的确是来自 tweet 的数据包，因为 Host 域包含了 twitter.com 这个值❷。

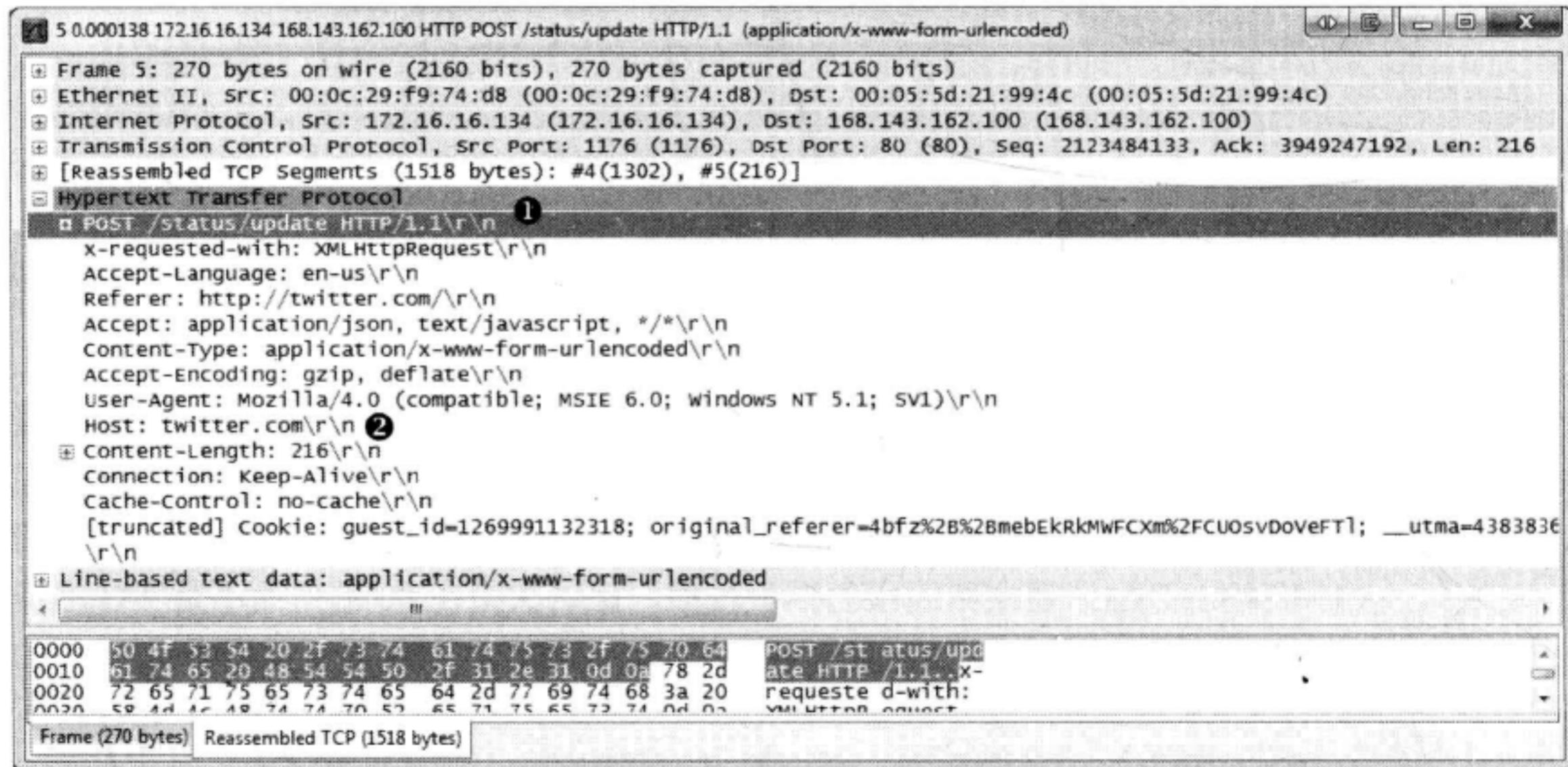


图 8-4 更新 Twitter 的 HTTP POST

注意图 8-5 中包含在数据包的 Line-based Text Data 域中的信息❶。当分析这个数据时,你会看见一个叫做 Authenticity Token 的域,紧随其后的是一个 URL 的 status 域,包含这个值:

This+is+a+tweet+for+practical+packet+analysis%2c+second+edition

这个 status 域的值就是我提交的未加密的 tweet。

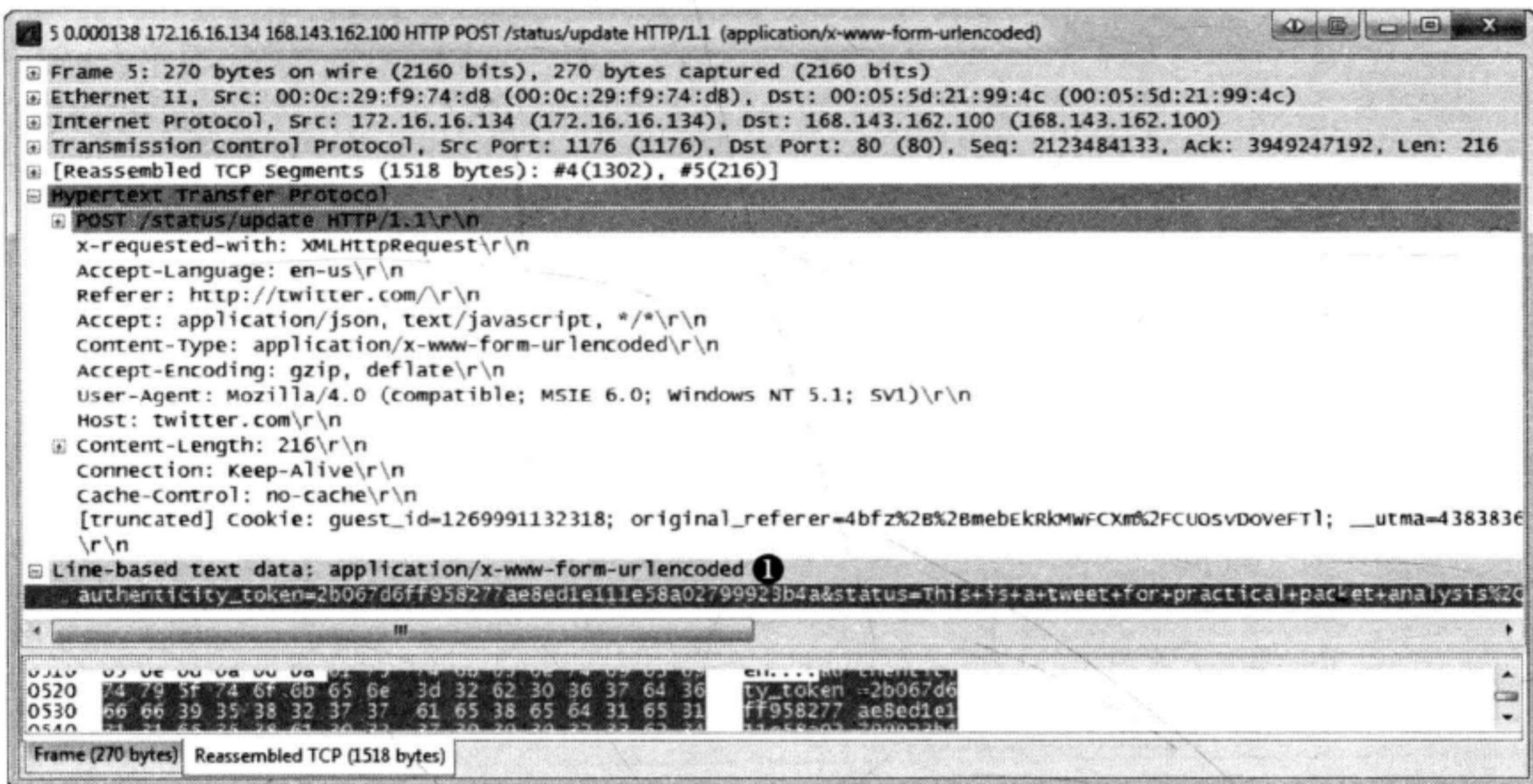


图 8-5 明文显示的 tweet

这就有一个小小的安全问题,因为有些人希望保护他们的 tweet,不想被任何人看见。这并不意味着任何人都能读取这条 tweet,但是同一个网络中的用户可以拦截这些流量并清楚地看到 tweet 中包含的内容。

### 3. Twitter 直达消息

现在我们考虑一个具有安全内涵的场景:Twitter 直达消息,该场景允许用户分享被认为是私密的消息。这个 `twitter_dm.pcap` 文件是一条 Twitter 直达消息的捕获记录。如图 8-6 所示,直达消息确切地说并不是私密的。

图 8-6 的数据包 7 显示了内容仍然是明文发送的。很显然它同样处在 Line-based Text Data 域中❶,与之前在捕获记录中看到的一样。

我们从这里学到的关于 Twitter 的知识未必有多么了不起,但是它应该能让你重新考虑是否敢在不安全网络上通过私密 Twitter 消息来发送敏感数据。

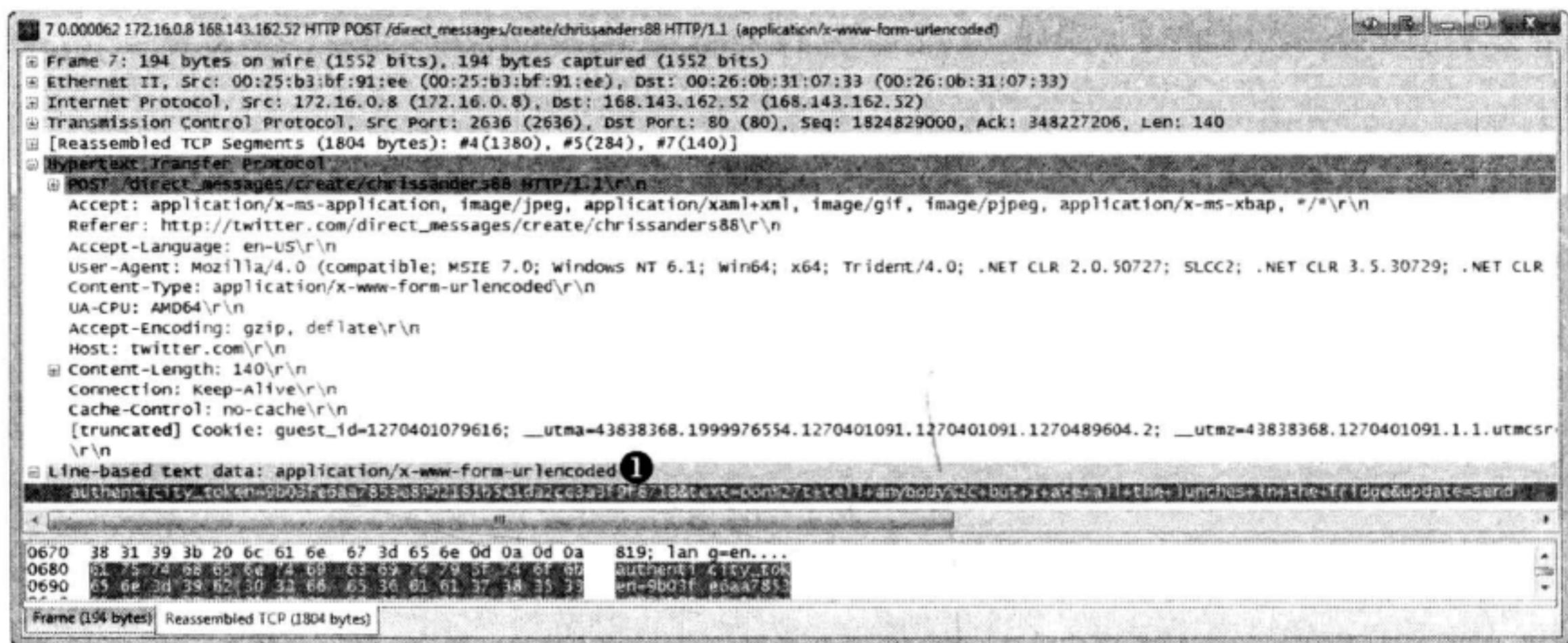


图 8-6 明文显示的直达消息

## 8.1.2 捕获 Facebook 流量

刷完 Twitter 之后，我喜欢登录 Facebook，看看朋友们在忙些什么，以便我能通过他们的感受来体验生活。现在，让我们使用 Wireshark 来捕获并分析 Facebook 流量。

### 1. Facebook 登录过程

我们从 `facebook_login.pcap` 捕获的登录过程开始。这份捕获记录始于提交凭证的那一刻，如图 8-7 所示。与 Twitter 登录过程类似，我们看到了 443 端口❶上的 TCP 握手过程。我们的 172.16.0.122 工作站❷与 69.63.180.173❸初始化通信，服务器处理 Facebook 身份认证过程。握手完成后，开始 SSL 握手❹，并提交登录凭证。

| No. | Time     | Source        | Destination   | Protocol | Info  |
|-----|----------|---------------|---------------|----------|---|
| 1   | 0.000000 | 172.16.0.122  | 69.63.180.173 | TCP      | Syn=2917405622 Win=5840 Len=0 MSS=1460 TSV=301989713 TSER=0 WS=6  |
| 2   | 0.000000 | 69.63.180.173 | 172.16.0.122  | TCP      | 443 > 54595 [SYN, ACK] Seq=2894038305 Ack=2917405623 Win=4140 Len=0 MSS=1360 WS=0 TSV=3479125768 TSER=301989713 |
| 3   | 0.000033 | 172.16.0.122  | 69.63.180.173 | TCP      | 54595 > 443 [ACK] Seq=2917405623 Ack=2894038305 Win=92 Len=0 TSV=301989735 TSER=3479125768                      |
| 4   | 0.000343 | 172.16.0.122  | 69.63.180.173 | TLSv1    | Client Hello ❹  |
| 5   | 0.089522 | 69.63.180.173 | 172.16.0.122  | TLSv1    | Server Hello, Certificate, Server Hello Done  |
| 6   | 0.000031 | 172.16.0.122  | 69.63.180.173 | TCP      | 54595 > 443 [ACK] Seq=2917405792 Ack=2894039242 Win=121 Len=0 TSV=301989758 TSER=3479125858                     |
| 7   | 0.002846 | 172.16.0.122  | 69.63.180.173 | TLSv1    | Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message  |
| 8   | 0.090444 | 69.63.180.173 | 172.16.0.122  | TLSv1    | Change Cipher Spec, Encrypted Handshake Message   |
| 9   | 0.000533 | 172.16.0.122  | 69.63.180.173 | TLSv1    | Application Data  |
| 10  | 0.189619 | 69.63.180.173 | 172.16.0.122  | TCP      | 443 > 54595 [ACK] Seq=2894039285 Ack=2917406956 Win=5473 Len=0 TSV=3479126142 TSER=301989781                    |
| 11  | 0.073201 | 69.63.180.173 | 172.16.0.122  | TLSv1    | Application Data  |

图 8-7 登录凭证通过 HTTPS 安全传输

与 Twitter 身份认证过程不同的是，在传输登录凭证后，我们并没有马上看到结束身份认证连接。相反，我们在数据包 12 的 HTTP 头部看到一个指向/home.php 的 GET 请求，如图 8-8 中的突出显示❶。

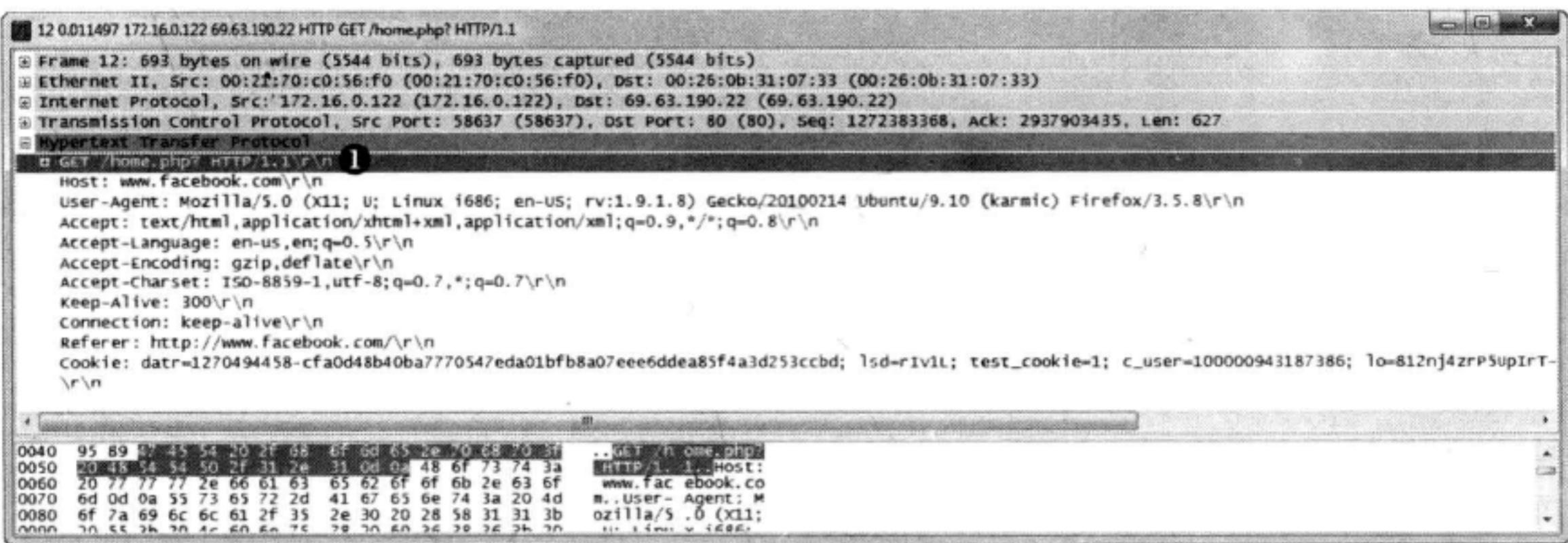


图 8-8 身份认证之后，产生了指向/home.php 的 GET 请求

如图 8-9 的捕获文件尾部的数据包 64 所示①，用于身份认证的连接在传送 home.php 的内容后才拆除。首先，80 端口上的 HTTP 连接被拆除（数据包 62）②，然后 443 端口上 HTTPS 连接被拆除。

| No.           | Time                     | Source        | Destination | Protocol | Info  |
|---------------|--------------------------|---------------|-------------|----------|---|
| ②             | 62 300.39816172.16.0.122 | 69.63.190.22  |             | TCP      | 58637 > 80 [FIN, ACK] Seq=1272384963 Ack=2937930926 Win=1002 Len=0 TSV=302065222 TSER=3479159094  |
| 63 0.000388   | 172.16.0.122             | 69.63.180.173 |             | TLSv1    | Encrypted Alert   |
| ① 64 0.000027 | 172.16.0.122             | 69.63.180.173 |             | TCP      | 54595 > 443 [FIN, ACK] Seq=2917406979 Ack=2894040466 Win=158 Len=0 TSV=302065222 TSER=3479126214  |
| 65 0.036439   | 69.63.190.22             | 172.16.0.122  |             | TCP      | 80 > 58637 [ACK] Seq=2937930926 Ack=1272384964 Win=7233 Len=0 TSV=3479459532 TSER=302065222       |
| 66 0.000082   | 69.63.190.22             | 172.16.0.122  |             | TCP      | 80 > 58637 [FIN, ACK] Seq=2937930926 Ack=1272384964 Win=7233 Len=0 TSV=3479459532 TSER=302065222  |
| 67 0.000023   | 172.16.0.122             | 69.63.190.22  |             | TCP      | 58637 > 80 [ACK] Seq=1272384964 Ack=2937930927 Win=1002 Len=0 TSV=302065232 TSER=3479459532       |
| 68 0.052635   | 69.63.180.173            | 172.16.0.122  |             | TCP      | 443 > 54505 [FIN, ACK] Seq=2894040466 Ack=2917406972 Win=5496 Len=0 TSV=3479427810 TSER=302065222 |
| 69 0.000078   | 172.16.0.122             | 69.63.180.173 |             | TCP      | 54595 > 443 [ACK] Seq=2917406980 Ack=2894040467 Win=158 Len=0 TSV=302065245 TSER=3479427810       |
| 70 0.459231   | 172.16.0.122             | 69.63.180.173 |             | TCP      | 54595 > 443 [FIN, ACK] Seq=2917406979 Ack=2894040467 Win=158 Len=0 TSV=302065360 TSER=3479427810  |
| 71 0.088948   | 69.63.180.173            | 172.16.0.122  |             | TCP      | 443 > 54595 [ACK] Seq=2894040467 Ack=2917406980 Win=5496 Len=0 TSV=3479428358 TSER=302065360      |

图 8-9 HTTP 连接和 HTTPS 连接相继被拆除

## 2. Facebook 私信

既然我们已经查看了 Facebook 的登录认证过程，让我们看一看它是如何处理私信的。这个 facebook\_message.pcap 文件包含从我的账户向另一个 Facebook 账户发送消息过程中捕获的数据包。当你打开这个文件时，一些数据包或许会让你感到惊讶。

开头两个数据包构成了负责发送消息的 HTTP 流量。当你展开数据包 2 的 HTTP 头部时，如图 8-10 所示，你会看到浏览器对一个相当长的 URL 字符串使用了 POST 方法①。如你所见，这个字符串包含了对 AJAX 的引用。

异步 JavaScript 和 XML（Asynchronous JavaScript and XML，简称 AJAX）是一种在客户端实现的、用于创建交互式 Web 应用的、在后台从服务器取回信息的方法。也许你会想，在私信送达客户端的浏览器后，这个会话会被重定向到另一个页面（如 Twitter 直达信息所做的），但是这并没有发生。在这个案例中，使用 AJAX 可能意味着消息是从某些类型的交互式弹出窗口中发出的，而不是来自独立的页

面，这意味着没有必要做重定向或刷新。这是用 AJAX 实现的优点之一。

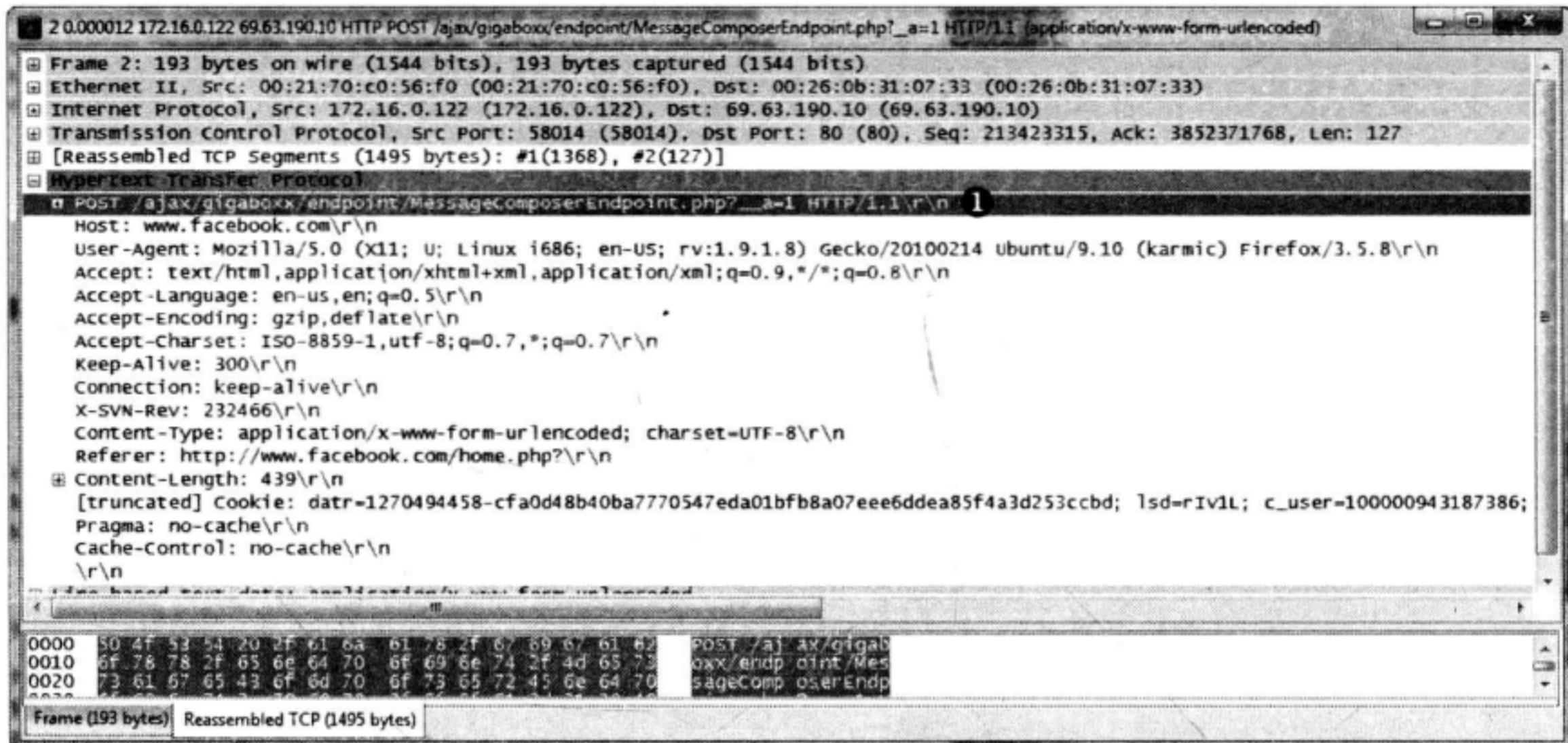


图 8-10 这个 HTTP POST 引用了 AJAX

- 如图 8-11 所示，通过展开数据包 2 的 Line-based Text Data 部分，你可以查看到私信内容。与 Twitter 一样，看来 Facebook 私信也是没有加密就发送的。

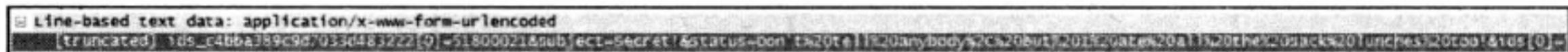


图 8-11 这条 Facebook 私信的内容可以明文查看

### 8.1.3 比较 Twitter 和 Facebook 的方法

你现在已经看到 Twitter 和 Facebook 这两个 Web 服务的身份认证和消息发送方法了。它们分别采用了不同的实现手段。程序员可能会说 Twitter 的身份认证方法比较好，因为它更快速、高效。安全研究人员则可能会说 Facebook 的方法更好，因为它保证所有内容都能成功传递。并且，由于在关闭身份认证连接之前不需要额外的认证，使中间人攻击（Man-in-the-middle，简称 MITM，中间人攻击是指恶意用户拦截通信双方流量的一种攻击手段）更难得逞。在现实中，这两个网站的认证方法差别很小，但却表明了当两个程序员着手为同样的任务编写程序时，差异是存在的。

尽管这很有意思，但此次分析的关键并不在于弄清楚 Twitter 和 Facebook 究竟是如何工作的，而只是想让你接触到可以比较和对比的流量。如果你需要

检查为什么相似服务的运转方式与预想不符，或者运转较慢，那么这个案例提供了一个很好的参考。

## 8.2 捕获 ESPN.com 流量

刷完社交网络后，下一步我就要看看最新的新闻头条以及运动赛事比分。一些网站有助于这有趣的分析，<http://www.espn.com> 便是其中之一。我已经把浏览 ESPN 网站的流量捕获到 `http_espn.pcap` 文件中了。

这个捕获文件包含很多数据包——确切地说是 956 个。但若是手工滚动分析整个文件以尝试辨别每个连接和异常，这数据量对我们而言就太大了。因此，我们将使用 Wireshark 的一些分析功能让该过程变得更加容易。

### 8.2.1 使用会话窗口

ESPN 主页包含了太多的花哨设计，因此很容易理解为什么它会用近千个数据包把数据传输给我们。不论何时当你要传输大量数据时，知道数据的来源都是很有用的，尤其是它究竟有一个来源还是多个来源。我们可以使用 Wireshark 的会话窗口（Statistics->Conversations）来实现。

图 8-12 显示了包含 14 个 IP 会话、25 个 TCP 连接、14 个 UDP 会话的例子，

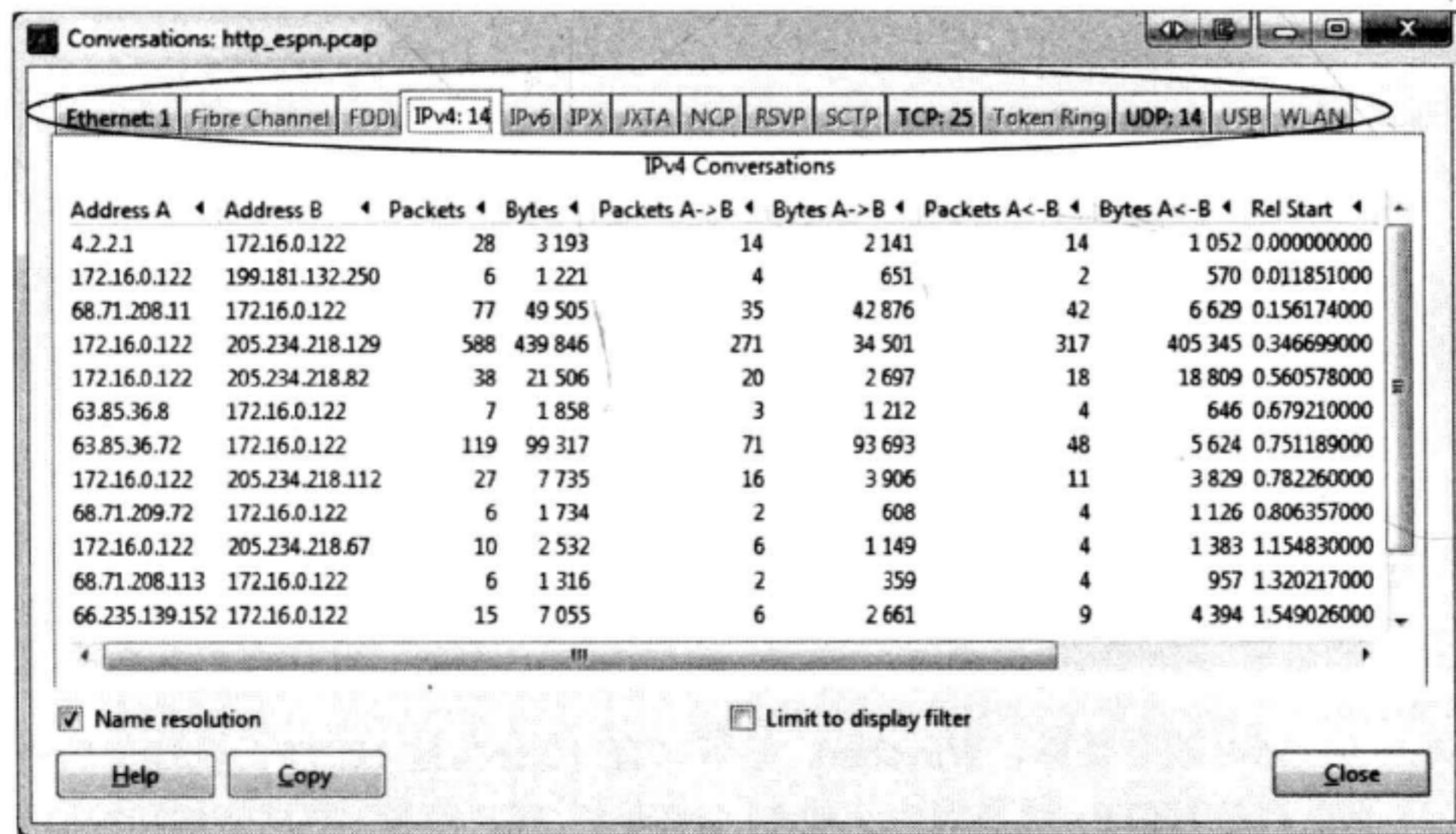


图 8-12 Conversations 窗口显示了多个不同的连接

所有会话都详细显示在会话主窗口中。对于一个站点来说，这算多的了。

## 8.2.2 使用协议分层统计窗口

为了更好地观察情形，我们可以看看这些 TCP 和 UDP 连接上使用的应用层协议。如图 8-13 所示，打开协议分层统计窗口（Statistics → Protocol Hierarchy）。

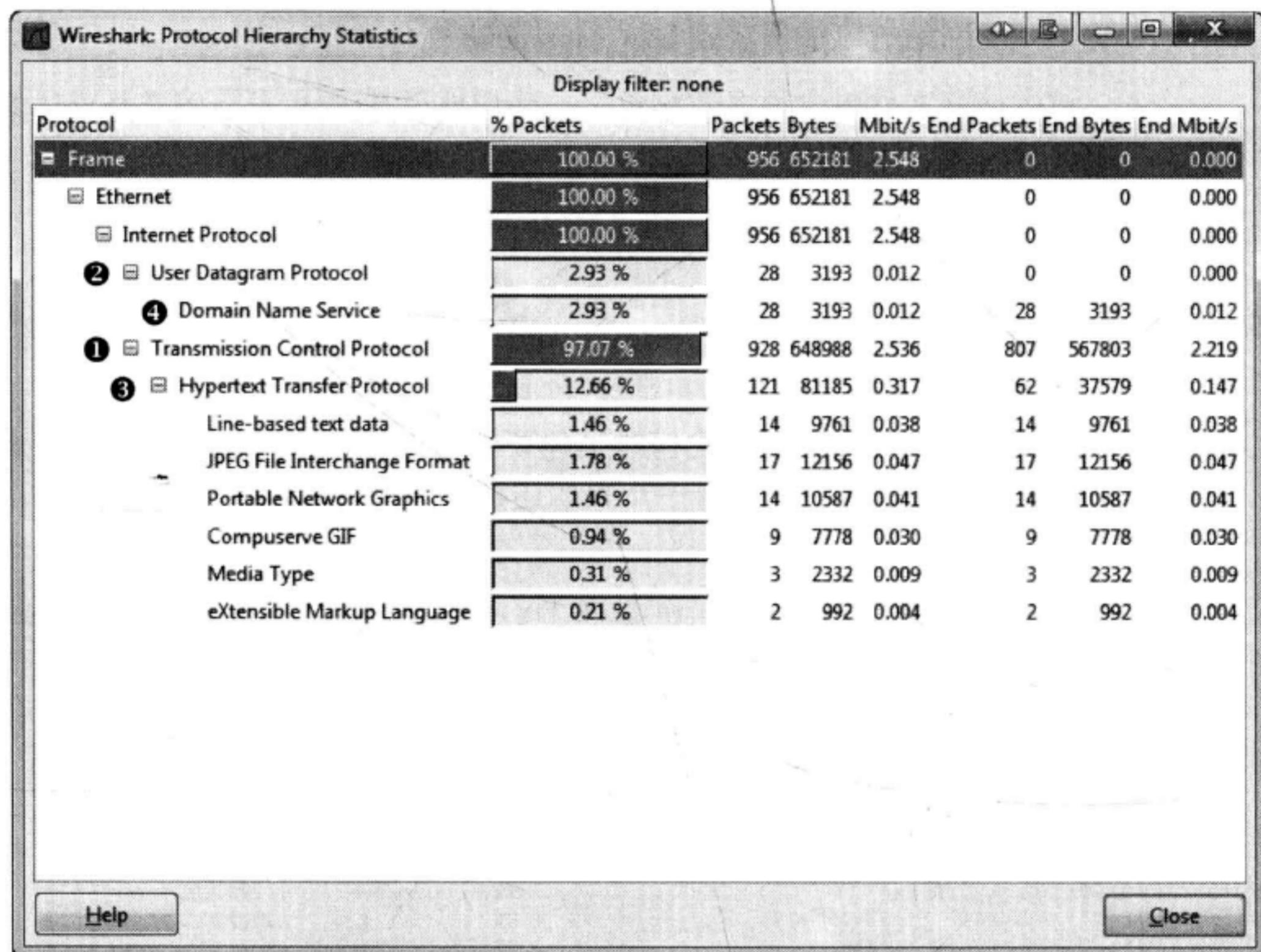


图 8-13 协议分层统计窗口显示了这个捕获记录内的协议分布

你可以看到，TCP 在捕获记录中占了 97.07% 的数据包①，UDP 占了剩下的 2.93%②。果不其然，所有 TCP 流量都是 HTTP③，甚至可以进一步按照 HTTP 传输的文件类型进行细分。

你可能会疑惑，Wireshark 显示只有 12.66% 是 HTTP，我却说所有的 TCP 流量都是 HTTP，这是因为其他 84.41% 是纯 TCP 流量（数据传输和控制数据包）。UDP 标题下的条目显示，图中所有 UDP 流量都是 DNS④。

单独依据这些信息，我们可以做些归纳。首先，我们从前面的例子知道，DNS 事务很少，所以 28 个 DNS 数据包（如图 8-13 所示，列在 Domain Name Service 条目旁边的 Packets 列）意味着我们有 14 个 DNS 事务。我们将数据包总数除以 2，表示成对的请求与响应，就得到了这个数字。如果你往会话窗口的 UDP 标题下面看，那里的确显示 14 个会话，代表 14 个 DNS 事务，这验证了我们的猜想。

然而 DNS 查询并不会独自发生，而捕获记录中的其他流量就只有 HTTP。这告诉我们，ESPN 网站的 HTML 代码有可能通过域名引用了其他域或子域，从而导致执行多个查询。

让我们看看是否能找到一些证据来证实我们的推测。

### 8.2.3 查看 DNS 流量

查看 DNS 流量的一个简单方法是创建一个过滤器。如图 8-14 所示，在 Wireshark 窗口内的 filter 栏里输入 dns，将显示所有的 DNS 流量。

图 8-14 中的 DNS 流量全部以查询/响应的形式出现。为了更好地查看被查询的域名，我们创建一个只显示查询数据包的过滤器。在 Packet List 面板中选择一个查询，然后在 Packet Details 面板中展开数据包的 DNS 头部，然后右击 Flags: 0x0100 (Standard query) 域，移动到 **Apply as Filter**，选择 **Selected**。

| No. | Time     | Source       | Destination  | Protocol | Info  |
|-----|----------|--------------|--------------|----------|---|
| 1   | 0.000000 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A www.espn.com   |
| 2   | 0.011663 | 4.2.2.1      | 172.16.0.122 | DNS      | Standard query response A 199.181.132.230   |
| 3   | 0.144300 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A espn.go.com  |
| 10  | 0.155758 | 4.2.2.1      | 172.16.0.122 | DNS      | Standard query response A 68.71.208.11  |
| 21  | 0.326066 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A a.espncdn.com  |
| 22  | 0.337568 | 4.2.2.1      | 172.16.0.122 | DNS      | Standard query CNAME a.espncdn.com.edgesuite.net CNAME a1831.g.akamai.net A 205.234.218.129 A 205.234.218.82          |
| 224 | 0.542076 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A a1.espncdn.com   |
| 225 | 0.549050 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A a2.espncdn.com   |
| 226 | 0.553531 | 4.2.2.1      | 172.16.0.122 | DNS      | Standard query response CNAME a.espncdn.com.edgesuite.net CNAME a1831.g.akamai.net A 205.234.218.82 A 205.234.218.129 |
| 227 | 0.560189 | 4.2.2.1      | 172.16.0.122 | DNS      | Standard query response CNAME a.espncdn.com.edgesuite.net CNAME a1831.g.akamai.net A 205.234.218.129 A 205.234.218.82 |
| 389 | 0.650057 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A www.masters.com  |
| 417 | 0.679056 | 4.2.2.1      | 172.16.0.122 | DNS      | Standard query response CNAME www.masters.com.edgesuite.net CNAME a1073.g.akamai.net A 63.85.36.8 A 63.85.36.49       |
| 425 | 0.737456 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A adsat1.espn.go.com   |
| 426 | 0.738032 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A log.go.com   |
| 427 | 0.749732 | 4.2.2.1      | 172.16.0.122 | DNS      | Standard query response CNAME adimages.go.com.edgesuite.net CNAME a1412.g.akamai.net A 63.85.36.72 A 63.85.36.58      |
| 429 | 0.758282 | 172.16.0.122 | 4.2.2.1      | DNS      | Standard query A assets.espn.go.com   |

图 8-14 DNS 流量以标准的查询/响应形式出现

这将激活过滤器 dns.flags == 0x0100，使窗口上只显示查询数据包，这样就更容易读取我们正在分析的记录。而且，如图 8-14 所示，那里的确有 14 个独立的请求（每个数据包表示一个请求），所有的域名好像都与 ESPN 有关，或者与它主页显示的内容有关。

## 8.2.4 查看 HTTP 请求

最后，我们可以通过查看 HTTP 请求来验证这些查询的来源。为此，选择 **Statistics -> HTTP**，选择 **Requests**，然后单击 **Create Stat**（要确定做这步之前你已经清除了刚才创建的过滤器）。

图 8-15 显示了 HTTP Requests 窗口。显示在这里的 14 个连接（每一行代表一个与特定域名的连接）解释了 DNS 请求代表的所有域名。

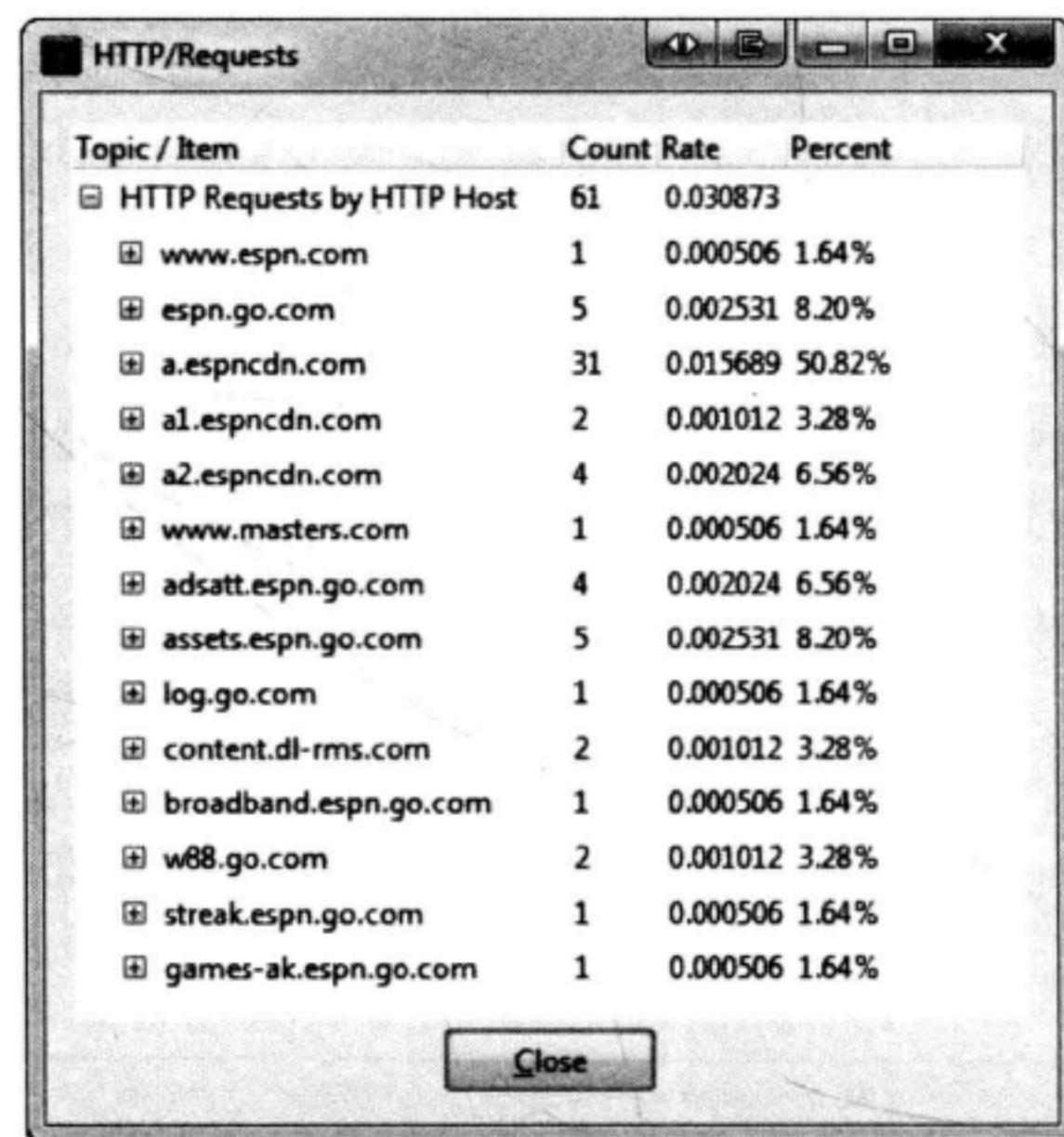


图 8-15 这个窗口汇总了全部 HTTP 请求，显示了所访问的域

看到这么多连接，我们很可能会想查一下，这个高度复杂的过程到底花了多少时间。最简单的方法是查看这些流量的概述信息。为此，选择 **Statistics -> Summary**。如图 8-16 所示，Summary 窗口显示了整个过程在大约 2s 内发生❶，这是完全可以接受的。

太神奇了！我们查看一个网页的简单请求，竟然被分解成面向 14 个独立的域和子域的连接，与种种不同的服务器做了交互，而整个过程只在 2s 内完成！在访问你最喜欢的网站时，像这样捕获流量把它分解，是一种有趣的练习。你只有去查看这些报文，才会知道你的数据究竟是从哪儿来的。

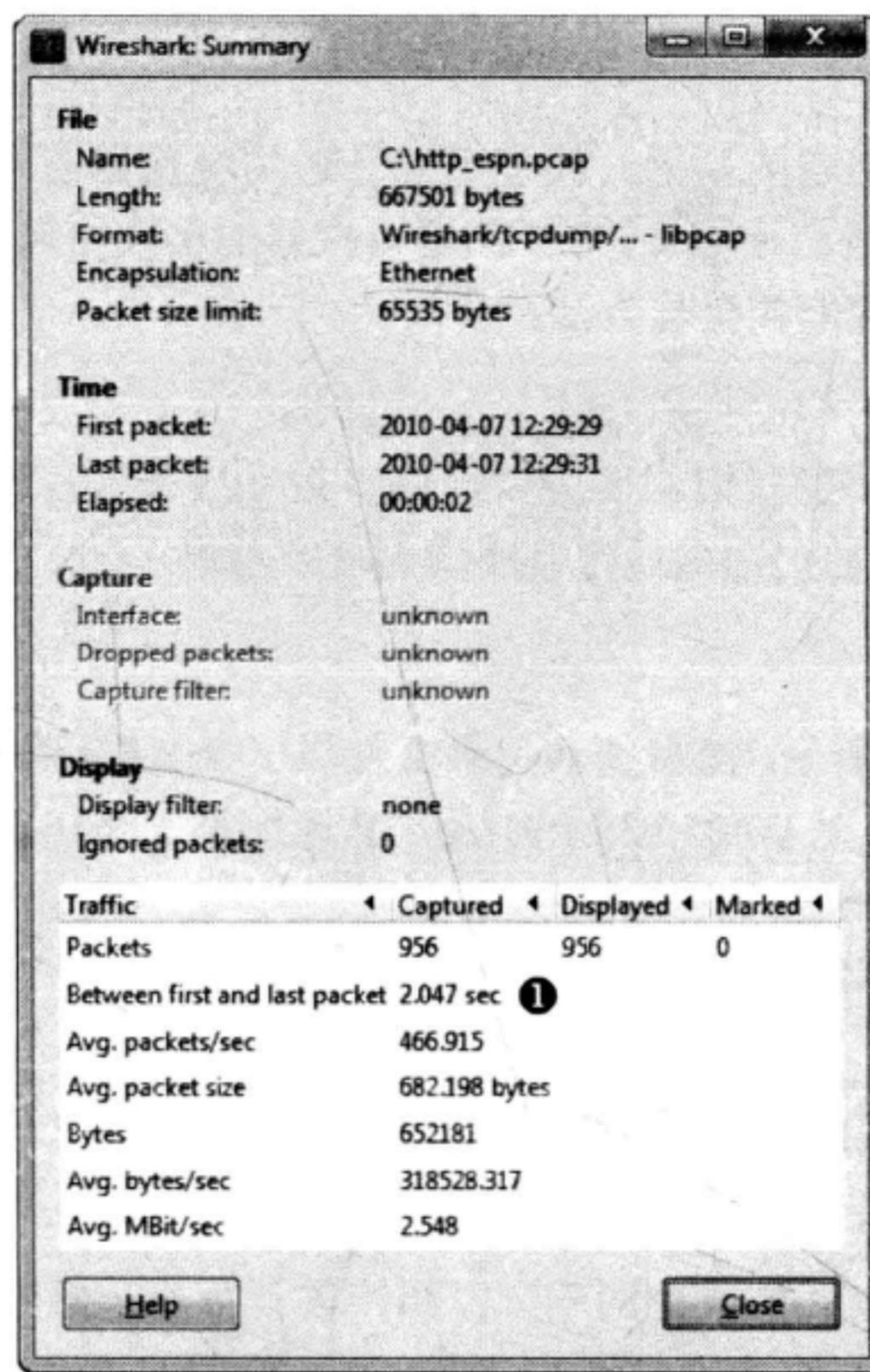


图 8-16 文件的 Summary 窗口显示全过程在 2s 内完成

## 8.3 现实世界问题

现在,我们将讨论存在问题的流量的例子。让我们来看看各种各样的 Internet 接入问题,以及类似打印机故障、从分支机构连接等经典问题。

### 8.3.1 无法访问 Internet: 配置问题

第一个问题的场景相当简单: 用户不能访问 Internet。我们已经确认该用户可以访问所有内网资源,包括其他工作站的共享内容,以及运行在本地服务器上的应用程序。

这个网络架构非常简单,所有客户机和服务器都连接到一系列的简单交换机上。Internet 连接由一个路由器处理,作为默认网关,IP 地址信息由 DHCP 提供。这种情景在小型办公室中非常常见。

## 1. 偷听线路

为了找出问题的原因，我们可以一边用嗅探器监听线路，一边让用户尝试浏览 Internet。我们使用 2.5 节“部署嗅探器的实践指南”中的信息（见图 2-15），来决定放置嗅探器的最佳方法。

我们网络上的交换机不支持端口镜像。为了完成测试，我们已经不可避免妨碍了用户，所以我们假设可以使他再次下线（这是说，使用网络分流器是监听线路的最佳办法）。最后得到捕获记录文件 nowebaccess1.pcap。

## 2. 分析

如图 8-17 所示，流量捕获记录文件以一个 ARP 请求和响应开始。用户的计算机的 MAC 地址是 00:05:b3:bf:91:ee，IP 地址是 172.16.0.8。在数据包 1 中，用户的计算机发送一个 ARP 广播数据包给网络上的所有计算机，试图得到默认网关 172.16.0.10 的 MAC 地址。

| No. | Time     | Source            | Destination       | Protocol | Info                                 |
|-----|----------|-------------------|-------------------|----------|--------------------------------------|
| 1   | 0.000000 | 00:25:b3:bf:91:ee | ff:ff:ff:ff:ff:ff | ARP      | who has 172.16.0.10? tell 172.16.0.8 |
| 2   | 0.000090 | 00:24:81:a1:f6:79 | 00:25:b3:bf:91:ee | ARP      | 172.16.0.10 is at 00:24:81:a1:f6:79  |

图 8-17 针对计算机默认网关的 ARP 请求和响应

根据数据包 2 中收到的响应，用户的计算机了解到 172.16.0.10 的 MAC 地址是 00:24:81:a1:f6:79。收到这个响应后，计算机就有了到达网关的路由，而网关应该可以引导它接入 Internet。

ARP 响应之后，计算机一定会在数据包 3 中请求将网站的域名解析成 IP 地址。如图 8-18 所示，计算机发送一个 DNS 查询数据包到它的首选 DNS 服务器 4.2.2.2①。

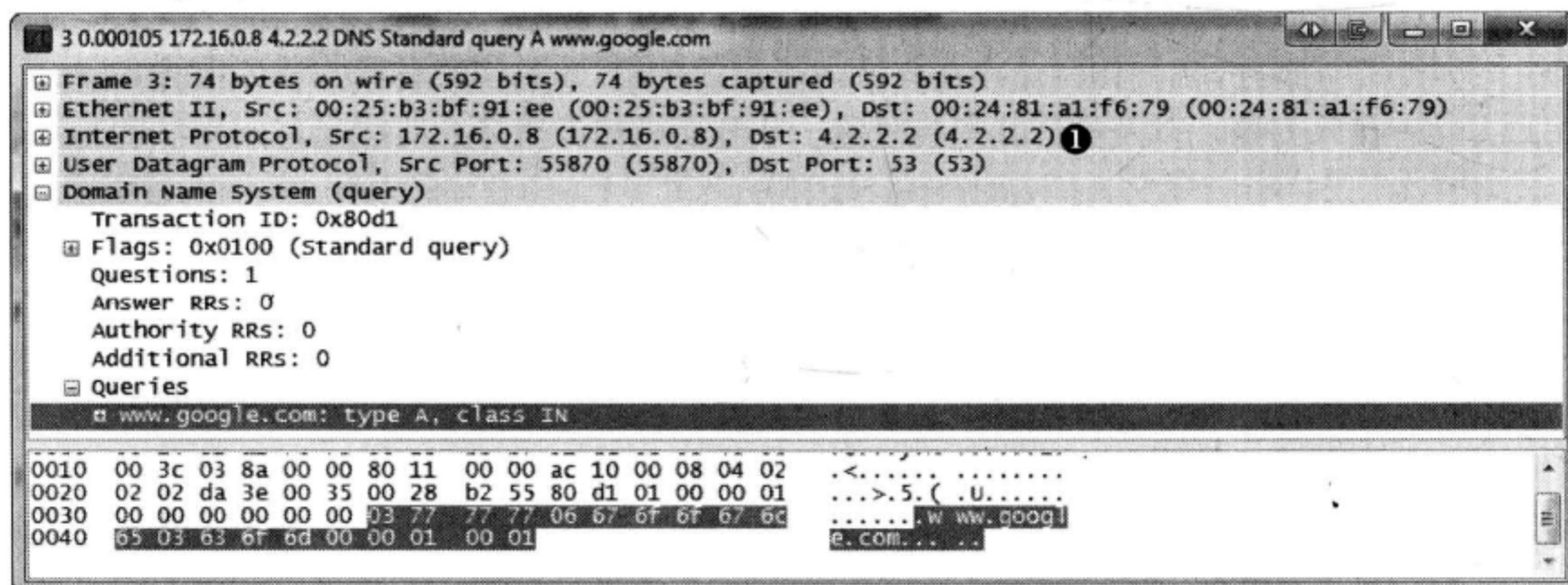


图 8-18 发送到 4.2.2.2 的 DNS 查询

正常情况下，DNS 服务器会很快响应 DNS 查询，但在这个例子中并非如此。我们没有看到任何响应，却发现同样的 DNS 查询再次发送到不同的目的地址。如图 8-19 所示，在数据包 4 中，第二个 DNS 查询被发送到预先配置好的备用 DNS 服务器 4.2.2.1①。

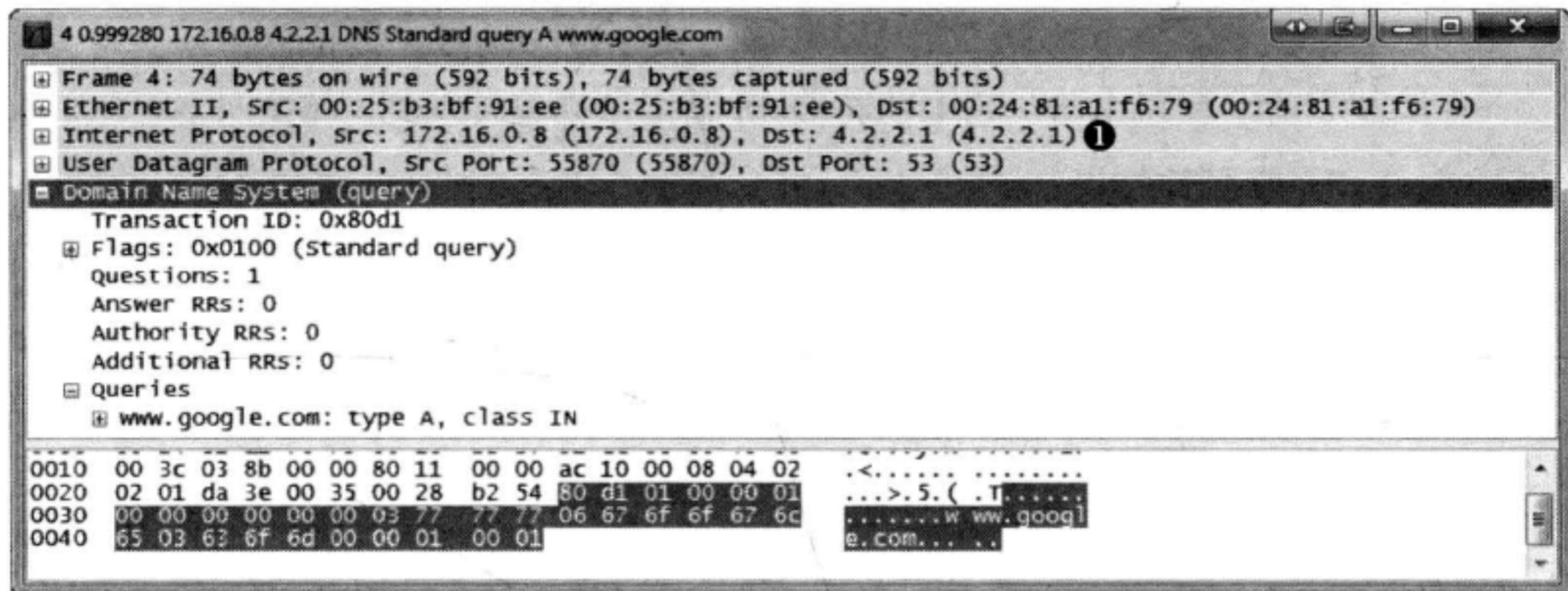


图 8-19 发送到 4.2.2.1 的第二个 DNS 查询

由于计算机仍然没有从 DNS 服务器收到响应，1s 后，查询被再次发送到 4.2.2.2。如图 8-20 所示，这个过程不断重复，在接下来的几秒钟，交替向配置好的首选 DNS 服务器①和备用 DNS 服务器②发送请求。整个过程大概花了 8s③，这正是用户的 Internet 浏览器报告该页无法访问之前所花的时间。

| No. | Time     | Source            | Destination       | Protocol | Info                                 |
|-----|----------|-------------------|-------------------|----------|--------------------------------------|
| 1   | 0.000000 | 00:25:b3:bf:91:ee | ff:ff:ff:ff:ff:ff | ARP      | who has 172.16.0.10? Tell 172.16.0.8 |
| 2   | 0.000090 | 00:24:81:a1:f6:79 | 00:25:b3:bf:91:ee | ARP      | 172.16.0.10 is at 00:24:81:a1:f6:79  |
| 3   | 0.000105 | 172.16.0.8        | 4.2.2.2 ①         | DNS      | Standard query A www.google.com      |
| 4   | 0.999280 | 172.16.0.8        | 4.2.2.1 ②         | DNS      | Standard query A www.google.com      |
| 5   | 1.999279 | 172.16.0.8        | 4.2.2.2           | DNS      | Standard query A www.google.com      |
| 6   | 3.999372 | 172.16.0.8        | 4.2.2.1           | DNS      | Standard query A www.google.com      |
| 7   | 3.999393 | 172.16.0.8        | 4.2.2.2           | DNS      | Standard query A www.google.com      |
| 8   | 7.999627 | 172.16.0.8        | 4.2.2.1           | DNS      | Standard query A www.google.com      |
| 9   | 7.999648 | 172.16.0.8        | 4.2.2.2           | DNS      | Standard query A www.google.com      |

图 8-20 直到通信结束，重复的 DNS 查询才停止

基于前面看到的数据包，我们可以开始查明问题的根源了。首先，我们看见一个 ARP 请求成功地抵达网络上我们认为的默认网关，所以我们知道网关设备在线并且能连接。我们也知道用户的计算机确实在网络上传输数据包，所以我们可以假设本机的协议栈没有问题。显然当进行 DNS 请求时，问题就发生了。

就这个网络来说，DNS 请求是由 Internet 上的外部服务器（4.2.2.2 或 4.2.2.1）解析的。这意味着，要使解析顺利进行的话，负责将数据包路由到 Internet 的路由器必须成功将 DNS 查询转发到服务器，而且服务器必须响应。否则，就无法

用 HTTP 请求 Web 页面。

我们知道其他用户上网都没有问题，这告诉我们网络路由器和远程 DNS 服务器也许不是问题的原因所在。剩下唯一可能的问题来源是用户自己的计算机。

进一步检查这台故障计算机后，我们发现它不接受 DHCP 分配的地址，而是手动配置了地址信息，并且默认网关地址设置错了。被设置为默认网关的地址并不是一台路由器，它不能将 DNS 查询数据包转发到网络之外。

### 3. 学到的知识

在这个情景中，问题出自一台错误配置的客户端。虽然这个问题相当简单，但它却严重影响了用户。对于缺少网络知识或者不能像我们这样能快速分析数据包的人而言，排除这么简单的配置错误将花费相当长的时间。你可以看到，数据包分析并不局限于大型和复杂的问题。

注意到由于我们不知道网络上默认网关的 IP 地址，Wireshark 不能准确地识别问题，但它可以告诉我们去哪里找，从而节省了宝贵时间。如果我们先与 ISP 联系、或者尝试用其他手段排除远程 DNS 服务器的因素，而不是检查网关路由器，- 我们就能将注意力集中到计算机本身，实际也就是问题的原因上。

---

#### 注意

如果我们能更熟悉这个特定网络的 IP 地址分配方案，就可以更快地分析出结果。如果我们注意到 ARP 请求发送到与网关路由器不同的 IP 地址上，我们就能立刻知道问题所在。网络出现问题经常是因为这些简单的配置错误，通过分析少量的数据包，通常都能快速解决。

---

## 8.3.2 无法访问 Internet：意外重定向

在这个情景中，我们又遇到一位不能在工作站上网的用户。然而，不像之前那个用户，她可以访问 Internet，只是不能访问 Google 主页 <http://www.google.com/>。每次她想访问 Google 的网站时，都被重定向到一个浏览器页面“该页无法显示”。这个问题只影响她一个人。

与之前的情景一样，这是一个只有一些简单交换机和一个简单路由器网关的小型网络。

### 1. 倾听线路

开始分析吧，我们一边监听流量，一边让用户尝试浏览 <http://www.google.com/>，

得到 nowebaccess2.pcap 文件。

## 2. 分析

如图 8-21 所示，捕获记录文件以一个 ARP 请求和响应开始。在数据包 1 中，用户计算机的 MAC 地址是 00:25:b3:bf:91:ee，IP 地址是 172.16.0.8，它向网段上的所有计算机发送一个 ARP 广播数据包，试图获得主机 172.16.0.102 的 MAC 地址。我们目前还不认识这个地址。

| No. | Time     | Source            | Destination       | Protocol | Info                                  |
|-----|----------|-------------------|-------------------|----------|---------------------------------------|
| 1   | 0.000000 | 00:25:b3:bf:91:ee | ff:ff:ff:ff:ff:ff | ARP      | who has 172.16.0.102? Tell 172.16.0.8 |
| 2   | 0.000334 | 00:21:70:c0:56:f0 | 00:25:b3:bf:91:ee | ARP      | 172.16.0.102 is at 00:21:70:c0:56:f0  |

图 8-21 对网络上另一个设备的 ARP 请求和响应

在数据包 2 中，用户的计算机了解到 IP 地址 172.16.0.102 的 MAC 地址是 00:21:70:c0:56:f0。根据之前的情形，我们也许会猜测这是网关路由器的地址，通过这个地址数据包可以被再次转发到外部 DNS 服务器。然而，如图 8-22 所示，下一个数据包并不是 DNS 请求，而是从 172.16.0.8 到 172.16.0.102 的 TCP 数据包。它设置了 SYN 标志，表明这是两台主机间建立 TCP 连接时握手的第一个数据包①。

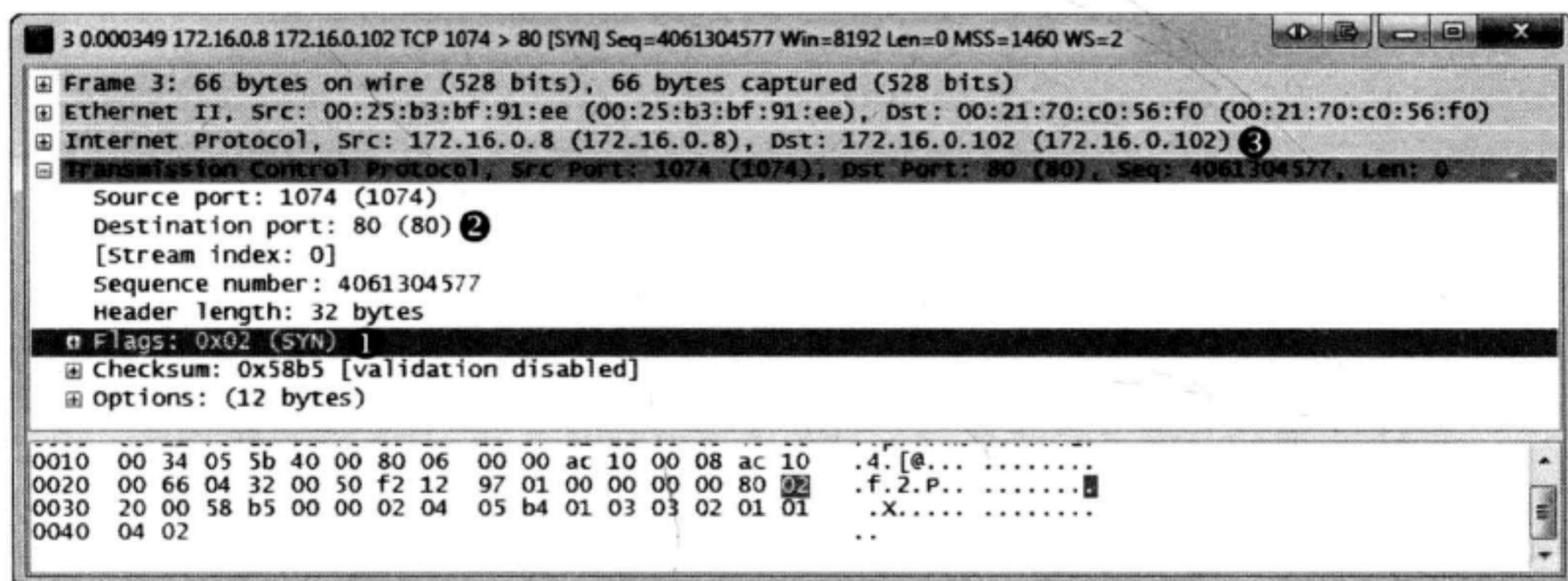


图 8-22 从一台内网主机发往另一台内网主机的 TCP SYN 数据包

显然，试图连接到 172.16.0.102 ③ 的 80 端口 ② 的 TCP 连接通常与 HTTP 流量有关。如图 8-23 所示，当主机 172.16.0.102 发送回带有 RST 和 ACK 标志 ① 的 TCP 数据包（数据包 4）后，连接请求就中断了。

回想起第 6 章说过，带有 RST 标志的数据包是用来结束 TCP 连接的。在这个场景中，主机 172.16.0.8 尝试与主机 172.16.0.102 的 80 端口建立 TCP 连接。

不幸的是，由于那台主机没有配置好在 80 端口监听请求的服务，那只能发送 TCP RST 数据包结束连接。这个过程又重复了两次。如图 8-24 所示，在通信最终结束前，用户计算机发送了一个 SYN 数据包并得到 RST 响应。

此时，用户在浏览器上看到了“该页无法显示”。

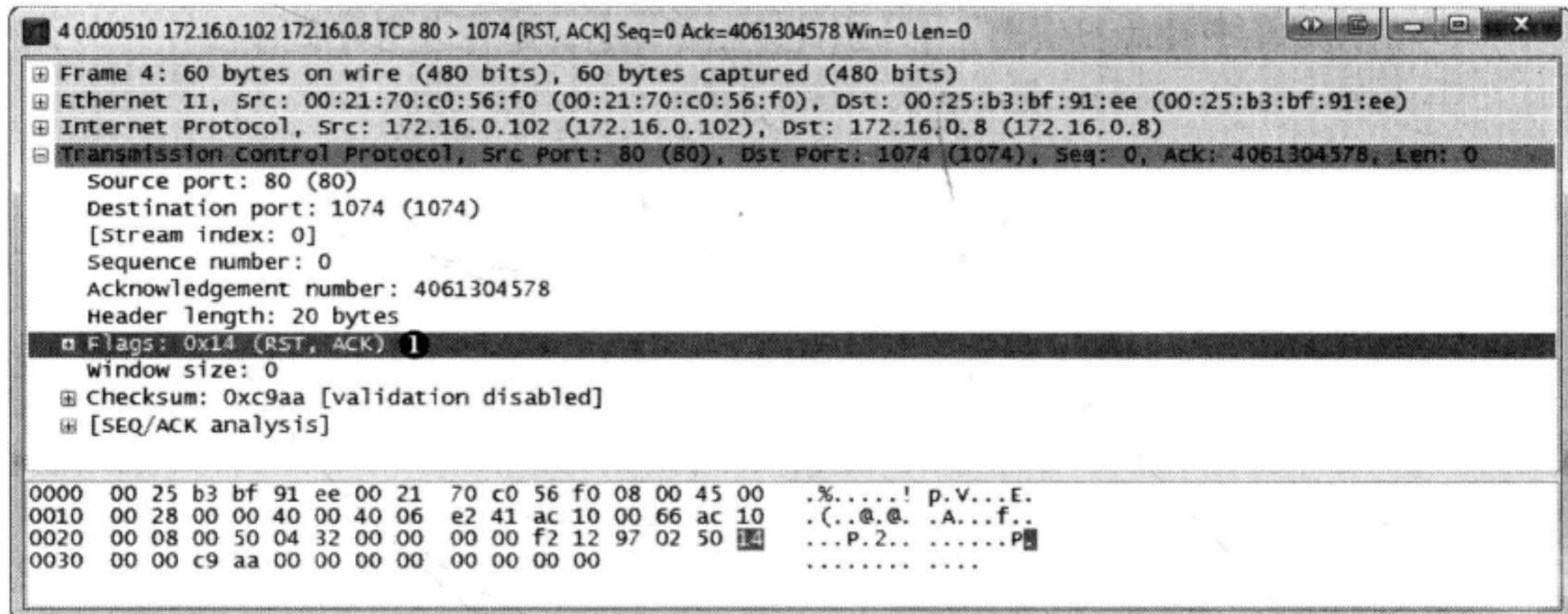


图 8-23 响应 TCP SYN 的 TCP RST 数据包

| No. | Time     | Source       | Destination  | Protocol | Info  |
|-----|----------|--------------|--------------|----------|---|
| 3   | 0.000349 | 172.16.0.8   | 172.16.0.102 | TCP      | 1074 > 80 [SYN] Seq=4061304577 win=8192 Len=0 MSS=1460 ws=2 |
| 4   | 0.000510 | 172.16.0.102 | 172.16.0.8   | TCP      | 80 > 1074 [RST, ACK] Seq=0 Ack=4061304578 win=0 Len=0       |
| 5   | 0.499162 | 172.16.0.8   | 172.16.0.102 | TCP      | 1074 > 80 [SYN] Seq=4061304577 win=8192 Len=0 MSS=1460 ws=2 |
| 6   | 0.499362 | 172.16.0.102 | 172.16.0.8   | TCP      | 80 > 1074 [RST, ACK] Seq=0 Ack=4061304578 win=0 Len=0       |
| 7   | 0.999190 | 172.16.0.8   | 172.16.0.102 | TCP      | 1074 > 80 [SYN] Seq=4061304577 win=8192 Len=0 MSS=1460      |
| 8   | 0.999507 | 172.16.0.102 | 172.16.0.8   | TCP      | 80 > 1074 [RST, ACK] Seq=0 Ack=4061304578 win=0 Len=0       |

图 8-24 TCP SYN 和 RST 数据包一共出现了 3 次

在查看其他工作正常的网络设备的配置信息后，数据包 1 和 2 中的 ARP 请求和响应引起了我们的注意。因为 ARP 请求并不是指向网关路由器的真实 MAC 地址，而是其他未知设备。在 ARP 请求和响应之后，我们期望看到发送给 DNS 服务器的请求，以得到 www.google.com 的 IP 地址，但最终并没有看到。以下是阻止 DNS 查询的两个条件。

- 发起连接的设备在 DNS 缓存中已经有域名到 IP 地址的映射。
- 发起连接的设备在 hosts 文件中已经有域名到 IP 地址的映射。

进一步检查这台计算机后，我们发现它的 hosts 文件有一个 www.google.com 表项，对应一个内网 IP 地址 172.16.0.102。这个错误表项就是用户问题的根源。

计算机通常都把 hosts 文件当做域名-IP 地址配对的可信来源，并且会在查询外部来源之前检索它。在这个场景中，用户计算机检查它的 hosts 文件，发现

有一个 `www.google.com` 的表项，就认为 `www.google.com` 在这个本地网段。接着，它向这个主机发送 ARP 请求，并得到响应，然后尝试向 `172.16.0.102` 的 80 端口发起 TCP 连接。然而，由于该系统并没有配置成 Web 服务器，因此它不可能接受这个连接请求。

将这个 hosts 文件的表项移除后，用户的计算机就能正常访问 `www.google.com` 了。

---

**注意** 在 Windows 系统上查看 hosts 文件，请打开 `C:\Windows\System32\drivers\hosts`。  
在 Linux 上则应查看 `/etc/hosts`。

---

实际上，这个场景非常普遍。恶意软件在几年前就使用这个方法，把用户重定向到存放恶意代码的网站。试想，如果黑客修改了你的 hosts 文件，每次你登录网上银行，实际上访问的却是一个伪造的网站，专门偷你账户里的钱，这该有多恐怖！

### 3. 学到的知识

继续分析流量，你会学到各种各样的协议如何工作、以及如何阻断它们。在这个场景中，主机没有发送 DNS 请求是因为客户端被配置错了，而不是因为其他外部限制或外部的错误配置。

在数据包层面上考察这个问题，我们可以迅速发现未知的 IP 地址，也能迅速发现 DNS 这个通信过程的关键部分消失了。通过这些信息，我们可以确定客户端才是问题的来源。

#### 8.3.3 无法访问 Internet：上游问题

与前两个场景一样，在这个场景中，有一位用户抱怨它的工作站无法上网。后来，他发现只是无法访问 `http://www.google.com/` 这个网站。进一步调查之后，我们发现这个问题影响到了机构的每一个人——谁也无法访问 Google。

这个网络配置和前两个场景一样，仍然是用一些简单交换机和一个路由器将网络连接到 Internet。

##### 1. 偷听线路

为了解决这个问题，我们首先访问 `http://www.google.com/` 以生成流量。这是个全网问题——意味着它也影响你的计算机，而且可能是感染恶意软件导致

的——所以你不应该直接在你的设备上嗅探。当你在现实中遇到类似这样的问题时，网络分流器就是最好的解决方案，因为它允许你在短暂中断服务后完全被动地获取流量。通过网络分流器获得的流量保存在 nowebaccess3.pcap 文件中。

## 2. 分析

这个数据包的捕获以 DNS 流量开始，而不是我们之前看到的 ARP 流量。因为捕获的第一个数据包发往一个外部地址，并且数据包 2 包含来自那个地址的响应，我们可以假设 ARP 过程已经完成了，并且网关路由器的 MAC-IP 地址映射已经存在于主机的 ARP 缓存中。

如图 8-25 所示，捕获中的第一个数据包从主机 172.16.0.8 发往地址 4.2.2.1①，并且它是一个 DNS 数据包②。查看该数据包的内容，我们发现这是一个查询 www.google.com 的 A 记录请求③。

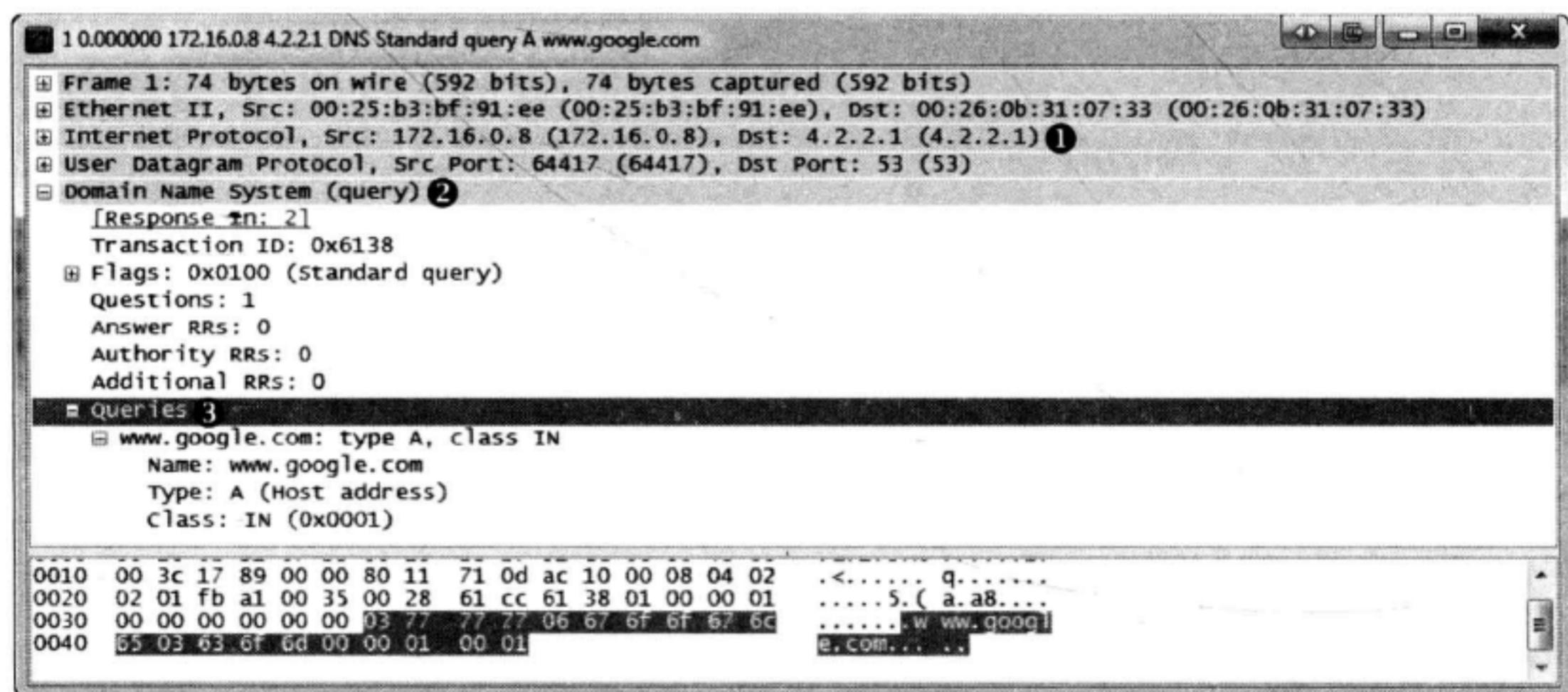


图 8-25 查询 www.google.com 的 A 记录

如图 8-26 所示，来自 4.2.2.1 的响应是捕获文件的第 2 个数据包。查看 Packet Details 面板，我们发现响应这个请求的名字服务器提供了多个回答①。此时看起来通信一切正常。

现在用户的计算机已经得到 Web 服务器的 IP 地址，它可以尝试与服务器通信了。如图 8-27 所示，通信过程从数据包 3 开始，这是一个从 172.16.0.8 发往 74.125.95.105 的 TCP 数据包①。这个目标地址来自数据包 2 里 DNS 查询响应提供的第 1 个 A 记录。TCP 数据包设置了 SYN 标志②，尝试连接远程服务器的 80 端口③。

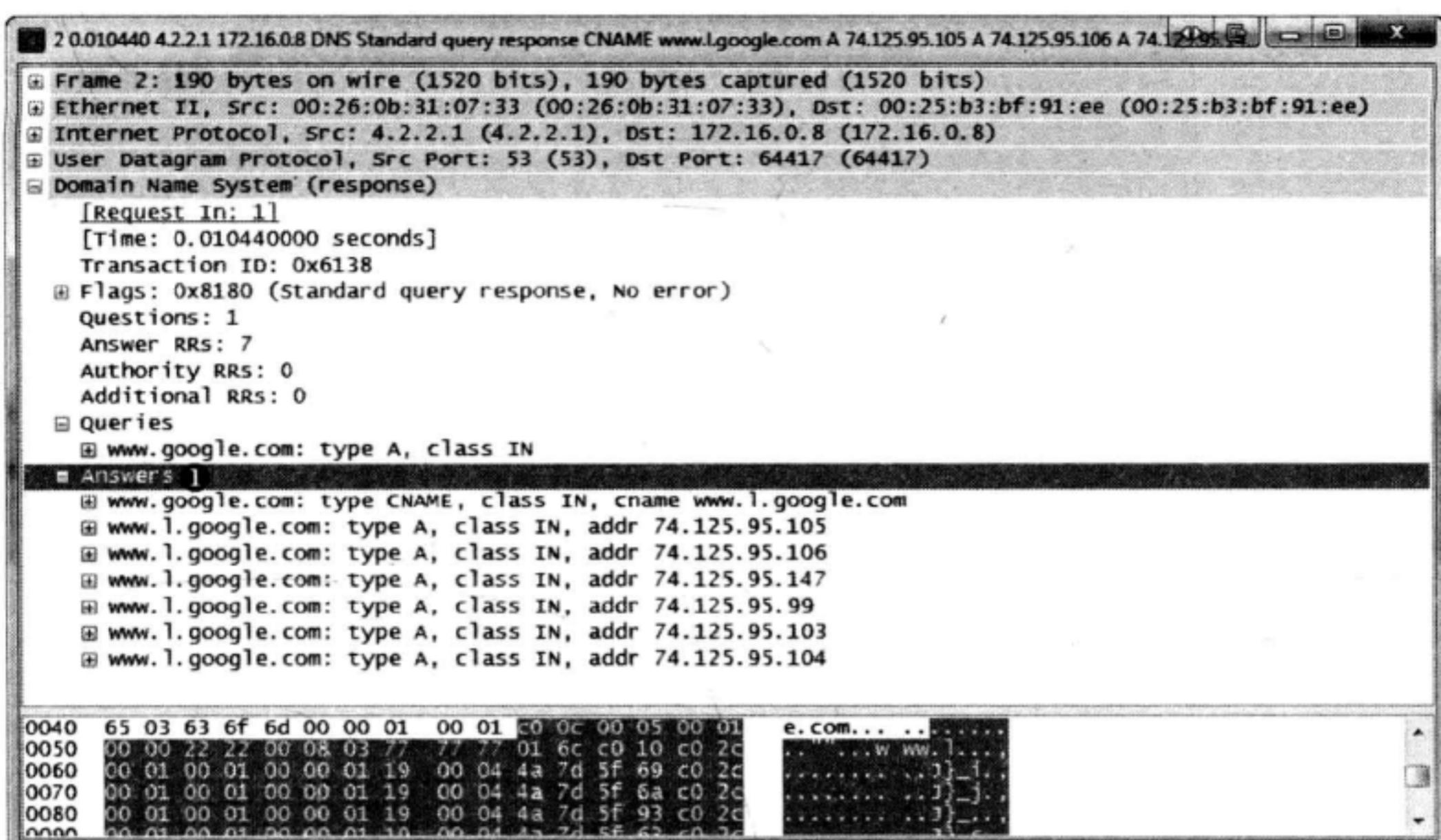


图 8-26 包含多个 A 记录的 DNS 响应

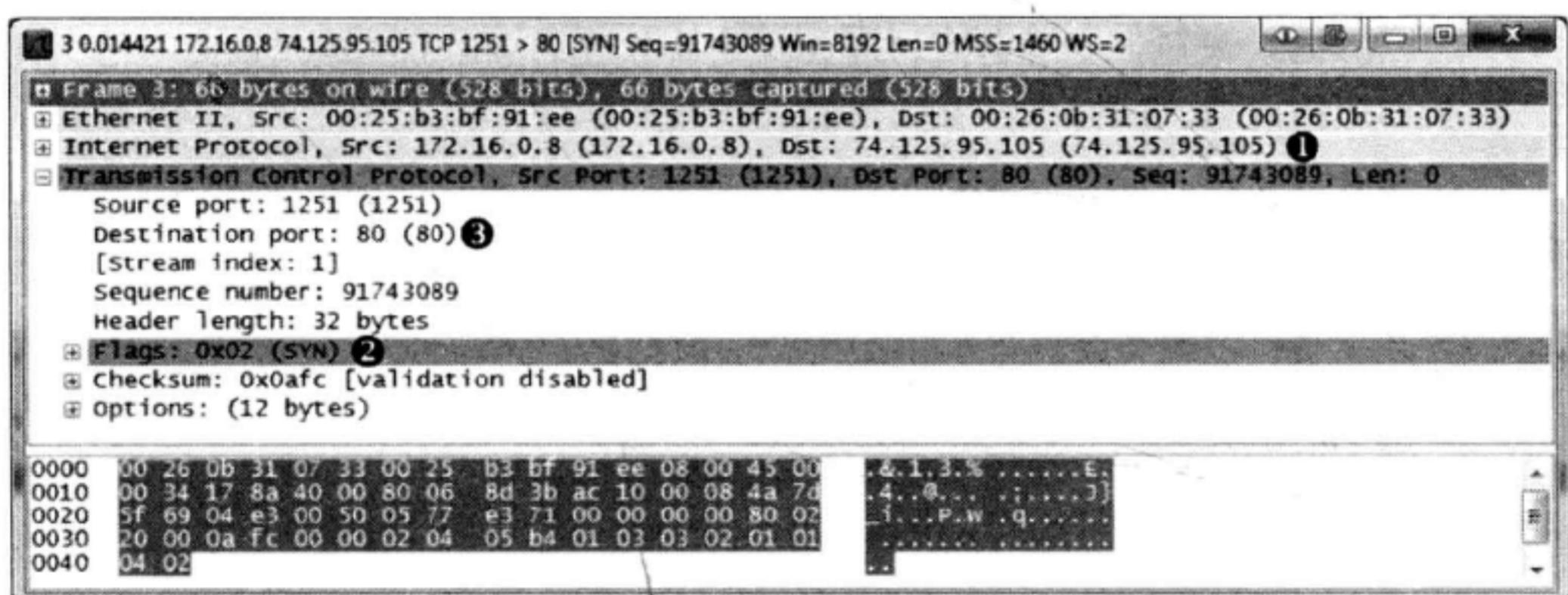


图 8-27 尝试连接 80 端口的 SYN 数据包

因为这是一个 TCP 握手过程，我们知道应该在响应中看到 TCP SYN/ACK 数据包，但相反，主机过一会儿又发了另一个 SYN 数据包到目标。这个过程大概在 1s 后再次发生，如图 8-28 所示，到这里通信停止了，浏览器报告找不到网站。

| No. | Time     | Source     | Destination   | Protocol | Info  |
|-----|----------|------------|---------------|----------|---|
| 3   | 0.014421 | 172.16.0.8 | 74.125.95.105 | TCP      | 1251 > 80 [SYN] Seq=91743089 win=8192 Len=0 MSS=1460 WS=2 |
| 4   | 0.019417 | 172.16.0.8 | 74.125.95.105 | TCP      | 1251 > 80 [SYN] Seq=91743089 win=8192 Len=0 MSS=1460 WS=2 |
| 5   | 1.016531 | 172.16.0.8 | 74.125.95.105 | TCP      | 1251 > 80 [SYN] Seq=91743089 win=8192 Len=0 MSS=1460 WS=2 |

图 8-28 TCP SYN 数据包尝试了 3 次都没有收到响应

这时，我们想到由于能成功向外部 DNS 服务器提交查询请求，所以网络内的工作站可以连接到外网。DNS 服务器响应了一些看起来有效的地址，然后我们的主机就尝试向其中一个地址建立连接。而且，我们尝试连接的本地工作站看起来功能正常。

问题是远程服务器没有响应我们的连接请求，连 TCP RST 数据包都没发过来。可能的原因有几种：Web 服务器配置错误、Web 服务器的协议栈崩溃、远程网络部署了数据包过滤设备（比如防火墙<sup>1</sup>）。假设本地网络没有数据包过滤设备，那所有可能的解决方法都在远程网络上，这超出了我们的控制范围。在这个案例中，Web 服务器工作不正常，我们的所有尝试都失败了。一旦 Google 那边修复故障<sup>2</sup>，通信就可以继续了。

### 3. 学到的知识

这个场景中的问题不是我们能修复的。我们的分析表明，问题不在于我们网络上的主机、路由器，也不在于提供域名解析服务的外部 DNS 服务器。问题在我们网络设施之外。

有时候发现这不是我们的问题不仅能缓解压力，也能在管理层来敲门时挽回颜面。我与很多运营商、设备厂商和软件公司打过交道，他们都说不是自己那边的问题，但你已经看到，数据包是不会说谎的。

## 8.3.4 打印机故障

我们的技术支持管理员遇到了一个打印机的难题。销售部门的用户报告说大批量打印机出故障了。当用户发送大量打印作业给它时，它会打印几页然后停止工作。他们改动了很多驱动配置选项，但没有任何效果。技术支持的员工们希望你去确认这不是网络问题。

### 1. 倾听线路

这个问题的焦点在于打印机，因此我们希望尽可能将嗅探器放到离打印机最近的地方。虽然我们不能在打印机上安装 Wireshark，但网络上使用了高级三层交换机，所以我们可以使用端口镜像功能。我们将打印机连接的端口镜像到一个空端口，然后将一台装有 Wireshark 的笔记本连接到该端口。安装完成后，我们让一位用户发送大量打印作业给打印机，而我们会监视端口输出内容，最

1 这是在中国大陆地区最可能遭遇的情况。——译者注

2 在中国大陆地区，通常情况下并非 Google 那边的故障，而是访问路径中存在某种防火墙，只能寻求绕过。——译者注

后得到捕获记录文件 inconsistent\_printer.pcap。

## 2. 分析

如图 8-29 所示，捕获文件的开头是发送打印作业的主机（172.16.0.8）与打印机（172.16.0.253）的 TCP 握手。握手之后，一个大小为 1460 字节的 TCP 数据包发送到打印机①。数据大小既可以在 Packet List 面板 Info 列的右边看到，也可以在 Packet Details 面板的 TCP 头部信息的底部看到。



当一个设备发送 TCP 数据包给远程设备，远程设备没有确认此次传输时，就发送一个 TCP 重传数据包。一旦到达重传门限，发送设备就假设远程设备没有收到数据，立刻重传数据包。在通信停止之前，这个过程重复了多次。

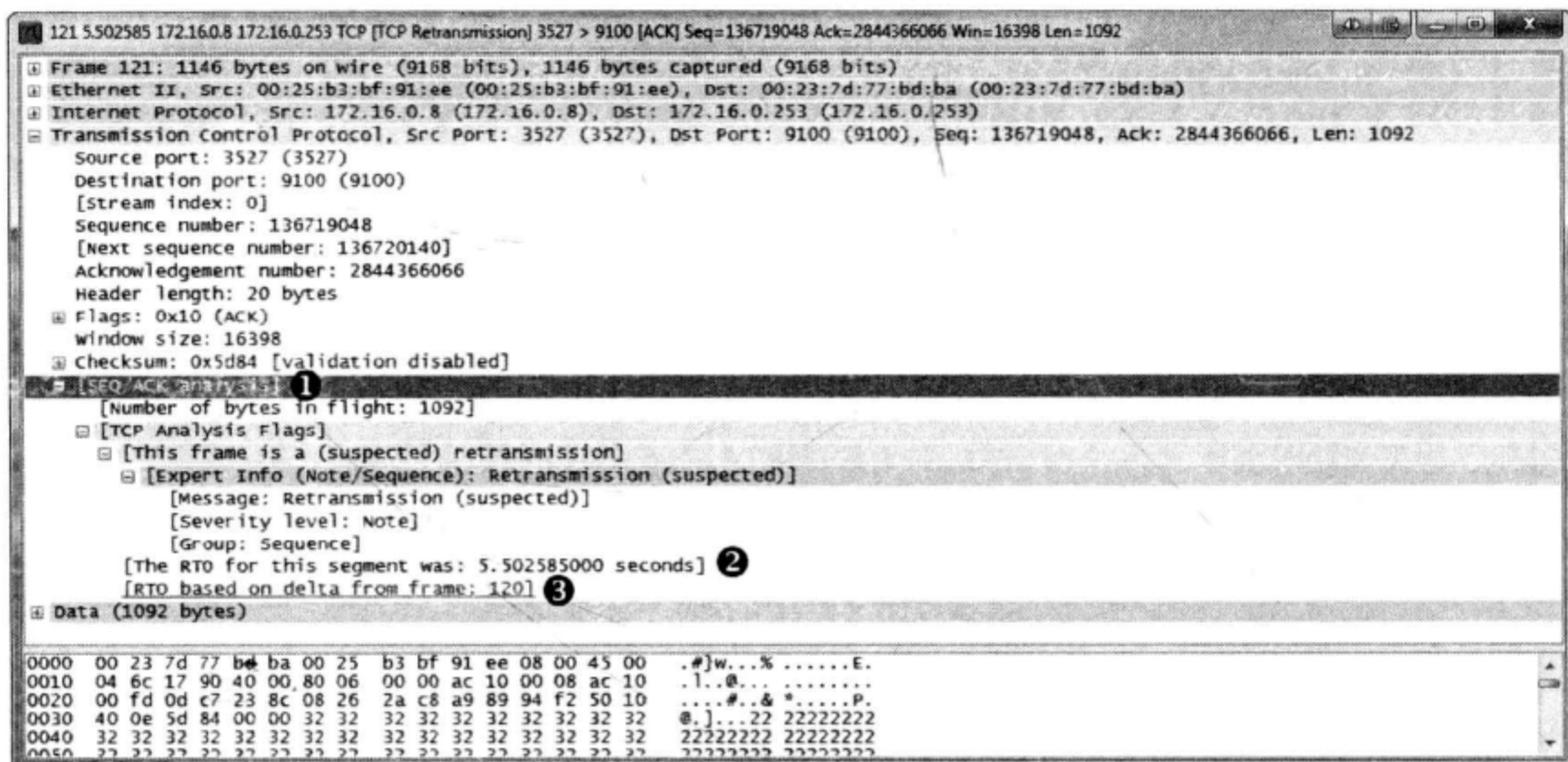


图 8-31 这些 TCP 重传数据包是故障的一个标志

在这个场景中，因为打印机没有确认传输的数据，客户工作站就向打印机发送重传数据包。如果你展开 TCP 头部的 SEQ/ACK analysis 部分以及下方的额外信息，如图 8-31 所示①，你可以从细节中看到为什么这是重传。根据 Wireshark 加工的细节，数据包 121 是数据包 120 的重传③。另外，重传数据包的重传超时（RTO）在 5.5s②左右。

当分析数据包间隔时间时，你可以更改时间显示格式以适应特定情形。在这个案例中，我们想看看之前的数据包在发送后多久发生了重传，于是选择 **View->Time Display Format** 并改成 **Seconds Since Previous Captured Packet** 这个选项。然后，如图 8-32 所示，你可以清楚地看见初始数据包（数据包 120）发送 5.5s 后发生了数据包 121 的重传①。

| No. | Time     | Source     | Destination  | Protocol | Info   |
|-----|----------|------------|--------------|----------|--|
| 121 | 5.502585 | 172.16.0.8 | 172.16.0.253 | TCP      | [TCP Retransmission] 3527 > 9100 [ACK] Seq=136719048 Ack=2844366066 win=16398 Len=1092 |
| 122 | 5.600089 | 172.16.0.8 | 172.16.0.253 | TCP      | [TCP Retransmission] 3527 > 9100 [ACK] Seq=136719048 Ack=2844366066 win=16398 Len=1092 |

图 8-32 查看数据包间隔时间有利于解决问题

下一个数据包是数据包 120 的另一个重传。这个数据包的 RTO 是 11.10s，包括上一个数据包的 5.5s RTO。Packet List 面板的 Time 列告诉我们，在上一次重传 5.6s 后发生了这次重传。这好像是捕获文件中的最后一个数据包，巧合的是，打印机大概在这个时间停止打印了。

好在这个分析场景只涉及内网的两台设备，所以我们只需要确定是客户工作站还是打印机的问题。我们可以看见数据正常流动了相当长的时间，然而在某一时刻，打印机停止响应工作站了。工作站尽了最大努力使数据到达目的地，重传就是个明证，但打印机就是没有响应。这个问题可以在其他工作站上重现，所以我们猜测打印机才是问题的来源。

进一步分析后，我们发现打印机的内存出故障了。当大量打印作业发送到打印机时，它只打印一定的页数，一旦访问到特定内存区域就停止工作。由此可见，内存问题导致打印机无法接收新数据，并中断了与主机的通信。

### 3. 学到的知识

虽然这个打印机问题，不是网络问题所致，但我们仍可以用 Wireshark 指出其问题。跟之前的场景不同，这里只有 TCP 流量。幸好，当两台设备停止通信时，TCP 给我们留下了有用的信息。

在这个案例中，当通信意外停止时，我们靠 TCP 内置的重传功能指出了问题的确切位置。继续学习下面的场景，我们会经常依赖于像这样的功能来解决更复杂的问题。

## 8.3.5 分公司之困

在这个场景中，一家公司在总部之外新开设了几家分公司。通过部署一台 Windows 域控制器服务器和一台备用域控制器，公司的所有 IT 设施几乎都放置在总部。域控制器负责处理 DNS 和分公司用户的认证请求。

域控制器是一个代理 DNS 服务器，它接收来自总部的上游 DNS 服务器的资源记录信息。

当部署团队将新设施延伸到分公司时，发现没有人能访问网络上的内部 Web 应用服务器。这些服务器位于总部办公室，通过广域网（Wide Area Network，简称 WAN）访问。问题影响到分公司的所有用户，并只限于这些内部服务器。所有用户都可以访问 Internet 以及分公司内的其他资源。

图 8-33 显示了在这个场景中要考虑的组件，包括了多个站点。

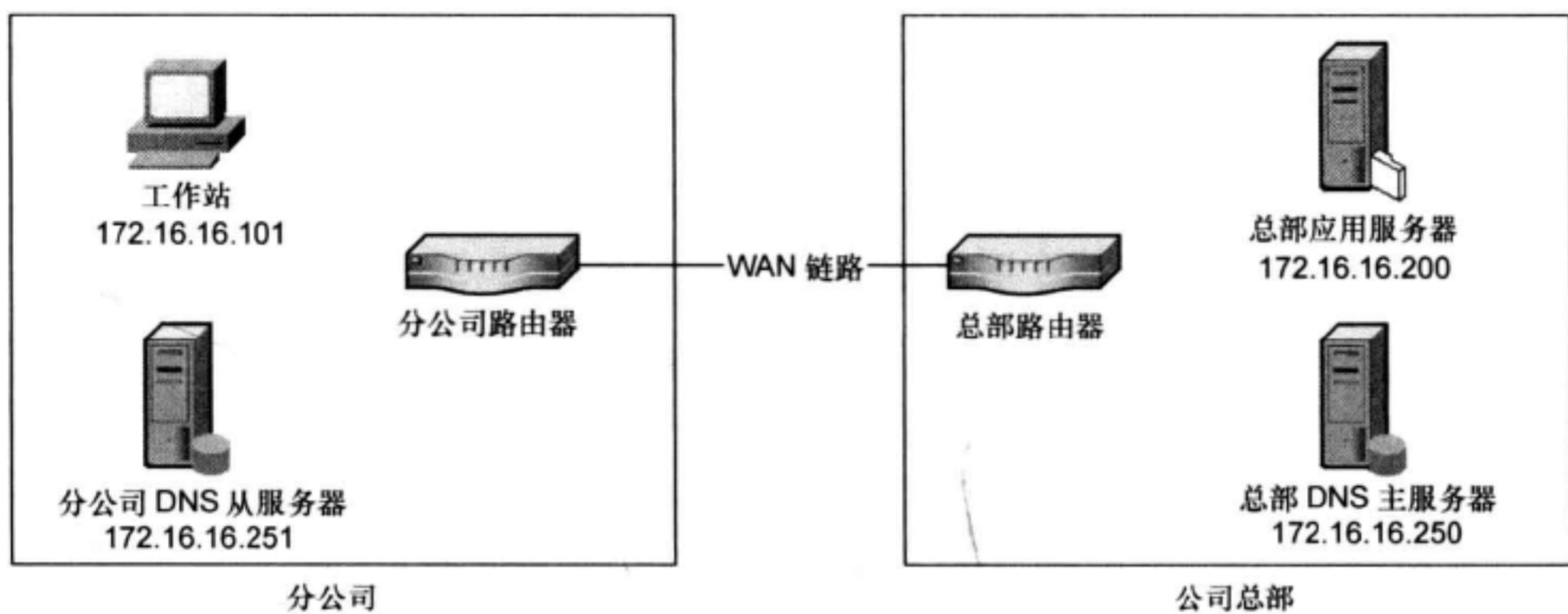


图 8-33 分公司之困问题的相关组件

## 1. 偷听线路

由于问题出在总部和分公司间的通信过程中，我们可以在多个地点收集数据来跟踪问题。问题可能出在分公司的客户端上，所以我们使用端口镜像功能查看其中一台计算机在线路上看到了什么数据包。收集完这个信息后，我们可以使用它推测出其他收集地点以帮助解决问题。从其中一个客户端捕获的初始数据包保存在 `stranded_clientside.pcap` 文件中。

## 2. 分析

如图 8-34 所示，当工作站 172.16.16.101 尝试访问托管在总部应用服务器 172.16.16.200 的应用程序时，产生了捕获文件的第一个数据包。这个捕获只有两个数据包。第一个数据包是发送到 172.16.16.251 ① 的 DNS 请求，查询应用服务器 ③ 的 A 记录 ②。这是总部 172.16.16.200 服务器的 DNS 域名。

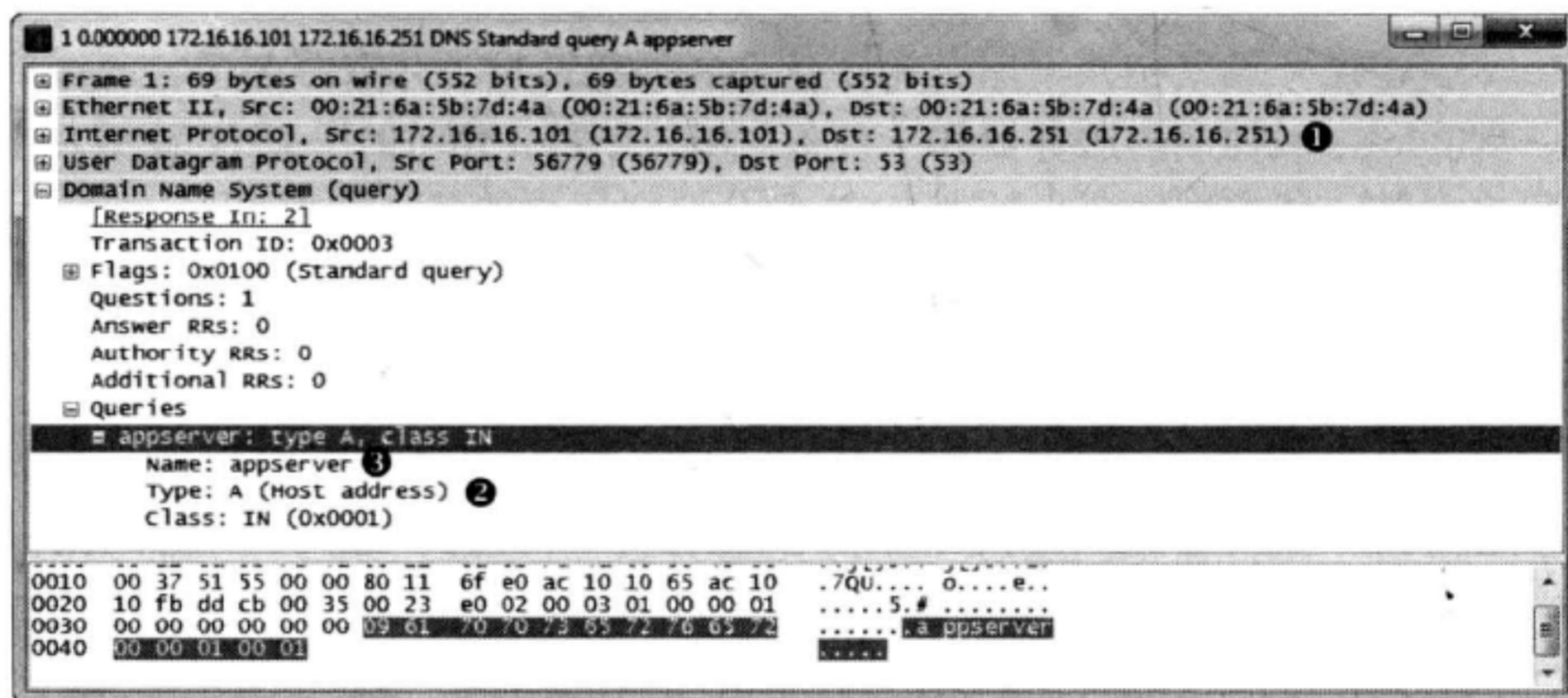


图 8-34 通信从查询应用服务器 A 记录的 DNS 请求开始

如图 8-35 所示，这个数据包的响应是一个服务器失败❶，表明 DNS 查询被阻止了。注意到这个数据包只是一个错误（服务器失败），并没有回答查询结果❷。

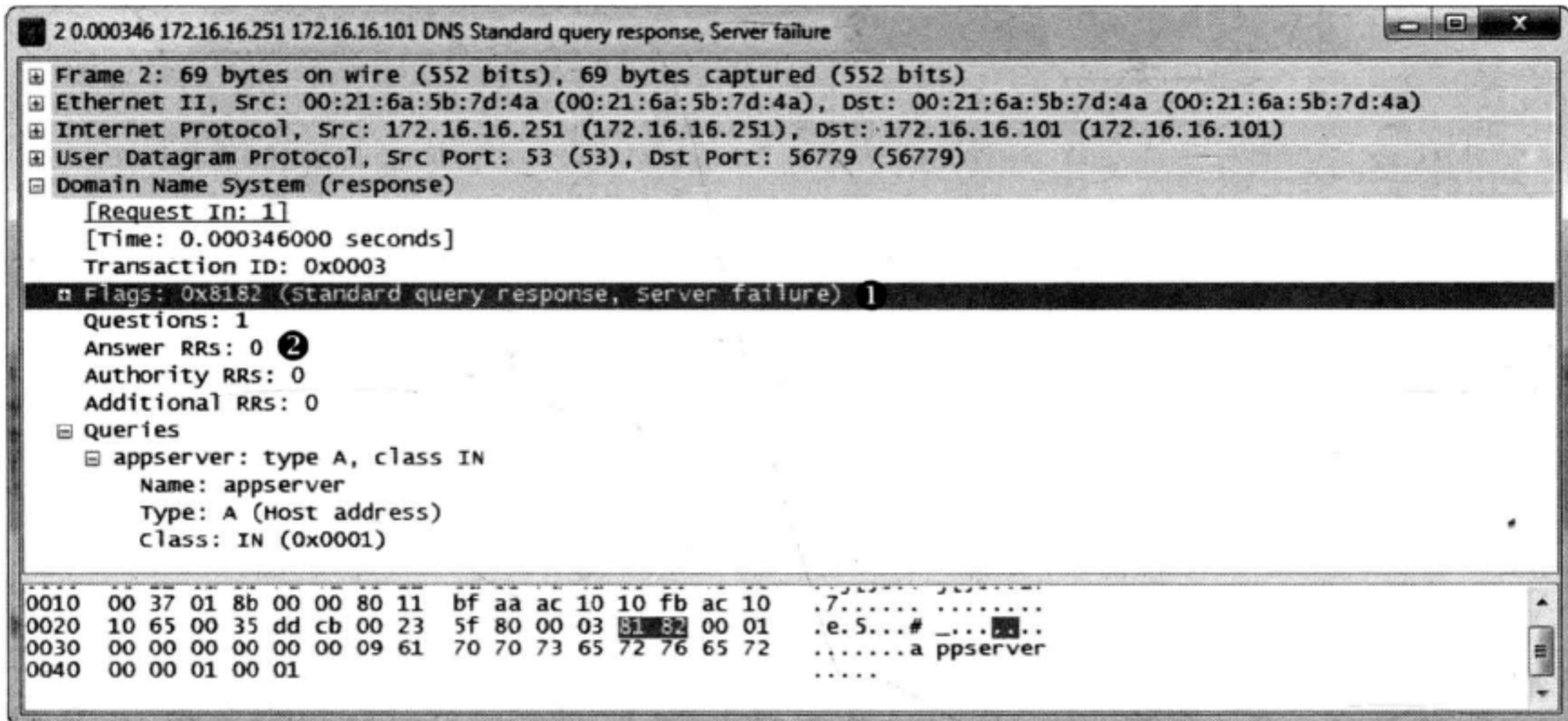


图 8-35 查询响应表明这是上游的问题

现在我们知道该通信故障与 DNS 有关。因为分公司的 DNS 查询由 DNS 服务器 172.16.16.251 解析，我们前往下一站。

为了从分公司的 DNS 服务器捕获合适流量，我们将嗅探器留在原地，只改变端口镜像设置。现在服务器的流量就被镜像到我们的嗅探器了。捕获结果在 `stranded_branchdns.pcap` 文件中。

如图 8-36 所示，这个捕获的开头是我们之前看到的查询和响应，但还有一个额外的数据包。额外的数据包看起来很奇怪，因为它尝试与中心办公室的首选 DNS 服务器（172.16.16.250）❶的标准 DNS 服务端口 53❷进行通信，但它却不是我们过去看见的 UDP 类型❸。

为了找出这个数据包的用途，回顾我们在第 7 章对 DNS 的讨论。DNS 通常使用 UDP，但当响应超过一定大小时就使用 TCP。在那种情况下，我们会看见一些触发 TCP 流量的 UDP 流量。另外，TCP 也用于 DNS 的区域传送过程，使资源记录在 DNS 服务器之间传输，这里就是该种情况。

分公司的 DNS 服务器是总部 DNS 服务器的从属服务器，意味着分公司的 DNS 服务器依赖于总部服务器获得资源记录。分公司用户试图访问的应用

服务器放置在总部，意味着总部 DNS 服务器是它的权威 DNS 服务器。要使分公司服务器能解析用户对应用服务器的 DNS 请求，总部 DNS 服务器必须把 DNS 资源记录传输给分公司 DNS 服务器，这可能是捕获文件中 SYN 数据包的来源。

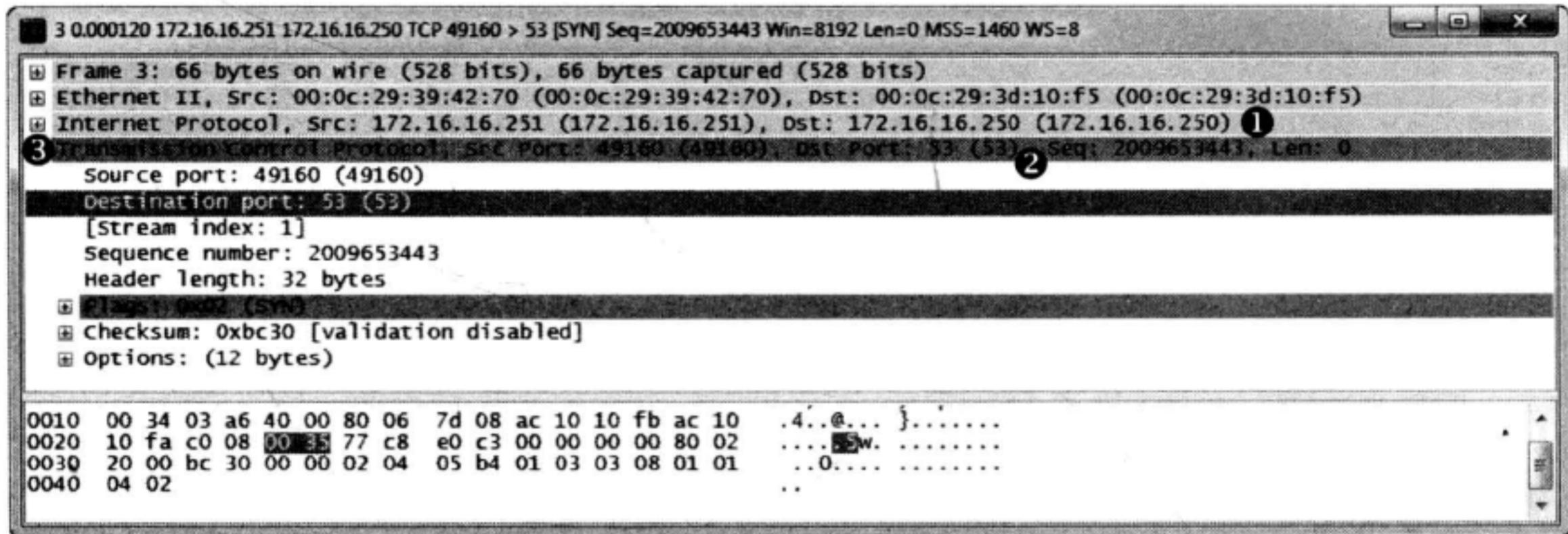


图 8-36 这个 SYN 数据包使用了 53 端口，但不是 UDP

SYN 数据包没有得到响应，这告诉我们总部和分公司 DNS 服务器之间区域传送失败导致了 DNS 故障。现在我们可以进一步找出区域传送失败的原因。办公室之间的路由器或中心办公室的 DNS 服务器可能是罪魁祸首。为了找出问题，我们可以嗅探中心办公室 DNS 服务器的流量，看看 SYN 数据包是不是到达了服务器。

我没有给出中心办公室 DNS 服务器的流量捕获文件，因为根本就没有。SYN 数据包从来没有到达服务器。派遣技术人员查看连接两个办公室的路由器配置后，我们发现中心办公室的路由器被配置成只允许 53 端口的 UDP 流量进入，而 53 端口的 TCP 流量则被阻止了。这个简单的配置错误阻止了服务器间的区域传送，从而导致分支办公室的客户端无法解析对中心办公室设备的查询。

### 3. 学到的知识

看完这个“犯罪剧”，你一定学到了很多关于调查网络通信问题的知识。当“犯罪”发生后，侦探开始问讯受害者。找到线索，顺藤摸瓜，直到找到罪魁祸首。

在这个场景中，我们一开始先查看了受害者（工作站），然后找到了 DNS 通信问题这个线索。这个线索将我们带到分支 DNS 服务器，然后又到中心 DNS

服务器，最终找到路由器，也就是问题的来源。

在分析时，请尝试从数据包中找出线索。线索不一定能告诉你谁是“罪犯”，但通常它们最终能帮你找出来。

### 8.3.6 生气的开发者

在 IT 界，开发者和系统管理员经常争吵。开发者总是将程序故障归咎于糟糕的网络设置和设备。系统管理员则倾向于把网络错误和网络缓慢归咎于糟糕的代码。

在这个场景中，程序员开发了一个应用程序，用于跟踪多个商店的销售并报告回中心数据库。为了节约正常工作时间的带宽，他没有设计成实时应用程序。而是等报告数据累积一天后，才在晚上以逗号分隔值（comma-separated value，简称 CSV）文件的形式传回，插入中心数据库中。

然而，这个新开发的应用程序工作不太正常。服务器接收到了各个商店传回的文件，但插入数据库的数据是错误的。一些地区的数据丢失了，有的数据还存在错误，而且文件某些部分还丢失了。系统管理员很烦恼，因为程序员抱怨这是网络的问题。程序员一口咬定文件在从商店传到中心数据库时被损坏了。我们就要证明他是错的。

#### 1. 倾听线路

为了收集所需数据，我们可以在其中一个商店或中心办公室捕获数据包。故障影响到了所有商店，因此如果这确实是网络导致的问题，那肯定是在中心办公室那边——它是所有商店通信的汇聚点。

网络交换机支持端口镜像，所以我们用端口镜像功能嗅探服务器的流量。我们只捕获单个商店上传 CSV 文件到收集服务器的流量。结果保存在 tickedoffdeveloper.pcap 文件中。

#### 2. 分析

除了网络上的基本信息流量之外，我们完全不了解程序员开发的应用程序。捕获文件看起来是以一些 FTP 流量开始的，因此我们将调查这是不是传输 CSV 文件采用的机制。简洁、干净的通信最适合查看通信流量图了。选择 **Statistics -> Flow Graph**，然后单击 OK。图 8-37 显示了结果图像。

在流量图的基础上，我们看见一个 172.16.16.128 与 172.16.16.121❶之间的

基本 FTP 连接②。由于 172.16.16.128 发起连接，我们猜测它是客户端，172.16.16.121 则是汇总与处理数据的服务器。流量图确认这些流量只用到了 FTP 协议。

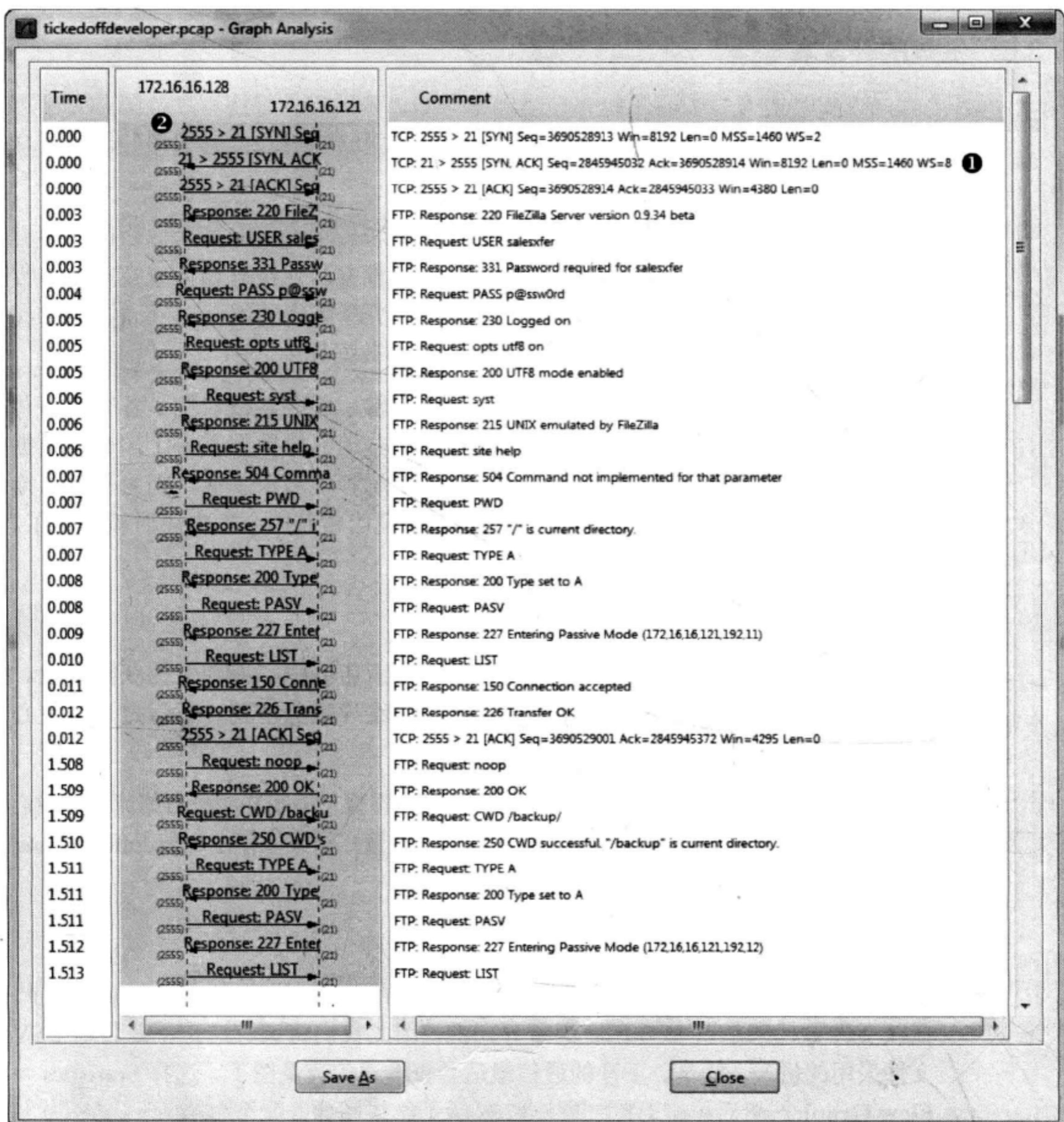


图 8-37 流量图提供了 FTP 通信的快速视图

我们知道这里会有一些数据传输，因此我们用 FTP 的知识，定位开始传输数据的位置。FTP 连接和数据传输是由客户端发起的，因此我们应该寻找用于上传数据到 FTP 服务器的 FTP STOR 命令。最简单的方法是生成一个过滤器。

这个捕获文件里到处都是 FTP 请求命令，因此我们不需要面对表达式生成器中的数百个协议和选项，只要在 Packet List 面板中直接生成过滤器就行了。为此，我们首先需要选择一个出现有 FTP 请求命令的数据包。我们将选择数据包 5，这是最接近列表顶部的一个。然后展开 Packet Details 面板的 **FTP section** 和 **USER section**。右击 **Request Command: USER** 域，并选择 **Prepare a Filter**。最后，选择 **Selected**。

这将生成一个筛选含有 FTP USER 请求命令的数据包的过滤器，并出现在过滤器对话框中。接着，如图 8-38 所示，编辑过滤器，将单词 USER 替换成 STOR ①。

The screenshot shows the Wireshark interface with a filter bar at the top containing "Filter: ftp.request.command == \"STOR\" ①". Below the filter bar is a table with columns: No., Time, Source, Destination, Protocol, and Info. A single row is selected, labeled ②, showing details: Time 64 4.369659, Source 172.16.16.128, Destination 172.16.16.121, Protocol FTP, and Info Request: STOR store4829-03222010.csv.

| No.  | Time     | Source        | Destination   | Protocol | Info                                 |
|------|----------|---------------|---------------|----------|--------------------------------------|
| ② 64 | 4.369659 | 172.16.16.128 | 172.16.16.121 | FTP      | Request: STOR store4829-03222010.csv |

图 8-38 这个过滤器有助于识别数据从哪里开始传输

现在按回车键应用这个过滤器，你会看见捕获文件的数据包 64 ②里只有一个 STOR 命令。

既然我们已经知道数据从哪里开始传输了，就单击 Packet List 面板上方的 **Clear** 按钮清除过滤器。

查看从数据包 64 开始的捕获文件，我们看见这个数据包指定了传输 store4829-03222010.csv 文件①，如图 8-39 所示。

The screenshot shows the Wireshark interface with a detailed view of frame 64. The packet details pane shows the following structure:  
Frame 64: 83 bytes on wire (664 bits), 83 bytes captured (664 bits)  
Ethernet II, Src: 00:21:6a:5b:7d:4a (00:21:6a:5b:7d:4a), Dst: 00:0c:29:ea:17:be (00:0c:29:ea:17:be)  
Internet Protocol, Src: 172.16.16.128 (172.16.16.128), Dst: 172.16.16.121 (172.16.16.121)  
Transmission Control Protocol, Src Port: 2555 (2555), Dst Port: 21 (21), Seq: 3690529135, Ack: 2845945907, Len: 29  
File Transfer Protocol (FTP)  
STOR store4829-03222010.csv\r\nRequest command: STOR  
Request arg: store4829-03222010.csv ①

The bytes pane shows the raw hex and ASCII data for the packet.

| Hex  | Dec   | ASCII             |
|------|---|-------------------|
| 0020 | 10 79 09 fb 00 15 db f9 01 6f a9 a1 b0 33 50 18 | .y..... .o...3P.  |
| 0030 | 10 41 fe 31 00 00 53 54 4f 52 20 f3 74 6f 72 65 | .A.1..ST OR Store |
| 0040 | 34 38 32 39 2d 30 33 32 32 32 30 31 30 2e 63 73 | 4829-032 22010.cs |
| 0050 | 76 0d 0a  | M..               |

图 8-39 使用 FTP 传输 CSV 文件

STOR 命令后面的数据包使用了不同的端口，但被识别成 FTP 数据传输的

一部分。我们已经验证数据在传输了，但我们仍然没有证明程序员是错的。为此，我们需要从捕获的数据包中提取传输文件，以展示文件在网络中传输后并没有被损坏。

当文件以未加密格式在网络中传输时，它会被分解成多个段，并在目的地组装。在这个场景中，我们在数据包到达目的地并且尚未被组装时捕获它们。数据就在那儿了，我们只需要当做数据流提取文件组装它。为此，选择 FTP 数据流的任一个数据包（比如数据包 66），并单击 **Follow TCP Stream**。结果显示在 TCP 流中，如图 8-40 所示。

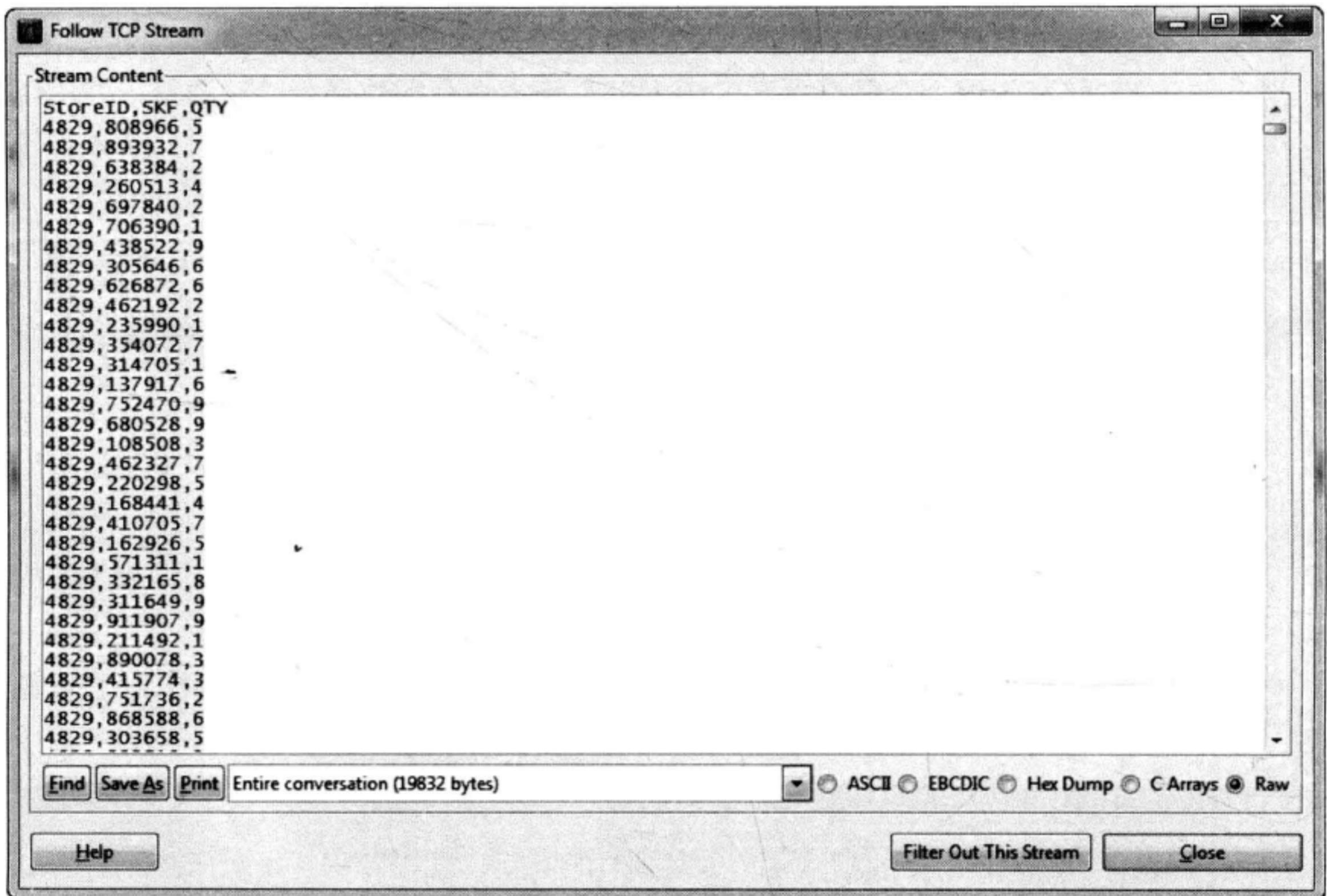


图 8-40 TCP 流显示了传输的数据

由于数据在 FTP 中明文传输，所以我们能看到它，但却不能断定文件是完整的。为了重组数据并提取为原始格式，我们单击 **Save As** 按钮并指定数据包 64 中显示的文件名，如图 8-41 所示。然后单击 **Save**。

保存操作的结果应该是一个 CSV 文件，这是对商店系统传过来的文件的字节层次的拷贝。我们比较原始文件和提取文件的 MD5 哈希值，就能验证是否一

致。MD5 哈希值应该是一样的，如图 8-42 所示。

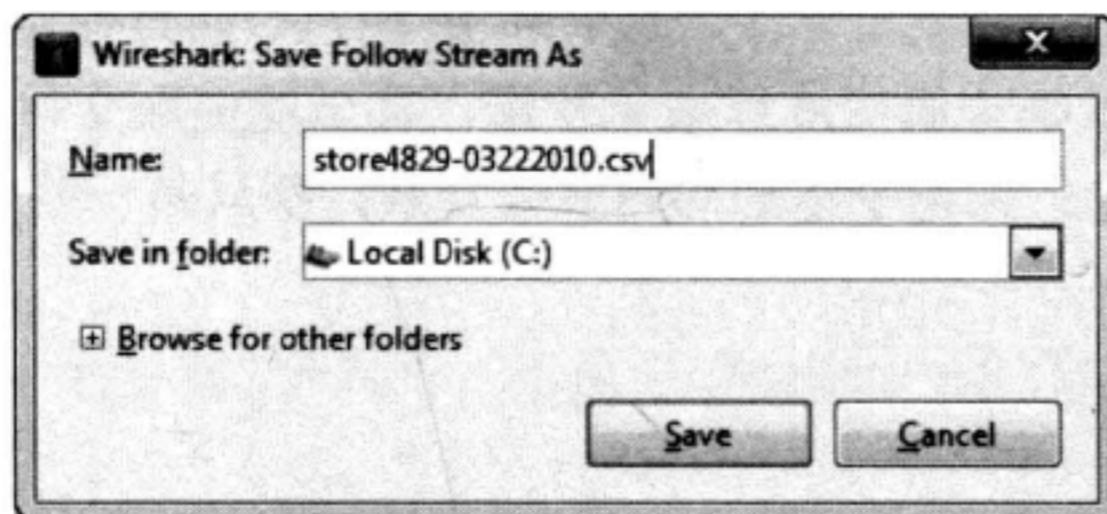


图 8-41 将数据流保存为原始文件名



图 8-42 原始文件和提取文件的 MD5 哈希值相等

比较完文件后，我们可以证明网络并不是应用程序数据库出错的原因。文件从商店传输到收集服务器时是完整的，所以肯定是应用程序处理文件时出错了。

### 3. 学到的知识

数据包层面分析的一个好处是你不必处理杂乱无章的应用程序。糟糕的程序远比好的程序要多，但在数据包层面，它们就无所谓了。在这个案例中，程

程序员会担忧应用程序所依赖的所有不明组件。但最终，他那耗费数百行代码写成的数据传输不过就是 FTP、TCP 和 IP 而已。通过使用我们了解的基本协议知识，我们可以确认通信过程毫无差错，甚至能提取文件证明网络正常。记住这个关键的结论：无论手里的问题多么复杂，都只是一些数据包而已。

## 8.4 小结

在本章，我们讲述了几个基本场景，数据包分析使我们更好地理解通信故障。通过分析常见协议，我们可以及时查出并解决网络问题。你在现实网络中可能不会遇到跟这里完全相同的场景，但本章讨论的分析技术应该能给你一些有益的启发。

# 第9章

## 让网络不再卡



作为一名网络管理员，你将花费很多时间用于修复运行缓慢的计算机和服务。但是人们抱怨网络缓慢，并不意味着就是网络的问题。

在开始处理网络缓慢的问题之前，你首先要确定网络是否真的很慢。你将在本章中学到这些技巧。

首先，我们会讨论 TCP 的错误恢复和流量控制机制。然后，我们会探索如何检测网络缓慢的根源。最后，我们会讨论用基线测试网络以及网络上运行的设备与服务的方法。读完本章后，你在识别、诊断、解决慢速网络方面，应该会有非常大的进步。

---

### 注意

很多技术都可以用来排除网络缓慢故障。本章内容主要集中在 TCP，因为

---

在大多数时间你只需要面对它。TCP 允许你执行被动的回溯分析，而不用生成额外的流量（比如 ICMP）。

---

## 9.1 TCP 的错误恢复特性

TCP 的错误恢复特性是我们定位、诊断，并最终修复网络高延迟的最好工具。在计算机网络中，“延迟”是数据包传输与接收时间差的衡量参数。

延迟可以被测量为单程延迟（从单个来源到一个目的地）或往返延迟（从来源到达目的地并返回来源）。当设备间通信很快，并且数据包从一端点到另一端点所花时间很少时，就说通信是低时延的。相反，当数据包在来源和目的地间传输要花费大量时间时，就说通信是高时延的。高时延是所有珍视自己声誉（以及工作）的网络管理员们的头号敌人。

在第 6 章中，我们讨论了 TCP 如何使用序号和确认号保证可靠地传递数据包。在本章，我们将再次关注序号和确认号，观察当高时延导致这些号码乱序抵达（或根本没有接收到）时，TCP 是如何响应的。

### 9.1.1 TCP 重传

重传数据包是 TCP 最基本的错误恢复特性之一，它被设计用来对付数据包丢失。

数据包丢失可能有很多原因，包括出故障的应用程序、流量负载沉重的路由器，或者临时性的服务中断。数据包层次上的移动速度非常快，而且数据包丢失通常是暂时的，因此 TCP 能否检测到数据包丢失并从中恢复显得至关重要。

决定是否有必要重传数据包的主要机制叫做重传计时器。这个计时器负责维护一个叫重传超时（Retransmission timeout, RTO）的值。每当使用 TCP 传输一个数据包时，就启动重传计时器。当收到这个数据包的 ACK 时，计时器停止。从发送数据包到接收 ACK 确认之间的时间被称为往返时间（Round-trip time, RTT）。将若干个这样的时间平均下来，可算出最终的 RTO 值。

在最终算出 RTO 值之前，传输操作系统将一直依赖于默认配置的 RTT 值。此项设定用于主机间的初始通信，并基于接收到的数据包 RTT 进行调整，以形成真正的 RTO。

一旦 RTO 值确定下来，重传定时器就被用于每个传输的数据包，以确定数据包是否丢失。图 9-1 阐述了 TCP 重传过程。

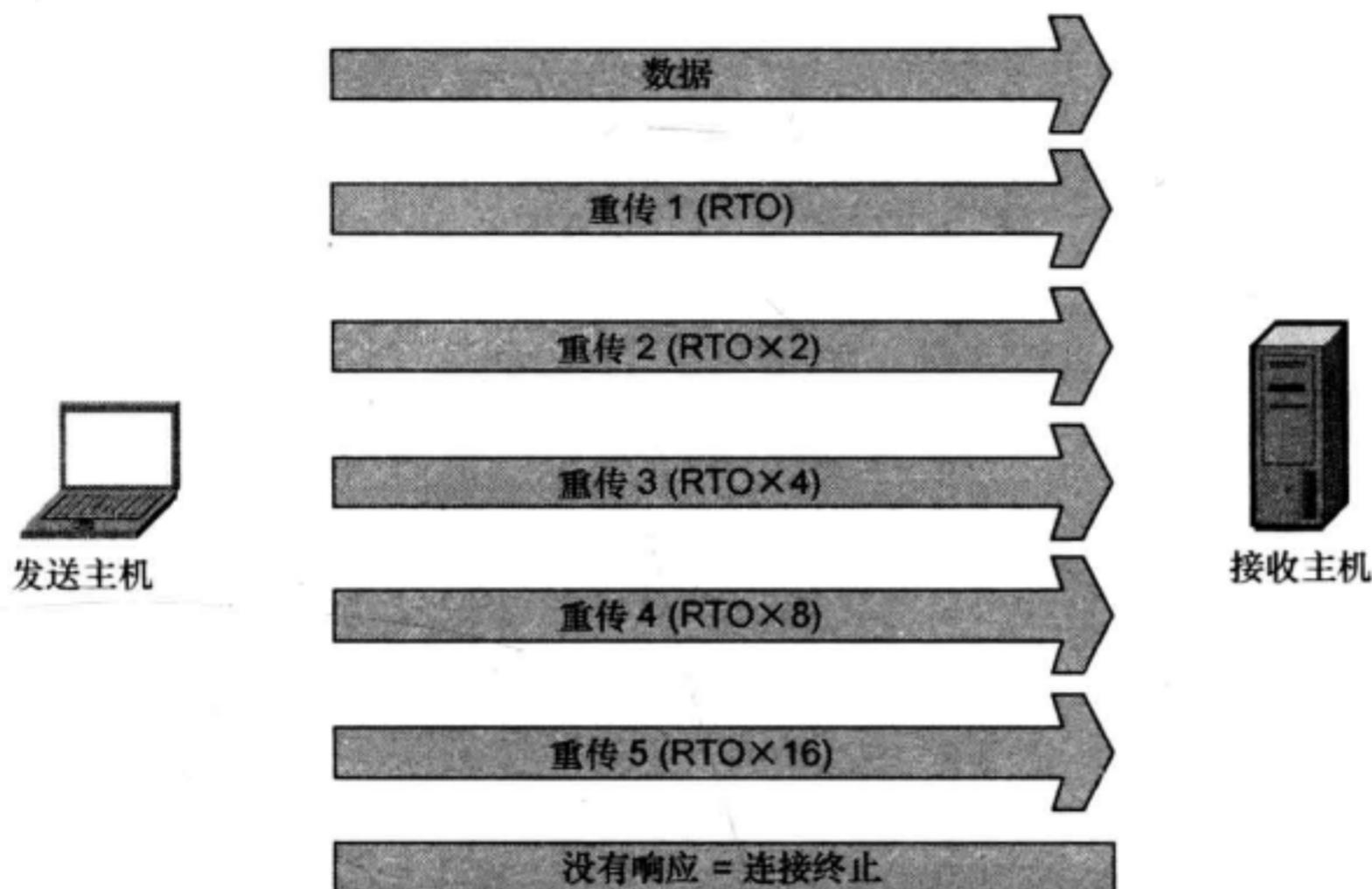


图 9-1 TCP 重传过程的概念视图

当数据包被发送出去，但接收方没有发送 TCP ACK 数据包时，传输主机就假设原来的数据包丢失了，并重传它。重传之后，RTO 值翻倍；如果在到达极限值之前一直没有接收到 ACK 数据包，那将发生另一次重传。如果下一次重传还是没有收到 ACK，RTO 值将翻倍。每次重传，RTO 值都将翻倍，这个过程会持续到收到一个 ACK 数据包，或者发送方达到配置的最大重传次数为止。

最大重传次数取决于传输操作系统上的配置。默认情况下，Windows 主机最多重传 5 次。大部分 Linux 主机则默认最多重传 15 次。这个选项在两个操作系统中都是可配置的。

要看 TCP 重传的例子，请打开 `tcp_retransmissions.pcap` 文件，它包含了 6 个数据包。第一个数据包如图 9-2 所示。

这是一个 TCP PSH/ACK 数据包❶，包含 648 字节的数据❷，从 10.3.30.1 发送到 10.3.71.7 ❸。这是一个典型的数据包。

在正常情况下，你会期待发送第一个数据包之后，很快就能看到响应的 TCP ACK 数据包。然而，在这个例子中，下一个数据包是一次重传。通过在 Packet List 面板中查看这个数据包，你就能得出这个结论。Info 列明确表明了 [TCP Retransmission]，并且这个数据包以黑底红字出现。图 9-3 显示了 Packet List 面板中列出的重传例子。

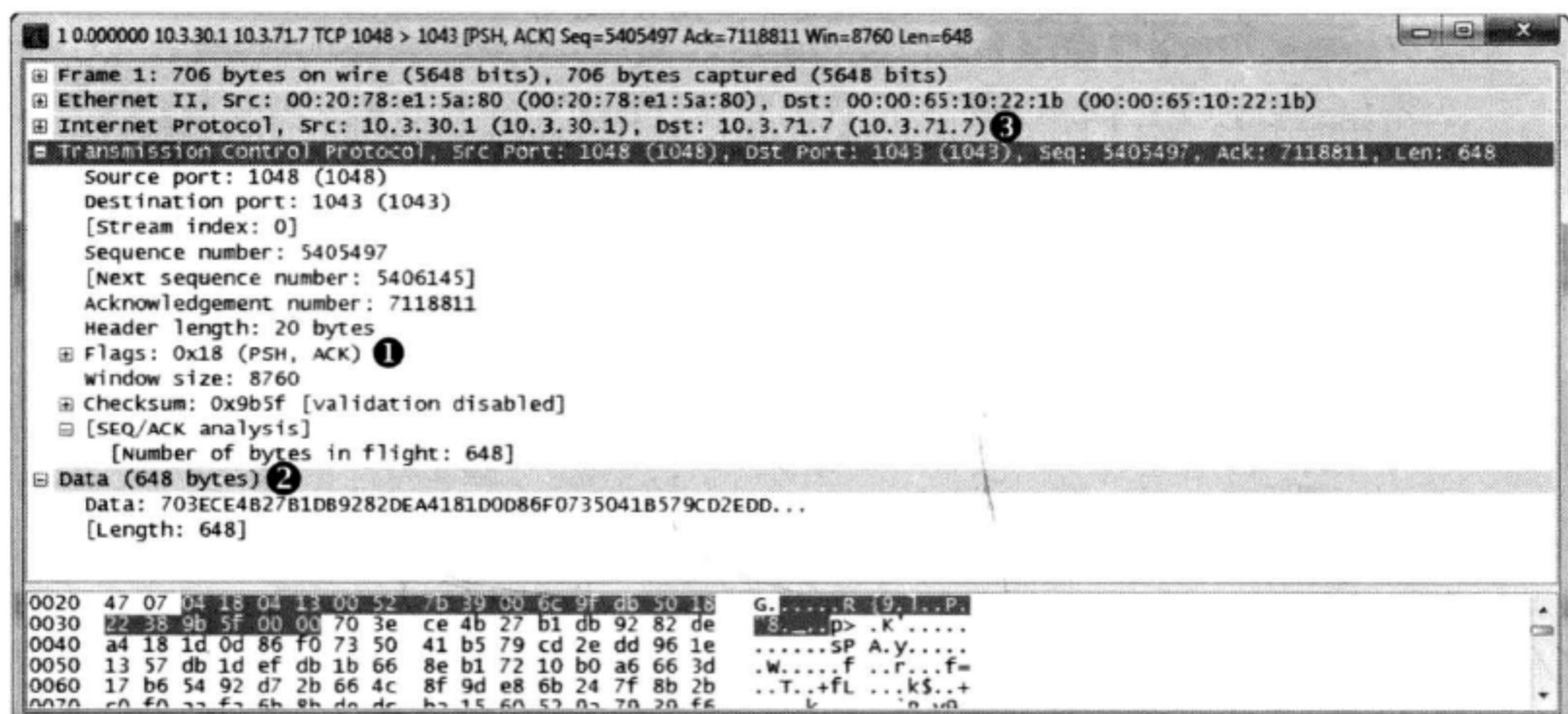


图 9-2 包含数据的简单 TCP 数据包

| No. | Time     | Source    | Destination | Protocol | Info  |
|-----|----------|-----------|-------------|----------|---|
| 1   | 0.000000 | 10.3.30.1 | 10.3.71.7   | TCP      | 1048 > 1043 [PSH, ACK] Seq=5405497 Ack=7118811 Win=8760 Len=648         |
| 2   | 0.206000 | 10.3.30.1 | 10.3.71.7   | TCP      | TCP Retransmission 1048 > 1043 Seq=5405497 Ack=7118811 Win=8760 Len=648 |
| 3   | 0.412000 | 10.3.30.1 | 10.3.71.7   | TCP      | TCP Retransmission 1048 > 1043 Seq=5405497 Ack=7118811 Win=8760 Len=648 |
| 4   | 0.618000 | 10.3.30.1 | 10.3.71.7   | TCP      | TCP Retransmission 1048 > 1043 Seq=5405497 Ack=7118811 Win=8760 Len=648 |
| 5   | 0.824000 | 10.3.30.1 | 10.3.71.7   | TCP      | TCP Retransmission 1048 > 1043 Seq=5405497 Ack=7118811 Win=8760 Len=648 |

图 9-3 Packet List 面板中的重传

如图 9-4 所示, 你也可以通过查看 Packet Details 和 Packet Bytes 面板确定它是否是重传数据包。

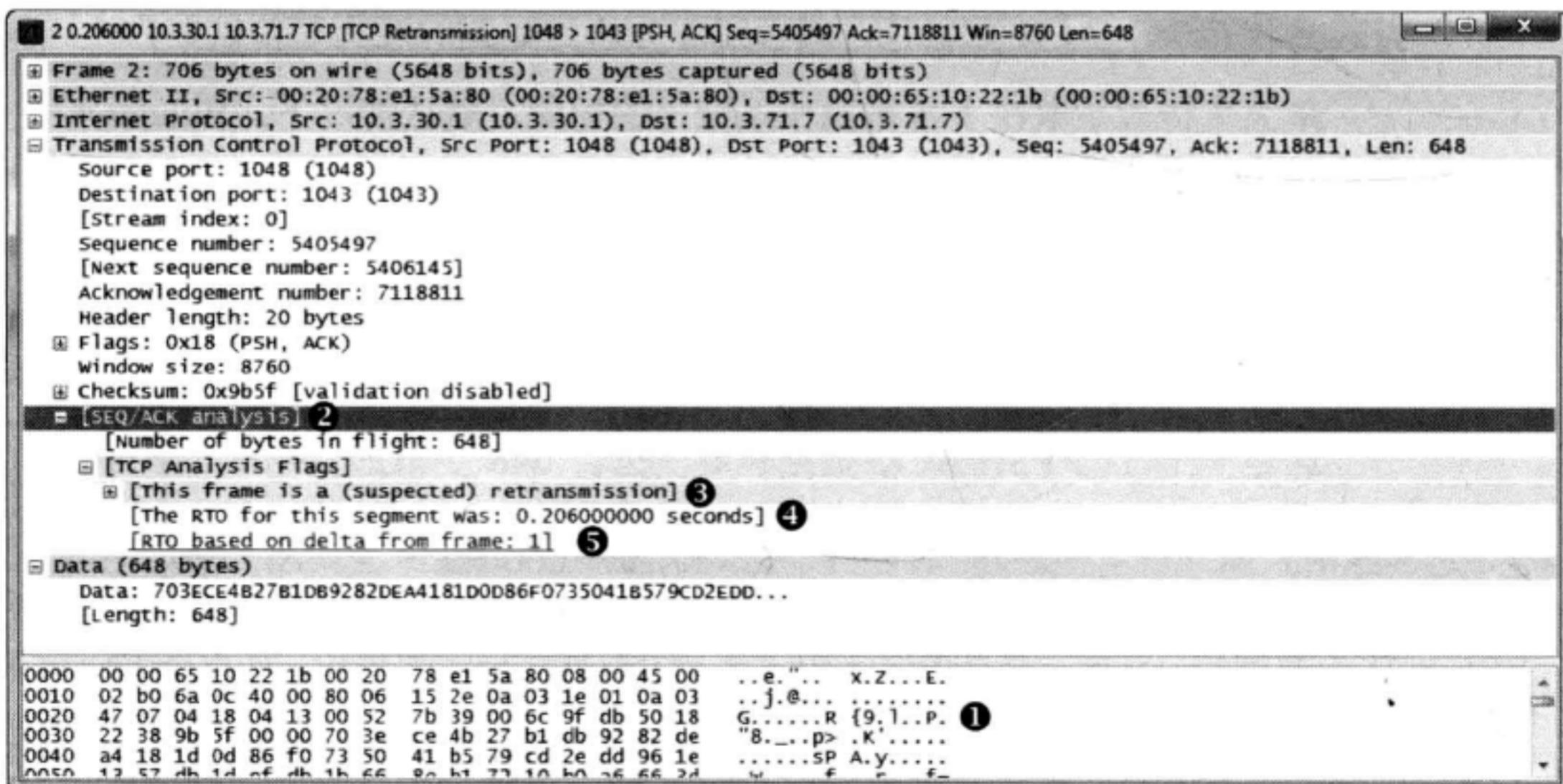


图 9-4 一个重传数据包

注意除了 IP identification 和 Checksum 域之外，这个数据包与最初的数据包完全一致。为了验证这个结论，在 Packet Bytes 面板中比较这个重传数据包和最初的数据包❶。

在 Packet Details 面板中，注意到重传数据包的 SEQ/ACK Analysis 标题下有一些额外的信息❷。这个有用的信息是由 Wireshark 提供的，实际上并不包含在数据包里。SEQ/ACK analysis 告诉我们这确实是一个重传❸，RTO 值 0.206s ❹ 是基于与数据包 1❺ 的时间差值算出来的。

查看剩下的数据包应该是类似的结果，唯一的不同在于 IP identification 和 Checksum 域，以及 RTO 值。为了显示每个数据包之间的时间间隔，如图 9-5 所示，查看 Packet List 面板的 Time 列。在这里，你将看到每一次重传后 RTO 值翻倍，时间呈指数增长。

| No. | Time     |
|-----|----------|
| 1   | 0.000000 |
| 2   | 0.206000 |
| 3   | 0.600000 |
| 4   | 1.200000 |
| 5   | 2.400000 |
| 6   | 4.805000 |

图 9-5 Time 列显示了 RTO 值的增长

传输设备使用 TCP 的重传特性来检测数据包丢失并从中恢复。下一步，我们将查看 TCP 的重复确认特性，它被接收方用于检测数据包丢失并从中恢复。

### 9.1.2 TCP 重复确认和快速重传

当接收方收到乱序数据包时，就发送重复的 TCP ACK 数据包。TCP 在其头部使用序号和确认号字段，以确保数据被可靠接收并以发送顺序重组。

注意

“TCP 数据包”的准确术语其实应该是“TCP 分段”，但大多数人倾向于把它们称为“数据包”。

建立一个新的 TCP 连接时，初始序号（Initial Sequence Number, ISN）是握手过程中交换的最重要信息之一。一旦设置好连接两端的 ISN，接下来传输的每一个数据包都将按照数据载荷的大小增长序号。

举个例子，一台主机的 ISN 是 5000，它发送一个 500 字节的数据包给接收方。一旦接收到此数据包，接收方会根据以下规则响应一个包含确认号 5500 的

TCP ACK 数据包：

接收数据的序号 + 接收数据的字节数 = 发出的确认号

在这个运算中，返回到发送方的确认号实际上就是接收方期待下次接收的数据包序号。图 9-6 中可以看到一个这样的例子。

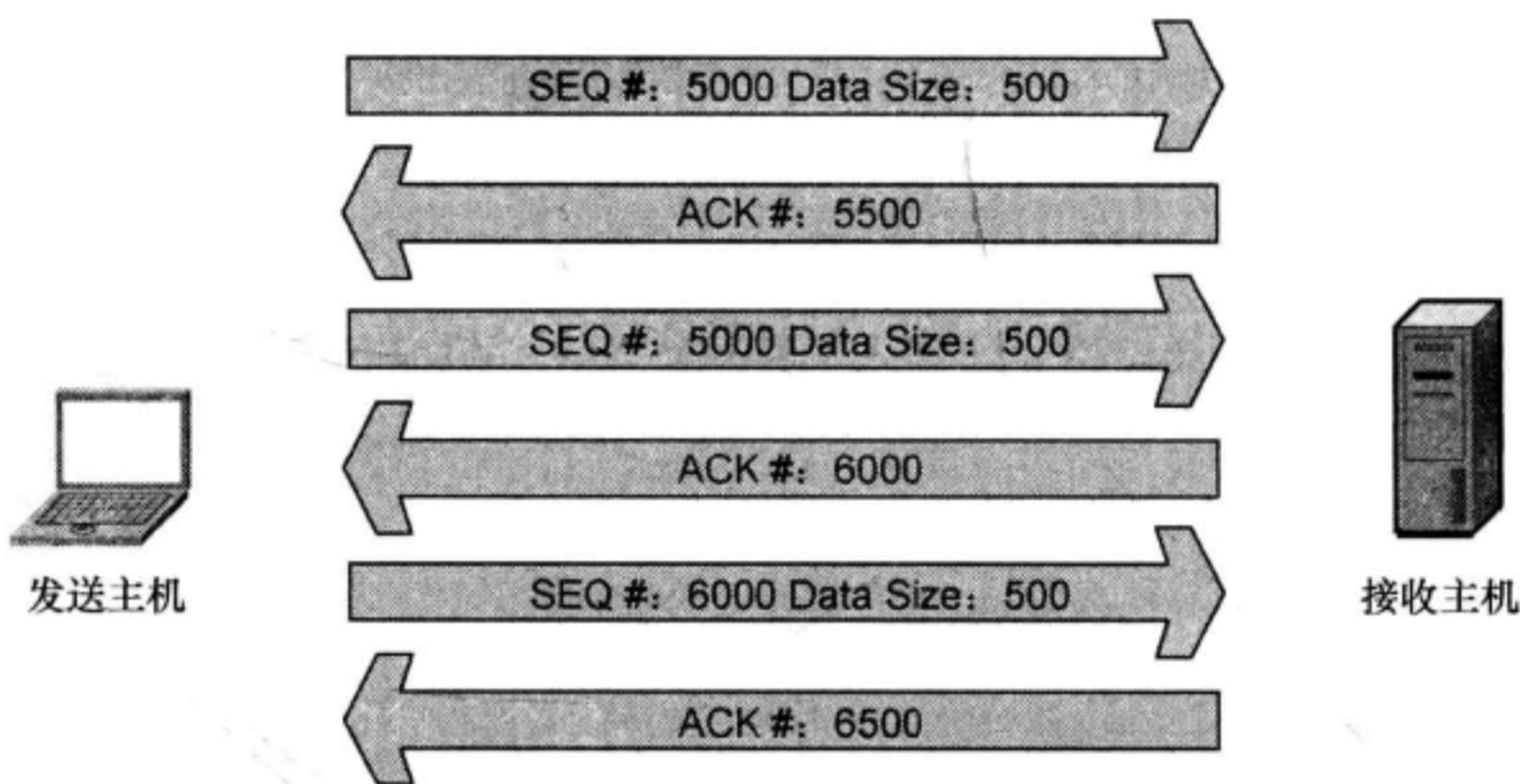


图 9-6 TCP 序号和确认号

序号使数据接收方检测数据包丢失成为可能。当接收方追踪正在接收的序号时，如遇到不合顺序的序号，它就知道数据包丢失了。

当接收方收到一个预料之外的序号时，它假设有一个数据包在传输中丢失了。为了正确地重组数据，接收方必须要得到丢失的数据包，因此它重新发送一个包含丢失数据包序号的 ACK 数据包，以使发送方重传该数据包。

当传输主机收到 3 个来自接收方的重复 ACK 时，它假设这个数据包确实在传输中丢失了，并立刻发送一个快速重传。一旦触发快速重传，其他所有正在传输的数据包都要靠边，直到把快速重传数据包发送出去为止。图 9-7 描述了这个过程。

你将在 `tcp_dupack.pcap` 文件中发现重复 ACK 和快速重传的例子。捕获记录中的第一个数据包如图 9-8 所示。

在网络中这个 TCP ACK 数据包从数据接收方（172.31.136.85）去往发送方（195.81.202.68）①，会对在前一个数据包（没有包含在该捕获文件中）中发送的数据进行确认。

---

#### 注意

默认情况下，Wireshark 使用相对序号来简化对这些数字的分析，但在接下来的几节中，并未在例子和截图中使用这个特性。使用如下方法可关闭此项

功能，选择 **Edit->Preferences**，在 Preferences 窗口中选择 **Protocols**，然后选择 **TCP** 区段。最后取消 **Relative sequence numbers** 和 **window scaling** 旁边的复选框。

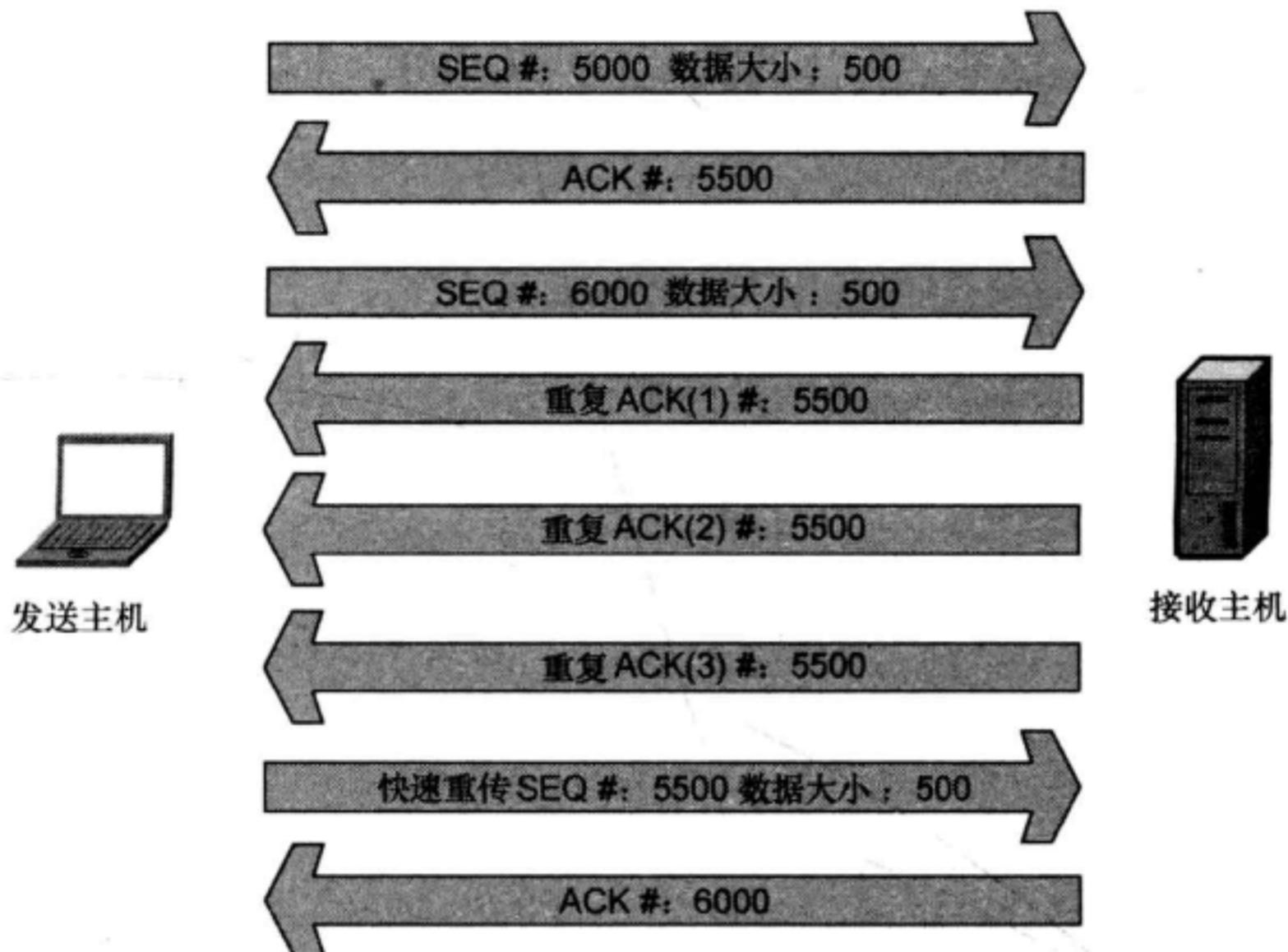


图 9-7 来自接收方的重复 ACK 导致快速重传

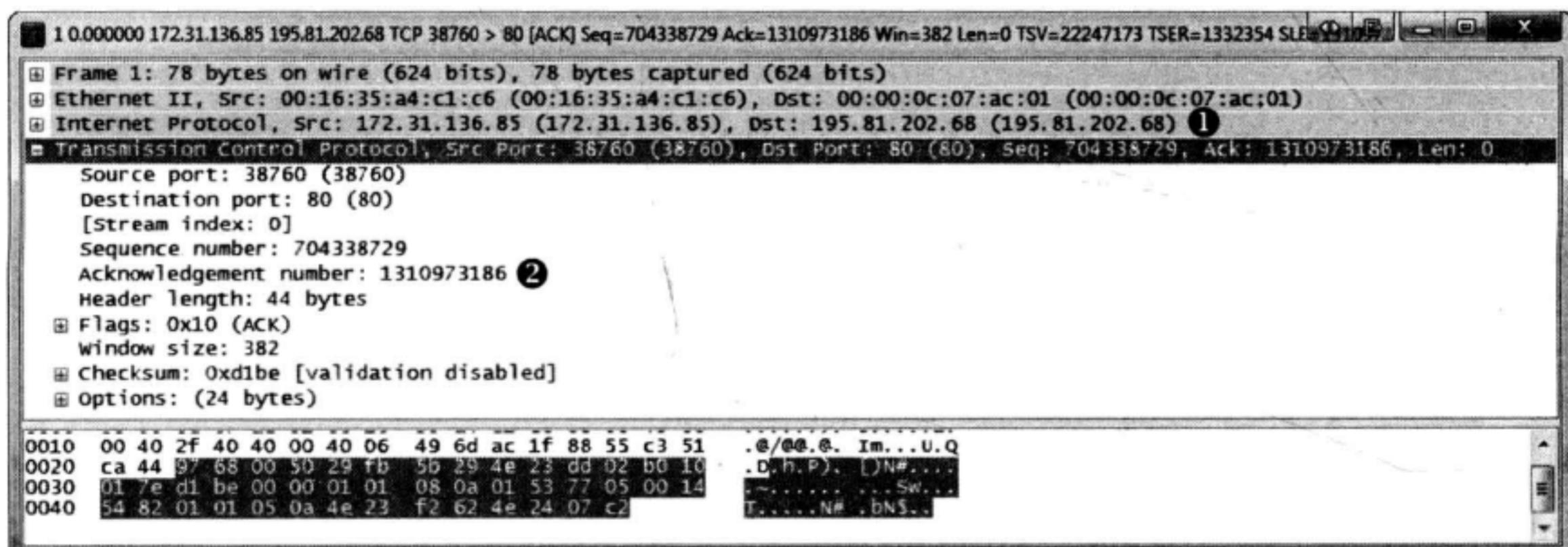


图 9-8 ACK 显示了下一个期待的序号

此数据包中的确认号是 1310973186②，这应该是接收的下一个数据包的序号，如图 9-9 所示。

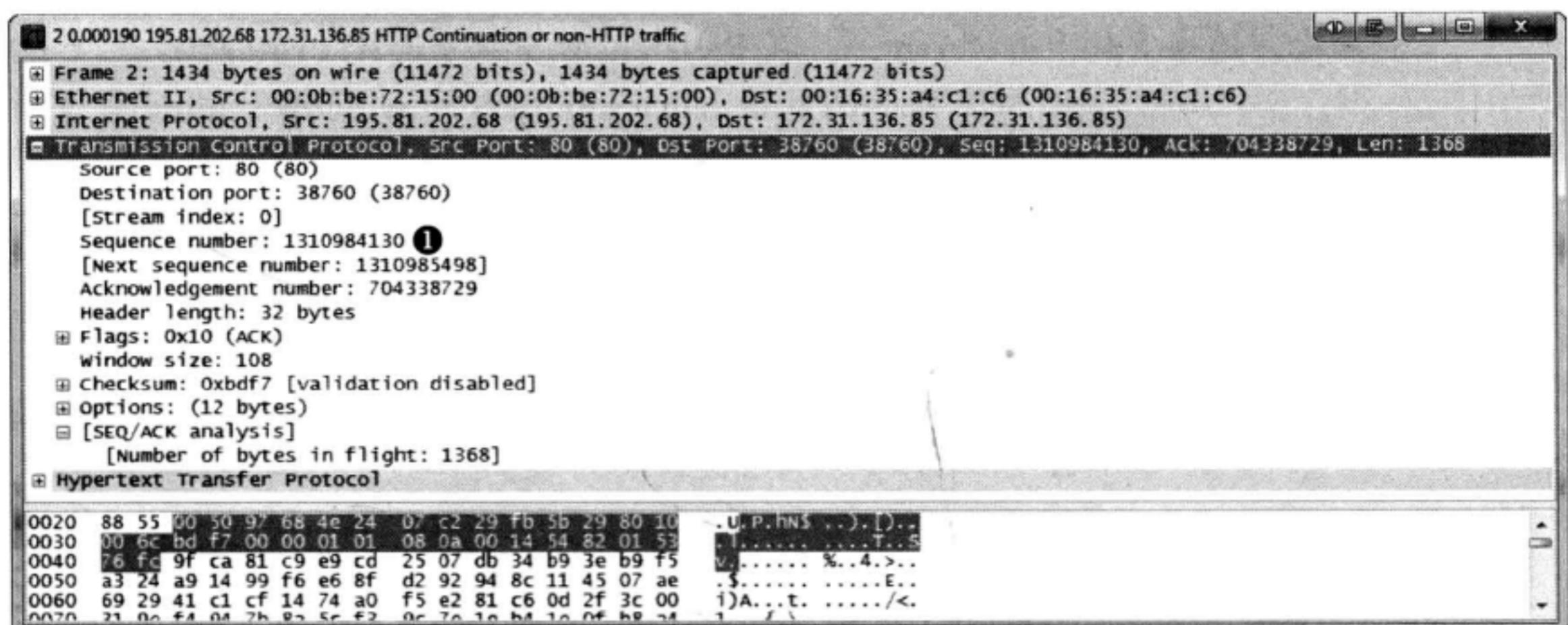


图 9-9 此数据包的序号与预料的不同

对我们和接收方而言都是一种不幸,下一个数据包的序号是 1310984130 ❶,并非是我们所期待的。这表明期待的数据包在传输中莫名其妙地丢失了。如图 9-10 所示,接收主机注意到这个数据包的序号不符,就在捕获记录的第 3 个数据包中发送一个重复的 ACK。



图 9-10 第一个重复 ACK 数据包

通过查看以下信息的任意一个,你就可以确定这是一个重复的 ACK 数据包。

- Packet Details 面板中的 Info 列。这个数据包以黑底红字呈现。

- SEQ/ACK Analysis 标题下的 Packet Details 面板。若展开此标题，你会发现这个数据包被列为数据包 1 的重复 ACK。

如图 9-11 所示，接下来的几个数据包继续这个过程。

| No. | Time       | Source        | Destination   | Protocol | Info   |
|-----|------------|---------------|---------------|----------|--|
| 1   | 0.000000   | 172.31.136.85 | 195.81.202.68 | TCP      | 38760 > 80 [ACK] Seq=704338729 Ack=1310973186 Win=382 Len=0 TSV=22247173 TSER= |
| 2   | 0.000190   | 195.81.202.68 | 172.31.136.85 | HTTP     | Continuation or non-HTTP traffic   |
| ②   | 4 0.000093 | 195.81.202.68 | 172.31.136.85 | HTTP     | Continuation or non-HTTP traffic   |
| ③   | 6 0.000121 | 195.81.202.68 | 172.31.136.85 | HTTP     | Continuation or non-HTTP traffic   |

图 9-11 由于乱序数据包的影响，生成了额外的重复 ACK

捕获文件的第 4 个数据包是发送主机以错误序号发送的另一个数据区块①。因此，接收主机发送第二个重复 ACK②。接收方又收到一个包含错误序号的数据包③。这导致它传输第三个、也是最后一个重复 ACK④。

发送方收到来自接收方的第三个重复 ACK 之后，就强制停止所有的数据包传输，并重新发送丢失的数据包。图 9-12 显示了丢失数据包的快速重传。

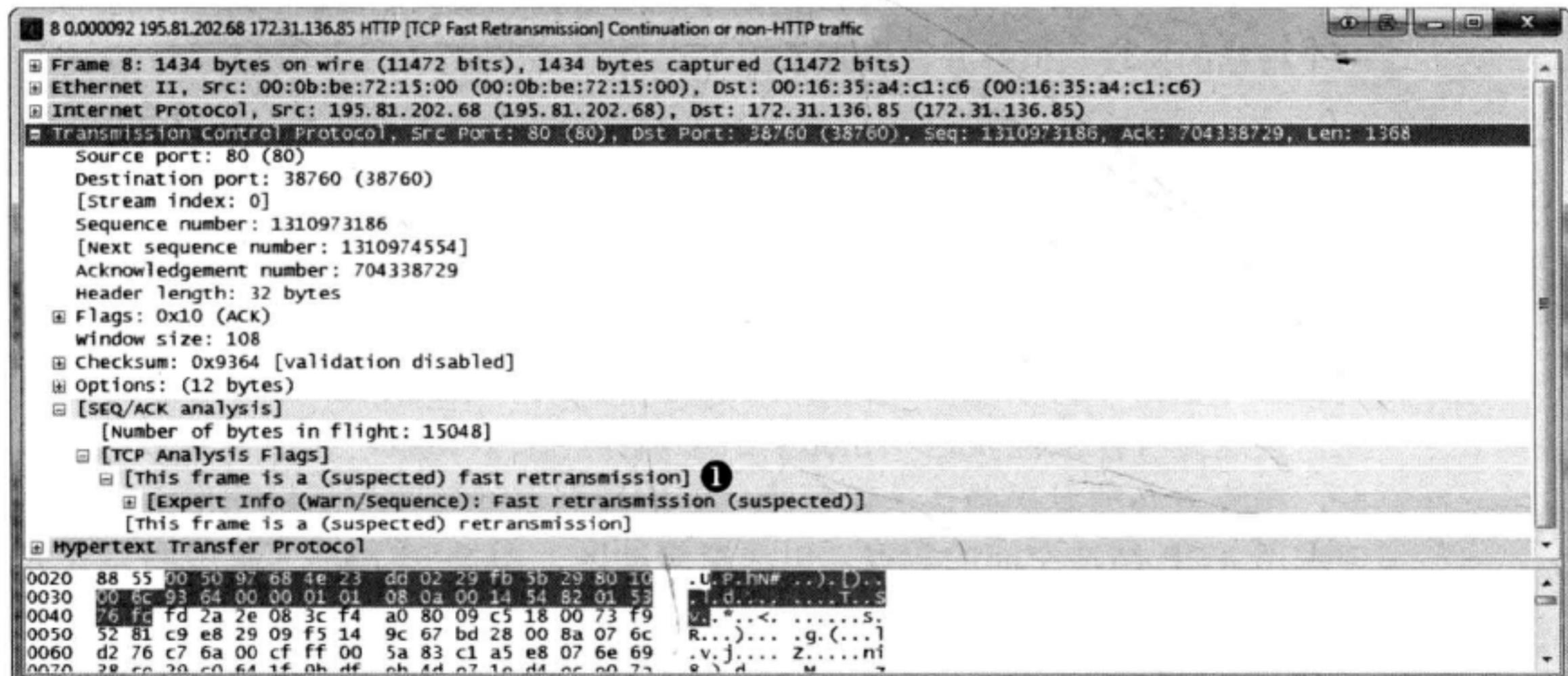


图 9-12 重复 ACK 引发了丢失数据包的快速重传

在 Packet List 面板的 Info 列中再次出现了重传数据包。正如前面的例子，数据包被清楚地标记为黑底红字。这个数据包的 SEQ/ACK 分析部分告诉我们这有可能是一次快速重传①（再次注意，数据包的快速重传标记信息并非数据包本身的值，而是 Wireshark 的功能）。捕获记录的最后一个数据包是确认收到快速重传的 ACK 数据包。

## 注意

当发生数据包丢失时，可能影响 TCP 通信数据流的功能是选择性确认（Selective Acknowledgement）。在上面的捕获记录里，通信双方已经在三次握手过程中协商开启了选择性 ACK。因此，一旦数据包丢失并收到重复 ACK，即使在丢失数据包之后还成功接收了其他数据包，也只需要重传丢失的数据包。如果不启用选择性 ACK，那就必须重新传输丢失数据包之后的每一个数据包。选择性 ACK 使得数据丢失的恢复更加高效。由于大部分现代 TCP/IP 协议栈的实现都支持选择性 ACK，因此你会发现这个功能通常都会被启用。

## 9.2 TCP 流控制

重传和重复 ACK 都是 TCP 反应性的功能，被设计用来从数据包丢失中恢复。如果 TCP 没有包含某些形式的用于预防数据包丢失的前瞻性功能，它将是一个糟糕的协议，但幸好它做到了。

TCP 实现了滑动窗口机制，用于检测何时发生了数据包丢失，并调整数据传输速率加以避免。滑动窗口机制利用数据接收方的接收窗口来控制数据流。

接收窗口是数据接收方指定的值，存储在 TCP 头部中（以字节为单位），它告诉发送设备自己希望在 TCP 缓冲空间中存储多少数据。这个缓冲空间是数据在可以向上传递到等待处理数据的应用层协议之前的临时存储空间。因此，发送方一次只能发送 Window Size 域指定的数据量。为了传输更多的数据，接收方必须发送确认，以告知之前的数据已经接收到了。它也必须要处理占用 TCP 缓冲空间的数据，以清空缓冲区。图 9-13 阐明了接收窗口是如何工作的。

在图 9-13 中，客户端正在向接收窗口大小为 5000 字节的服务器发送数据。客户端发送了 2500 字节，将服务器的缓冲空间减少到 2500 字节，然后再发送 2000 字节，进而将缓冲区减少到 500 字节。然后服务器送出这些数据的确认。它处理缓冲区的数据，得到可用的空缓冲区。这个过程不断重复，客户端又发送了 3000 字节和另外 1000 字节，将服务器的缓冲区减少到 1000 字节。客户端再次确认这些数据，并处理缓冲区的内容。

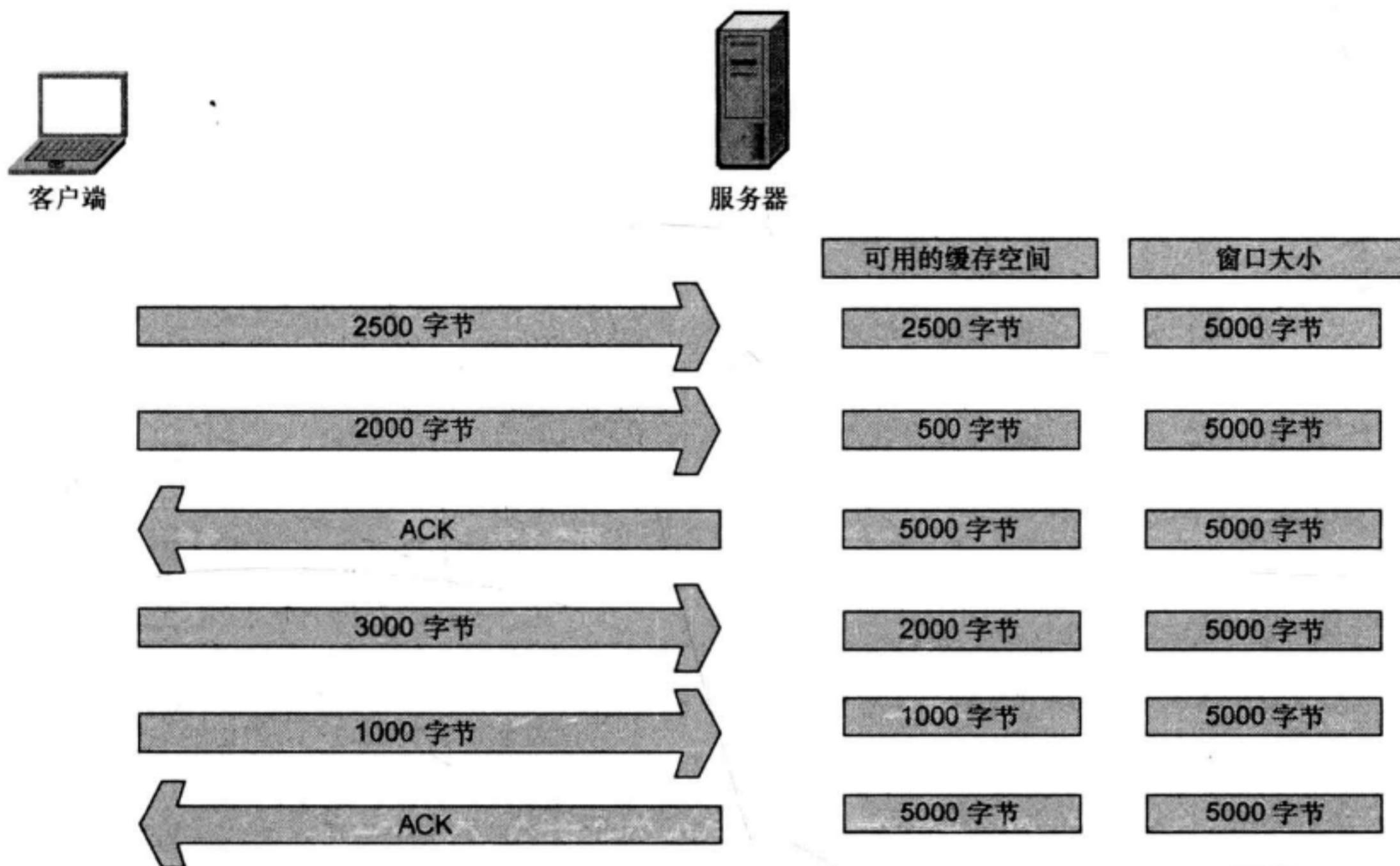


图 9-13 接收窗口使得接收方不被数据淹没

### 9.2.1 调整窗口大小

调整窗口大小的过程相当明确，但有时候它也不尽人意。每当接收到数据时，TCP 栈就生成并发送一个确认作为响应，但是接收方不可能总是迅速地处理缓冲区的数据。

当一台繁忙的服务器处理来自多个客户端的数据包时，服务器可能会缓慢地清空缓冲区，腾不出空间来接收新数据。如果没有流量控制，这将导致数据包丢失和数据损坏。幸好，当服务器太繁忙，以致不能以接收窗口宣告的速率处理数据时，它可以调整接收窗口的大小。它通过减小向发送方返回 ACK 数据包的 TCP 头部窗口大小值，达到这个目的。图 9-14 展示了这样的例子。

在图 9-14 中，服务器一开始声明的窗口大小是 5000 字节。客户端发送了 2000 字节，紧接着再发送 2000 字节，只留下 1000 字节的可用缓冲空间。服务器意识到它的缓冲区很快就要被塞满了。它知道如果按此速率传输数据，那数据包很快就会丢失。为了校正这个问题，服务器向客户端发送一个确认，包含更新的窗口大小 1000 字节。因而，客户端会减少发送的数据，服务器端可以按照能接受的速率处理缓冲区的内容，从而允许数据恒定流动。

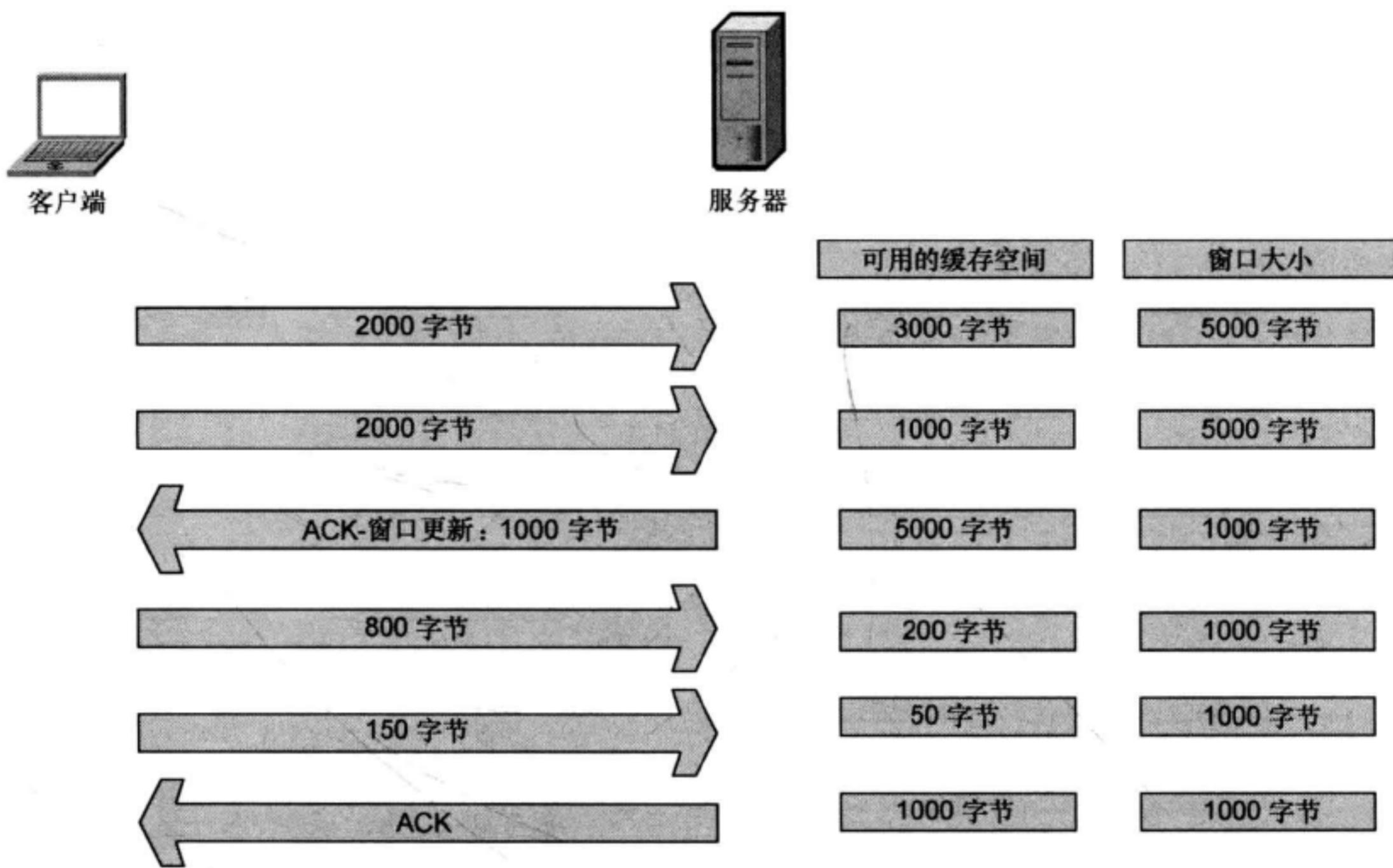


图 9-14 服务器变繁忙时，可以调整窗口大小

窗口重设过程是双向工作的。当服务器能更快地处理数据时，它可以发送一个 ACK 数据包，指明更大的窗口大小。

## 9.2.2 用零窗口通知停止数据流

某些情况下，服务器可能无法处理客户端发送的数据。这可能是因为内存不足、缺少处理能力或者其他问题。这可能导致数据包中止、通信过程停止，但接收窗口能够将负面影响最小化。

当出现这种情况时，服务器可以发送一个数据包，指明窗口大小是零。当客户端收到这个数据包时，它会停止所有数据传输，但仍通过传输“保活数据包”保持与服务器的连接。客户端周期性地发送保活数据包，以检查服务器接收窗口的状态。一旦服务器能再次处理数据，它会响应一个非零的窗口大小以恢复通信。图 9-15 显示了零窗口通知的例子。

在图 9-15 中，服务器开始用 5000 字节的窗口接收数据。在从客户端接收到 4000 字节的数据后，服务器开始承受沉重的处理负担，不能再处理来自客户端的任何数据。服务器发送一个数据包，将窗口大小域设为 0。客户端暂停数据

传输，并发送一个保活数据包。收到保活数据包之后，服务器响应一个数据包，通知客户端它现在可以接收数据了，而且它现在的窗口大小是 1000 字节。客户端继续发送数据。

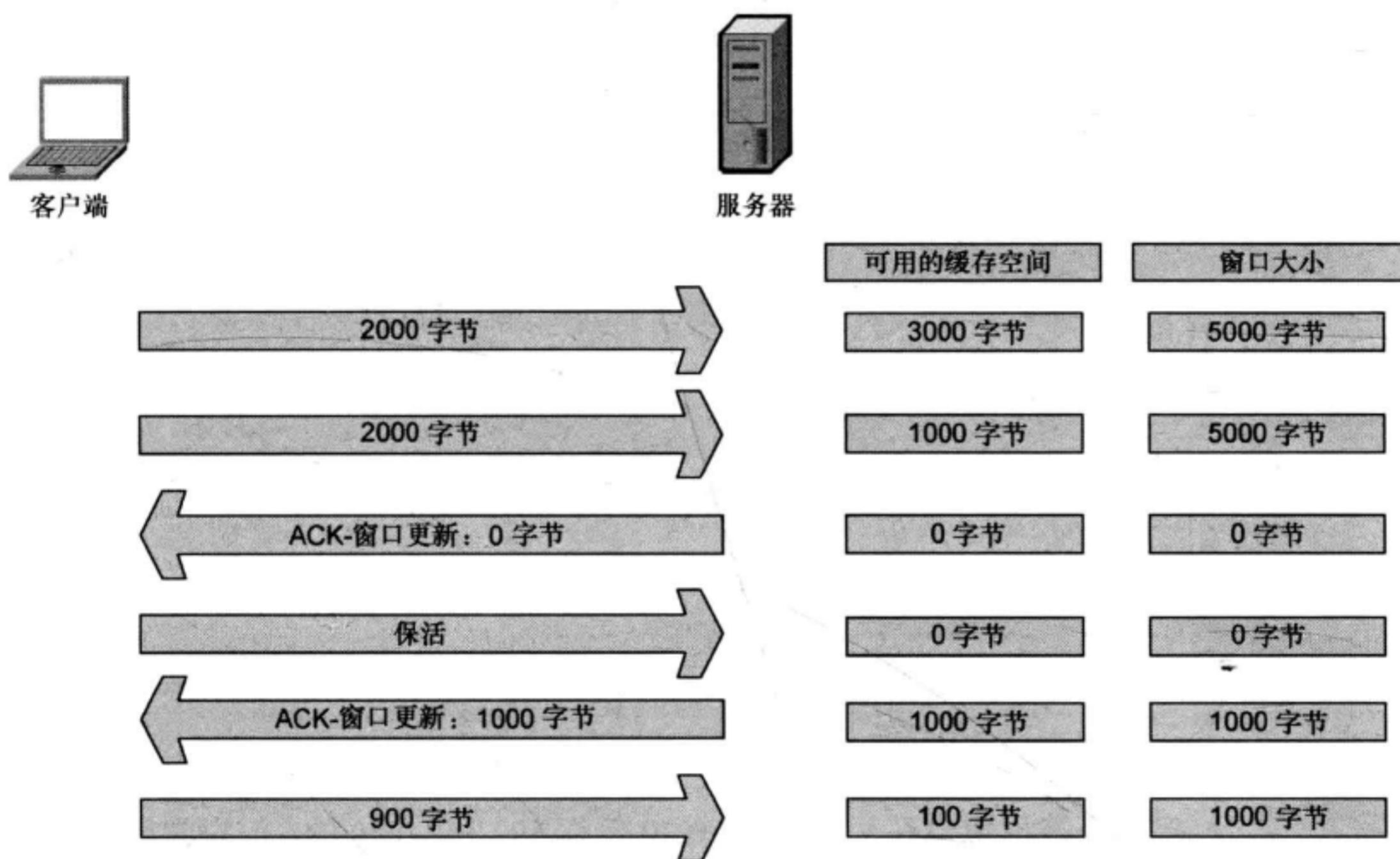


图 9-15 当窗口大小设为 0 字节时，数据传输就停止了

### 9.2.3 TCP 滑动窗口实战

看完 TCP 滑动窗口的理论之后，我们将在捕获文件 `tcp_zerowindowrecovery.pcap` 中探究它。

在这个文件中，我们从 192.168.0.20 发送给 192.168.0.30 的几个 TCP ACK 数据包开始。我们主要对 Windows Size 域感兴趣，可以在 Packet List 面板的 Info 列以及 Packet Details 面板的 TCP 头部看到它。从图 9-16 可以立即发现，在前面的 3 个数据包中，这个域的值不断减小。

| No. | Time     | Source       | Destination  | Protocol | Info   |
|-----|----------|--------------|--------------|----------|--|
| 1   | 0.000000 | 192.168.0.20 | 192.168.0.30 | TCP      | 2235 > 1720 [ACK] Seq=1422793785 Ack=2710996659 Win=8760 Len=0 |
| 2   | 0.000237 | 192.168.0.20 | 192.168.0.30 | TCP      | 2235 > 1720 [ACK] Seq=1422793785 Ack=2710999579 Win=5840 Len=0 |
| 3   | 0.000193 | 192.168.0.20 | 192.168.0.30 | TCP      | 2235 > 1720 [ACK] Seq=1422793785 Ack=2711002499 Win=2920 Len=0 |

图 9-16 这些数据包的窗口大小在递减

这个值从第一个数据包的 8760 字节减少到第二个数据包的 5840 字节，接着又减为第三个数据包的 2920 字节①。窗口大小值递减是主机延迟增加的典型指标。注意一下 Time 列的信息，这是在极短的时间内发生的②。当窗口大小像这样快速减小时，它很可能会减至零，如图 9-17 所示，数据包 4 正是这样的情形。

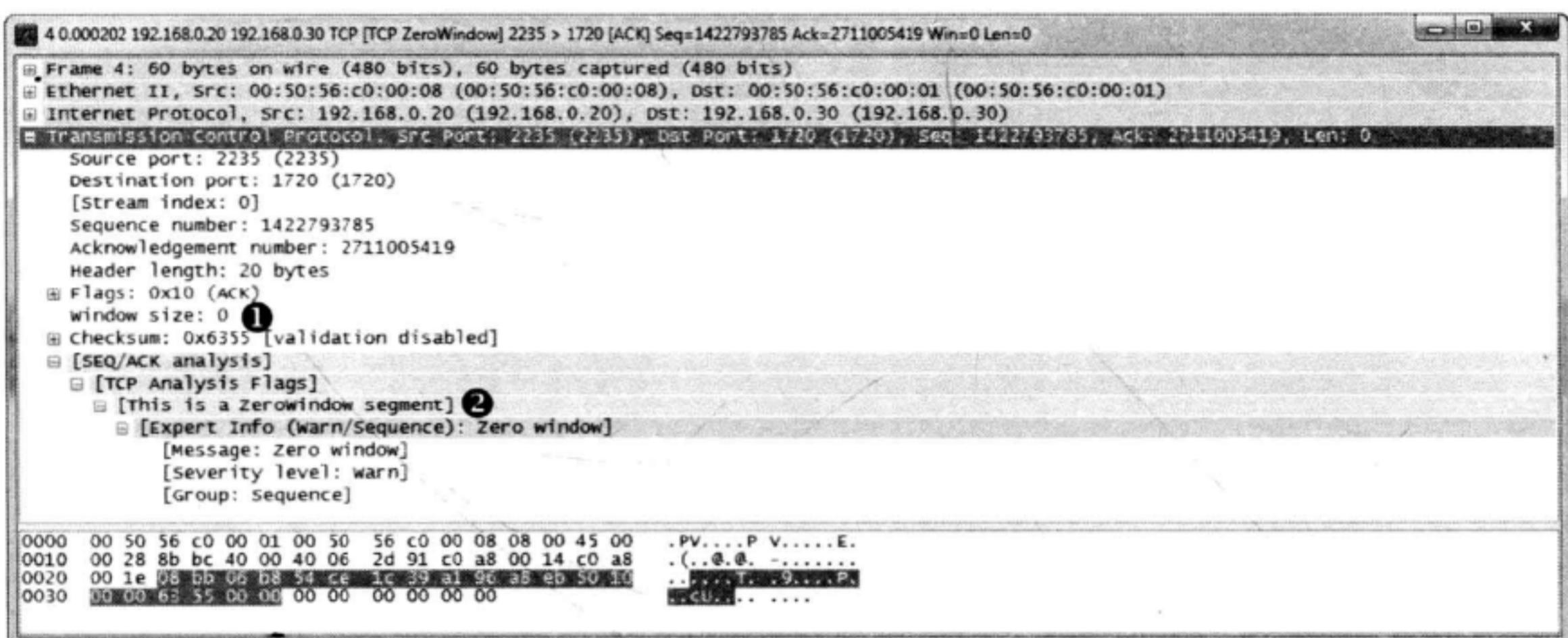


图 9-17 零窗口数据包说明了主机不能再接收任何数据

第四个数据包也是从 192.168.0.20 发往 192.168.0.30 的，但它的目的是告诉 192.168.0.30 它不能再接收任何数据。在 TCP 头部就可以看到这个数值 0①，而且 Wireshark 也在 Packet List 面板的 Info 列以及 TCP 头部 SEQ/ACK Analysis 部分，告诉我们这是一个零窗口数据包②。

一旦发送了零窗口数据包，192.168.0.30 的设备就不再发送任何数据，直到它从 192.168.0.20 收到一个窗口更新，通知它窗口大小已经增长了。幸好，在这个捕获文件里，导致零窗口的问题是暂时的。因此，如图 9-18 所示，发送的下一个数据包就是窗口更新。

在这个例子中，窗口大小增长到了非常健康的 64,240 字节①。Wireshark 再一次在 SEQ/ACK Analysis 标题下面告诉我们，这是一个窗口更新。

一旦收到这个更新数据包，192.168.0.30 主机就可以再次发送数据，如数据包 6 和 7 所示。这个过程非常迅速。就算它只是多持续一点点时间，也可能会引起网络“打嗝”，导致数据传输变慢或失败。

最后再看看滑动窗口，查看一下 `tcp_zerowindowdead.pcap` 文件。捕获记录

中的第一个数据包是从 195.81.202.68 发送到 172.31.136.85 的正常 HTTP 流量。如图 9-19 所示，紧接着就是一个从 172.31.136.85 返回的零窗口数据包。

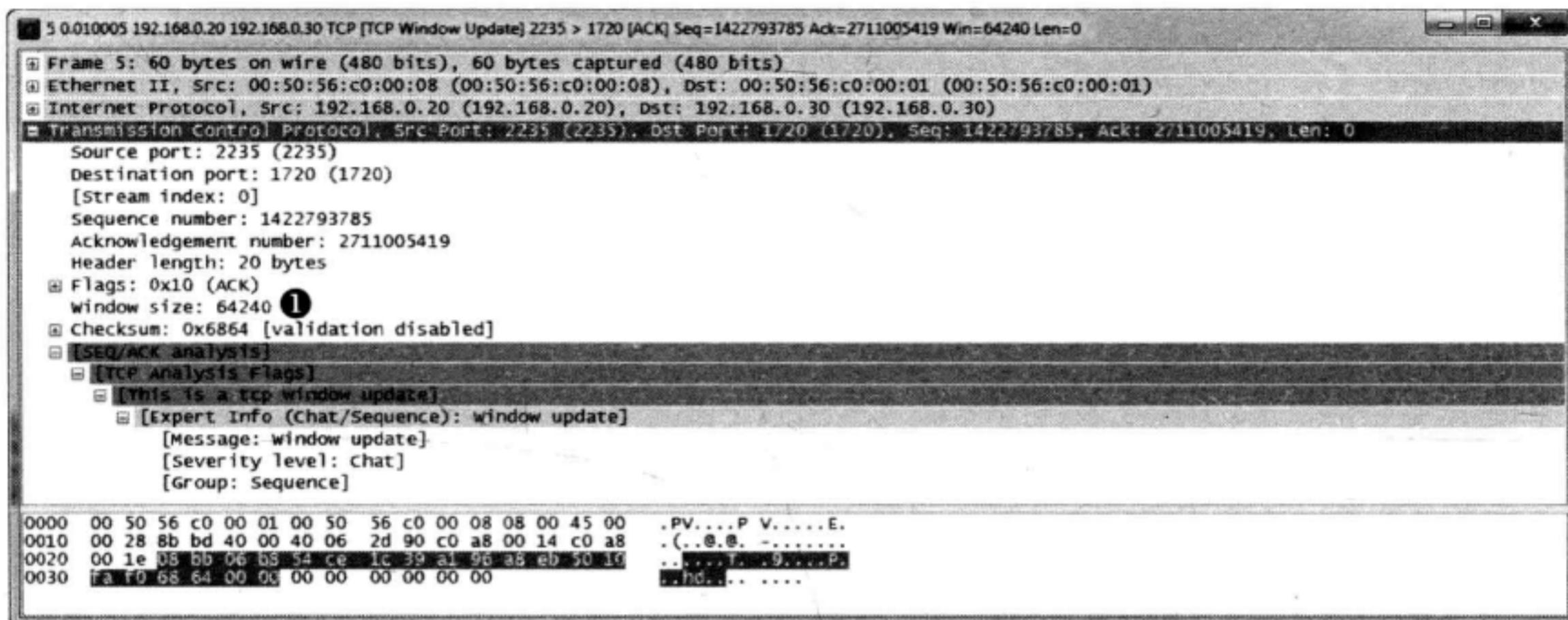


图 9-18 TCP 窗口更新数据包告诉其他主机它又可以传输数据了

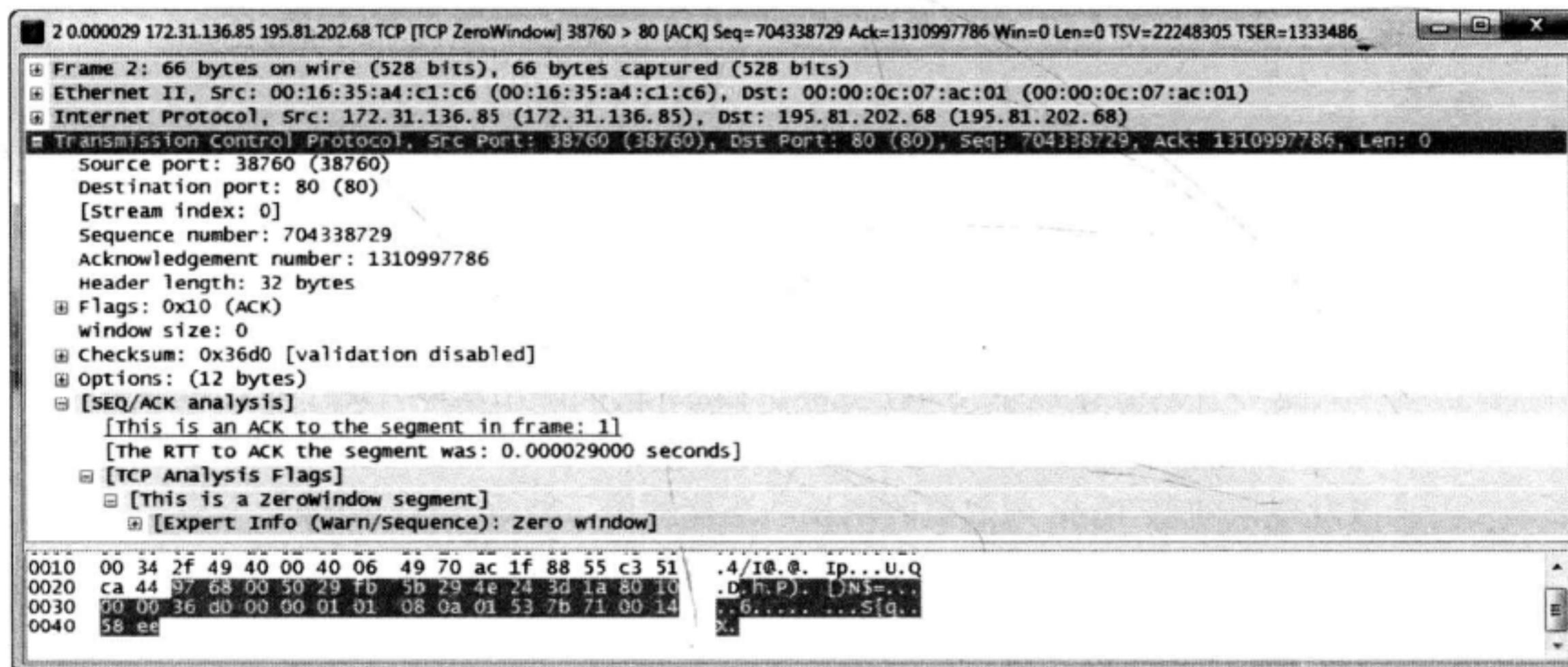


图 9-19 零窗口数据包使数据传输暂停

这看起来与图 9-17 里的零窗口数据包非常相似，但结果却很不相同。在图 9-20 中，我们并没有看到 172.31.136.85 主机发送使通信恢复的窗口更新，而是看到一个保活数据包。

Wireshark 在 Packet Details 面板中 TCP 头部的 SEQ/ACK Analysis 标题下，将这个数据包标记为保活数据包❶。我们从 Time 列可得知，在收到上一个数据

包 3.4s 后，出现了这个数据包。如图 9-21 所示，这个过程又持续了几次，一台主机发送零窗口数据包，另一台则发送保活数据包。

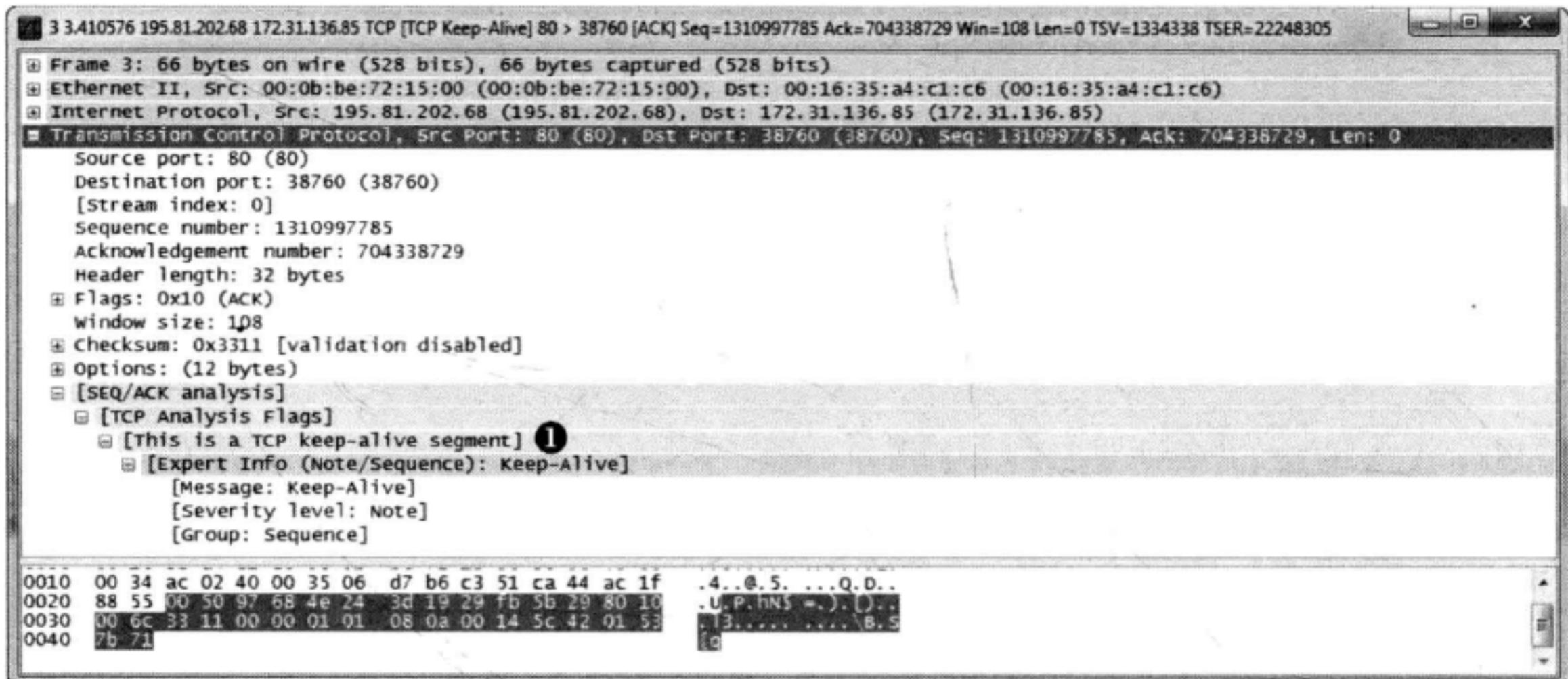


图 9-20 保活数据包保证零窗口主机仍然在线

| No. | Time     | Source         | Destination    | Protocol | Info   |
|-----|----------|----------------|----------------|----------|--|
| 2   | 0.000024 | 172.31.1.36.85 | 195.81.202.68  | TCP      | TCP ZeroWindow 80 > 38760 [ACK] Seq=704338729 Ack=1310997785 Win=0 Len=0 TSV=1334338 TSER=22248305   |
| 3   | 0.000075 | 195.81.202.68  | 172.31.1.36.85 | TCP      | TCP Keep-Alive 80 > 38760 [ACK] Seq=1310997785 Ack=704338729 Win=108 Len=0 TSV=1334338 TSER=22248305 |
| 4   | 0.000031 | 172.31.1.36.85 | 195.81.202.68  | TCP      | TCP ZeroWindow 80 > 38760 [ACK] Seq=704338729 Ack=1310997785 Win=0 Len=0 TSV=1334338 TSER=22248305   |
| 5   | 0.000031 | 195.81.202.68  | 172.31.1.36.85 | TCP      | TCP Keep-Alive 80 > 38760 [ACK] Seq=1310997785 Ack=704338729 Win=108 Len=0 TSV=1334338 TSER=22248305 |
| 6   | 0.000029 | 172.31.1.36.85 | 195.81.202.68  | TCP      | TCP ZeroWindow 80 > 38760 [ACK] Seq=704338729 Ack=1310997785 Win=0 Len=0 TSV=1334338 TSER=22248305   |
| 7   | 0.000031 | 195.81.202.68  | 172.31.1.36.85 | TCP      | TCP Keep-Alive 80 > 38760 [ACK] Seq=1310997785 Ack=704338729 Win=108 Len=0 TSV=1334338 TSER=22248305 |
| 8   | 0.000047 | 172.31.1.36.85 | 195.81.202.68  | TCP      | TCP ZeroWindow 80 > 38760 [ACK] Seq=704338729 Ack=1310997785 Win=0 Len=0 TSV=1334338 TSER=22248305   |

图 9-21 零窗口和保活数据包不断出现

这些保活数据包以 3.4s、6.8s、13.5s 的间隔出现①。这个过程可能会持续相当长的时间，这取决于通信设备采用了哪个操作系统。在这个例子中，你可以发现，随着 Time 列数值的增长，连接暂停了将近 25s。想像一下尝试向域控制器认证或者从网上下载文件时，25s 的延迟真是难以接受！

## 9.3 从 TCP 错误控制和流量控制中学到的

让我们在上下文中讨论重传、重复 ACK、滑动窗口机制。下面是处理延迟问题的一些注意事项。

### 重传数据包

发生重传是因为客户端检测到服务器没有接收到它发送的数据。因此，

你能否看到重传取决于你在分析的通信的哪一端。如果你从服务器端捕获数据，并且它确实没有接收到客户端发送、重传的数据包，那你将被蒙在鼓里，因为你看不到重传的数据包。如果你怀疑服务器端受到了数据包丢失的影响，（可能的话）你应该考虑到在客户端那边捕获流量以观察是否有重传数据包。

### 重复 ACK 数据包

我倾向于把重复 ACK 看作重传的表面对立词 (*pseudo-opposite*)，因为当服务器检测到来自客户端的数据包丢失时就发送它。在多数情况下，你可以在通信两端捕获到重复 ACK 的流量。记住当接收到乱序数据包时才触发重复 ACK。例如，如果服务器只接收到了第一个和第三个数据包，就会发送一个重复 ACK 以引发第二个数据包的快速重传。由于你已经接收到第一个和第三个数据包，那不管什么情况导致第二个数据包丢失都可能只是暂时的，因此在大部分情况下，你可以成功发送并接收重复 ACK。当然，有时候这个情况也未必成立。因此，当你在服务器端怀疑有数据包丢失却没有看到任何重复 ACK 时，可以考虑在通信的客户端捕获数据包。

### 零窗口和保活数据包

滑动窗口与服务器接收、处理数据的故障直接相关。服务器的某些问题可以直接导致窗口大小减少或达到零窗口状态，因此，如果你发现这样的情况，就应该集中调查那里。通常你会在网络通信的两端看见窗口更新数据包。

## 9.4 定位高延迟的原因

有时候，数据包丢失并不是延迟的原因。也许你会发现，当两台主机间通信很慢时，并没有 TCP 重传或重复 ACK 这样的常见特征。在这些情况下，你需要采用其他技术来定位高延迟的原因。

在这里，最有效的办法之一是查看初始连接握手以及接下来的两个数据包。例如，考虑客户端和 Web 服务器之间的一个简单连接，客户端尝试浏览托管在服务器上的一个站点。我们关注通信序列里的前 6 个数据包，包括 TCP 握手、初始 HTTP GET 请求、对这个 GET 请求的确认，以及从服务器发往客户端的第一个数据包。

## 注意

为了重现本节内容,请确保你已经在 Wireshark 中设置了恰当的时间显示格式。在 Wireshark 中选择 **View > Time Display Format > Seconds Since Previous Displayed Packet**。

### 9.4.1 正常通信

我们将在本章的稍后部分详细讨论网络基线。目前,只知道你需要一个正常通信的基线,并与高延迟的情况作比较。在这些例子中,我们将使用 `latency1.pcap` 文件。由于我们已经讨论过了 TCP 握手和 HTTP 通信的细节,在这里我们将跳过它们。实际上,我们根本不再看 **Packet Details** 面板。如图 9-22 所示,我们只关心 **Time** 列。

| No. | Time     | Source        | Destination   | Protocol | Info  |
|-----|----------|---------------|---------------|----------|---|
| 1   | 0.000000 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=2                     |
| 2   | 0.030107 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 WS=6 |
| 3   | 0.000075 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 Win=4218 Len=0                    |
| 4   | 0.000066 | 172.16.16.128 | 74.125.95.104 | HTTP     | GET / HTTP/1.1  |
| 5   | 0.048778 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=109 Len=0                     |
| 6   | 0.022176 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]  |

图 9-22 这些流量发生得相当快,被认为是正常的

这个通信序列是相当快的,全过程花了不到 0.1s。

接下来我们查看的几个捕获文件将包含相同的流量模式,只是在数据包时序上有些许不同。

### 9.4.2 慢速通信——线路延迟

现在我们转向捕获文件 `latency2.pcap`。如图 9-23 所示,注意除了时间值之外,所有数据包都和上一个文件中的相同。

| No. | Time     | Source        | Destination   | Protocol | Info  |
|-----|----------|---------------|---------------|----------|---|
| 1   | 0.000000 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=2                     |
| 2   | 0.878530 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 WS=6 |
| 3   | 0.016604 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 Win=4218 Len=0                    |
| 4   | 0.000335 | 172.16.16.128 | 74.125.95.104 | HTTP     | GET / HTTP/1.1  |
| 5   | 1.155228 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=109 Len=0                     |
| 6   | 0.015866 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]  |

图 9-23 数据包 2 和 5 有很高的延迟

当我们开始逐个查看这 6 个数据包时,很快遇到了延迟的第一个标志。客户端 (172.16.16.128) 发送初始 SYN 数据包,开始 TCP 握手。在收到服务器端 (74.125.95.104) 返回的 SYN/ACK 数据包之前有 0.87s 的延迟。这是我们受到线路延迟影响的第一个迹象,这是客户端和服务器之间的设备导致的。

由于数据包传输的特性，我们可以确定这是线路延迟的问题。当服务器收到一个 SYN 数据包时，由于不涉及任何传输层以上的处理，发送一个响应只需要非常小的处理量。即使服务器正承受巨大的流量负载，通常它也会迅速地向 SYN 数据包响应一个 SYN/ACK。这排除了服务器导致高延迟的可能性。

客户端的可能性也被排除了，因为它在此时除了接收 SYN/ACK 数据包以外什么也没干。排除了客户端和服务器的原因，那网络缓慢的原因应该在捕获记录的前两个数据包里。

继续看，我们发现完成三次握手的 ACK 数据包传输很快，客户端发送的 HTTP GET 请求也同样如此。产生这两个数据包的处理过程是收到 SYN/ACK 后在客户端本地发生的，所以只要客户端没有沉重的处理负载，这两个数据包应该立刻就能发出去。

在数据包 5，我们看到它的时间值也高得令人难以置信。看来，我们发送初始 HTTP GET 请求之后，经过 1.15s 才收到从服务器返回的 ACK 数据包。收到 HTTP GET 请求后，服务器在发送数据之前先发送了一个 TCP ACK，同样这也需要服务器耗费太多处理资源。这是线路延迟的另一个标志。

每次遇上线路延迟，你几乎都会在通信过程中初始握手的 SYN/ACK 以及其他 ACK 数据包中看到这样的情景。虽然这个信息并没有告诉你网络高延迟的确切原因，但它起码告诉你不是客户端或服务器的问题，所以你能意识到延迟是因为中间的一些设备出了问题。此刻，你可以开始检查受影响主机之间的防火墙、路由器、代理服务器等设备，以确定问题所在。

### 9.4.3 慢速通信——客户端延迟

我们查看的下一个延迟情景包含在 latency3.pcap 文件中，如图 9-24 所示。

| No. | Time     | Source        | Destination   | Protocol | Info  |
|-----|----------|---------------|---------------|----------|---|
| 1   | 0.000000 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [SYN] Seq=2082691767 win=8192 Len=0 MSS=1460 WS=2                     |
| 2   | 0.023790 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 win=5720 Len=0 MSS=1406 WS=6 |
| 3   | 0.014894 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 win=4218 Len=0                    |
| 4   | 1.345023 | 172.16.16.128 | 74.125.95.104 | HTTP     | GET / HTTP/1.1  |
| 5   | 0.046121 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 win=109 Len=0                     |
| 6   | 0.016182 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]  |

图 9-24 这个捕获记录中的缓慢数据包是初始 HTTP GET

刚开始这个捕获记录很正常，迅速完成了 TCP 握手，没有任何延迟。但握手完成之后，数据包 4 的 HTTP GET 请求有点问题。该数据包继上一个接收的数据包有 1.34s 的延迟。

我们应该查看数据包 3 和 4 之间到底发生了什么以找到延迟的原因。数据

包 3 是 TCP 握手过程中客户端发往服务器的最后一个 ACK，数据包 4 则是客户端发往服务器的 GET 请求。它们的共同点在于都是客户端发送的，与服务器无关。发送完 ACK 之后本应该很快就发送 GET 请求，因为所有动作都是以客户端为中心的。

不幸的是，ACK 并没有快速切换到 GET。创建和传输 GET 数据包确实需要应用层的处理，而这处理过程的延迟表明客户端无法及时执行该动作。这意味着通信高延迟的根源在于客户端。

#### 9.4.4 慢速通信——服务器延迟

我们查看的最后一个延迟情景使用了 latency4.pcap 文件，如图 9-25 所示。这是服务器延迟的一个例子。

| No. | Time     | Source        | Destination   | Protocol | Info  |
|-----|----------|---------------|---------------|----------|---|
| 1   | 0.000000 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [SYN] Seq=2082691767 Win=8192 Len=0 MSS=1460 WS=2                     |
| 2   | 0.018583 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [SYN, ACK] Seq=2775577373 Ack=2082691768 Win=5720 Len=0 MSS=1406 WS=6 |
| 3   | 0.016197 | 172.16.16.128 | 74.125.95.104 | TCP      | 1606 > 80 [ACK] Seq=2082691768 Ack=2775577374 Win=4218 Len=0                    |
| 4   | 0.000172 | 172.16.16.128 | 74.125.95.104 | HTTP     | GET / HTTP/1.1  |
| 5   | 0.047936 | 74.125.95.104 | 172.16.16.128 | TCP      | 80 > 1606 [ACK] Seq=2775577374 Ack=2082692395 Win=109 Len=0                     |
| 6   | 0.982983 | 74.125.95.104 | 172.16.16.128 | TCP      | [TCP segment of a reassembled PDU]  |

图 9-25 直到最后一个数据包才表现出高延迟

在这个捕获记录中，两台主机间的 TCP 握手过程很快就顺利完成，这是个很好的开端。下一对数据包带来了更好的消息，初始 GET 请求和响应的 ACK 数据包也很快传输完毕了。直到最后一个数据包，我们才发现了高延迟的迹象。

第六个数据包是服务器响应客户端 GET 请求的第一个 HTTP 数据包，但它竟然在服务器为 GET 请求发送 TCP ACK 之后延迟了 0.98s。数据包 5 到 6 的切换情况与我们在上一个情景中所见的握手 ACK 与 GET 请求之间的切换很相似。然而，在这个例子中，服务器才是我们关注的焦点。

数据包 5 是服务器响应客户端 GET 请求的 ACK。一旦发送这个数据包，服务器就应该立即开始发送数据。这个数据包中的数据访问、打包、传输是由 HTTP 协议完成的，由于这是一个应用层协议，需要服务器作一点处理。延迟收到这个数据包表明服务器不能及时处理这个数据，最终把延迟的根源指向了服务器。

#### 9.4.5 延迟定位框架

我们使用 6 个数据包成功地定位了从客户端到服务器的网络高延迟的原因。

这些场景看起来也许有点复杂，但图 9-26 应该能帮你更快解决延迟问题。这些原则几乎可应用于任何基于 TCP 的通信。

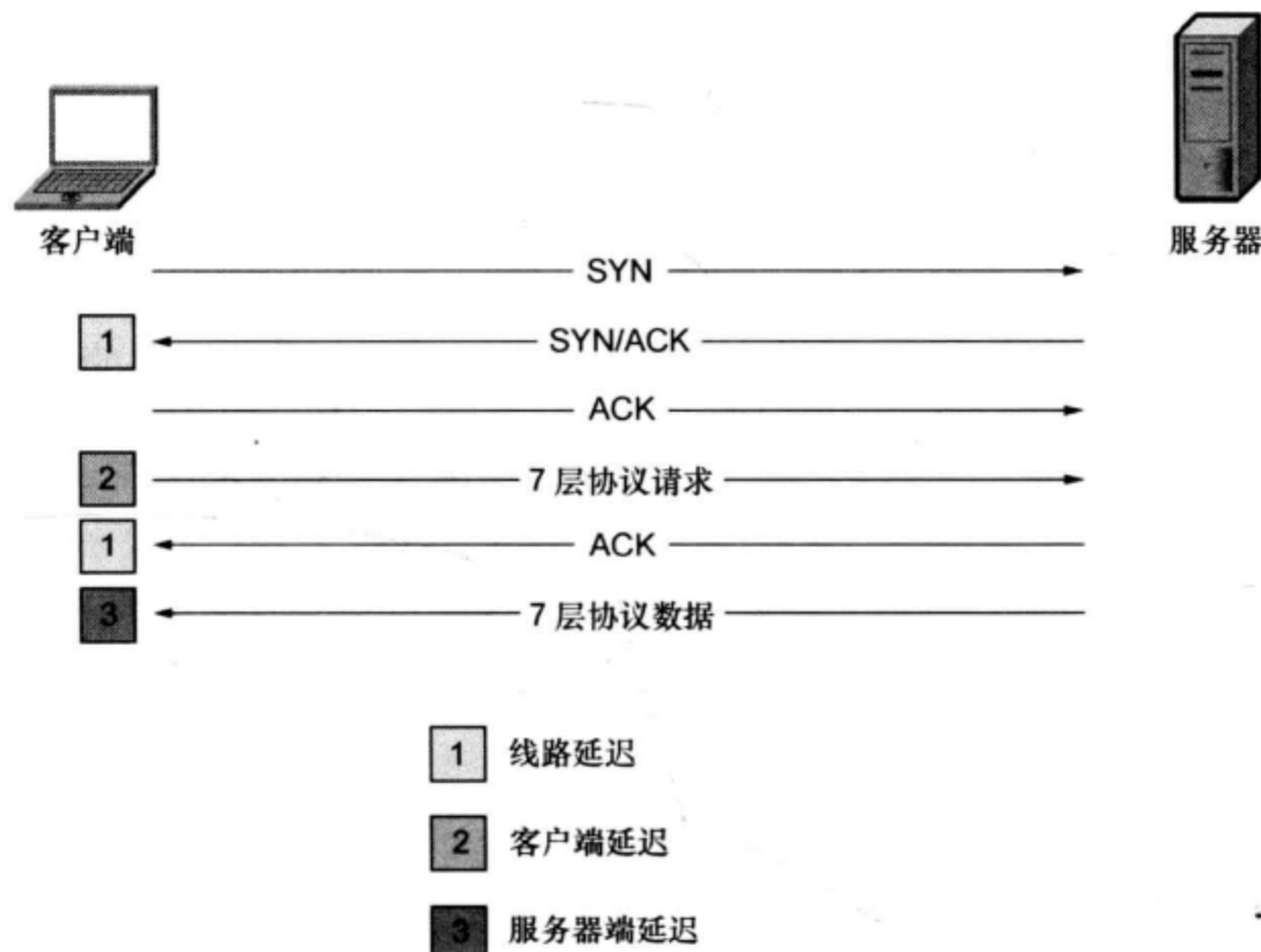


图 9-26 你可以用这张图解决自己的延迟问题

#### 注意

注意我们还没有讨论 UDP 延迟。因为 UDP 的设计目标是快速但不可靠，所以它没有内置任何延迟检测并从中恢复的功能。相反，它依赖于应用层协议（和 ICMP）来解决数据可靠传输的问题。

## 9.5 网络基线

当所有的努力都失败时，网络基线将成为检修网络缓慢故障最关键的数据之一。对我们的目的而言，网络基线包含来自网络不同端点的流量样本，包括大量我们认可的“正常”网络流量。网络基线的作用是在网络或设备工作不正常时作为比较的基准。

例如，考虑一个场景，网络上几个客户反映登录某个本地 Web 应用服务器时反应迟钝。如果你捕获这些流量并与网络基线对照，你会发现 Web 服务器一切正常，但嵌入到 Web 应用中的外部内容引发了额外的外部 DNS 请求，而这

些请求比正常速度慢两倍。

也许不靠网络基线的帮助，你也能注意到异常的外部 DNS 服务器，但当你处理微妙的变化时，就不一定了。10 个 DNS 请求都比正常的多耗费 0.1s 秒来处理，跟 1 个 DNS 请求多耗费 1s 来处理一样糟糕，但没有网络基线的话，检测前者要难得多。

由于网络各不相同，网络基线的组件将有很大差异。接下来的几节提供了网络基线组件的几个例子。你也许会发现所有这些条目都能应用到你的网络基础设施，或只是很少一部分能适用。不管怎样，你应该把你的基线中的每一个组件置入这 3 个基本基线目录中：站点、主机、应用程序。

### 9.5.1 站点基线

站点基线的目的是获得网络上每个物理站点的整体流量快照。理想情况下，这将是 WAN 内的每一个段。

这个基线应该包含以下几个组件。

#### 使用的协议

在网络边缘（路由器/防火墙）捕获网段上所有设备的流量时，请使用协议分层统计窗口（**Statistics->Protocol Hierarchy**）来查看所有设备的流量。然后，你可以对照看看是否缺少本应出现的协议，或者网络上是否出现了新的协议。你也可以在协议的基础上，用它来发现高于正常数量的特定类型的流量。

#### 广播流量

其包含网段上的一切广播流量。在站点内任一点监听都可以捕获所有广播流量，通过它可以了解正常情况下谁将大量广播流量发送到网络中，这样你将很快确定是否出现了过多（或过少）的广播流量。

#### 身份验证序列

这包括任意客户端到所有服务的身份验证过程的流量，比如动态目录、Web 应用程序，以及特定组织的软件。身份验证通常是服务运行缓慢的一个方面。通过基线你可以确定身份验证是否是通信缓慢的原因。

## 数据传输率

这通常包括在网络上测量这个站点到其他站点传输的大量数据。你可以使用 Wireshark 的捕获概述和绘图功能确定传输速率和连接的一致性。这可能是你最重要的站点基线。每当在网段上建立或拆除连接很慢时，你可以运行与基线同样的数据传输并比较结果。这可以告诉你连接是否真的很慢，甚至可能帮助你查找缓慢的原因。

### 9.5.2 主机基线

使用主机基线并不意味着你必须在网络上测试每台主机。主机基线只需要在高流量或关键任务服务器上执行。基本上，只要某台服务器缓慢会招来管理层愤怒的电话，你就必须在那台主机上建立基线。

主机基线包含以下几个组件。

#### 使用的协议

当捕获这台主机的流量时，这个基线提供了使用协议分层统计窗口的好机会。然后，你可以对照看看是否缺少本应出现的协议，或者主机上是否出现了新的协议。你也可以在协议的基础上，用它来发现高于正常数量的特定类型的流量。

#### 空闲/繁忙流量

这个基线只是简单包含了高峰和非高峰时段正常操作流量的总体捕获记录。了解到一天之中不同时段的连接数量及其占用的带宽大小，有助于你确定缓慢是用户负载还是其他原因造成的。

#### 启动/关闭

为了获取这个基线，你需要在主机启动和关闭时创建一个流量捕获记录。一旦计算机不能启动、不能关闭，或这两个过程异常缓慢，你就可以使用它确认问题是否跟网络有关。

#### 身份验证序列

这个基线要求在主机上捕获所有服务的身份验证过程的流量。身份验证通常是

服务运行缓慢一个方面。通过基线你可以确认身份验证是否是通信缓慢的原因。

### 关联/依赖

这个基线需要持续更长时间的捕获，以确定这台主机依赖于哪些主机（以及哪些主机依赖于这台主机）。你可以通过会话窗口（Statistics->Conversations）查看这些关联和依赖。Web 服务器依赖于 SQL 服务器便是这样的例子。有时我们会意识到主机间的一些潜在依赖关系，这时主机基线便能派上用场。通过这个，你可以确定主机不能正常运转是因为配置错误，还是因为所依赖主机的高延迟。

## 9.5.3 应用程序基线

最后一个网络基线类别是应用程序基线。这个基线应该用于所有基于网络的关键业务应用程序。

应用程序基线包含以下几个组件。

### 使用的协议

我们在这个基线中再次使用了 Wireshark 的协议分层统计窗口，但这次是在运行应用程序的主机上捕获流量。然后，你可以通过比较这个列表，发现依赖于这些协议的应用程序是否正常运转。

### 启动/关闭

这个基线需要捕获应用程序启动和关闭时生成的流量。一旦应用程序不能启动、不能关闭，或这两个过程都异常缓慢，你就可以使用它确定原因。

### 关联/依赖

这个基线需要持续更久的捕获，以通过会话窗口确定这个应用程序依赖的其他主机和应用程序。有时我们会意识到应用程序间的一些潜在依赖关系，这时应用程序基线便能派上用场。通过这个，你可以确定应用程序不能正常运转是因为配置错误，还是因为所依赖应用程序的高延迟。

### 数据传输率

你可以在应用程序服务器正常运转期间，使用 Wireshark 的捕获概述和绘图

功能确定数据传输率和连接一致性。每当有人报告应用程序缓慢时，你可以使用这个基线来确定当前问题是否是高利用率或高用户负载造成的。

#### 9.5.4 基线的其他注意事项

下面是创建网络基线的一些额外注意事项。

- 创建每个基线都至少要经过 3 次：低流量期间一次（早晨）、高流量期间一次（下午 3 点左右）、无流量期间一次（深夜）。
- 有可能的话，尽量避免直接在需要创建基线的主机上捕获流量。因为在高流量期间，这可能会增加设备负载、影响性能，并可能因数据包丢失导致基线无效。
- 你的基线可能包含一些与网络有关的私密信息，一定要保护好它。将它存储在安全的地方，只有合适的人才有访问权限。但同时别放得太偏，以免需要时找不到。可以考虑将它存放在 U 盘或者加密分区里。
- 让所有.pcap 文件与你的基线关联，为更常见的参考值写一份“小抄”，比如关联关系或数据传输率。

### 9.6 小结

本章内容聚焦于如何解决慢速网络的问题。我们讲述了 TCP 的一些有用的可靠性检测和恢复功能，演示了如何定位网络通信高延迟的原因，并讨论了网络基线的重要性以及它的一些组件。使用这里讨论的技术，再加上 Wireshark 的绘图和分析功能（在第 5 章讨论过），你再接到电话抱怨网速很慢时，应该能应付自如。



# 第10章

## 安全领域的数据包分析



虽然本书主要集中于如何使用数据包分析技术解决网络故障，但在现实世界中，很多数据包分析工作都是为了解决安全问题。当入侵分析师检查来自可疑入侵者的网络流量，或取证人员调查恶意软件在主机上的感染程度时，就会用到数据包分析的方法。面向安全的数据包分析是一个很大的话题，都可以另写一本书了，本章只是带你尝尝鲜。

在本章，我们将扮演一位安全从业者，学习在网络层分析“肉鸡<sup>1</sup>”系统的各个方面。我们将涉及网络侦察、重定向恶意的流量，以及利用系统漏洞。接着，我们将扮演一位入侵分析师，剖析来自入侵检测系统的警报流量。即使

<sup>1</sup> “肉鸡”是网络安全领域中的行话，指被恶意入侵者远程控制的机器。因为机器被远程控制后，将“任人宰割”，所以俗称“肉鸡”。——译者注

你不是安全从业者，通过阅读本章，你也可以获得一些对网络安全的关键视角。

## 10.1 网络侦察

攻击者采取的第一步行动是深入研究目标系统。这一步又叫“网络踩点”，通常使用各式各样的公开资源来完成，比如目标公司的主页或者 Google。这个研究完成后，攻击者通常开始扫描目标的 IP 地址（或者域名）的开放端口或运行服务。

通过扫描，攻击者可以确定目标是否在线并且可达。例如，想象一下，一位银行大盗盯上了位于 Main Street 123 号的目标——本市规模最大的银行。他花费数星期时间精心策划此次抢劫，却在抵达目的地后才发现银行已经搬迁到了 Vine Street 555 号。还可以想象一个更糟糕的场景，劫匪计划在正常上班时间步行进入银行，以便对金库下手，刚到银行门口却发现今天歇业。确保目标在线并且可达是我们必须要解决的一个问题。

扫描的另一个重要收获是，它告诉了攻击者目标开放了哪些端口。回到我们刚才类比的银行劫匪，想一想，如果劫匪出现在银行门口，却对整幢楼的布局一无所知，会怎么样？他不知道从哪里进入大楼，因为他不知道物理防御的弱点在哪里。

在本节中，我们会讨论如何用一些典型的扫描技术识别主机和它们开放的端口号，以及网络上的漏洞。

---

### 注意

到目前为止，本书说的“连接两端”都是指发送者和接收者，或者客户端和服务器。而本章提到的“连接两端”却是指攻击者或受害者。

---

### 10.1.1 SYN 扫描

首先对系统做 TCP SYN 扫描，又称为隐秘扫描或半开扫描。SYN 扫描是最常见的扫描类型，有以下几个原因。

- 快速可靠。
- 在所有平台上都很准确，与 TCP 协议栈的实现无关。
- 比其他扫描技术更安静，不容易被发现。

TCP SYN 扫描依赖于三次握手过程，可以确定目标主机的哪些端口是开放

的。攻击者发送 TCP SYN 数据包到受害者的一个范围的端口上，就像要在这些端口上建立用于正常通信的连接似的。如图 10-1 所示，一旦受害者收到这个数据包，就可能会发生一些事情。

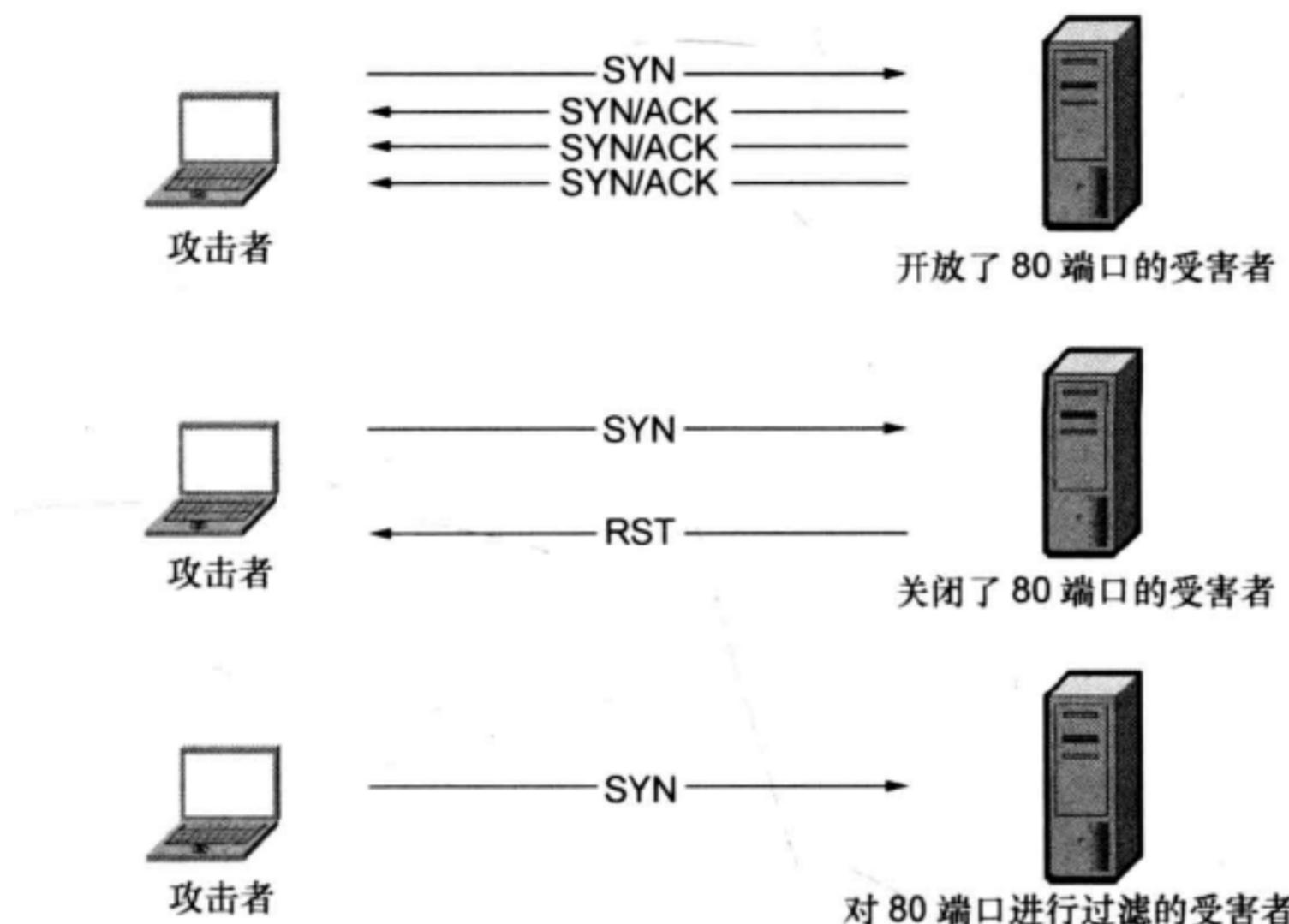


图 10-1 一次 TCP SYN 扫描的结果

如果受害者机器上某个服务正在监听的端口收到了 SYN 数据包，它将向攻击者回复一个 TCP SYN/ACK 数据包，也就是 TCP 握手的第二部分。这样攻击者就能知道这个端口是开放的，并且有一个服务在上面监听。正常情况下会发送一个 TCP ACK 包以完成连接握手，但此刻攻击者并不想这样，因为他还不想与主机通信。所以，攻击者并不打算完成 TCP 握手。

如果没有服务在被扫描的端口上监听，那攻击者就收不到 SYN/ACK。按照受害者操作系统的不同配置，攻击者可能会收到响应的 RST 数据包，表示端口关闭了，或者，攻击者看不到任何响应。这意味着端口被某个中间设备过滤了，例如防火墙或主机本身。另一方面，也有可能是因为响应数据包在传输过程中丢失了。这个结果通常表明端口是关闭的，但说服力并不强。

捕获文件 synscan.pcap 提供了用 NMAP 工具进行 SYN 扫描的绝佳例子。NMAP 是 Fyodor 创立的网络扫描程序，非常健壮。它可以执行你能想到的任何一种扫描方式。你可以从 <http://www.nmap.com/download.html> 免费下载 NMAP。

我们捕获的样本大概包含 2000 个数据包，说明这种扫描有一定的规模。确定这个扫描范围大小的最好办法之一就是查看 Conversations 窗口，如图 10-2 所

示。在图中你会看到攻击者（172.16.0.8）和受害者（63.13.134.52）之间只有一个 IPv4 会话①。你也会看到，那里有 1994 个 TCP 会话②——通信基本上是每一个端口对应一个新会话。

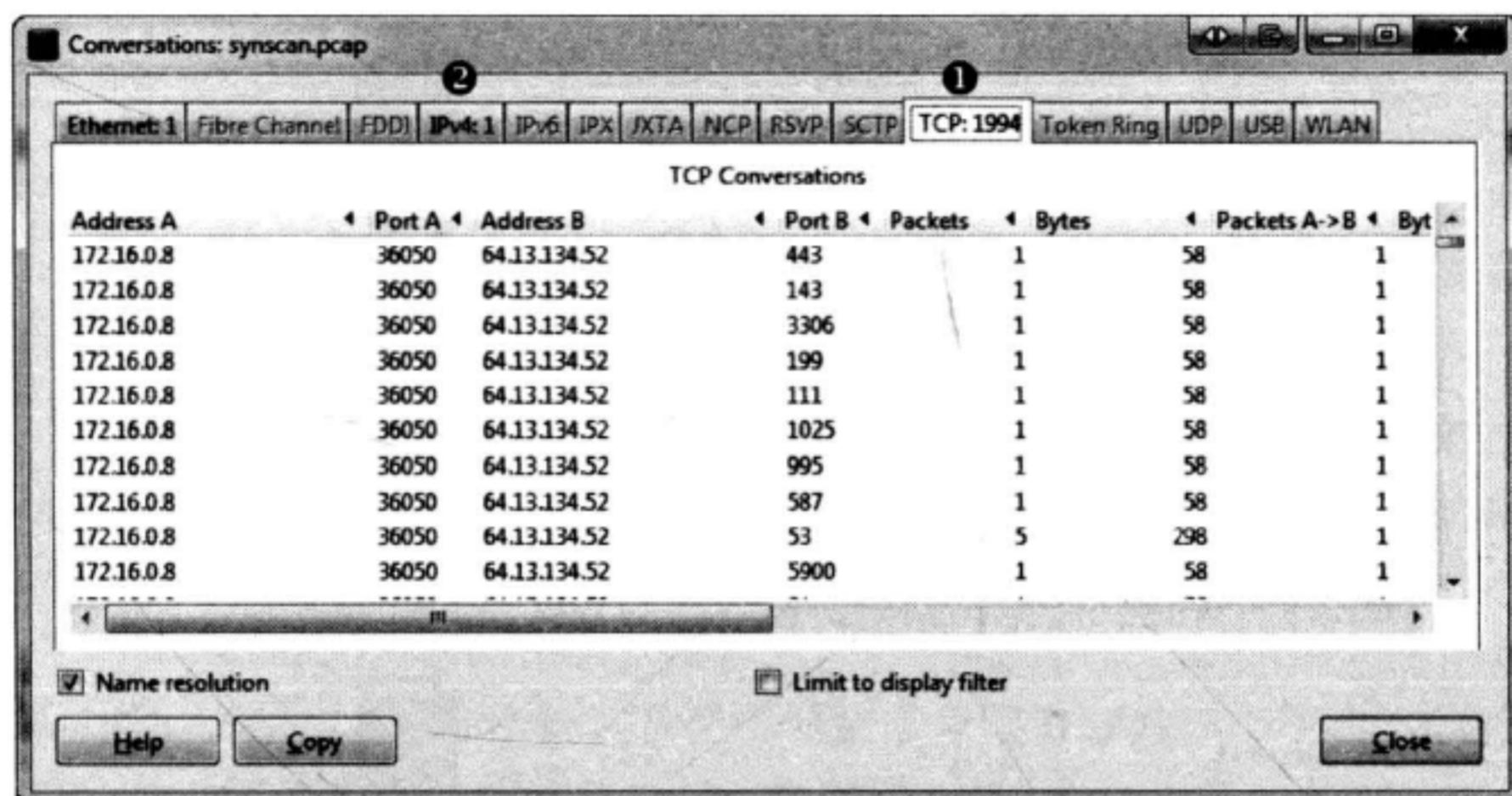


图 10-2 Conversations 窗口显示了正在进行的 TCP 通信

- 扫描是在极短时间内完成的，因此在捕获文件上滚动鼠标并不是寻找 SYN 数据包响应的好办法。在接收到响应之前，已经发送了更多的 SYN 数据包。幸好，我们可以创建过滤器，来帮助我们寻找正确的流量。

### 10.1.1.1 在 SYN 扫描中使用过滤器

举一个筛选的例子。让我们看看第 1 个数据包——发送到受害者 443 端口（HTTPS）的 SYN 数据包。为了查看是否有对这个数据包的响应，我们可以创建一个过滤器，以显示所有源端口或目标端口为 443 的流量。下面是如何进行快速设置的方法。

1. 在捕获文件中选择第一个数据包。
2. 在 Packet Details 面板中展开 TCP 头部。
3. 右键单击 Destination Port 域，选择 Prepare as Filter，单击 Selected。
4. 这将在 filter 对话框放置一个过滤器，针对所有目标端口为 443 的数据包。现在，由于我们也想要源端口为 443 的数据包，所以单击屏幕顶端的 filter 栏，删除过滤器的 dst 部分。

结果过滤器给出了两个数据包，都是攻击者发给受害者的 TCP SYN 数据包，如图 10-3 所示。

| No. | Time     | Source     | Destination  | Protocol | Info   |
|-----|----------|------------|--------------|----------|--|
| 1   | 0.000000 | 172.16.0.8 | 64.13.134.52 | TCP      | 36050 > 443 [SYN] Seq=3713172248 win=3072 Len=0 MSS=1460 |
| 32  | 0.000065 | 172.16.0.8 | 64.13.134.52 | TCP      | 36051 > 443 [SYN] Seq=3713237785 win=2048 Len=0 MSS=1460 |

图 10-3 两次尝试用 SYN 数据包建立连接

两个数据包都没有得到响应，有可能是因为响应数据包被受害者主机或中间设备过滤了，或者端口是关闭的。但最终来说，对 443 端口的扫描结果是不确定的。

我们可以用同样技术来分析其他数据包，看看有没有不同的结果。首先，单击过滤器旁边的 Clear 按钮，清空之前创建的过滤器。然后选择列表中的第 9 个数据包。这是目标端口为 53 的 SYN 数据包，通常与 DNS 有关。使用在前面提到的方法，创建一个基于目标端口的过滤器，并删除 dst 部分，这样它就应用到所有与 TCP 53 端口有关的流量了。当你应用这个过滤器时，你会看见 5 个数据包，如图 10-4 所示。

| No. | Time     | Source       | Destination  | Protocol | Info  |
|-----|----------|--------------|--------------|----------|---|
| 9   | 0.000052 | 172.16.0.8   | 64.13.134.52 | TCP      | 36050 > 53 [SYN] Seq=3713172248 win=3072 Len=0 MSS=1460 |
| 10  | 0.000052 | 64.13.134.52 | 172.16.0.8   | TCP      | 53 > 36050 [SYN, ACK] Seq=1 win=1460 Len=0 MSS=1460     |
| 11  | 0.000052 | 64.13.134.52 | 172.16.0.8   | TCP      | 53 > 36050 [SYN, ACK] Seq=1 win=1460 Len=0 MSS=1460     |
| 12  | 0.000052 | 64.13.134.52 | 172.16.0.8   | TCP      | 53 > 36050 [SYN, ACK] Seq=1 win=1460 Len=0 MSS=1460     |
| 13  | 0.000052 | 64.13.134.52 | 172.16.0.8   | TCP      | 53 > 36050 [SYN, ACK] Seq=1 win=1460 Len=0 MSS=1460     |

图 10-4 表明端口是开放的 5 个数据包

第 1 个是我们在捕获之初选择的 SYN 数据包。第二个则是来自受害者的响应。这是一个 TCP SYN/ACK 数据包——实施三次握手时期待的响应。在正常情况下，下一个数据包应该是发送初始 SYN 的主机发送的 ACK。然而，在这个例子中，攻击者并不想建立连接，因而没有发送响应。受害者重传了 3 次 SYN/ACK 包才放弃。由于尝试与主机的 53 端口通信时收到了 SYN/ACK 响应，我们可以确定有一个服务在监听该端口。

让我们在数据包 13 上再次重复此过程。这是一个目标端口为 113 的 SYN 数据包，通常与 Ident 协议有关，经常用于 IRC 的身份识别和验证服务。如果你在这个数据包上应用同一类型的过滤器，会发现 4 个数据包，如图 10-5 所示。

| No. | Time     | Source       | Destination  | Protocol | Info   |
|-----|----------|--------------|--------------|----------|--|
| 13  | 0.000070 | 172.16.0.8   | 64.13.134.52 | TCP      | 36050 > 113 [SYN] Seq=3713172248 win=4096 Len=0 MSS=1460         |
| 14  | 0.061491 | 64.13.134.52 | 172.16.0.8   | TCP      | 113 > 36050 [RST, ACK] Seq=2462244745 Ack=3713172249 Win=0 Len=0 |
| 15  | 0.000152 | 64.13.134.52 | 172.16.0.8   | TCP      | 113 > 36061 [RST, ACK] Seq=1027049353 Ack=3696394777 Win=0 Len=0 |

图 10-5 SYN 之后紧随一个 RST，表明端口是关闭的

第一个数据包是初始 SYN，紧接着是来自受害者的 RST。这是受害者目标端口不接受连接的迹象，表明很可能没有服务运行在上面。

### 10.1.1.2 识别开放和关闭的端口

理解了 SYN 扫描能引起的不同响应类型后，自然而然会想到去找一个方法——如何快速识别哪些端口是开放的还是关闭的。答案再次落到了 Conversations 窗口中，在这个窗口中，你可以通过数据包编号排序 TCP 会话，单击 Packet 列两次就可以让最高值靠前，如图 10-6 所示。

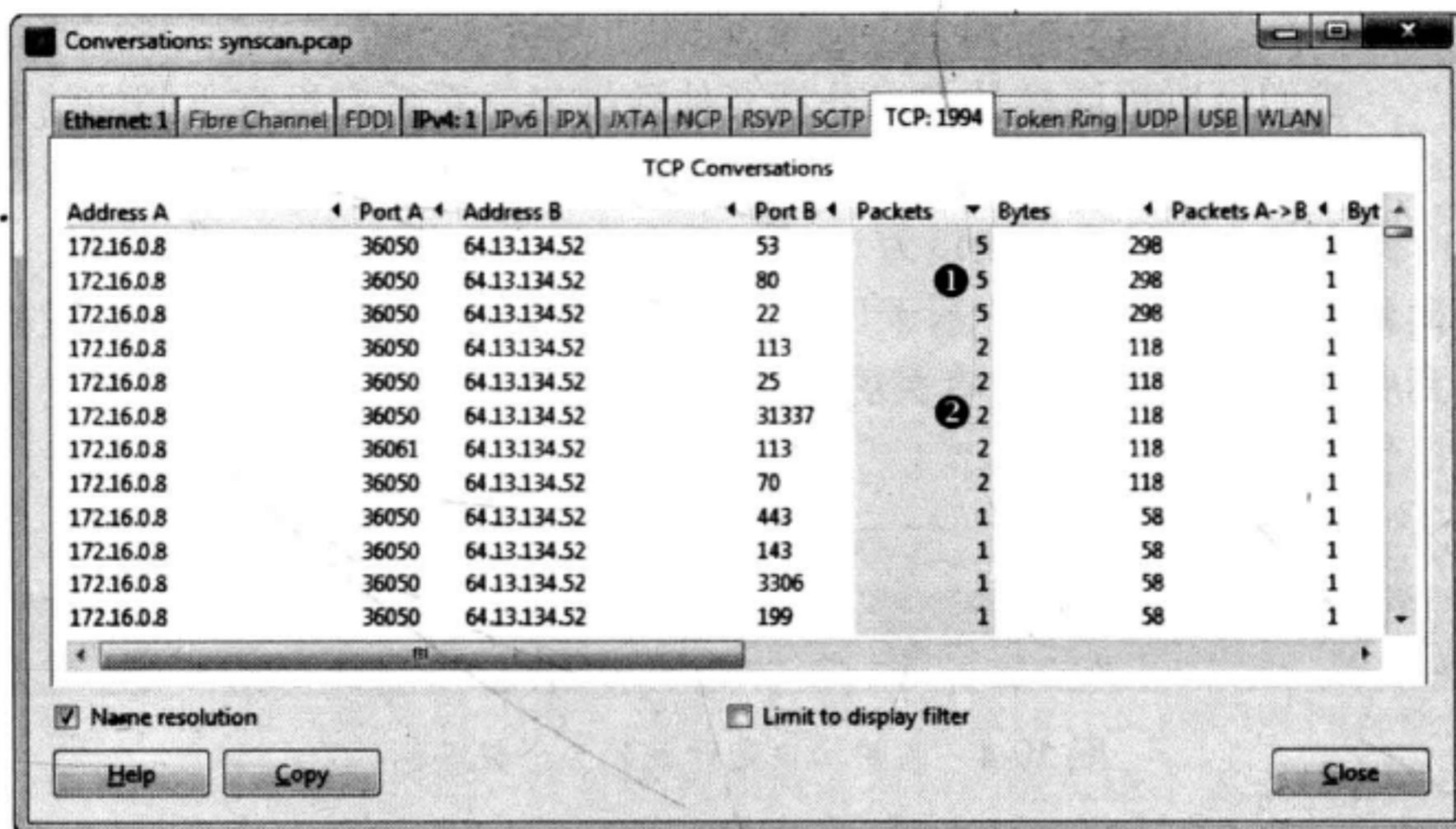


图 10-6 用 Conversations 窗口寻找开放端口

3 个被扫描的端口在各自会话中包含 5 个数据包①。我们知道 53、80 和 22 端口是开放的，因为这 5 个数据包表示初始 SYN、对应的 SYN/ACK 及来自受害者的 3 次重传 SYN/ACK。

有 5 个端口的通信只包含 2 两个数据包②。第一个是初始 SYN，第二个是来自受害者的 RST。这表明 113、25、31337、113 和 70 端口是关闭的。

Conversations 窗口剩下的项只包含 1 个数据包，意味着受害者主机并没有响应初始 SYN 包。剩下这些端口很可能是关闭的，但我们不能确定。

### 10.1.2 操作系统指纹术

了解目标的操作系统对攻击者有极大价值。了解到目标使用的操作系统，将确保攻击者实施具有针对性的攻击手段。同时这个信息也有助于攻击者成功进入系统后，在目标文件系统找到关键文件和目录。

“操作系统指纹术”是指在没有物理接触的情况下，用来确定机器运行的操

作系统的一组技术。操作系统指纹术分为两种类型：被动式和主动式。

### 10.1.2.1 被动式指纹技术

被动式指纹技术通过分析目标发送的数据包的某些域来确定目标使用的操作系统。这项技术之所以称为“被动”，是因为你只监听目标主机发送的数据包，但并不主动向目标发送任何流量。对黑客来说，这是最理想的操作系统指纹技术，因为它非常隐蔽。

那就是说，我们只需要基于主机发送的数据包，就能确定它使用哪一种操作系统了？这其实非常容易，由于 RFC 文件定义的协议并未规定全部技术参数的值，这完全是有可能的。虽然 TCP、UDP 和 IP 头部的各个域都有特定含义，但并没有定义这些域的默认值。这意味着不同操作系统实现的 TCP/IP 协议栈都必须为这些域定义它自己的默认值。表 10-1 列出了一些与操作系统实现有关的常见域及其默认值。

表 10-1 常见的被动式指纹值

| 协议<br>头部 | 域      | 默认值       | 操作系统   |
|----------|--------|-----------|--|
| IP       | 初始 TTL | 64        | Nmap、BSD、Mac OS 10、Linux                           |
|          |        | 128       | Novell、Windows                                     |
|          |        | 255       | Cisco IOS、Palm OS、Solaris                          |
| IP       | 不分片标志  | Set       | BSD、Mac OS 10、Linux、Novell、Windows、Palm OS、Solaris |
|          |        | Not set   | Nmap、Cisco IOS                                     |
| TCP      | 最长段大小  | 0         | Nmap   |
|          |        | 1440      | Windows、Novell                                     |
|          |        | 1460      | BSD、Mac OS 10、Linux、Solaris                        |
| TCP      | 窗口大小   | 1024–4096 | Nmap   |
|          |        | 65535     | BSD、Mac OS 10                                      |
|          |        | 2920–5840 | Linux  |
|          |        | 16384     | Novell   |
|          |        | 4128      | Cisco IOS  |
|          |        | 24820     | Solaris  |
|          |        | Variable  | Windows  |
|          |        | Set       | Linux、Windows、OpenBSD                              |
| TCP      | SackOK | Set       | Nmap、FreeBSD、Mac OS 10、Novell、Cisco IOS、Solaris    |
|          |        | Not set   |  |

捕获文件 `passiveosfingerprinting.pcap` 中的数据包是这项技术的绝佳例子。该文件含有两个数据包。它们都是发送到目标端口 80 的 TCP SYN 数据包，但来自不同的主机。仅仅使用这些数据包中的值，并参考表 10-1，我们就能确定每台主机使用的操作系统架构。每一个数据包的细节如图 10-7 所示。

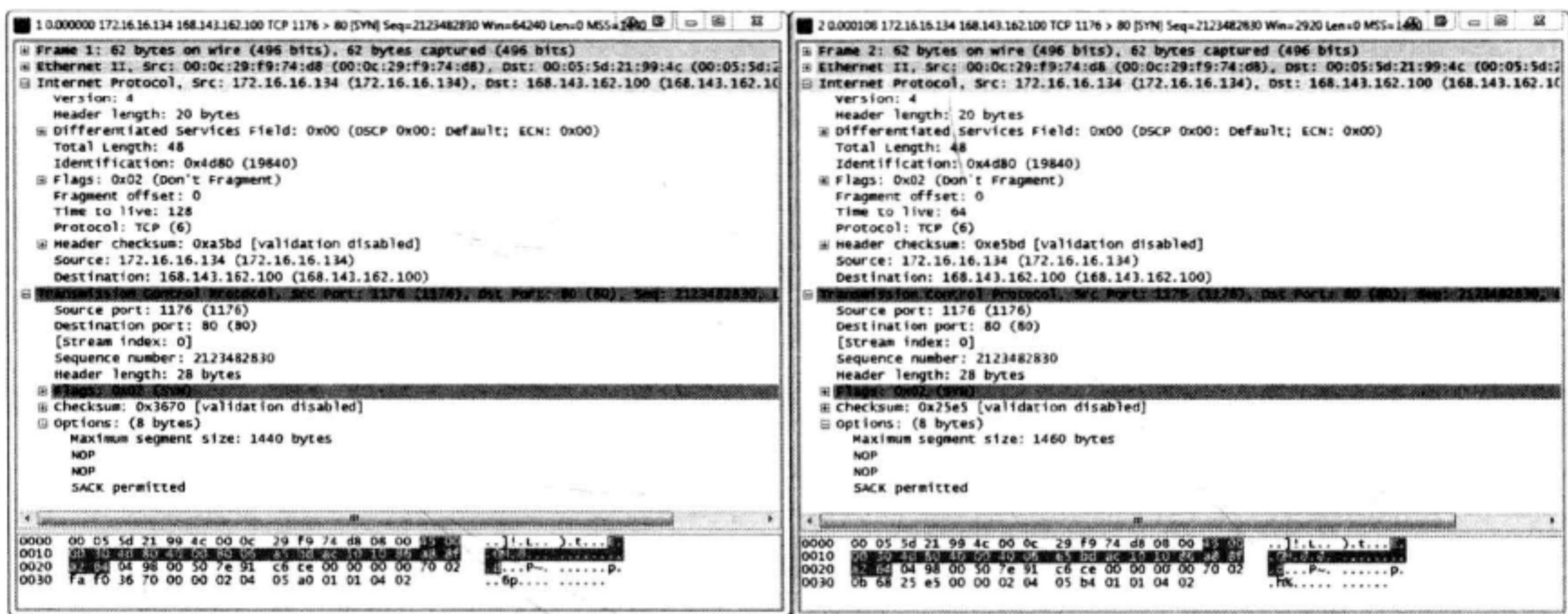


图 10-7 这些数据包可以告诉我们它们来自哪种操作系统

参考表 10-1，我们将这些数据包的相关域分类，创建了表 10-2。

表 10-2 操作系统指纹术的数据包分类

| 协议<br>头部 | 域      | 数据包 1 的值    | 数据包 2 的值   |
|----------|--------|-------------|------------|
| IP       | 初始 TTL | 128         | 64         |
| IP       | 不分片标志  | Set         | Set        |
| TCP      | 最长段大小  | 1440 Bytes  | 1460 Bytes |
| TCP      | 窗口大小   | 64240 Bytes | 2920 Bytes |
| TCP      | SackOK | Set         | Set        |

基于这些值，我们可以得出结论：发送数据包 1 的设备运行 Windows 的可能性最大，而发送数据包 2 的设备运行 Linux 的可能性最大。

要记住，表 10-1 列出的被动式操作系统指纹技术的常见识别域并不完整。很多实现上的“怪癖”可能会导致真实值与期望值的偏差。所以，你不能完全依赖被动式操作系统指纹技术得到的结果。

---

**注意**

有一个叫 p0f 的工具使用了操作系统指纹识别技术。该工具分析捕获数据包的相关域，然后输出可能的操作系统。使用像 p0f 这样的工具，你不仅能了解到操作系统架构，有时甚至能了解适当的版本号或者补丁级别。你可以从 <http://lcamtuf.coredump.cx/p0f.shtml> 下载 p0f。

---

### 10.1.2.2 主动式指纹技术

当被动监听流量不能得到想要的结果时，可能需要一个更直接的方法。这种方法叫做“主动式指纹技术”。它是指攻击者主动向受害者发送特意构造的数据包以引起响应，然后从响应数据包中获知受害者机器操作系统的技术。当然，由于这种方法要与受害者直接通信，它并不是最隐蔽的，但它可以做到非常高效。

捕获文件 activeosfingerprinting.pcap 包含了使用 Nmap 扫描工具发起主动式指纹扫描的例子。文件中有一些是 Nmap 发送的探测数据包，这些探测数据包引起的响应可用于识别操作系统。Nmap 记录下对这些探测数据包的响应并创建一个指纹，与指纹数据库对比后得出结论。

---

**注意**

Nmap 使用的主动式操作系统指纹技术十分复杂。想了解 Nmap 如何执行主动式操作系统指纹技术，请阅读 Nmap 的权威指南——《Nmap Network Scanning》。这是 Nmap 扫描器作者 Gordon “Fyodor” Lyon 的著作。

---

## 10.2 漏洞利用

攻击者是吃饭还是喝粥，就要看漏洞利用阶段的表现了。攻击者已经研究并侦察好目标，并发现了准备利用的漏洞，期望以此进入目标系统。在本章剩下的篇幅里，我们将关注在一些漏洞利用技术中捕获的数据包。这些漏洞包括近期的 Microsoft 漏洞、通过 ARP 缓存中毒攻击重定向流量，以及盗窃数据的远程访问特洛伊木马。

### 10.2.1 极光行动

2010 年 1 月，极光行动利用了一个当时未知的 IE 浏览器漏洞。这个漏洞允许攻击者取得 Google 及其他公司的目标机器的 root 级别的远程控制权限。<sup>1</sup>

---

<sup>1</sup> Google 宣称遭受源自中国的攻击，并随后宣布将搜索引擎服务从中国大陆转移到中国香港。——译者注

用户只需要使用包含该漏洞的 IE 浏览器访问一个网站，就能执行这个恶意代码。然后，攻击者就能立刻获得用户机器的管理员权限。攻击者使用了“网络钓鱼”引诱受害者。所谓“网络钓鱼”(Spear phishing)是指攻击者向受害者发送一封电子邮件，诱导受害者单击邮件中的链接，从而将其引导至一个恶意网站。通常，网络钓鱼信息看起来都像来自可信的源，因此它们经常能成功。

在极光行动的案例中，我们从用户单击钓鱼邮件中链接的那一刻开始讲起。这些数据包包含在文件 aurora.pcap 中。

这个捕获文件以受害者（192.168.100.206）和攻击者（192.168.100.202）之间的三次握手开始。初始化连接的目标是 80 端口，这使我们相信它是 HTTP 流量。第四个数据包证实了我们的假设，它是一个对 /info 的 HTTP GET 请求❶，如图 10-8 所示。

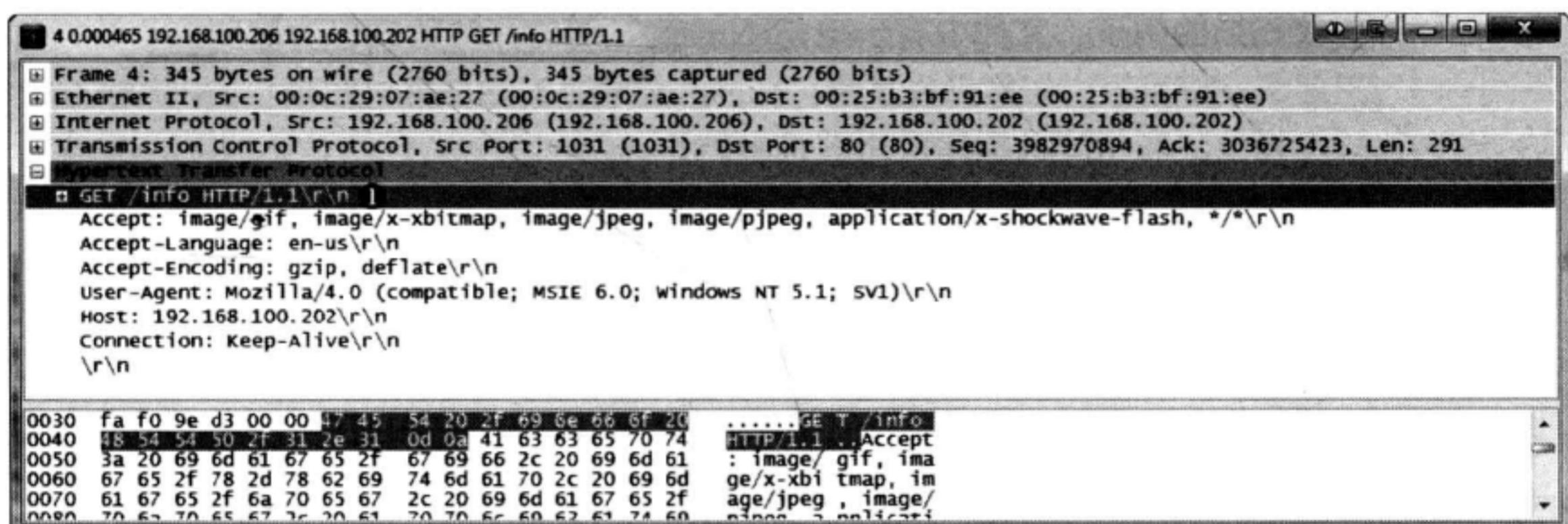


图 10-8 受害者做了一次针对/info 的 GET 请求

攻击者的机器确认收到了 GET 请求，并在数据包 6 中返回了一个 302 (Moved Temporarily) 码。这个状态码通常用于将浏览器重定向到另一个页面，在这个案例中正是如此。与 302 返回码❷一起来的还有一个 Location 域，它指明重定向位置是/info?rFfWELUjLJHnP❸，如图 10-9 所示。

收到 HTTP 302 数据包后，客户端在数据包 7 中初始化了另一个针对 /info?rFfWELUjLJHnP 这个 URL 的 GET 请求，然后在数据包 8 中收到了一个 ACK。ACK 之后的几个包表示从攻击者发往受害者的数据。为了更好地查看那些数据，右击当前 TCP 流的任何一个数据包，如数据包 9，选择 Follow TCP Stream。在这个数据流的输出中，我们看到了初始 GET 请求、302 重定向，以及第二个 GET 请求，如图 10-10 所示。

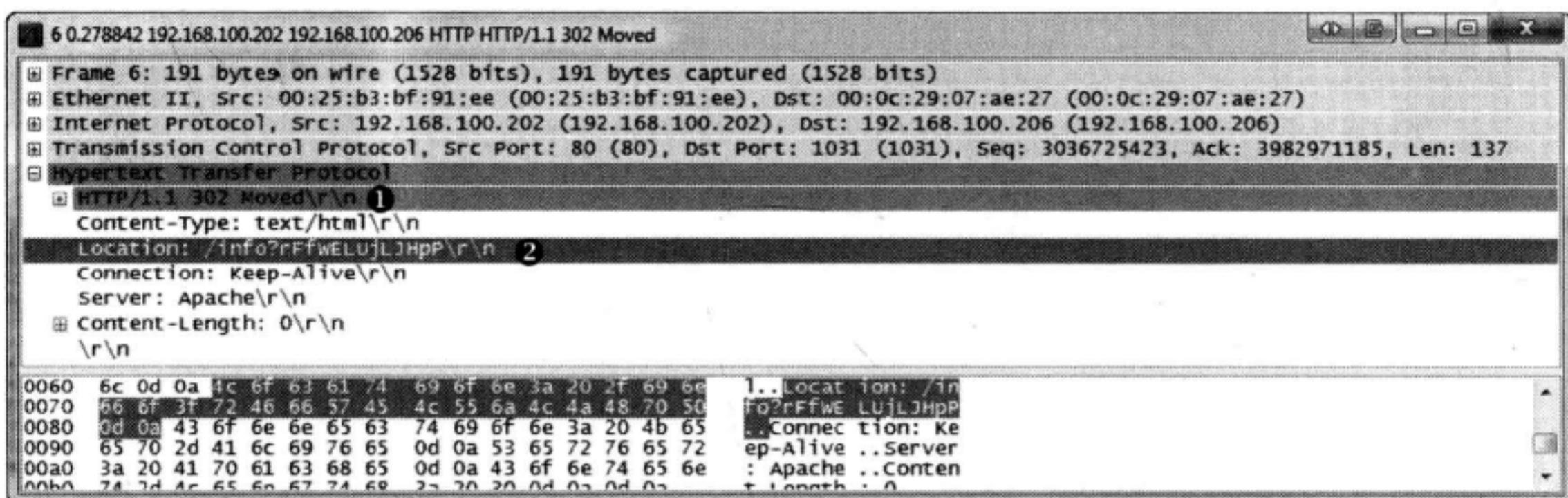


图 10-9 客户端浏览器被这个数据包重定向

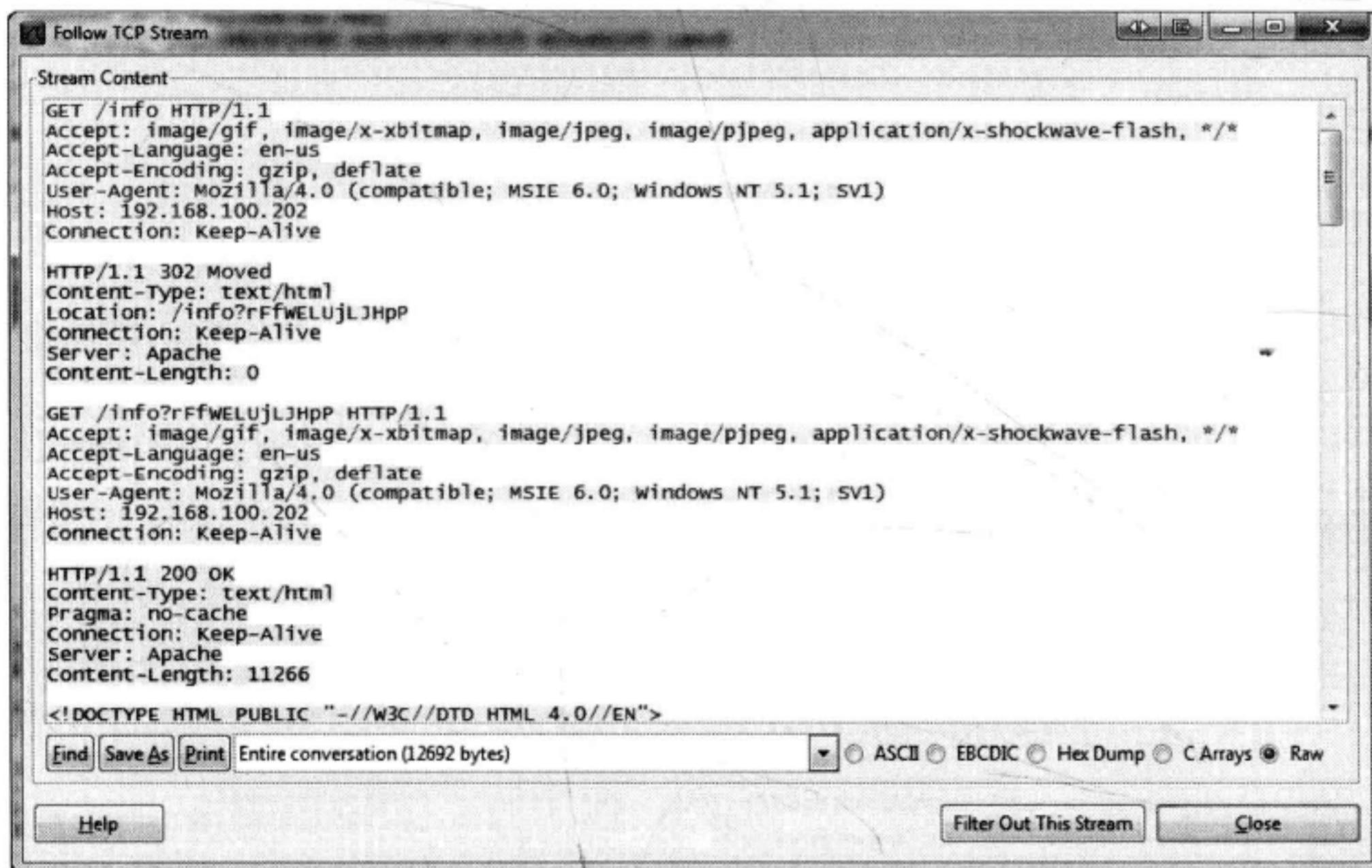


图 10-10 发送到客户端的数据流

在此之后，事情变得十分奇怪了。攻击者对 GET 请求响应了一些看起来非常奇怪的内容。图 10-11 显示了这些内容的第一部分。

这些内容好像是`<script>`标签内的一系列随机数字和字母❶。HTML 内的`<script>`标签表示使用了一种高级脚本语言。在这个标签内，你通常会看到各种脚本语句。但这些乱码表明真正的内容可能已经被特殊编码以逃避检测了。由于我们知道这是一个漏洞利用的流量，我们可以假设这乱七八糟的文本包含了十六进制填充以及真正利用漏洞服务的 shellcode。



图 10-11 <script> 标签内的杂乱内容看起来像是被编码了

图 10-12 显示了攻击者发送的第二部分内容。在已编码文本之后，我们最终看到了一些可读文本。即便没有丰富的编程知识，我们也能看出这些文本像是在一些变量的基础上做了一些字符串解析。这是在闭合标签</script>之前的最后一些文本比特。

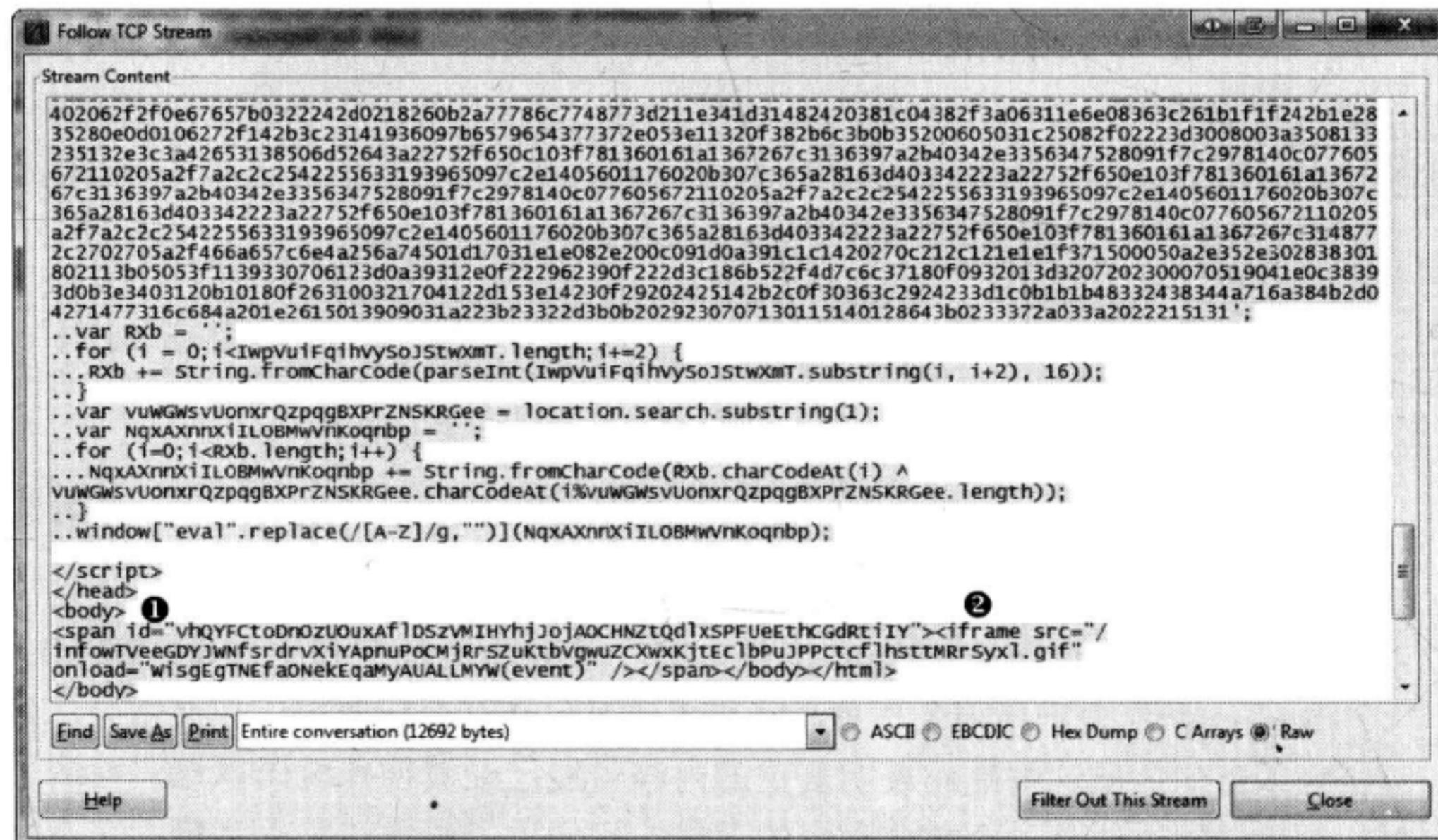


图 10-12 服务器发送的这部分内容包含了可读文本和可疑的 iframe

攻击者发给客户端的最后一段数据包含两个部分。第一部分是<span id="vh YFCtoDnOzUOuxAflDSzVMIHYhjJojAOCHNZtQdlxSPFUeEthCGdRtiIY">**①**。第二部分包含在<span></span>标签内，是<iframe src="/infowTVeeGDYJWNfsrdrv iYApnuPoCMjRrSZuKtbVgwuZCXwxKjtEclbPuJPPctcflhsttMRrSyxl.gif" onload="WisgEg NEfaONekEqaMyAUALLMYW(event)" />**②**。这些内容也有可能是恶意行为的标志，因为这些文本出奇得长、包含不可读的随机字符串和可能被混淆过的文本。

<span>标签内包含了一个 iframe，这是攻击者惯用的手法，用于在 HTML 页面中嵌入额外内容。<iframe>标签创建了一个内联帧，并不被用户察觉。在这个案例中，<iframe>标签引用了一个名字古怪的 GIF 文件。如图 10-13 所示，当受害者的浏览器发现对这个文件的引用时，它在数据包 21 中发送了一个 GET 请求**①**，紧接着 GIF 文件就被传送过来了**②**。这个 GIF 文件有可能被用来以某种方式，触发已经下载到受害者机器上的漏洞利用代码。

| No. | Time     | Source          | Destination     | Protocol | Info  |
|-----|----------|-----------------|-----------------|----------|---|
| 21  | 0.455107 | 192.168.100.206 | 192.168.100.202 | HTTP     | <b>①</b> GET /infowTVeeGDYJWNfsrdrv iYApnuPoCMjRrSZuKtbVgwuZCXwxKjtEclbPuJPPctcflhsttMRrSyxl.gif HTTP/1.1 |
| 22  | 0.199959 | 192.168.100.202 | 192.168.100.206 | TCP      | 80 > 1031 [ACK] Seq=3036736951 Ack=3982971911 Win=64518 Len=0   |
| 23  | 0.001166 | 192.168.100.202 | 192.168.100.206 | HTTP     | <b>②</b> HTTP/1.1 200 OK (GIF89a)   |
| 24  | 0.161592 | 192.168.100.206 | 192.168.100.202 | TCP      | 1031 > 80 [ACK] Seq=3982971911 Ack=3036737098 Win=64093 Len=0   |

图 10-13 受害者请求并下载 iframe 中指明的 GIF 文件

当受害者向攻击者的 4321 端口初始化连接时，文件中最奇怪的一部分出现在数据包 25 中。从 Packet Details 面板中查看第二个通信流并不能得出太多信息，因此我们再次查看 TCP 通信流以更清楚地了解传输的数据。图 10-14 显示了 Follow TCP Stream 窗口的输出。

```

Follow TCP Stream

Stream Content:
....1.d.R0.R..R..r(..J&1.1..<a|...  

...RW.R..B<...@x..tJ..P.H..X ...<I.4...1.1....  

..8.u..}$.u.X$..f..K.X.....D$[[aYZQ..X_Z....]hcmd...www1.j.YV..F.D$<...D  

$...DTPVVVFVNvVSVhY.?....NvF.Oh.....vh.....<.|  

...u..G.roj.S..Microsoft Windows XP [Version 5.1.2600]  

(C) Copyright 1985-2001 Microsoft Corp. ①

C:\Documents and Settings\Administrator\Desktop>dir ②
Volume in drive C has no label.
Volume Serial Number is 84AA-C05E

Directory of C:\Documents and Settings\Administrator\Desktop

07/13/2010 05:33 PM <DIR> .
07/13/2010 05:33 PM <DIR> .. ③
07/13/2010 05:33 PM <DIR> data
07/13/2010 05:33 PM <DIR> docs
07/13/2010 05:34 PM 227 passwords.txt
1 File(s) 227 bytes
4 Dir(s) 19,271,598,080 bytes free

C:\Documents and Settings\Administrator\Desktop>

Find Save As Print Entire conversation (899 bytes) ASCII EBCDIC Hex Dump C Arrays Raw
Help Filter Out This Stream Close

```

图 10-14 攻击者通过这个连接与命令窗口交互

在这里，我们看到了应该立即引发警报的东西：一个 Windows 的命令行解释器①。这个 shell 是由受害者发给服务器的，这表明攻击者成功利用了漏洞：一旦漏洞利用程序启动，客户端就给攻击者发送回一个命令行解释器。在这个捕获中，我们甚至能看到攻击者与受害者的交互：输入 dir 命令②以列出受害者机器的目录内容③。

攻击者进入这个命令行解释器后，对受害者机器就有了无限制的管理权限，他几乎能做任何想做的事情。受害者只不过是轻点了一下鼠标，几秒钟之内就把计算机的完全控制权限交给了攻击者。

像这样的漏洞利用程序在线路上传输信息时通常都编码成不可识别，以避免被网络 IDS 发现。就其本身而言，没有对这个漏洞利用程序的事先了解，也没有该程序的代码样本，在没有进一步分析之前很难说清受害者系统上到底发生了什么。幸好，我们可以从数据包捕获中辨认出恶意代码的一些蛛丝马迹。这包括<script>标签里的一些混淆文本、奇怪的 iframe，以及明文表示的命令行解释器。

我们在这里总结一下极光漏洞利用程序是如何工作的。

- 受害者收到一封来自攻击者的邮件，看起来合理，实际上有针对性，单击里面的一个链接，向攻击者的恶意网站发送一个 GET 请求。
- 攻击者的 Web 服务器向受害者发出 302 重定向，受害者的浏览器向重定向 URL 自动发起一个 GET 请求。
- 攻击者的 Web 服务器向客户端发送一个含有混淆的 JavaScript 代码（包括一个漏洞利用程序）的 Web 页面，以及一个含有恶意 GIF 图像链接的 iframe。
- 受害者向恶意图像发起一个 GET 请求，将它从服务器上下载下来。
- 利用 IE 浏览器的漏洞，使用恶意 GIF 文件反混淆之前发送的 JavaScript 代码，并在受害者机器上执行。
- 一旦漏洞被成功利用，就执行隐藏在混淆代码中的 payload，在 4321 端口打开一个从受害者向攻击者的新会话。
- 该 payload 会产生一个命令行解释器并返回给攻击者，以便攻击者与目标进行交互。

从防御的观点看，我们可以使用这个捕获文件为 IDS 创建一个签名，也许

能有助于检测这种攻击。举个例子，我们可以过滤捕获文件的非混淆部分，比如<script>标签里混淆文本末尾的明文代码。另外一个思路是为所有包含 302 重定向到特定 URL 的 HTTP 流量写一个签名。为能在生产环境中使用这个签名，还需要一些额外的调整，但这已经是个很好的开端。

---

**注意** 对尝试防御网络未知威胁的人而言，基于恶意流量样本创建流量签名是非常关键的一步。这里描述的捕获是提升编写签名技能的好方法。要想了解更多有关入侵检测和攻击特征的信息，请访问 Snort 项目主页 <http://www.nort.org/>。

---

### 10.2.2 ARP 缓存中毒攻击

在第 2 章中，我们讨论了将 ARP 缓存中毒攻击作为监听主机流量的方法。ARP 缓存中毒攻击是网络工程师高效实用的工具。然而，若有恶意企图，它也是一个非常致命的中间人攻击（man-in-the-middle，MITM）方法。

在 MITM 攻击中，攻击者重定向两台主机间的流量，试图在传输过程中对流量进行拦截或修改。MITM 攻击有多种形式，包括会话劫持、DNS 欺骗，以及 SSL 劫持。

ARP 缓存中毒攻击之所以有效，是因为特意构造的 ARP 数据包使两台主机相信它们是在互相通信，而实际上它们却是与一个在中间转发数据包的第三方通信。

文件 arppoison.pcap 包含了 ARP 缓存中毒攻击的一个例子。当你打开它时，第一眼你会发现这些流量看起来很正常。然而，如果你跟进这些数据包，就会发现我们的受害者 172.16.0.107 在浏览 Google 并执行搜索。搜索的结果导致了一些 HTTP 流量，并夹杂一些 DNS 查询。

我们知道 ARP 缓存中毒攻击是发生在第 2 层的技术，所以如果只是在 Packet List 面板里随意浏览，恐怕很难发现任何异常。因此，我们在 Packet List 面板里增加几列，过程如下。

1. 选择 **Edit->Preferences**。
2. 单击 Preferences 窗口左边的 **Columns**。
3. 单击 **Add**。

4. 输入 **Source MAC** 并按回车键。
5. 在 **Field type** 下拉列表里，选择 **Hw src addr (resolved)**。
6. 单击新增加的项，拖动它到 **Source** 列后面。
7. 单击 **Add**。
8. 输入 **Dest MAC** 并按回车键。
9. 在 **Field type** 下拉列表里，选择 **Hw dest addr (resolved)**。
10. 单击新增加的项，拖动它到 **Destination** 列后面。
11. 单击 **OK** 使改动生效。

当你完成这些步骤时，你的屏幕应该如图 10-15 一样。你现在应该有额外的两列，分别显示了数据包的来源 MAC 地址和目标 MAC 地址。

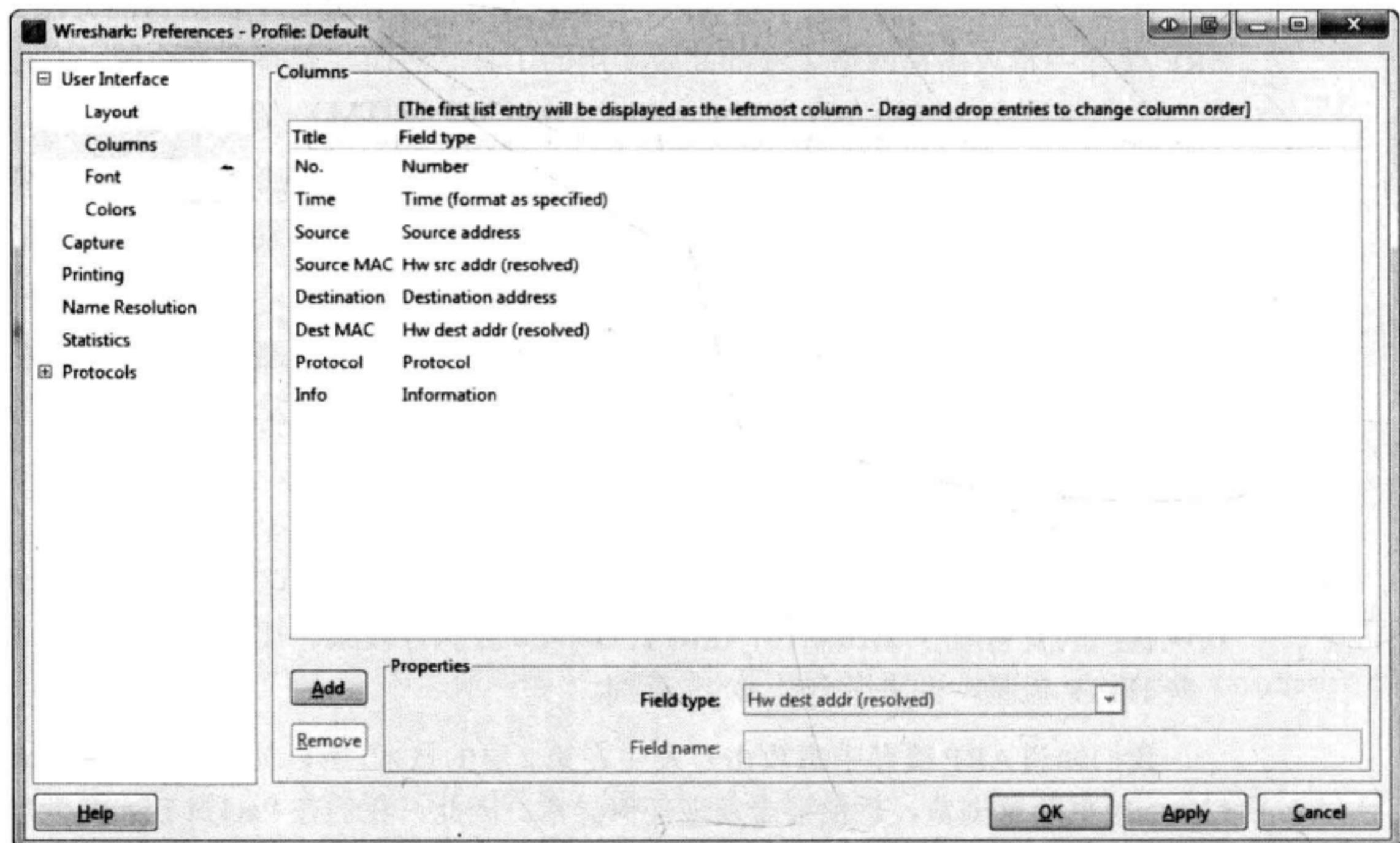


图 10-15 column 配置屏幕，包含了新增列的来源和目标硬件地址列

如果你还打开了 MAC 地址解析，应该会看到通信设备的 MAC 地址表明它是 Dell 或 Cisco 硬件。这是很重要的，因为当我们滚动整个捕获记录时，这些

信息在数据包 54 开始改变了。我们看到了一些奇怪的 ARP 流量，在 Dell 主机（受害者）和新出现的 HP 主机（攻击者）之间传输，如图 10-16 所示。

| No. | Time     | Source            | Source MAC        | Destination       | Dest MAC          | Protocol | Info                                  |
|-----|----------|-------------------|-------------------|-------------------|-------------------|----------|---------------------------------------|
| 54  | 4.171500 | HewlettP_bf:91:ee | HewlettP_bf:91:ee | Dell_c0:56:f0     | ① Dell_c0:56:f0   | ARP      | who has 172.16.0.107? tell 172.16.0.1 |
| 55  | 0.000053 | Dell_c0:56:f0     | Dell_c0:56:f0     | HewlettP_bf:91:ee | HewlettP_bf:91:ee | ARP      | 172.16.0.107 is at 00:21:70:c0:56:f0  |
| 56  | 0.000013 | HewlettP_bf:91:ee | HewlettP_bf:91:ee | Dell_c0:56:f0     | Dell_c0:56:f0     | ARP      | ③ 172.16.0.1 is at 00:25:b3:bf:91:ee  |

图 10-16 Dell 设备和 HP 设备间奇怪的 ARP 流量

在进一步深入之前，注意一下这次通信中涉及的端点，由表 10-3 列出。

表 10-3 监视的端点

| 角色  | 设备类型  | IP 地址        | MAC 地址            |
|-----|-------|--------------|-------------------|
| 受害者 | Dell  | 172.16.0.107 | 00:21:70:c0:56:f0 |
| 路由器 | Cisco | 172.16.0.1   | 00:26:0b:21:07:33 |
| 攻击者 | HP    | 未知           | 00:25:b3:bf:91:ee |

是什么使流量变得奇怪呢？回忆一下我们在第 6 章对 ARP 的讨论，ARP 数据包有两种类型：请求和响应。请求数据包在网络上广播给所有主机，以发现包含特定 IP 地址的机器的 MAC 地址。接着，响应信息作为单播数据包发给请求的设备。在这个背景下，我们从通信序列中发现了一些奇怪的事情，如图 10-16 所示。

首先，数据包 54 是 MAC 地址为 00:25:b3:bf:91:ee 的攻击者发送的 ARP 请求，它作为单播数据包直接发送给 MAC 地址为 00:21:70:c0:56:f0 的受害者①。这种类型的请求本应该广播给网络上所有主机，但它却只是直接发给了受害者。我们又注意到尽管这个数据包是攻击者发送的，并且在 ARP 头部包含了攻击者的 MAC 地址，但它却列出了路由器的 IP 地址，而不是它自己的。

紧随这个数据包的是受害者发给攻击者的响应，包含它的 MAC 地址信息②。最诡异的事情出现在数据包 56 中：攻击者给受害者发送了一个包含未请求的 ARP 响应数据包，告诉它 172.16.0.1 对应的 MAC 地址是 00:25:b3:bf:91:ee③。问题是 172.16.0.1 对应的 MAC 地址不是 00:25:b3:bf:91:ee 而应该是 00:26:0b:21:07:33。因为在之前的数据包捕获中我们看到过路由器 172.16.0.1 与受害者的通信，所以我们知道事实本应如此。由于 ARP 协议内在的不安全性（它的 ARP 表接收未请求的更新），现在受害者会将本应发送到路由器的流量发送给攻击者。

## 注意

因为这些数据包是从受害者机器上捕获的，所以你实际上没有看到事情的全貌。要使攻击生效，攻击者必须给路由器发送同样序列的数据包，骗它认为攻击者就是受害者。但我们需要在路由器（或攻击者）那捕获才能看到这些数据包。

一旦两头都上当，受害者和路由器间的通信就会流经攻击者，如图 10-17 所示。

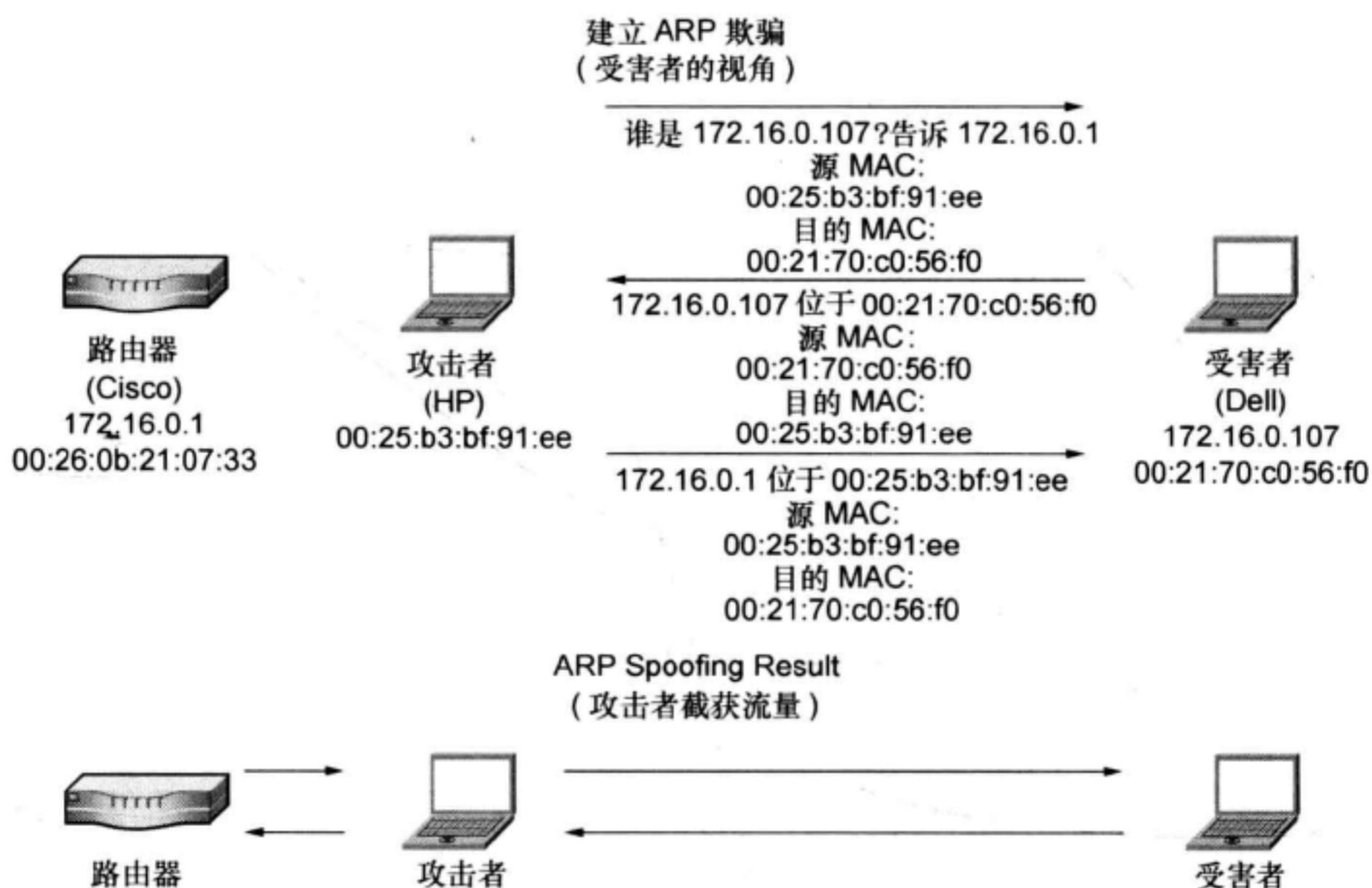


图 10-17 ARP 欺骗导致 MITM 攻击

数据包 57 可以确认攻击取得成功。当你用出现奇怪 ARP 流量前的数据包（比如数据包 40，如图 10-18 所示）与它比较时，你会发现远程服务器（Google）的 IP 地址是一样的❶，但目标 MAC 地址却变化了❷。MAC 地址的变化告诉我们，现在的流量抵达路由器之前将被路由到攻击者。

这个攻击如此狡猾，以至它很难被检测到。要想发现它，你通常需要专门配置 IDS 的帮助，或者在设备上运行能检测 ARP 表项突然变化的软件。因为你很可能会想利用 ARP 缓存中毒攻击来捕获网络上的数据包以便分析，所以了解如何使用这种技术也是很重要的。

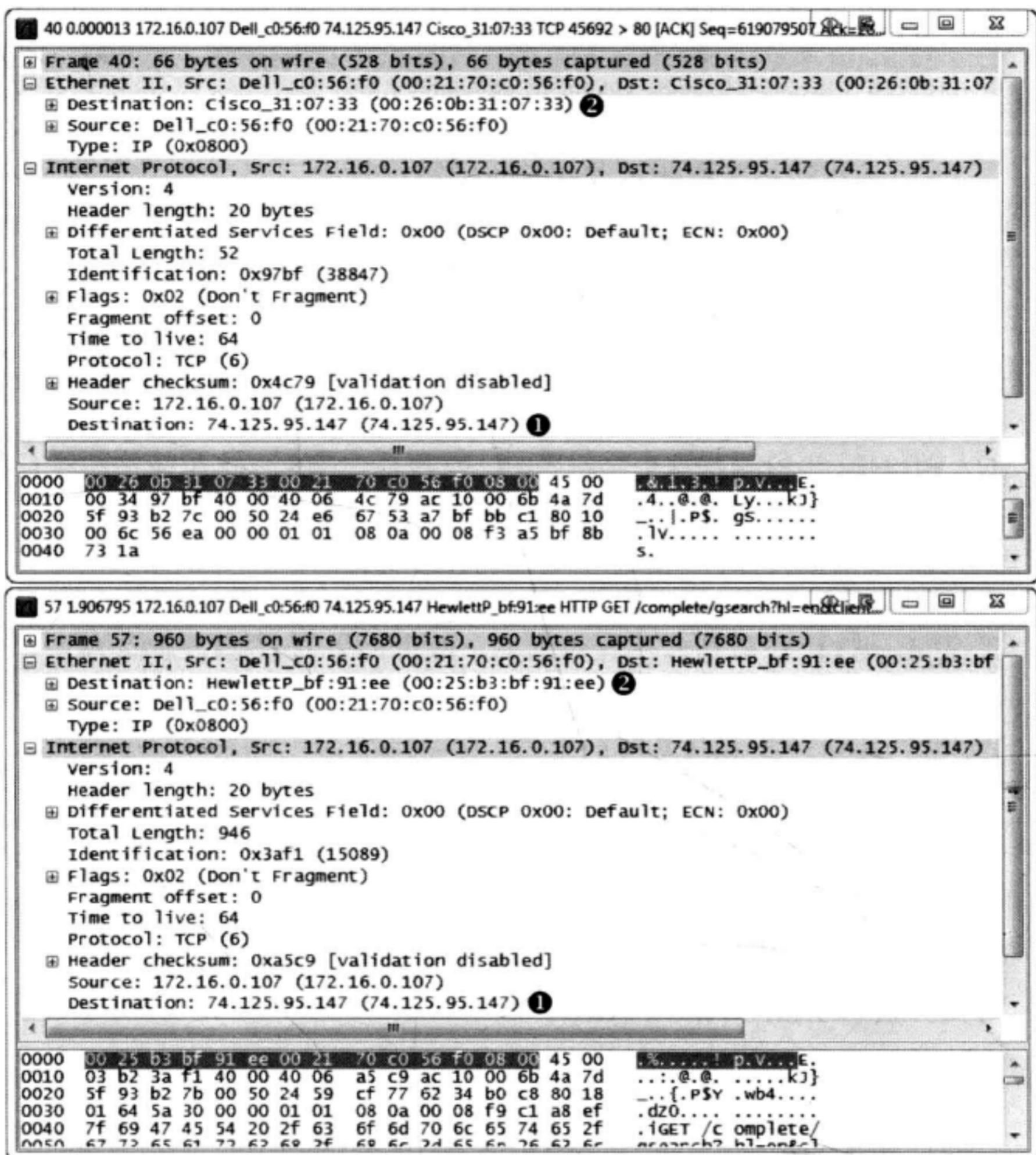


图 10-18 MAC 地址的变化说明这次攻击是成功的

### 10.2.3 远程访问特洛伊木马

到目前为止，我们已经利用经验知识查看了一些安全事件。这是学习攻击形态的好办法，但却不太符合实际。在真实世界里，守护网络安全的人不可能查看网络中的每一个数据包。他们会使用各式各样的 IDS 设备，基于预定义的攻击签名，来提醒他们留意网络流量里的异常情况，并进行进一步检查。

在下一个案例中，我们将像分析真正的网络威胁一样，从一个简单的警报开始。在这个例子中，我们的 IDS 生成了以下这个警报。

```
[**] [1:132456789:2] CyberEYE RAT Session Establishment [**]
[Classification:A Network Trojan was detected] [Priority:1]
07/18-12:45:04.656854 172.16.0.111:4433 -> 172.16.0.114:6641
TCP TTL:128 TOS:0x0 ID:6526 IpLen:20 DgmLen:54 DF
***AP*** Seq:0x53BAEB5E Ack:0x18874922 Win:0xFAF0 TcpLen:20
```

下一步我们将查看触发此次警报的特征规则。

```
alert tcp any any -> $HOME_NET any (msg:"CyberEYE RAT Session Establishment";
content:"|41 4E 41 42 49 4C 47 49 7C|"; classtype:trojan-activity;
content:"|41 4E 41 42 49 4C 47 49 7C|"; classtype:trojan-activity;
sid:132456789; rev:2;)
```

这个规则是这样定义的：当它发现一个进入内网的数据包含有十六进制内容 41 4E 41 42 49 4C 47 49 7C 时，就产生警报。这个内容转换成可读的 ASCII 码是 ANA BILGI。当检测到这个字符串时，警报就会响起，这可能预示着 CyberEYE 远程访问木马（Remote-access Trojan, RAT）的出现。RAT 是秘密运行在受害者计算机上的恶意应用程序，它连接到攻击者，使攻击者能远程管理受害者的机器。

#### 注意

CyberEYE 是来自土耳其的工具，用于产生 RAT 程序和管理“肉鸡”。有趣的是，这里看到的 Snort 规则有一个敏感字符串“ANA BILGI”，它其实是土耳其文字，表示“基本信息”的意思。

现在我们将在 ratinfected.pcap 文件中查看与警报有关的流量。Snort 通常只捕获触发警报的单个数据包，但幸好我们有主机间的完整通信序列。让我们搜索 Snort 规则中提到的十六进制字符串，直接跳到关键部分。

1. 选择 **Edit->Find Packet**。
2. 选择 **Hex Value** 单选按钮。
3. 在文本框输入 **41 4E 41 42 49 4C 47 49 7C**。
4. 单击 **Find**。

如图 10-19 所示，你最先在数据包 4 的数据部分发现了以上字符串①。

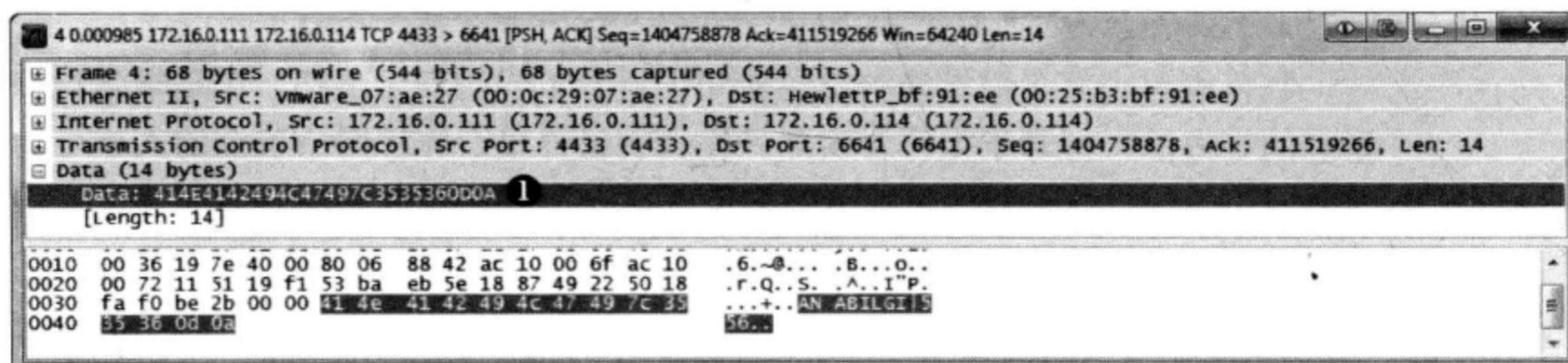


图 10-19 在数据包 4 中发现了 Snort 警报中的字符串内容

若多次选择 **Edit->Find Next**, 你会看见这个字符串也在数据包 5、10、32、156、280、405、531 和 652 出现了。尽管这个捕获文件里的所有通信都是在攻击者 (172.16.0.111) 和受害者 (172.16.0.114) 之间产生的, 但看起来这个字符串出现在了不同的会话中。数据包 4 和 5 使用 4433 端口和 6641 端口进行通信, 而其他大部分实例出现在 4433 端口和其他随机选择的临时端口之间。通过查看 Conversations 窗口的 TCP 标签, 我们可以确认存在多个会话, 如图 10-20 所示。

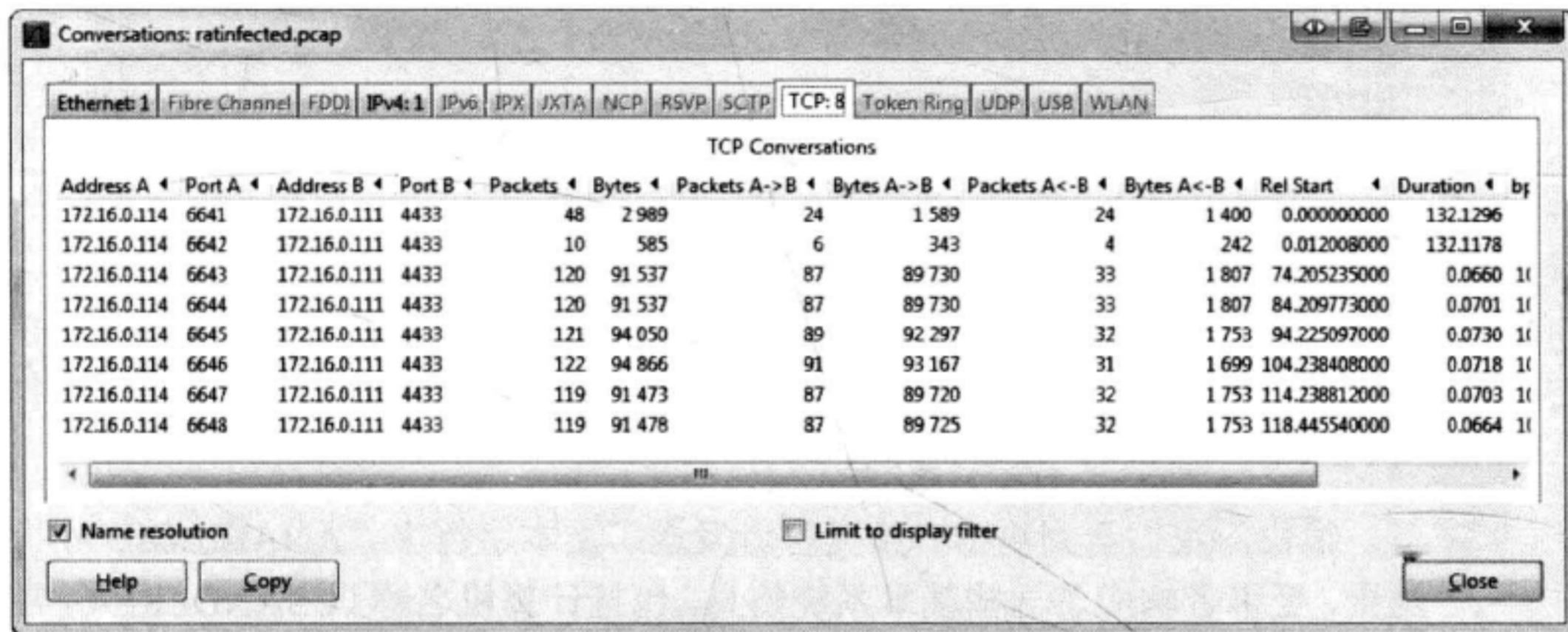


图 10-20 攻击者和受害者之间存在 3 个独立的会话

我们可以给捕获文件里的不同会话刷上不同颜色, 将它们从视觉上分开。

1. 在 Packet List 面板上面的过滤器栏里, 输入过滤器(`tcp.flags.syn = 1`) && (`tcp.flags.ack = 0`)。然后单击 **Apply**。这将筛选出流量中每一个会话的初始 SYN 数据包。
2. 右击第一个数据包, 选择 **Colorize Conversation**。
3. 选择 **TCP**, 然后选择一种颜色。
4. 为剩下的 SYN 数据包重复这个过程, 分别选择不同的颜色。
5. 完成之后, 选择 **Clear** 移除过滤器。

为每个会话着色后, 我们可以看看它们之间有什么关联关系, 这将帮助我们更好地跟踪两台主机之间的通信过程。第一个会话 (ports 6641/4433) 是两台主机通信的开端, 这是一个很好的开始。右击会话里的任何一个数据包, 选择 **Follow TCP Stream** 可以看到被传输的数据, 如图 10-21 所示。

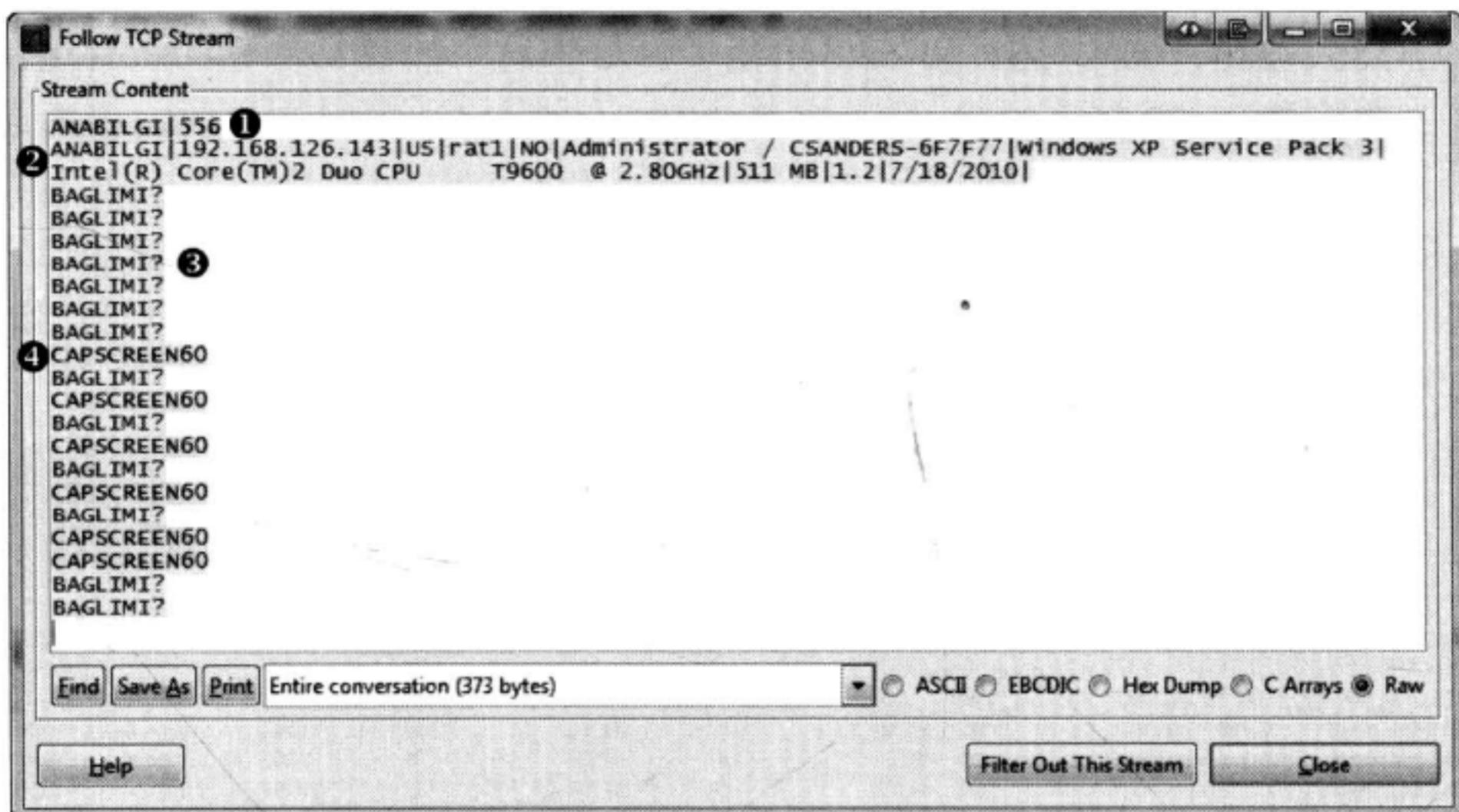


图 10-21 第一个会话产生了有趣的结果

很快，我们看到攻击者给受害者发送了文本字符串“ANABILGI|556”①。结果，受害者响应了一些基本系统信息，包括计算机名称(CSANDERS-6F7F77)和使用的操作系统(Windows XP Service Pack 3)②，并开始给攻击者发送回一些重复的字符串“BAGLIMI?”③。攻击者返回的消息只有字符串“CAPSCREEN60”④，并出现了6次。

攻击者返回的字符串“CAPSCREEN60”很有意思，看看它要带我们去哪里。我们再次使用 search 对话框在数据包中搜索这个文本字符串，指明 String 选项。

我们搜索到数据包 27 首次出现了这个字符串。这个信息的有趣之处在于，客户端一收到攻击者发送的这个字符串，就确认接收这个数据包，并在数据包 29 发起了一个新会话。

现在，如果我们跟随这个新会话(如图 10-22 所示)的 TCP 流输出，就能看到熟悉的字符串“ANABILGI|12”，跟在后面的是字符串“SH|556”，最后是字符串“CAPSCREEN|C:\WINDOWS\jpgevhook.dat|84972”①。注意到字符串“CAPSCREEN”之后指明的文件路径后面跟着不可读的文本。这里最吸引人的是不可读文本有一个前缀字符串“JFIF”②，Google 搜索一下知道，JPG 文件的开头部分通常就包含它。

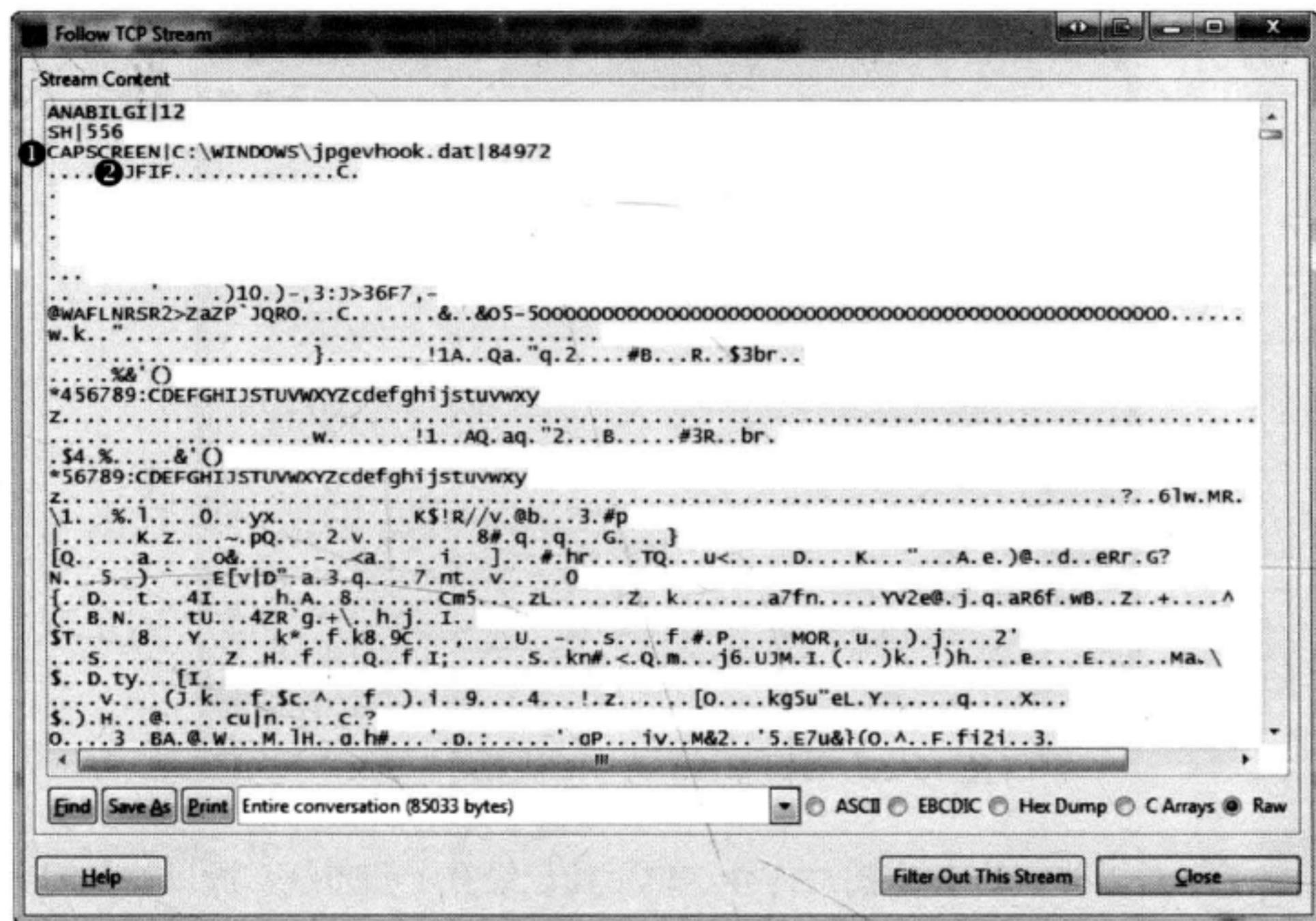


图 10-22 攻击者发起了对一个 JPG 文件的请求

到这里，我们可以负责任地说，攻击者发起新会话是为了传输这个 JPG 图像。但更重要的是，我们开始从流量中推断出一个命令结构。看起来，攻击者发送“CAPSCREEN”命令就可以发起 JPG 图像的传输。实际上，不管什么时候发送“CAPSCREEN”命令，结果都是一样的。为了验证这个结论，可以查看每个会话的流，或使用下面介绍的 Wireshark 的 IO 绘图功能。

1. 选择 Statistics->IO Graphs。
2. 在 5 个过滤器栏中分别插入 tcp.stream eq 2、tcp.stream eq 3、tcp.stream eq 4、tcp.stream eq 5 和 tcp.stream eq 6。
3. 单击 **Graph 1**、**Graph 2**、**Graph 3**、**Graph 4** 和 **Graph 5** 按钮分别启用各个过滤器的数据点。
4. 将 y 轴的单位改成 Bytes/Tick。

图 10-23 显示了结果图像。

从这个图像来看，似乎每个会话包含同样大小的数据，并出现了同样多的时间。现在我们可以总结，这个活动重复了好几次。

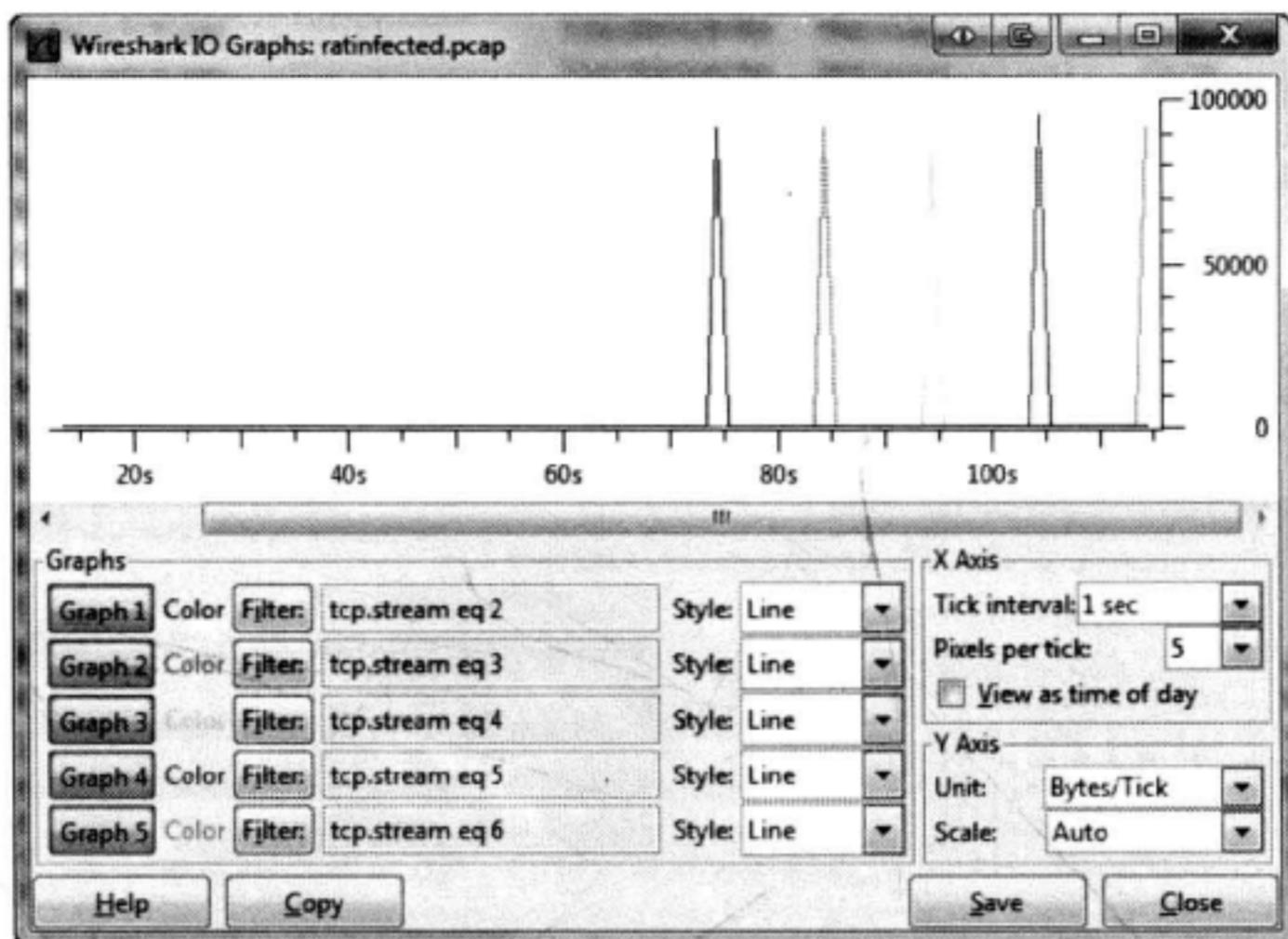


图 10-23 该图像显示同样活动重复了多次

对于传输的 JPG 图片内容，你可能已经有了些想法，所以让我们看看是否能查看这些 JPG 文件。执行以下步骤可以从 Wireshark 中提取 JPG 数据。

1. 首先，对特定数据包的 TCP 流进行重组，如之前图 10-22 所示的文本。
2. 然后通信被分离出来，我们只能看到受害者发送给攻击者的流数据。选择下拉菜单旁边的箭头，那里写着 **Entire Conversation (85033 bytes)**。确保选择合适的流量方向，就是 **172.16.0.114:6643 --> 172.16.0.111:4433 (85020 bytes)**。
3. 选择 **Save As** 按钮保存数据，确保扩展名是.jpg。

你现在试一下，一定会发现图片无法打开。因为我们还需要再做一步。不像在第 8 章从 FTP 流量中提取完整文件那样，这里的流量在真实数据之外还增加了一些额外内容。在这个例子中，TCP 流的前两行实际上是木马命令序列的一部分，而不属于 JPG 数据（如图 10-24 所示）。当我们保存数据流时，这些额外的数据也被保存了下来。结果，文件浏览器查找 JPG 文件头时遇到了预料之外的信息，导致它不能打开图片。

用十六进制编辑器修复这个问题很简单。这个过程叫“文件修复”（File carving）。在图 10-25 中，我已经用 WinHex 选定 JPG 文件前面的一些字节。你可以使用任何十六进制编辑器删除这些字节并保存图片文件。

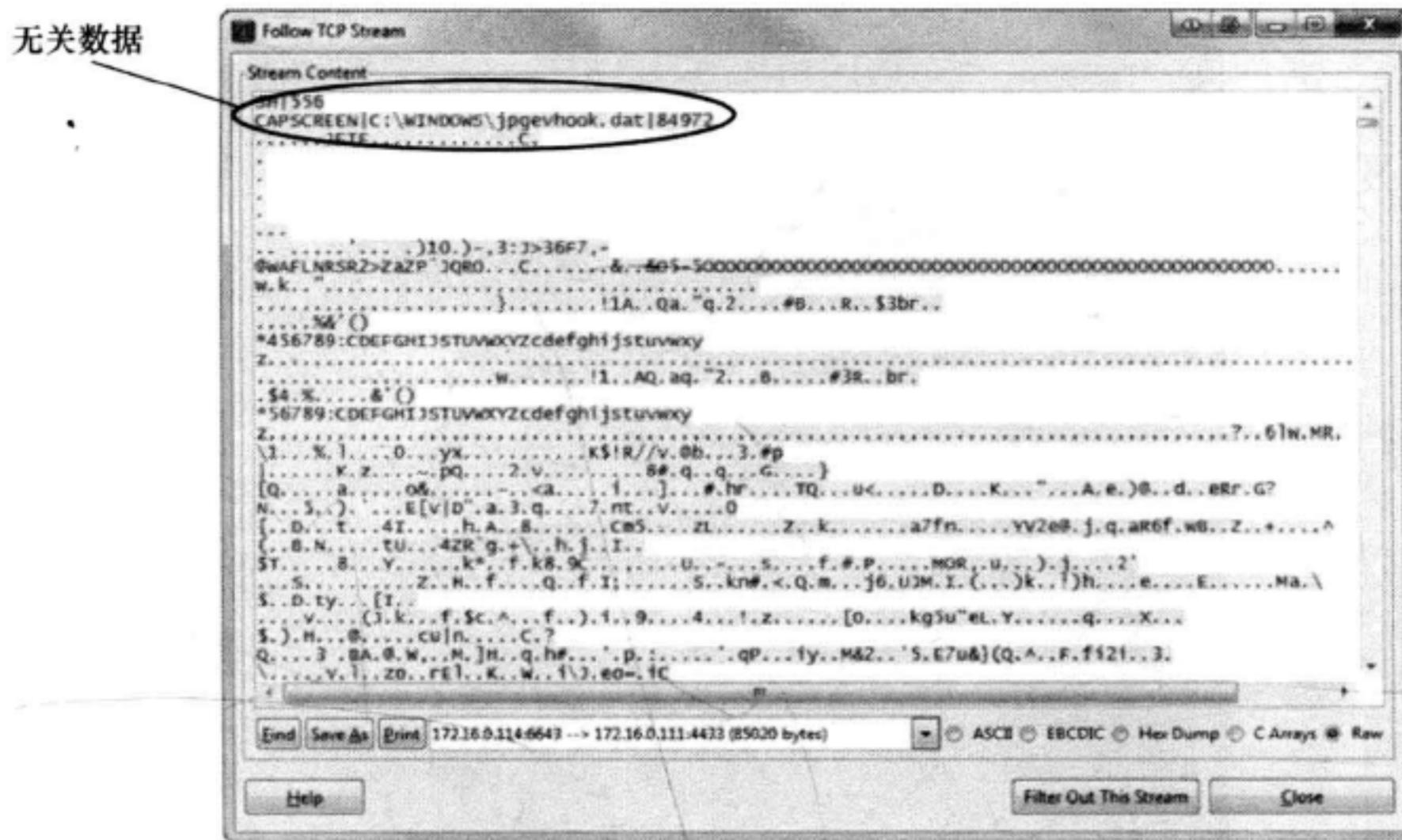


图 10-24 特洛伊木马增加的额外数据导致文件无法正确打开

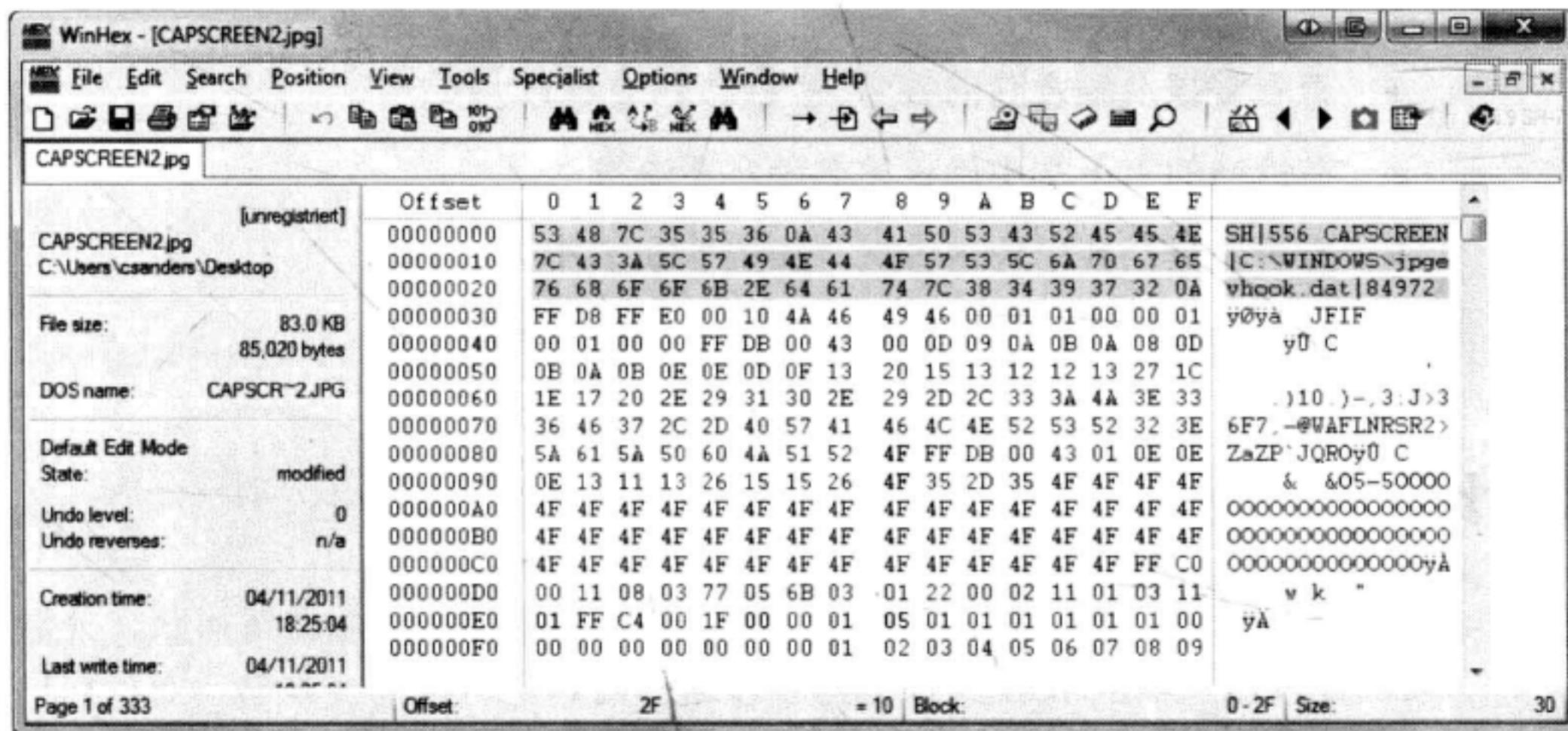


图 10-25 移除 JPG 文件中的附加字节

移除不需要的数据后，文件应该能够打开了。很显然，木马将受害者的桌面进行截屏，并发送给攻击者（见图 10-26）。

这些通信序列完成之后，通信就随着正常 TCP 拆除顺序而结束了。

这个场景展示的思考过程，就是一位入侵分析师分析 IDS 警报流量时应遵循的典型步骤。



图 10-26 发送的 JPG 文件是受害者计算机的屏幕截图

- 查看警报及触发它的签名。
- 在恰当的上下文中确定签名确实在流量中。
- 查看流量，找出攻击者对“肉鸡”动了什么手脚。
- 在“肉鸡”泄露更多敏感信息之前，控制局面。

## 10.3 小结

在本章中，我们讲了如何在安全相关的场景中解析数据包捕获、分析常见攻击技术以及响应 IDS 警报等内容。这些内容足可以写一本书。而我们只是学习了一些常用的扫描和枚举类型、一种常见的中间人攻击技术、两个如何利用系统漏洞的例子，以及一旦发生这些情况会有什么后果而已。

# 第 11 章

## 无线网络数据包分析



与传统有线网络相比，无线网络稍微有些不同。虽然我们仍然在处理 TCP、IP 等常见的通信协议，但是当移到 OSI 模型最底层时，游戏便发生了一点变化。在这里，由于无线网络和物理层的本质属性，数据链路层变得尤为重要。这给我们捕获和访问数据增加了新的限制。

考虑到这些额外因素，你应该不会惊讶于这一整章都将讨论无线网络中的数据包捕获和分析。本章我们将讨论为什么无线网络在数据包分析中比较特殊，以及如何克服这些困难。当然，我们会通过捕获无线网络的实际例子来进行说明。

### 11.1 物理因素

在无线网络中捕获和分析传输数据，首先考虑的是物理传输介质。到目前

为止，我们都没有考虑物理层，因为我们一直在物理的线缆上通信。现在我们通过不可见的无线电波通信，数据包就从我们身边飞过。

### 11.1.1 一次嗅探一个信道

当从无线局域网（Wireless Local Area Network, WLAN）捕获流量时，最特殊的莫过于无线频谱是共享介质。不像有线网络的每个客户端都有它自己的网线连接到交换机，无线通信的介质是客户端共享的空间。单个 WLAN 只占用 802.11 频谱的一部分。这允许同一个物理空间的多个系统在频谱不同的部分进行操作。

#### 注意

无线网络的基础是美国电子和电气工程师协会（Institute of Electrical and Electronics Engineers, IEEE）开发的 802.11 标准。整章涉及的“无线网络”“WLAN”等术语均指 802.11 标准中的网络。

空间上的分离是通过将频谱划分为不同信道实现的。一个信道只是 802.11 无线频谱的一部分。在美国，有 11 个信道可用（有些国家允许使用更多的信道）。这是很重要的，因为 WLAN 同时只能操作一个信道，就意味着我们只能同时嗅探一个信道，如图 11-1 所示。所以，如果你要处理信道 6 的 WLAN，就必须将系统配置成捕获信道 6 的流量。

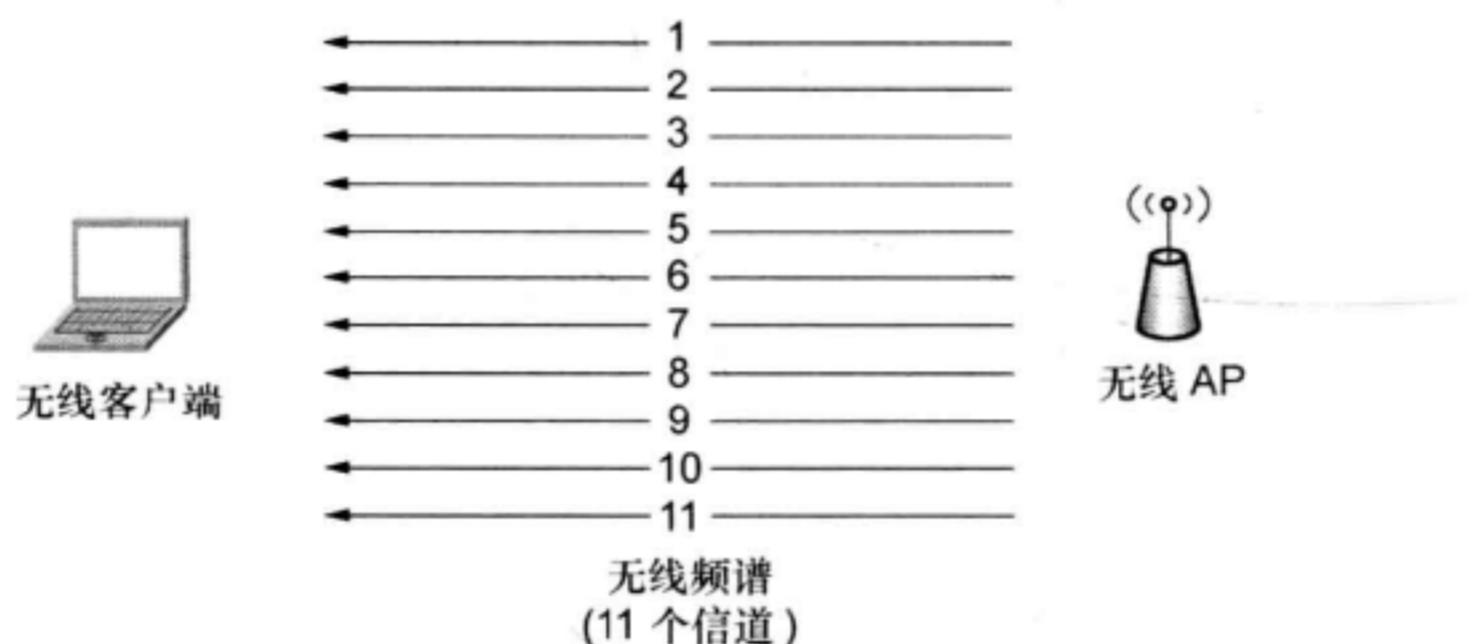


图 11-1 嗅探无线网络很麻烦，因为同一时间只能处理一个信道

#### 注意

传统的无线嗅探只能同时处理一个信道，但有一个例外：某些无线扫描应用程序使用“跳频”技术，可以迅速改变监听信道以收集更多数据。其中最流行的工具是 Kismet (<http://www.kismetwireless.net/>)，可以每秒跳跃 10 个信道，从而高效地嗅探多个信道。

### 11.1.2 无线信号干扰

当有其他因素干扰信号时，无线通信不能保证空气中传输的数据是完整的。无线网络有一定的抗干扰特性，但并不完全可靠。因此，当从无线网络捕获数据包时，你必须注意周边环境，确保没有大的干扰源，比如大型反射面、大块坚硬物体、微波炉、2.4Ghz 无绳电话、厚墙面，以及高密度表面等。这些可能导致数据包丢失、数据包重复或数据包损坏。

同时你还要考虑信道间干扰。虽然同一时刻只能嗅探一个信道，但还是有个小小的忠告：无线频谱被分为多个不同的传输信道，但因为频谱空间有限，信道间有些许重叠，如图 11-2 所示。这意味着，如果信道 4 和信道 5 上都有流量，当你在其中一个信道上嗅探时，会捕获到另一个信道上的数据包。通常，同一地域上的多个网络被设置成使用 1、6 和 11 这 3 个不重叠信道，所以你可能不会遇到这个问题，但以防万一，你还是要了解这是怎么回事。

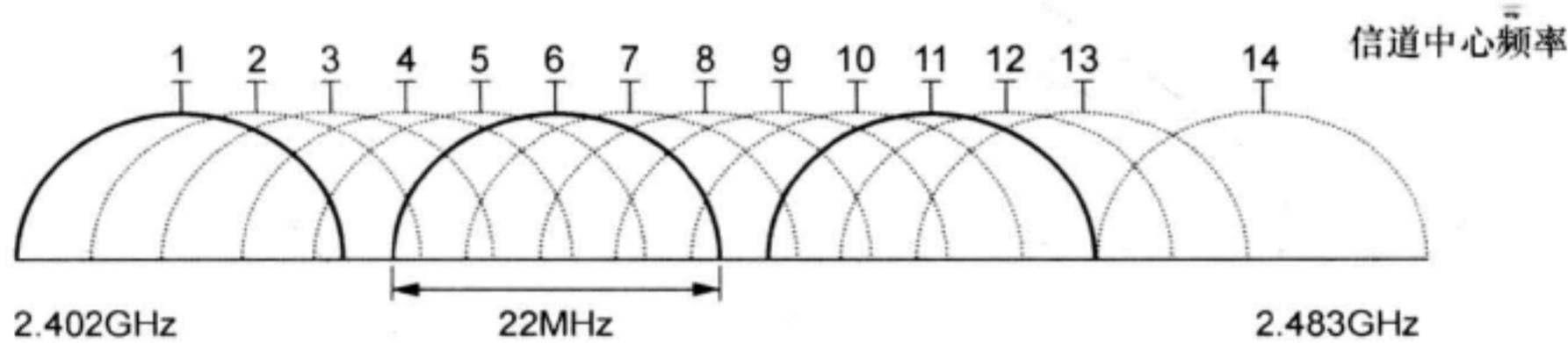


图 11-2 由于频谱空间有限，信道之间有重叠

### 11.1.3 检测和分析信号干扰

无线信号干扰的问题不是在 Wireshark 上观察数据包就能解决的。如果你致力于维护 WLAN，就应该定期检测信号干扰。这可以用频谱分析仪来完成，它可以显示频谱上的数据或干扰。

商业的频谱分析仪价格昂贵甚至高达数千美元，但对于日常使用则有更好的方案。MetaGeek 开发了一个叫 Wi-Spy 的产品，这是一个 USB 硬件设备，用于监测整个 802.11 频谱上的干扰。与 MetaGeek 的 Chanalyzer 软件搭配后，这个硬件可以输出图形化频谱，有助于解决无线网络的问题。Chanalyzer 的示例输出如图 11-3 所示。

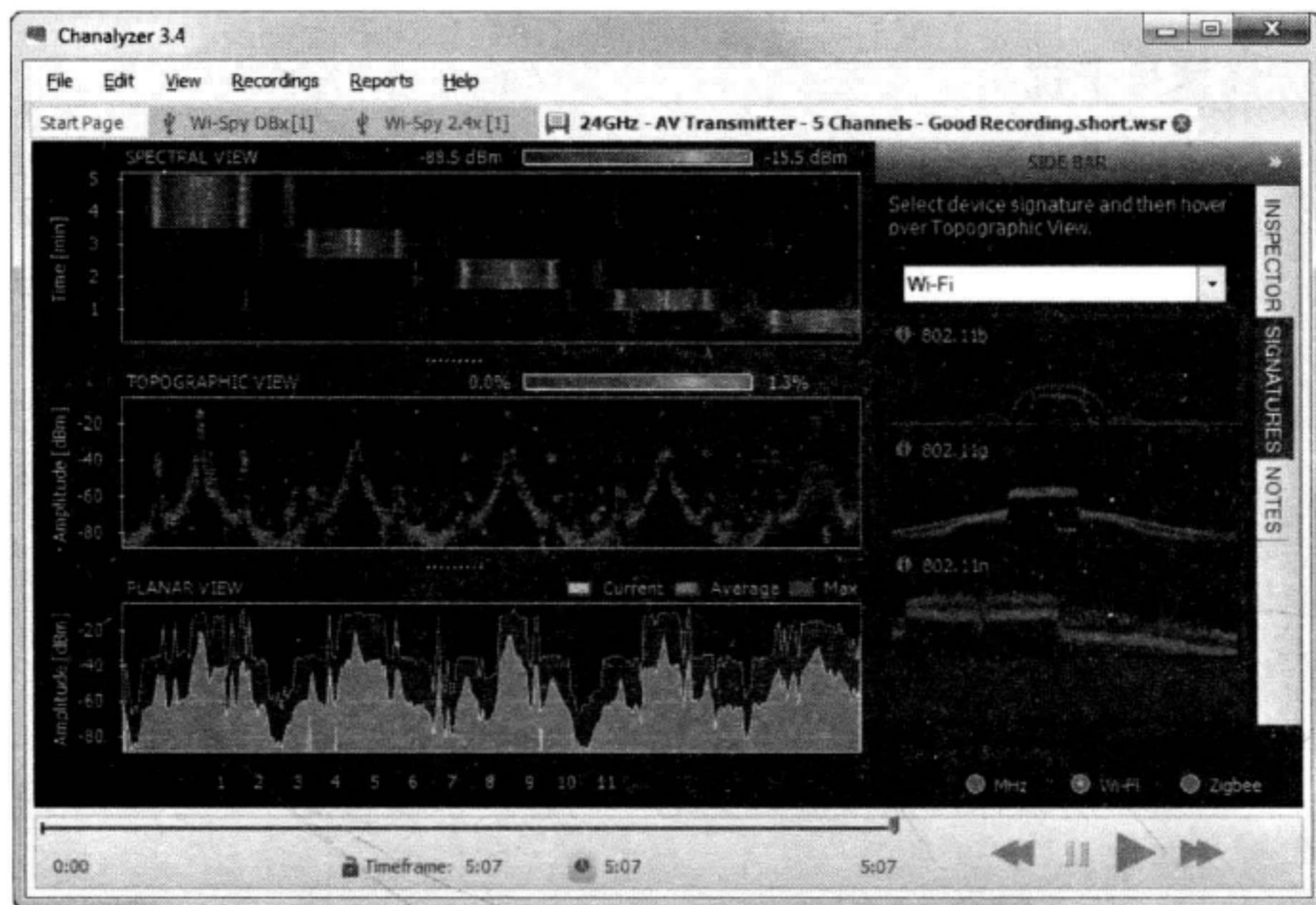


图 11-3 这个 Chanalyzer 显示同一地点有多个 WLAN 在工作

## 11.2 无线网卡模式

在开始嗅探无线数据包之前，我们需要了解无线网卡的不同工作模式。

无线网卡一共有 4 种工作模式。

**被管理模式(Managed mode):** 当你的无线客户端直接与无线接入点 (Wireless Access Point, WAP) 连接时，就使用这个模式。在这个模式中，无线网卡的驱动程序依赖 WAP 管理整个通信过程。

**Ad hoc 模式:** 当你的网络由互相直连的设备组成时，就使用这个模式。在这个模式中，无线通信双方共同承担 WAP 的职责。

**主模式 (Master mode):** 一些高端无线网卡还支持主模式。这个模式允许无线网卡使用特制的驱动程序和软件工作，作为其他设备的 WAP。

**监听模式 (Monitor mode):** 就我们的用途而言，这是最重要的模式。当你希望无线客户端停止收发数据，专心监听空气中的数据包时，就使用监听模式。

要使 Wireshark 捕获无线数据包，你的无线网卡和配套驱动程序必须支持监听模式（也叫 RFMON 模式）。

大部分用户只使用无线网卡的被管理模式或 ad hoc 模式。图 11-4 展示了各种模式如何工作。

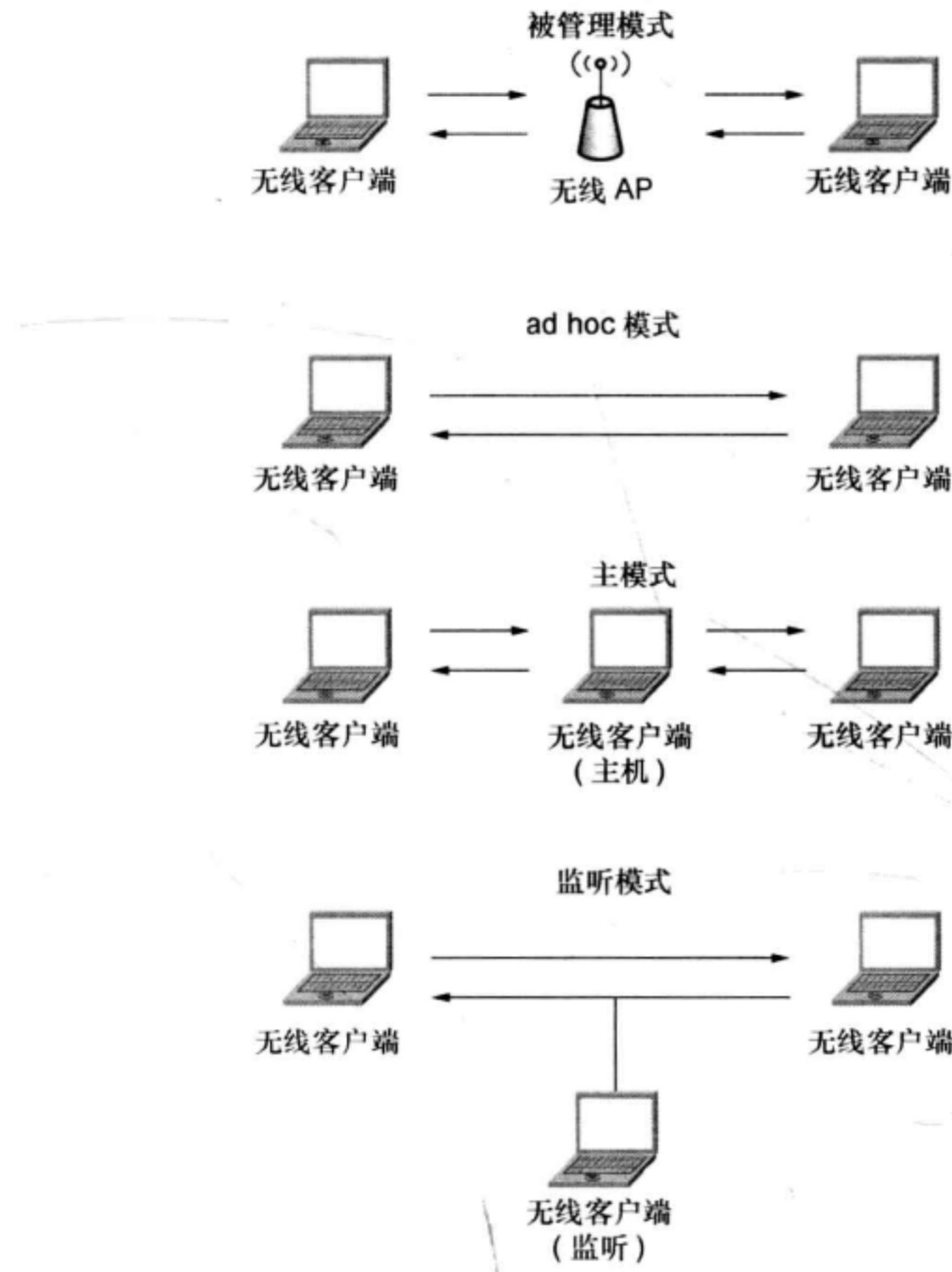


图 11-4 不同的无线网卡模式

#### 注意

经常有人问我推荐哪款无线网卡做数据包分析。我强烈推荐自己使用的 ALFA 1000mW USB 无线适配器。这是市场上最好的产品之一，确保你捕获到每一个可能的数据包。大部分在线计算机硬件零售商都销售此款产品。

## 11.3 在 Windows 上嗅探无线网络

即使你有支持监听模式的无线网卡，大部分基于 Windows 的无线网卡驱动也不允许你切换到这个模式（WinPcap 也不支持这么做）。你需要一些额外的硬件来完成工作。

### 11.3.1 配置 AirPcap

AirPcap（Riverbed 旗下 CACE Technologies 公司的产品，<http://www.cacetech.com/>）被设计用来突破 Windows 强加给无线数据包分析的限制。AirPcap 像 U 盘一样小巧，如图 11-5 所示，用于捕获无线流量。AirPcap 使用第 3 章讨论的 WinPcap 驱动和一个特制的客户端配置工具。



图 11-5 AirPcap 的设计非常紧凑，适合与笔记本电脑一同携带

AirPcap 的配置程序很简单，只有一些配置选项。如图 11-6 所示，AirPcap 控制面板提供了以下几个选项。

**Interface:** 你可以在这里选择要捕获的设备。一些高级的分析场景会要求你使用多个 AirPcap 设备，同时嗅探多个信道。

**Blink Led:** 选中这个按钮会使 AirPcap 设备上的 LED 指示灯闪烁。当存在多个 AirPcap 设备时，可用来识别正在使用的适配器。

**Channel:** 在下拉菜单里，你可以选择希望 AirPcap 监听的信道。

**Include 802.11 FCS in Frames:** 默认情况下，一些系统会去掉无线数据包的最后 4 个校验和比特。这个被称为帧校验序列（Frame Check Sequence，FCS）的校验和用来确保数据包在传输过程中没被破坏。除非你有特别的理由，否则请勾选这个复选框（包含 FCS 校验和）。

**Capture Type:** 这里有两个选项：802.11 Only 和 802.11 + Radio。802.11 Only

选项包含标准的 802.11 数据包头。802.11 + Radio 选项包含这个包头以及前端的 radiotap 头部，因而包含额外信息，比如数据率、频率、信号等级和噪声等级。选择 802.11 + Radio 以观察所有可用的数据信息。

**FCS Filter:** 即便你没有选择 Include 802.11 FCS in Frames，这个选项也可以过滤 FCS 认为已经被损坏的数据包。使用 Valid Frames 选项可以只显示 FCS 认为成功接收的那些数据包。

**WEP Configuration:** 这个区域（在 AirPcap Control Panel 的 Keys 选项卡可见）允许你输入所嗅探网络的 WEP 密码。为了能解密 WEP 加密的数据，你需要在这里填入正确的 WEP 密码。WEP 密码将在 11.8 节“无线网络安全”中讨论。

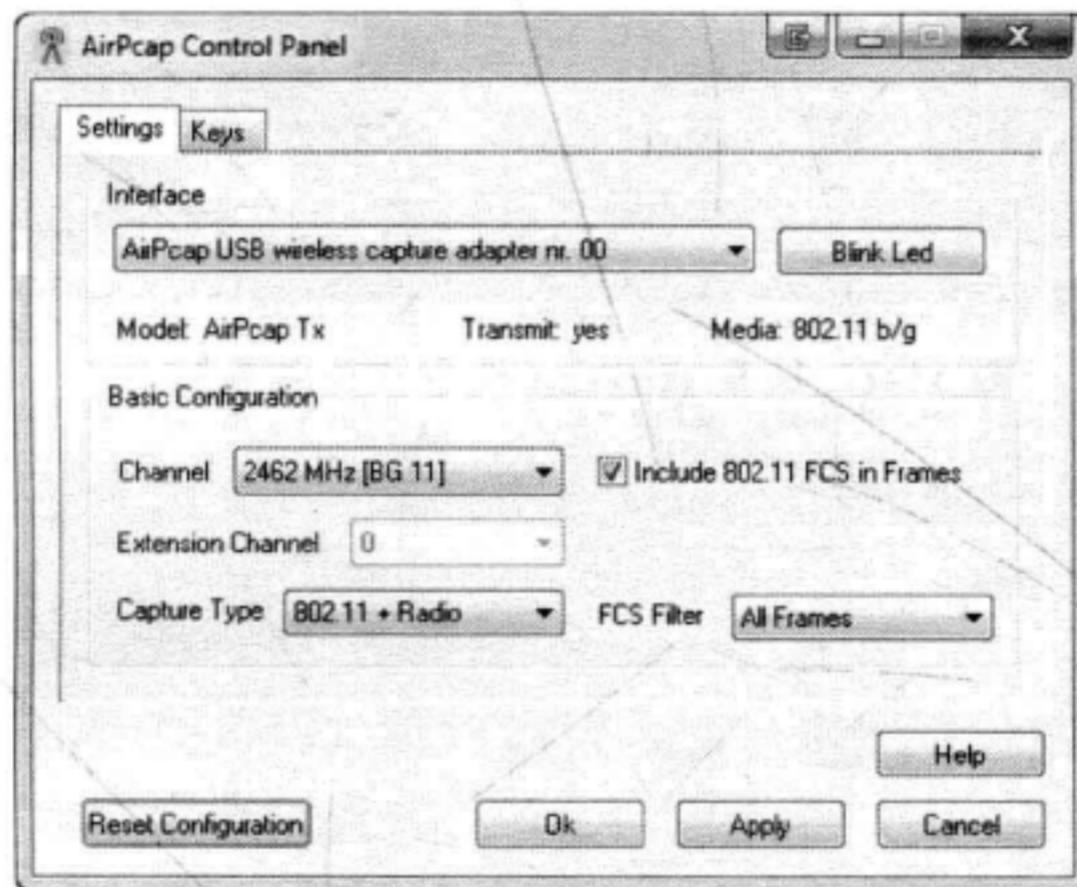


图 11-6 AirPcap 配置程序

### 11.3.2 使用 AirPcap 捕获流量

一旦安装并配置好 AirPcap，接下来的捕获过程你已经很熟悉了。只要启动 Wireshark 并选择 **Capture->Options**。接着，在 **Interface** 下拉框中选择 AirPcap 设备 ①，如图 11-7 所示。

除了 Wireless Settings 按钮外，屏幕上的一切都很熟悉。单击这个按钮会给出与 AirPcap 配置程序一样的选项，如图 11-8 所示。AirPcap 是完全嵌入 Wireshark 的，因此所有配置都可以在 Wireshark 中修改。

一切都配置好之后，就可以单击 **Start** 按钮开始捕获数据包了。

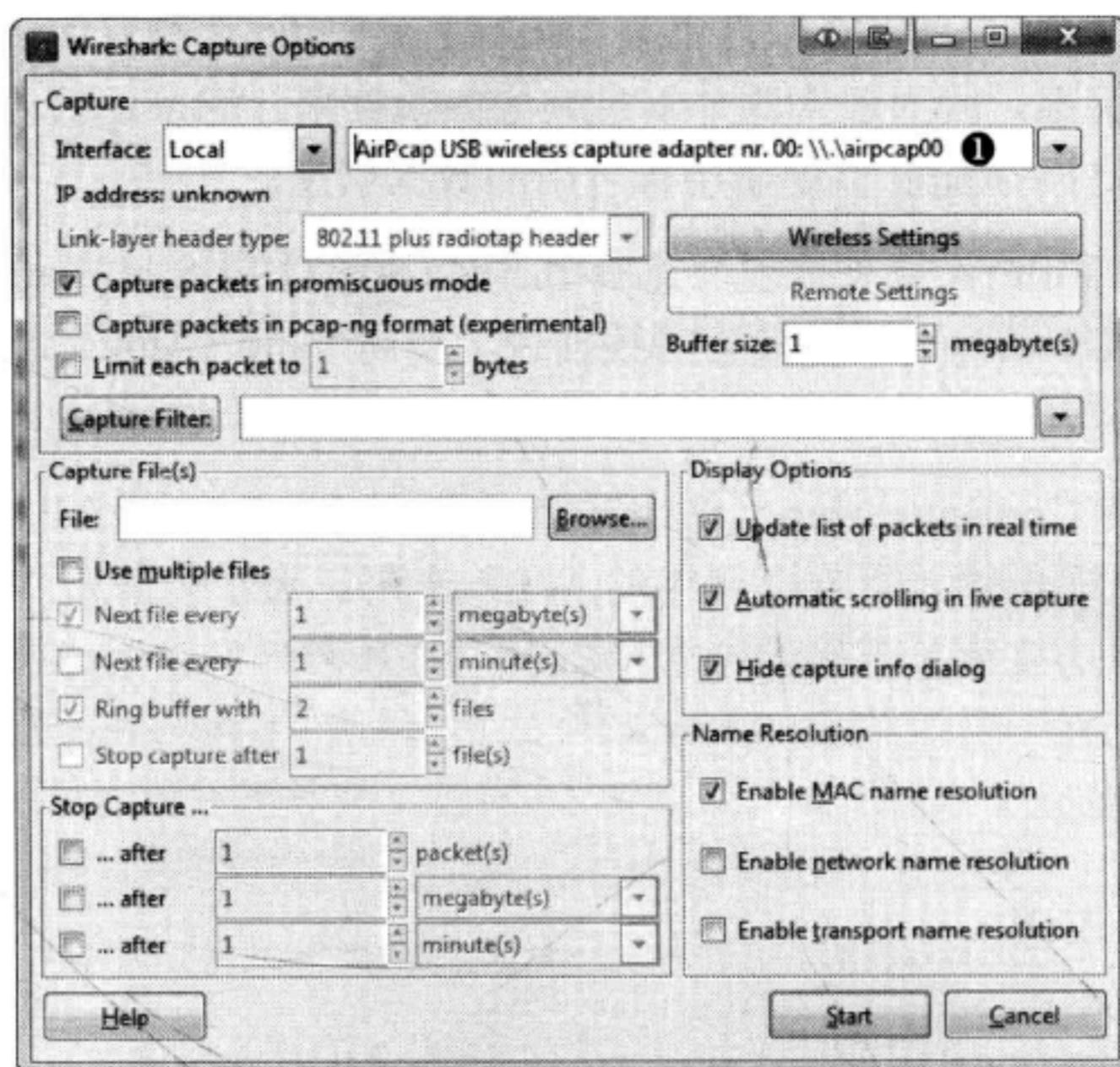


图 11-7 选择 AirPcap 设备作为捕获接口

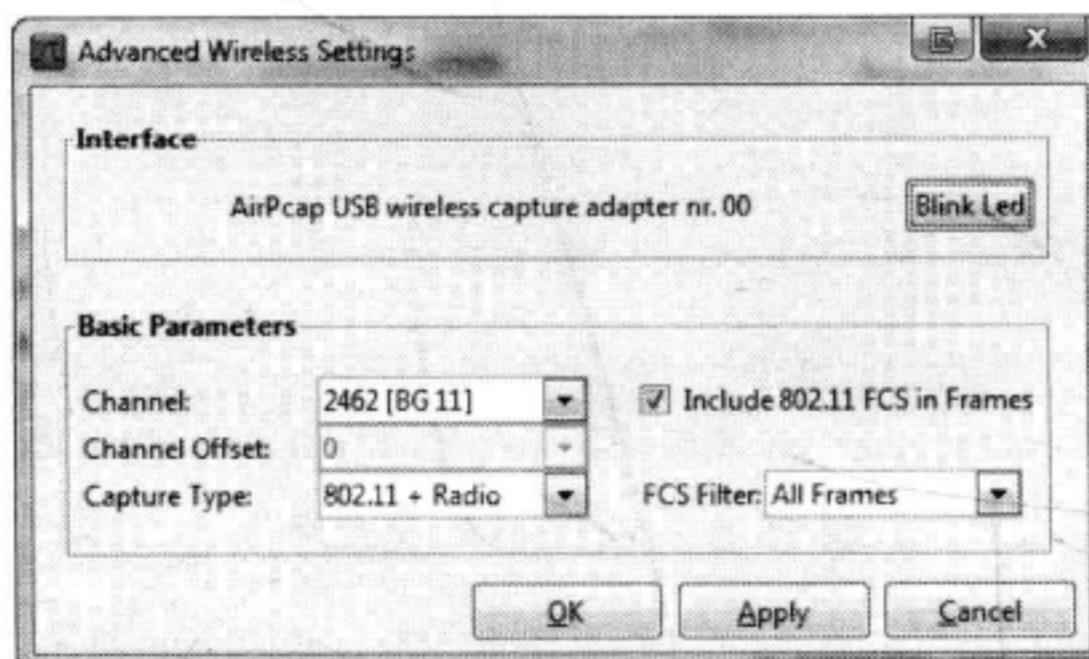


图 11-8 Advanced Wireless Settings 对话框允许你在 Wireshark 中配置 AirPcap

## 11.4 在 Linux 上嗅探无线网络

在 Linux 系统嗅探只需要简单地启用无线网卡的监听模式，然后启动 Wireshark 即可。不幸的是，不同型号无线网卡启用监听模式的流程各不相同，所以在这里我不能给出明确提示。实际上，有些无线网卡并不要求你启用监听模式。你最好 Google 一下你的网卡型号，确定是否需要启用它，以及如何启用。

在 Linux 系统中，通过内置的无线扩展程序启用监听模式是常用的办法之

一。你可以用 iwconfig 命令打开无线扩展程序。如果你在控制台上键入 iwconfig，应该会看到这样的结果。

---

```
$ iwconfig
eth0    no wireless extensions
lo0    no wireless extensions
eth1    IEEE 802.11gESSID:"Tesla Wireless Network"
        Mode:Managed Frequency:2.462 GHz Access Point:00:02:2D:8B:70:2E
        Bit Rate:54 Mb/s Tx-Power-20 dBm Sensitivity=8/0
        Retry Limit:7 RTS thr: off Fragment thr: off
        Power Management: off
        Link Quality=75/100 Signal level=-71 dBm Noise level=-86 dBm
        Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
        Tx excessive retries:0 Invalid misc:0 Missed beacon:2
```

---

iwconfig 命令的输出显示 eth1 接口可以进行无线配置。这是显然的，因为它显示了与 802.11g 协议有关的数据，反观 eth0 和 lo0，它们只返回了“no wireless extensions”。

这个命令提供了许多无线配置信息，仔细看下，有无线扩展服务设置 ID (Extended Service Set ID, ESSID)、频率等。我们注意到“eth1”下面一行显示，模式已经被设置为“被管理”，这就是我们想改动的地方。

要将 eth1 改成监听模式，你必须以 root 用户身份登录。可以直接登录或用切换用户 (su) 命令，如下所示。

---

```
$ su
Password:<enter root password here>
```

---

你成为 root 用户后，就可以键入命令来配置无线网卡选项。输入以下命令可以将 eth1 配置成监听模式。

---

```
# iwconfig eth1 mode monitor
```

---

网卡进入监听模式后，再次运行 iwconfig 命令应该能反映出变化。输入以下命令，以确保 eth1 接口可以工作。

---

```
# iwconfig eth1 up
```

---

我们也将使用 iwconfig 命令改变监听信道：输入以下命令，改变 eth1 接口的信道为信道 3。

---

```
# iwconfig eth1 channel 3
```

---

**注意** 你可以在捕获数据包的过程中随意修改信道，所以随便改吧，没事！也可以将 iwconfig 命令脚本化以简化过程。

---

完成这些配置后，启动 Wireshark 开始你的数据包捕获之旅！

## 11.5 802.11 数据包结构

无线数据包与有线数据包的主要不同在于额外的 802.11 头部。这是一个第 2 层的头部，包含与数据包和传输介质有关的额外信息。802.11 数据包有 3 种类型。

**管理：**这些数据包用于在主机之间建立第 2 层的连接。管理数据包还有些重要的子类型，包括认证（authentication）、关联（association）和信号（beacon）数据包。

**控制：**控制数据包允许管理数据包和数据数据包的发送，并与拥塞管理有关。常见的子类型包括请求发送（request-to-send）和准予发送（clear-to-send）数据包。

**数据：**这些数据包含有真正的数据，也是唯一可以从无线网络转发到有线网络的数据包。

一个无线数据包的类型和子类型决定了它的结构，因此各种可能的数据包结构不计其数。我们将考察其中一种结构，请看 80211beacon.pcap 文件里的单个数据包。这个文件包含一种叫 beacon 的管理数据包的例子，如图 11-9 所示。

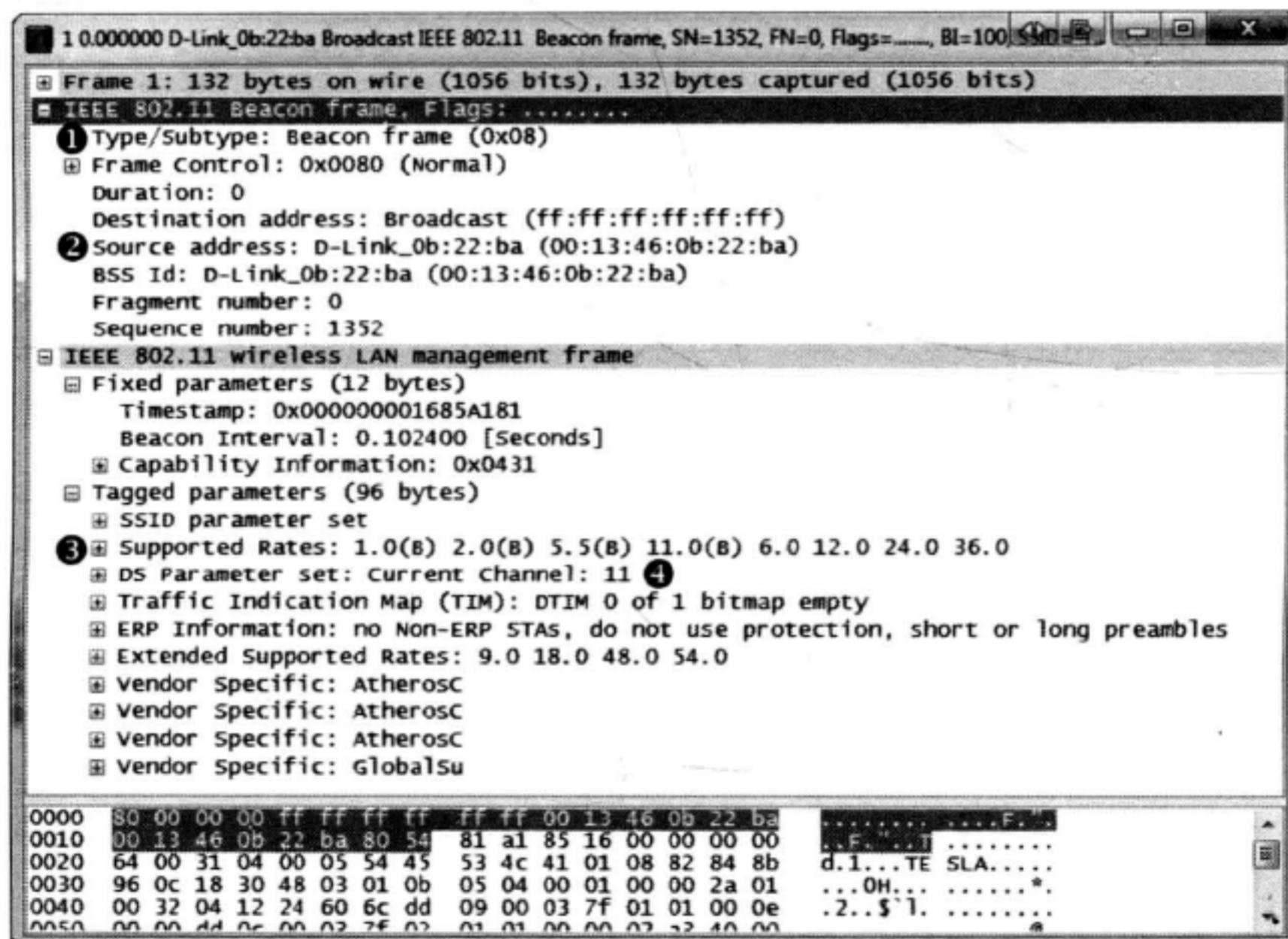


图 11-9 这是一个 802.11 beacon 数据包

beacon 是你能找到的最有信息量的无线数据包之一。它作为一个广播数据

包由 WAP 发送，穿过无线信道通知所有无线客户端存在这个可用的 WAP，并定义了连接它必须设置的一些参数。在我们的示例文件中，你可以看到这个数据包在 802.11 头部的 Type/Subtype 域被定义为 beacon❶。

在 802.11 管理帧头部发现了其他信息，包括以下内容。

**Timestamp** 发送数据包的时间戳。

**Beacon Interval** beacon 数据包重传间隔。

**Capability Information** WAP 的硬件容量信息。

**SSID Parameter Set** WAP 广播的 SSID（网络名称）。

**Supported Rates** WAP 支持的数据传输率。

**DS Parameter** WAP 广播使用的信道。

这个头部也包含了来源和目的地址以及厂商信息。

在这些知识的基础上，我们可以了解到示例文件中发送 beacon 的 WAP 的很多信息。显然这是一台 D-Link 设备❷，使用 802.11b 标准（B）❸，在信道 11 上工作❹。

虽然 802.11 管理数据包的具体内容和用途不一样，但总体结构跟这个例子相差不大。

## 11.6 在 Packet List 面板增加无线专用列

如你所见，Wireshark 通常在 Packet List 面板显示 6 个不同的列。分析无线数据包之前，让我们在 Packet List 面板增加 3 个新列。

- RSSI (for Received Signal Strength Indication) 列，显示捕获数据包的射频信号强度。
- TX Rate (for Transmission Rate) 列，显示捕获数据包的数据率。
- Frequency/Channel 列，显示捕获数据包的频率和信道。

当处理无线连接时，这些提示信息将会非常有用。例如，即使你的无线客户端软件告诉你信号强度很棒，捕获数据包并检查这些列，也许会得到与之前结果不符的数字。

按照以下步骤，在 Packet List 面板增加这些列。

1. 选择 **Edit->Preferences**。
2. 到 **Columns** 部分，并单击 **Add**。
3. 在 **Title** 域键入 **RSSI**，并在域类型下拉列表中选择 **IEEE 802.11 RSSI**。
4. 为 TX Rate 和 Frequency/Channel 列重复此过程，为它们取个恰当的 **Title**，并在 **Field type** 下拉列表选择 **IEEE 802.11 TX Rate** 和 **Channel/Frequency**。添加 3 列之后，Preferences 窗口应该像图 11-10 一样。

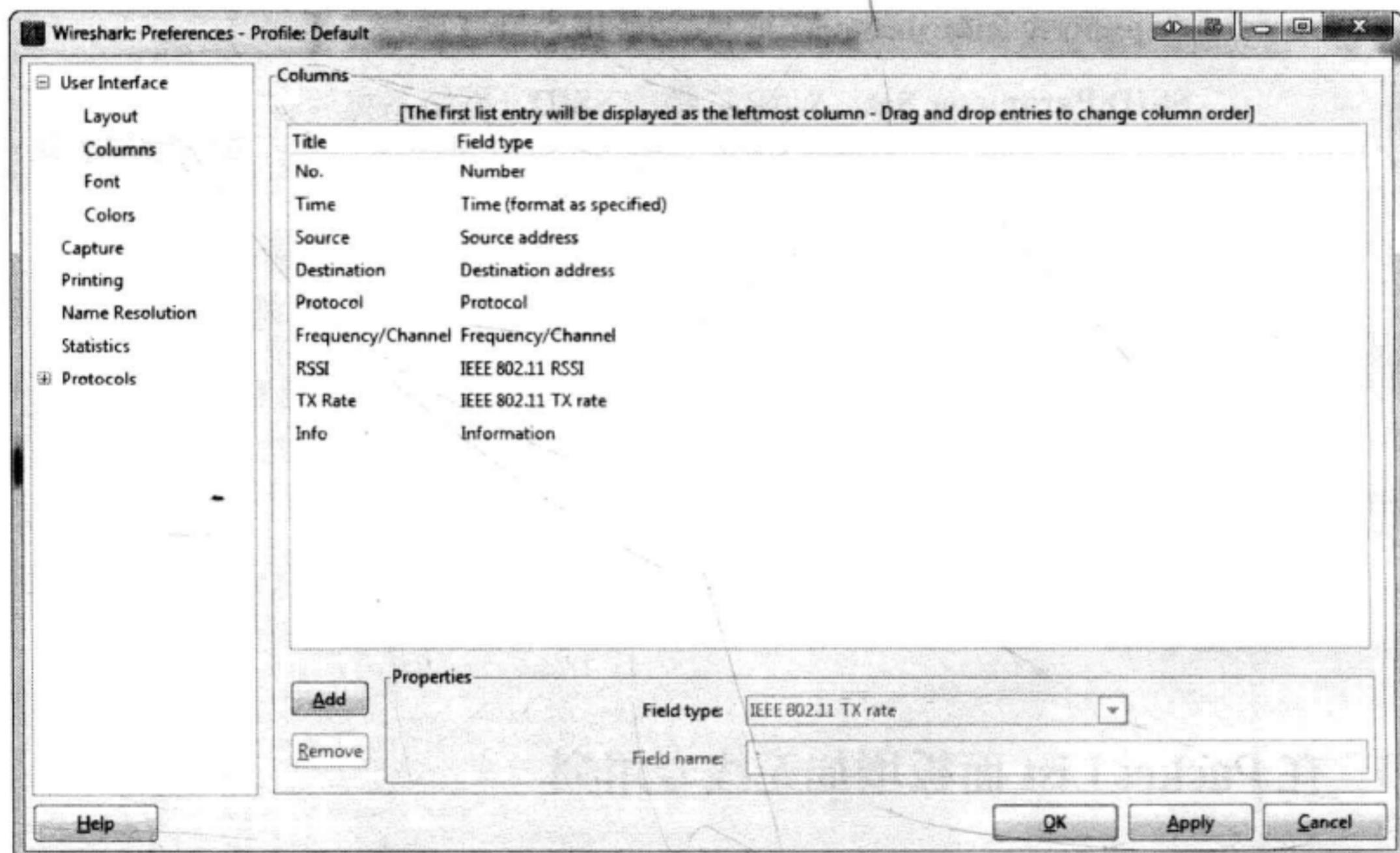


图 11-10 在 Packet List 面板增加与无线相关的列

5. 单击 **OK** 按钮使改动生效。
6. 重启 Wireshark 以显示新列。

## 11.7 无线专用过滤器

我们在第 4 章讨论过了使用捕获和显示过滤器的好处。在有线网络中筛选流量要容易得多，因为每个设备有它自己的专用线缆。然而，在无线网络中，所有无线客户端产生的流量同时存在于共享信道中，这意味着捕获任一个信道，都将包含几十个客户端的流量。本节要讨论的焦点就是数据包过滤器，可帮你找到特定流量。

### 11.7.1 筛选特定 BSS ID 的流量

网络上每一个 WAP 都有它自己的识别名，叫做“基础服务设备识别码”(Basic Service Set Identifier, BSS ID)。接入点发送的每一个管理分组和数据分组都包含这个名称。

一旦你知道想要查看的 BSS ID 名称，你需要做的只是找到那个 WAP 发送的数据包而已。Wireshark 在 Packet List 面板的 Info 列里显示了 WAP，因此找到这个信息易如反掌。

一旦得到感兴趣的 WAP 所传输的数据包，你需要在 802.11 头部中找它的 BSS ID 域。过滤器就基于这个地址来编写。找到 BSS ID MAC 地址后，你可以使用这个过滤器。

---

wlan.bssid.eq 00:11:22:33:44:55:66

---

这样你就能只看见流经该特定 WAP 的流量了。

### 11.7.2 筛选特定的无线数据包类型

在本章前面，我们曾讨论过你可能在网络上看见的无线数据包类型。你通常需要基于这些类型和子类型来筛选数据包。对于特定类型，可以用过滤器 wlan.fc.type 来实现；对于特定类型或子类型的组合，可以用过滤器 wlan.fc.type\_subtype 来实现。例如，为了过滤一个 NULL 数据包(在 16 进制中是类型 2，子类型 4)，你可以使用 wlan.fc.type\_subtype eq 0x24 这个过滤器。表 11-1 提供了 802.11 数据包类型和子类型的简要参考。

表 11-1 无线类型/子类型和相关过滤器语法

| 帧类型/子类型                | 过滤器语法                        |
|------------------------|------------------------------|
| Management frame       | wlan.fc.type eq 0            |
| Control frame          | wlan.fc.type eq 1            |
| Data frame             | wlan.fc.type eq 2            |
| Association request    | wlan.fc.type_subtype eq 0x00 |
| Association response   | wlan.fc.type_subtype eq 0x01 |
| Reassociation request  | wlan.fc.type_subtype eq 0x02 |
| Reassociation response | wlan.fc.type_subtype eq 0x03 |
| Probe request          | wlan.fc.type_subtype eq 0x04 |
| Probe response         | wlan.fc.type_subtype eq 0x05 |
| Beacon                 | wlan.fc.type_subtype eq 0x08 |
| Disassociate           | wlan.fc.type_subtype eq 0x0A |

续表

| 帧类型/子类型                    | 过滤器语法                        |
|----------------------------|------------------------------|
| Authentication             | wlan.fc.type_subtype eq 0x0B |
| Deauthentication           | wlan.fc.type_subtype eq 0x0C |
| Action frame               | wlan.fc.type_subtype eq 0x0D |
| Block ACK requests         | wlan.fc.type_subtype eq 0x18 |
| Block ACK                  | wlan.fc.type_subtype eq 0x19 |
| Power save poll            | wlan.fc.type_subtype eq 0x1A |
| Request to send            | wlan.fc.type_subtype eq 0x1B |
| Clear to send              | wlan.fc.type_subtype eq 0x1C |
| ACK                        | wlan.fc.type_subtype eq 0x1D |
| Contention free period end | wlan.fc.type_subtype eq 0x1E |
| NULL data                  | wlan.fc.type_subtype eq 0x24 |
| QoS data                   | wlan.fc.type_subtype eq 0x28 |
| Null QoS data              | wlan.fc.type_subtype eq 0x2C |

### 11.7.3 筛选特定频率

- 如果你在查看来自多个信道的流量，那基于信道的筛选就非常有用。例如，如果你本来只期待在信道 1 和 6 出现流量，可以输入一个过滤器显示信道 11 的流量。如果发现有流量，那你就知道一定是哪里弄错了——或许是配置错误，或许有无赖设备。使用这个过滤器语法，可以筛选特定频率。

---

```
radiotap.channel.freq == 2412
```

---

这将显示信道 1 的所有流量。你可以将 2412 值替换成想筛选的信道对应的频率。表 11-2 列出了信道和频率的对应表格。

表 11-2 802.11 无线信道和频率

| 信道 | 频率   |
|----|------|
| 1  | 2412 |
| 2  | 2417 |
| 3  | 2422 |
| 4  | 2427 |
| 5  | 2432 |
| 6  | 2437 |
| 7  | 2442 |
| 8  | 2447 |
| 9  | 2452 |
| 10 | 2457 |
| 11 | 2462 |

另外还有数百个实用的无线网络流量过滤器。你可以在 Wireshark wiki (<http://wiki.Wireshark.org/>) 上查看它们。

## 11.8 无线网络安全

部署和管理无线网络时最大的担忧就是传输数据的安全性。数据在空气中飞过，任何人都能得到它，因此数据加密是至关重要的。否则，任何人拿到 Wireshark 和 AirPcap 就都能看到数据了。

---

**注意** 当使用其他层次的加密技术时，比如 SSL 或 SSH，那么在那层的数据就是加密的，别人使用数据包嗅探器仍然读不到用户的通信内容。

---

最初推荐用在无线网络中加密传输数据的技术依据“有线等效加密”(Wired Equivalent Privacy, WEP) 标准。WEP 在前几年很成功，直到后来发现了它在密钥管理方面的几个漏洞。为了加强安全，几个新标准又被设计出来。这包括无线上网保护接入(Wi-Fi Protected Access, WPA) 和 WPA2 标准。尽管 WPA 和它更安全的版本 WPA2 仍然不可靠，但一般认为它们比 WEP 强多了。

在这节，我们来看一些 WEP 和 WPA 流量，以及认证失败的例子。

### 11.8.1 成功的 WEP 认证

80211-WEPauth.pcap 文件包含了成功连接 WEP 无线网络的例子。这个网络使用 WEP 安全机制。你必须向 WAP 提供一个密码，以通过认证并解密它发来的数据。你可以把 WEP 密码当成无线网络密码。

如图 11-11 所示，这个捕获文件以数据包 4 所示的从 WAP (00:11:88:6b:68:30) 发送到无线客户端 (00:14:a5:30:b0:af) 的质询开始❶。这个质询的目的是确认无线客户端是否有正确的 WEP 密码。展开 802.11 头部和 tagged parameters，你可以看到这个质询。

在数据包 5 中，这个质询被确认。然后无线客户端将用 WEP 密码解密的质询文本返回给 WAP❷，如图 11-12 所示。

在数据包 7 中，这个数据包被再次确认，并且 WAP 在数据包 8 中响应了无线客户端，如图 11-13 所示。响应里包含了一个说明认证成功的通知❸。

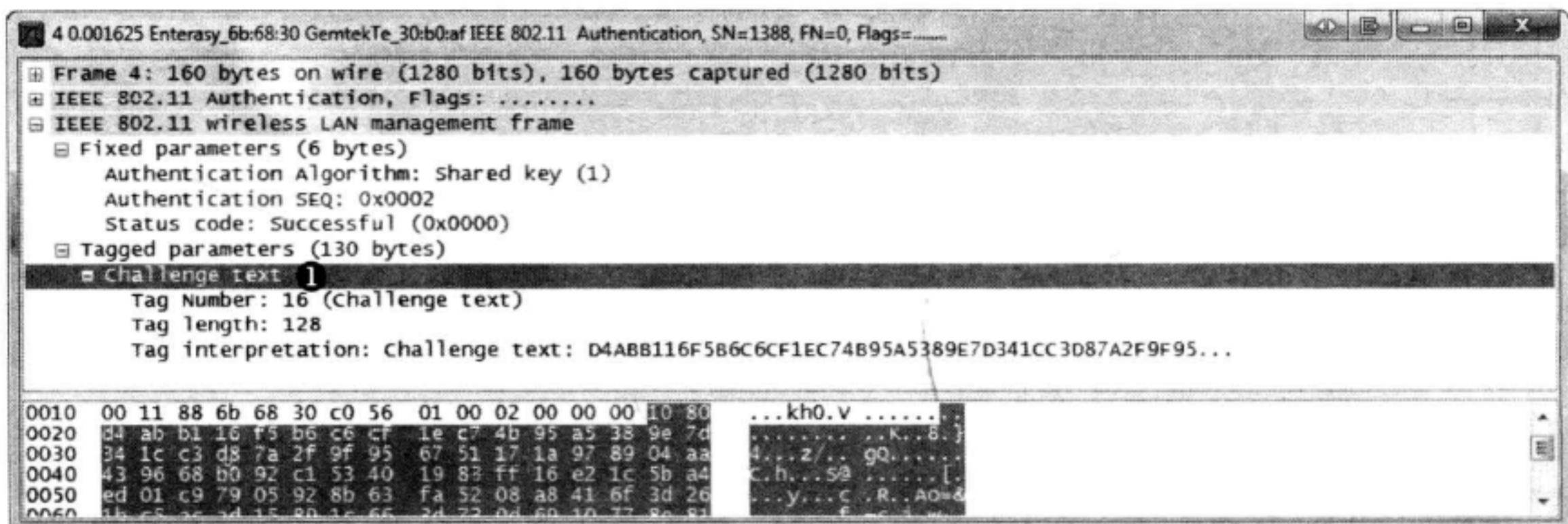


图 11-11 WAP 给无线客户端发送质询文本

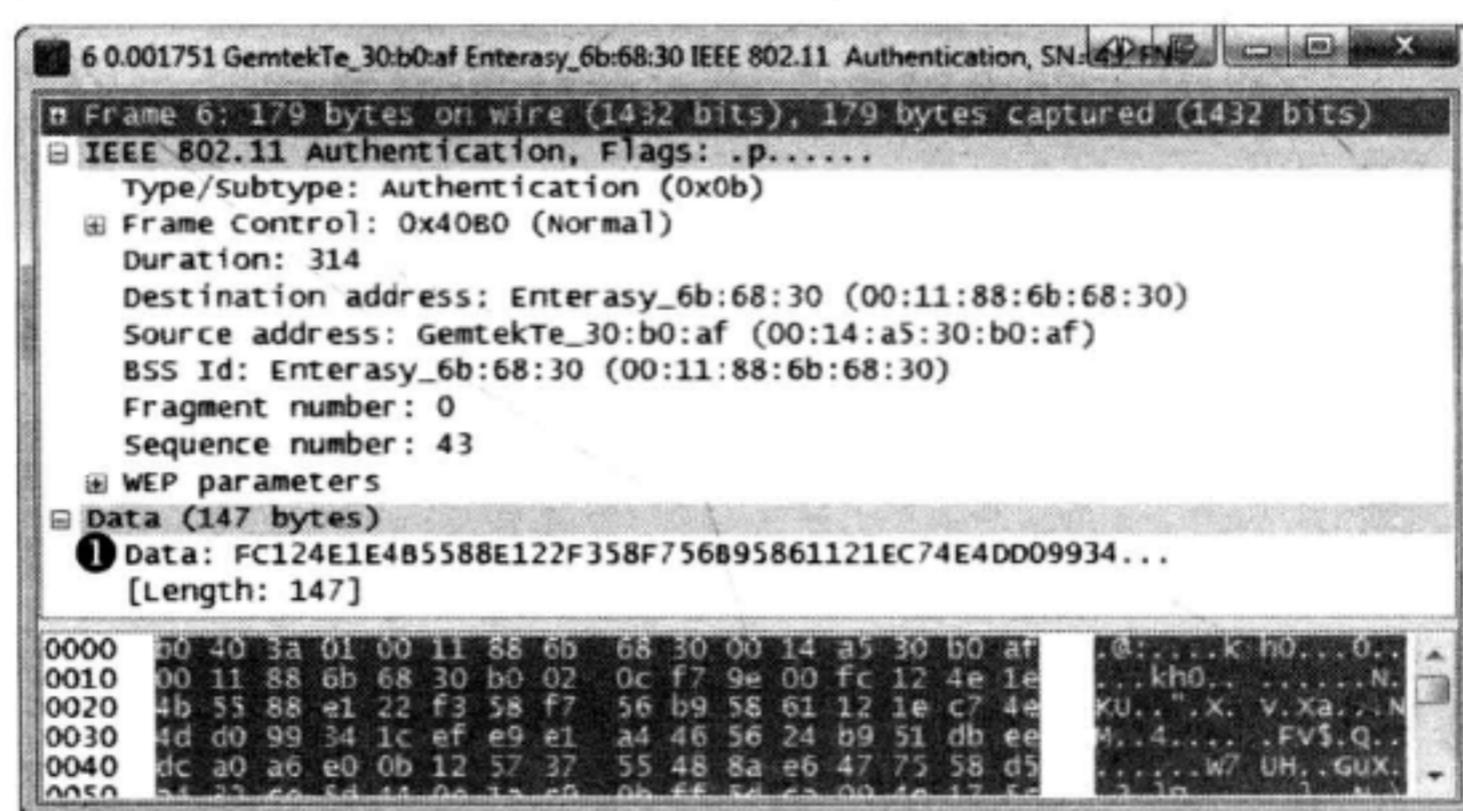


图 11-12 无线客户端向 WAP 发送已解密的质询文本

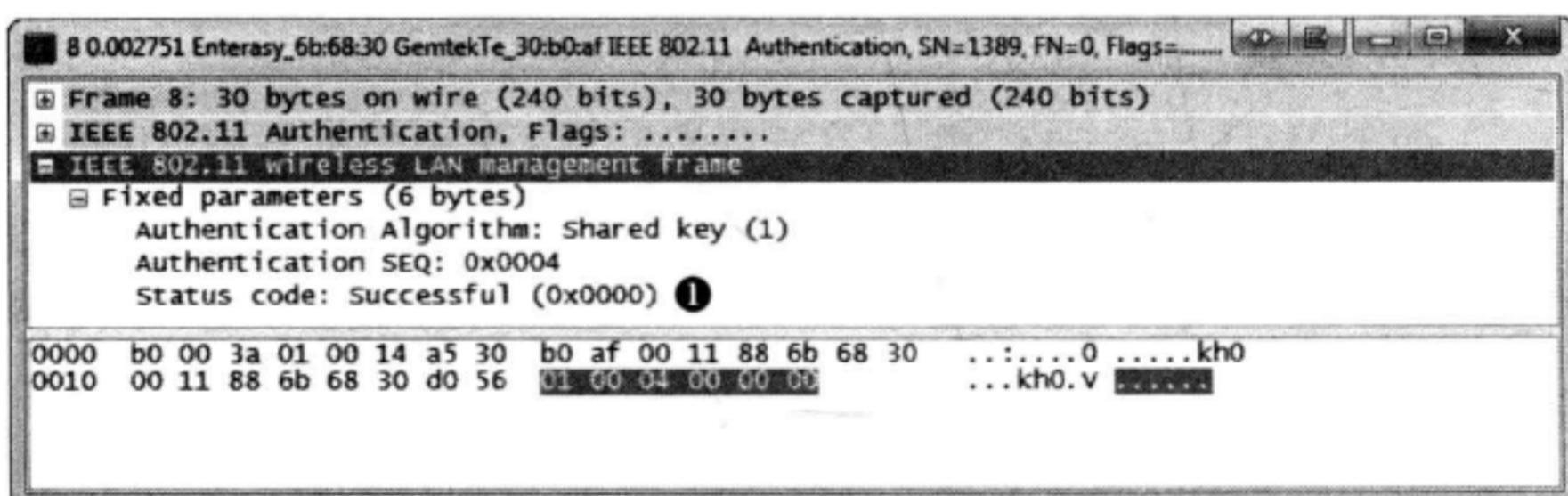


图 11-13 WAP 通知客户端认证成功了

成功认证后，客户端可以发送关联（association）请求、接收确认、完成连接过程，如图 11-14 所示。

| No. | Time     | Source            | Destination       | Protocol    | Channel | Info   |
|-----|----------|-------------------|-------------------|-------------|---------|--|
| 10  | 0.000876 | GemtekTe_30:b0:af | Enterasy_6b:68:30 | IEEE 802.11 |         | Association Request, SN=44, FN=0, Flags=....., SSID="DENVEROFFICE" |
| 11  | 0.000374 |                   |                   | IEEE 802.11 |         | Acknowledgement, Flags=.....                                       |
| 12  | 0.002627 | Enterasy_6b:67:28 | Broadcast         | IEEE 802.11 |         | Data, SN=1390, FN=0, Flags=p....F.                                 |
| 13  | 0.000624 | Enterasy_6b:68:30 | GemtekTe_30:b0:af | IEEE 802.11 |         | Association Response, SN=1391, FN=0, Flags=.....                   |
| 14  | 0.000374 |                   |                   | IEEE 802.11 |         | Acknowledgement, Flags=.....                                       |
| 15  | 0.683813 | GemtekTe_30:b0:af | Enterasy_6b:68:30 | IEEE 802.11 |         | Null function (No data), SN=45, FN=0, Flags=.....T                 |
| 16  | 0.000098 |                   |                   | IEEE 802.11 |         | Acknowledgement, Flags=.....                                       |
| 17  | 0.000053 | GemtekTe_30:b0:af | Broadcast         | IEEE 802.11 |         | Data, SN=46, FN=0, Flags=p....T                                    |

图 11-14 认证过程后紧跟一个简单的双数据包关联请求和响应

### 11.8.2 失败的 WEP 认证

在下一个例子中，一位用户输入他的 WEP 密码连接到 WAP，几秒后，无线客户端程序报告无法连接到无线网络，但没有给出原因。捕获的文件是 80211-WEPauthfail.pcap。

与成功连接时一样，通信从 WAP 在数据包 3 发送质询文本到无线客户端开始。这个消息被成功确认了。接着，在数据包 5 中，无线客户端使用用户提供的 WEP 密码发送了响应。

到这里，我们会想，应该有一个通知告诉我们认证成功了。但是我们在数据包 7 却看到了不一样的情况，如图 11-15 所示①。

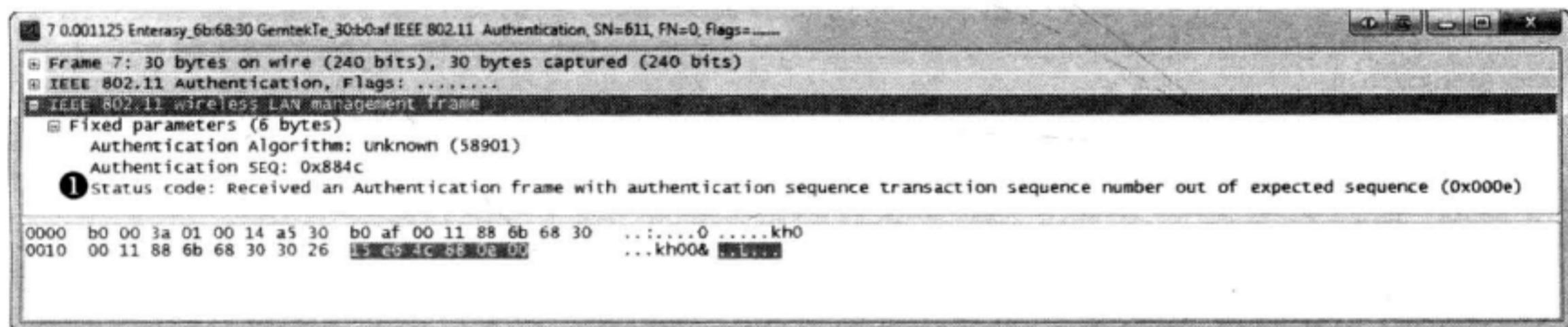


图 11-15 这个消息告诉我们认证不成功

这个消息告诉我们无线客户端对质询文本的响应不正确。这表明客户端用以解密质询文本的 WEP 密码肯定输错了。结果，连接过程就失败了。必须用正确的 WEP 密码重试才行。

### 11.8.3 成功的 WPA 认证

WPA 使用了与 WEP 完全不同的认证机制，但它仍然依赖于用户在无线客户端输入的密码来连接到网络。80211-WPAauth.pcap 文件中有一个成功的 WPA 认证的例子。

该文件第一个数据包是 WAP 发送的 beacon 广播。我们展开这个数据包的 802.11 头部，沿着 tagged parameters 往下看，展开 Vendor Specific 标题，如图 11-16 所示，能看到无线接入点的 WPA①属性部分。这让我们了解到无线接入点支持 WPA，以及版本与实现厂商。

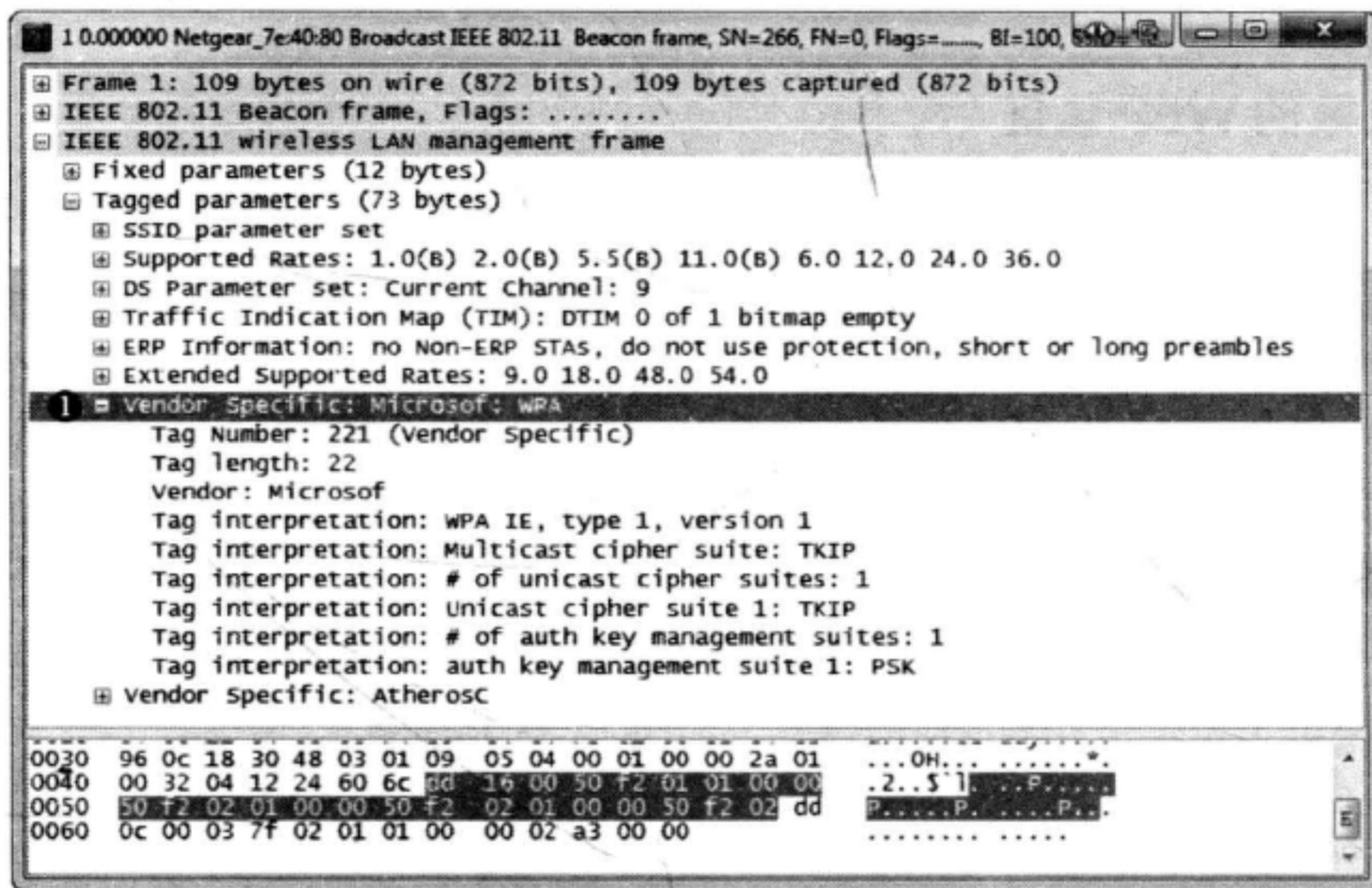


图 11-16 这个 beacon 让我们知道无线接入点支持 WPA 认证

无线客户端（00:0f:b5:88:ac:82）收到这个 beacon 广播后，就向无线接入点（00:14:6c:7e:40:80）发送一个探测请求，并得到了响应。无线客户端和无线接入点在数据包 4~7 之间生成认证与关联的请求及响应。

现在把目光转移到数据包 8 上。这是 WPA 开始握手的地方，一直持续到数据包 11。这个握手过程就是 WPA 质询响应的过程，如图 11-17 所示。

| No. | Time     | Source           | Destination      | Protocol | Channel | Info |
|-----|----------|------------------|------------------|----------|---------|------|
| 8   | 0.004096 | Netgear_7e:40:80 | Netgear_88:ac:82 | EAPOL    |         | Key  |
| 9   | 0.004101 | Netgear_88:ac:82 | Netgear_7e:40:80 | EAPOL    |         | Key  |
| 10  | 0.003580 | Netgear_7e:40:80 | Netgear_88:ac:82 | EAPOL    |         | Key  |
| 11  | 0.000004 | Netgear_88:ac:82 | Netgear_7e:40:80 | EAPOL    |         | Key  |

图 11-17 这些数据包是 WPA 握手的一部分

这里有两个质询与响应。每个数据包都可在基于 802.1x Authentication 头部下的 Replay Counter 域找到匹配对象，如图 11-18 所示。注意到前两个握手数据包的 Replay Counter 值是 1①，而后两个握手数据包的值是 2②。

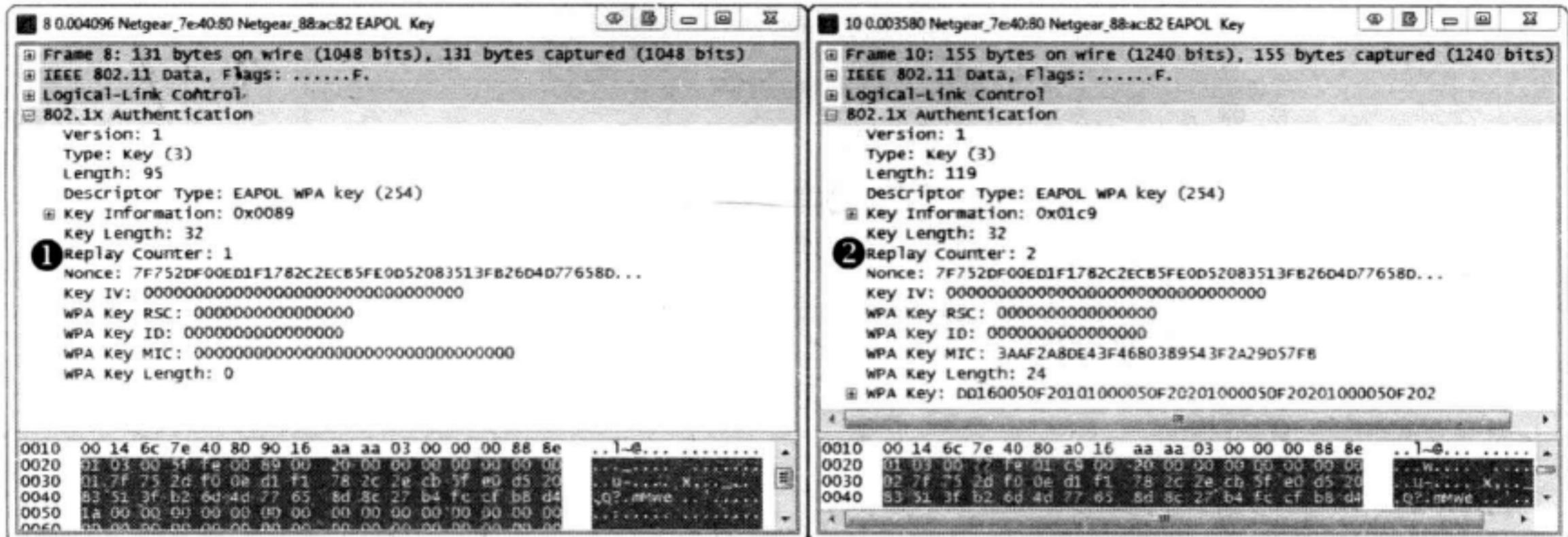


图 11-18 Replay Counter 域帮助我们匹配质询和响应

WPA 握手完成、认证成功后，数据就开始在无线客户端和 WAP 之间传输了。

#### 11.8.4 失败的 WPA 认证

与 WEP一样，用户输入 WPA 密码后，无线客户端程序报告无法连接到无线网络，但没有指出问题在哪里，我们来看看发生了什么。捕获的结果保存在 80211-WPAuthfail.pcap 文件中。

像刚才成功的 WPA 认证一样，捕获文件以同样的方式开始。这包括探测、认证和关联请求。WPA 握手从数据包 8 开始，但在这个例子中，我们看到了 8 个握手数据包，而不是之前在成功认证环节中看到的 4 个。

数据包 8 和 9 表示 WPA 握手的前两个数据包。然而在这个例子中，客户端发送回 WAP 的质询文本有误。结果，这个序列在数据包 10 和 11、12 和 13、14 和 15 中多次重复，如图 11-19 所示。使用 Replay Counter 值可以配对每个请求和响应。

| No. | Time      | Source           | Destination      | Protocol | Channel | Info |
|-----|-----------|------------------|------------------|----------|---------|------|
| 9   | 0.003547  | Netgear_88:ac:82 | Netgear_7e:40:80 | EAPOL    |         | Key  |
| 10  | 1.000549  | Netgear_7e:40:80 | Netgear_88:ac:82 | EAPOL    |         | Key  |
| 11  | 0.000476  | Netgear_88:ac:82 | Netgear_7e:40:80 | EAPOL    |         | Key  |
| 12  | 0.999489  | Netgear_7e:40:80 | Netgear_88:ac:82 | EAPOL    |         | Key  |
| 13  | 0.000511  | Netgear_88:ac:82 | Netgear_7e:40:80 | EAPOL    |         | Key  |
| 14  | 0.999013  | Netgear_7e:40:80 | Netgear_88:ac:82 | EAPOL    |         | Key  |
| 15  | -0.000037 | Netgear_88:ac:82 | Netgear_7e:40:80 | EAPOL    |         | Key  |

图 11-19 额外的 EAPOL 数据包表明 WPA 认证失败了

握手过程重试 4 次后，通信中止了。如图 11-20 所示，数据包 16 表明无线客户端没有通过认证①。

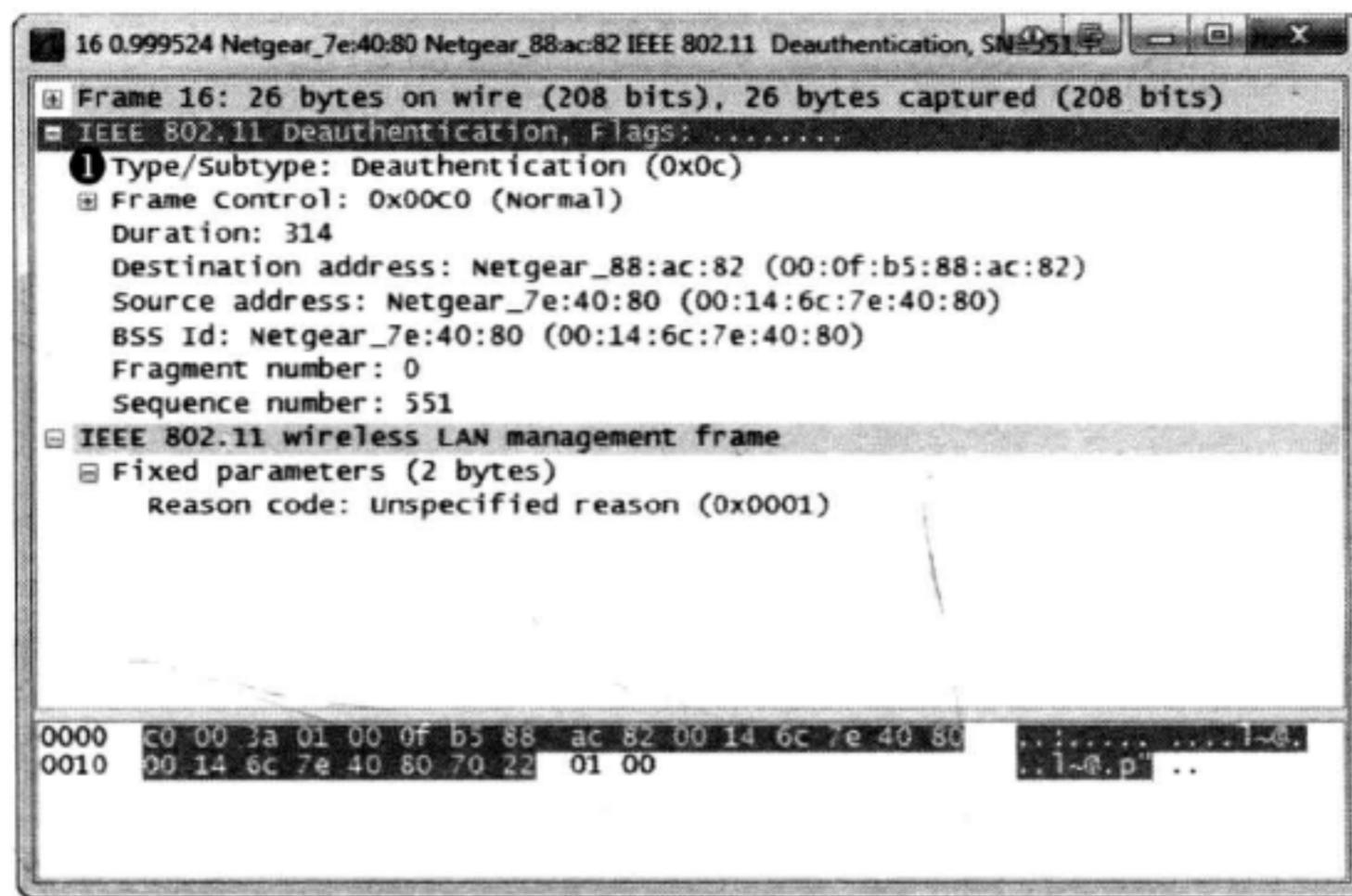


图 11-20 WPA 握手失败后，客户端认证失败

## 11.9 小结

虽然无线网络仍然普遍被认为不安全，但它在各个组织环境的部署却丝毫没有减缓。随着将焦点转移到无线通信，掌握有线网络那样的捕获并分析无线网络数据包的方法变得尤为重要。本章讲授的概念和技巧当然是不全面的，但它们能帮助你理解使用数据包分析技术解决无线网络问题的难点，让你赢在起跑线上。

# 附录 A

## 延伸阅读



数据包分析工具不仅只有 Wireshark，还有一大堆好用的工具，可以在解决网络缓慢、网络安全等常规问题及分析无线网络问题时大显身手。本章列出了一些有用的数据包分析工具，以及其他数据包分析的学习资源。

### A.1 数据包分析工具

除了 Wireshark 之外，还有一些实用的数据包分析工具。在这里，我会介绍一些我认为最有用的。

## A.1.1 tcpdump 和 Windump

虽然 Wireshark 很流行，但它可能没有 `tcpdump` 使用广泛。考虑到一些人群对数据包捕获和分析的实际需求，`tcpdump` 是完全基于文本的。

尽管 `tcpdump` 缺少图形特性，但它处理海量数据时非常可靠。因为你可以用管道将它的输出重定向输入给其他命令，比如 Linux 的 `sed` 和 `awk`。随着对数据分析的深入钻研，你会发现 Wireshark 和 `tcpdump` 都很有用。你可以从 <http://www.tcpdump.org/> 下载 `tcpdump`。

`Windump` 只是 `tcpdump` 在 Windows 平台的版本而已。你可以从 <http://www.winpcap.org/windump/> 下载到它。

## A.1.2 Cain & Abel

第 2 章已经讨论过，`Cain & Abel` 是 Windows 平台上最好的 ARP 缓存中毒攻击工具之一。`Cain & Abel` 实际上是一个非常健壮的工具套件，你一定能发现其他用途。它可以从 <http://www.oxid.it/cain.html> 取得。

## A.1.3 Scapy

`Scapy` 是一个非常强大的 Python 库，允许使用基于命令行脚本的方法创建、修改数据包。简单地说，`Scapy` 是已知最强大、最灵活的数据包操纵程序。你可以在 <http://www.secdev.org/projects/scapy/> 读到更多有关 `Scapy` 的资料，也可以下载 `Scapy` 并浏览 `Scapy` 的示例脚本。

## A.1.4 Netdude

如果你不需要像 `Scapy` 那样高级的工具，那么 `Netdude` 是 Linux 下一个好的替代品。虽然 `Netdude` 功能有限，但它提供了图形用户界面，因而出于研究目的，需要创建、修改数据包时，它显得极其方便。图 A-1 演示了使用 `Netdude` 的一个例子。你可以从 <http://netdude.sourceforge.net/> 下载到 `Netdude`。

## A.1.5 Colasoft Packet Builder

如果你是 Windows 用户，并且想要与 `Netdude` 类似的 GUI，那你可以考虑使用 `Colasoft Packet Builder`，一款超棒的免费工具。`Colasoft` 也提供了一个易用的用于数据包创建和修改的 GUI。你可以从 [http://www.colasoft.com/packet\\_builder/](http://www.colasoft.com/packet_builder/) 下载到它。

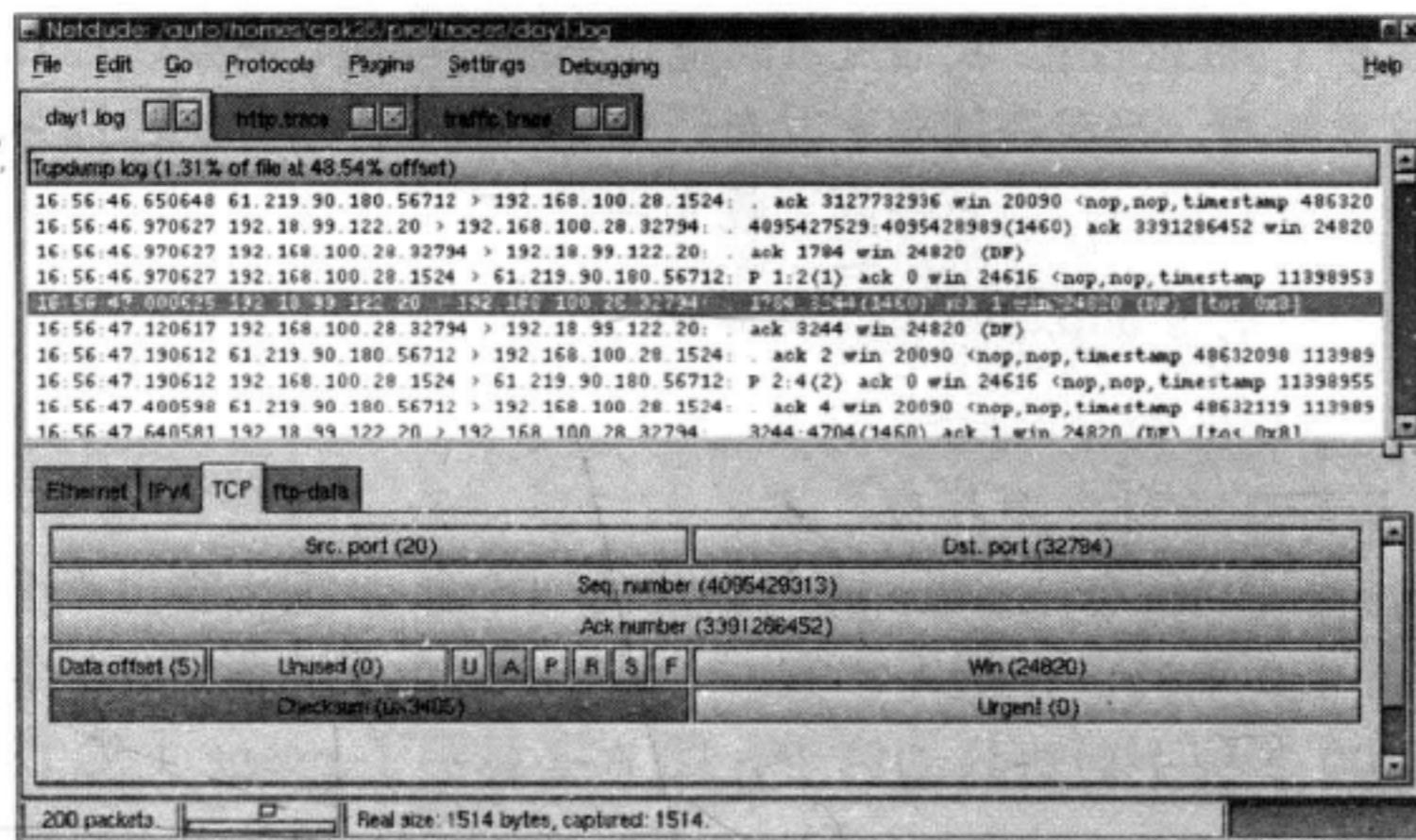


图 A-1 在 Netdude 上修改数据包

## A.1.6 CloudShark

CloudShark（由 QA Café 开发）是我最喜爱的工具之一，可以用它在线分享数据包捕获记录。如图 A-2 所示，CloudShark 网站可以在浏览器里以 Wireshark 的方式显示网络捕获文件。你可以上传捕获文件，并将链接发送给同事，以便共同分析。

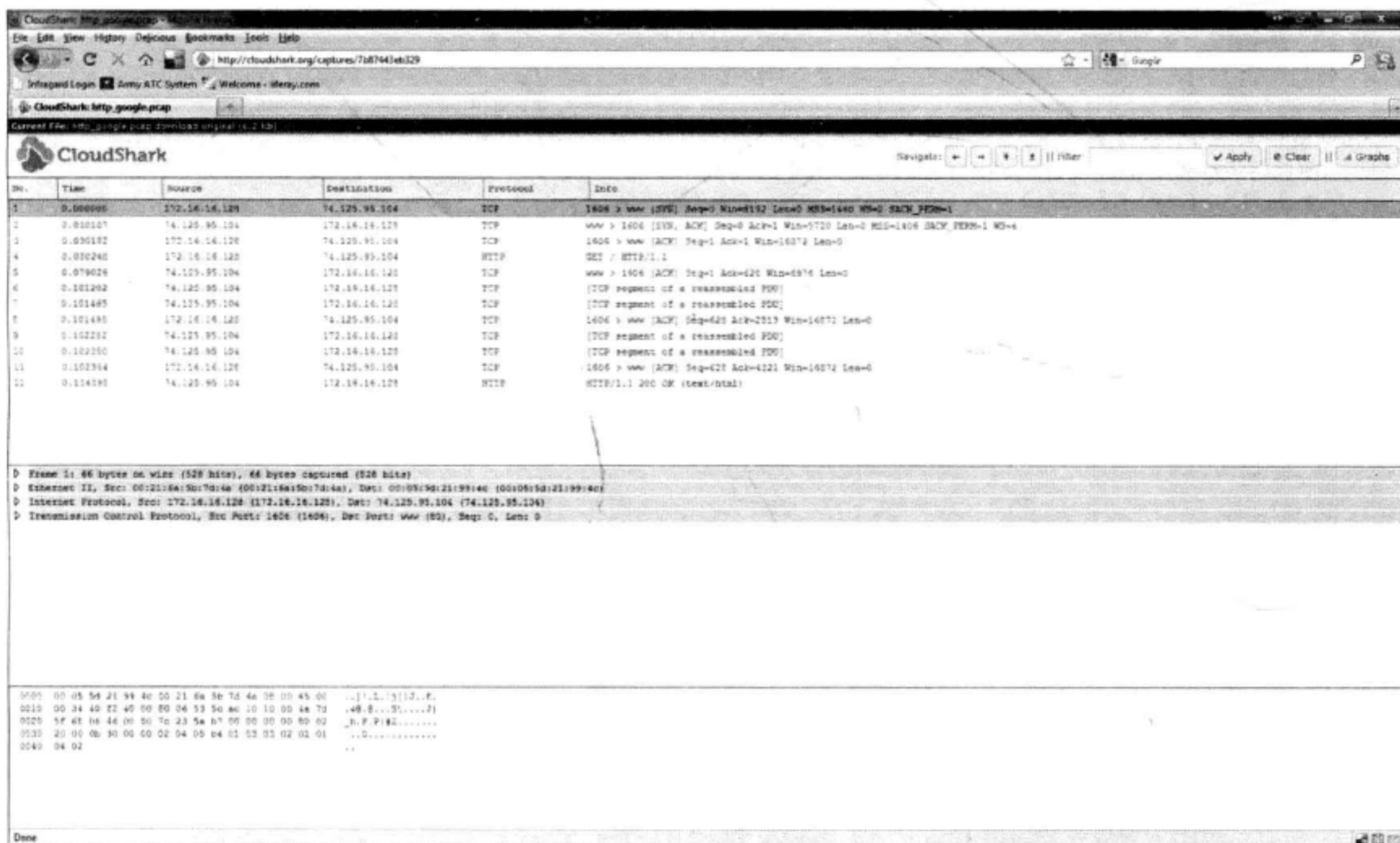


图 A-2 用 CloudShark 查看一个捕获文件示例

关于 CloudShark，我最赞赏的是它不需要注册，并能通过 URL 直接链接获取。这意味着，当我在博客上发布一个 PCAP 文件的链接时，其他人只需要单击就能查看数据包，而不需要下载文件后，再用 Wireshark 打开。

你可以通过 <http://www.cloudshark.org/> 访问 CloudShark。

### A.1.7 pcapr

pcapr 是 Mu Dynamics 的人创建的一个非常健壮的用于分享 PCAP 文件的 Web 2.0 平台。在撰写本文时，pcapr 包含了将近 3000 个 PCAP 文件，涉及 400 多种不同协议的例子。图 A-3 显示了 pcapr 上的 DHCP 流量捕获的例子。

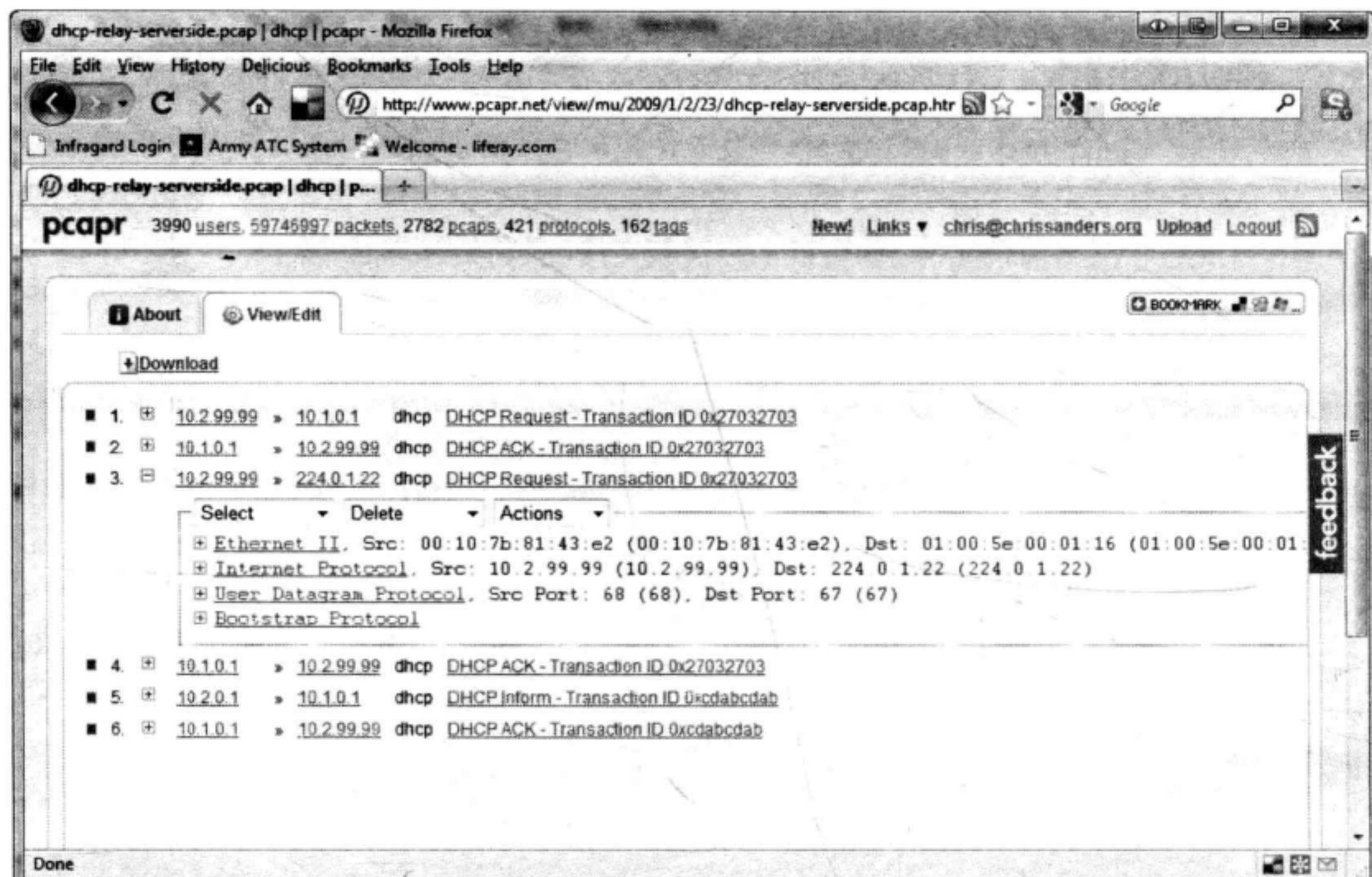


图 A-3 在 pcapr 上查看 DHCP 流量捕获

每次要查找某种确定类型的通信样例时，我都是首先在 pcapr 上搜索。如果你在自己的试验中创建了大量不同的捕获文件，不要犹豫，请将它们上传到 <http://www.pcapr.net/>，与 pcapr 社区分享。

## A.1.8 NetworkMiner

NetworkMiner 是一款主要用于网络取证的工具，但我发现它在其他一些情形下也非常实用。尽管它也可以用来捕获数据包，但它的强项在于如何解析数据包。NetworkMiner 会检测 PCAP 文件中网络各端的操作系统类型，并将文件解析成主机间的会话。它甚至允许你直接从捕获记录中提取传输的文件。NetworkMiner 可以从 <http://networkminer.sourceforge.net/> 免费下载。

## A.1.9 TcpReplay

每当我有一堆数据包需要在线路上重传并观察设备如何响应它们时，我就用 TcpReplay 来执行这个任务。TcpReplay 专门设计用来重传 PCAP 文件里的数据包。可以从 <http://tcpreplay.synfin.net/> 下载它。

## A.1.10 ngrep

如果你熟悉 Linux，毫无疑问，你肯定用过 grep 搜索数据。Ngrep 与之非常相似，允许你在 PCAP 数据上执行特定搜索。当捕获和显示过滤器都无法实现目标，或者实现太复杂时，就使用 ngrep。你可以在 <http://ngrep.sourceforge.net/> 读到更多有关 ngrep 的信息。

## A.1.11 libpcap

如果你计划开发一款应用程序，来进行一些高级的数据包解析，或是创建处理数据包，那你要对 libpcap 非常熟悉。简而言之，libpcap 是一个用于网络流量捕获的可移植的 C/C++ 库。Wireshark、tcpdump，以及其他大部分数据包分析工具都在一定层次上依赖于 libpcap。你可以在 <http://www.tcpdump.org/> 读到更多有关 libpcap 的信息。

## A.1.12 hping

hping 是你武器库中应有的“瑞士军刀”之一。hping 是一个命令行的数据包操纵和传输工具。它支持各种各样的协议，反应非常快且直观。你可以从 <http://www.hping.org/> 下载 hping。

### A.1.13 Domain Dossier

如果你需要查询域名或 IP 地址的注册信息，那 Domain Dossier 正合你意。它快速、简单、有效。你可以在 <http://www.centralops.net/co/DomainDossier.aspx> 访问到 Domain Dossier。

### A.1.14 Perl 和 Python

Perl 和 Python 虽然不是工具，但却是值得留意的脚本语言。当你熟练于数据包分析时，你会遇到没有自动化工具满足要求的情况。在那些情况下，首选 Perl 和 Python 语言编写工具，以在数据包上做些有趣的事情。对于大部分应用程序，我通常使用 Python，但这只是个人选择。

## A.2 数据包分析资源

从 Wireshark 的主页到教程、博客，有很多可用的数据包分析资源。我将在此列出我最喜欢的一些。

### A.2.1 Wireshark 主页

与 Wireshark 有关的首要资源就是它的主页：<http://www.Wireshark.org/>。主页包括软件文档、一个非常有用的包含了捕获文件样例的 wiki，以及 Wireshark 邮件列表的注册信息。

### A.2.2 SANS 安全入侵检测深入课程

作为一名 SANS 导师，我可能会有点偏袒，但我真不认为这个星球上有比“SANS SEC 503：Intrusion Detection In-Depth”更好的数据包分析课程。这个课程集中于数据包分析的安全方面。即便你不是重点关注安全，该课程前两天提供的对数据包分析和对 tcpdump 的介绍也是我所见最好的。

该课程由我的两位数据包分析英雄 Mike Poor 和 Judy Novak 讲授。它每年面授好几次。若你的旅行经费有限，没关系，该课程也通过基于 web 的点播格式在线讲授。

你可以在 <http://www.sans.org/> 阅读到更多关于 SEC 503 和其他 SANS 课程的

信息。

### A.2.3 Chris Sanders 的博客

我没有太多时间写博客，但偶尔也会在我的博客 <http://www.chrissanders.org/> 上写一些有关数据包分析的文章。如果没有别的，我的博客就作为链接到我写的其他文章和书籍的门户，另外它也提供了我的联系方式。

### A.2.4 Packetstan 博客

Mike Poor 和 Judy Novak 的博客是我目前最喜欢的与数据包相关的博客。他们的网站 <http://www.packetstan.com/> 包含了一些有趣流量的分类，并且每一篇内容都是 A+级别的。Mike 和 Judy 在他们领域是做得最好的两位，给了我很大鼓舞。

### A.2.5 Wireshark 大学

你会发现，Laura Chappell 是最多产的 Wireshark 布道者之一。她的网站包含了很多 Wireshark 使用技巧，以及她的著作 *Wireshark Network Analysis* 的信息和她教授的课程。在 <http://www.Wiresharktraining.com/> 能找到更多相关信息。

### A.2.6 IANA

互联网编号分配机构（Internet Assigned Numbers Authority, IANA）的网站是 <http://www.iana.org/>，它负责监督为北美分配 IP 地址和协议号码。它的网站提供了一些有价值的参考工具，比如查找端口号、查看有关顶级域名的信息、以及浏览合作网站查阅 RFC 文档。

### A.2.7 TCP/IP Illustrated ( Addison-Wesley )

对生活在数据包层次的人而言，Richard Stevens 博士撰写的系列图书是书架上的主要书目，已被多数人奉为 TCP/IP 圣经。这是我最喜欢的 TCP/IP 书籍，也是我写作本书时经常参考的文献。

### A.2.8 TheTCP/IP Guide ( No Starch 出版社 )

在 TCP/IP 领域里，我最喜欢的另一本书是 Charles Kozierok 写的。这本巨著厚达 1000 多页，内容非常详细，并且为视觉型学习者准备了大量很棒的图表。

# 本书第 1 版的书评

“各层次网络管理员的必备手册。”

——Linux Pro 杂志

“一本优秀、易懂且具有良好格式的 Wireshark 实用指南。”

——ARSGEEK.COM

“如果您需要掌握数据包分析的基础知识，本书将是您起步的好地方。”

——STATEOFSECURITY.COM

“本书能够让您有一技之长，它抓住了书名中的关键词——实用，很好地为读者们提供了进行数据包分析所需要知道的基本知识，然后又恰如其分地带领他们进入到使用 Wireshrak 软件解决现实问题的缤纷世界中。”

——LINUXSECURITY.COM

“您的网络中有未知主机在和其他主机聊天吗？您的电脑是否在和陌生人说话？您需要一个数据包嗅探器来找出这些问题的真正答案。Wireshark 是能够完成这件事情的最佳工具，而本书是学习这个工具最好的方式之一。”

——自由软件杂志

“新手入门的最佳读物！”

——DAEMON NEWS