



信息安全技术丛书



Life is tough,
But wireshark makes it easy.



Wireshark

网络分析就这么简单

林沛满 著



人民邮电出版社
POSTS & TELECOM PRESS

Wireshark

网络分析就这么简单

林沛满 著

人 民 邮 电 出 版 社
北 京

-
- ◆ 著 林沛满
责任编辑 傅道坤
责任印制 彭志环
- ◆ 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>
北京 印刷
- ◆ 开本: 800×1000 1/16
印张:
字数: 千字 2014 年 10 月第 1 版
印数: 1— 000 册 2014 年 10 月北京第 1 次印刷

定价: 元

读者服务热线: (010)81055410 印装质量热线: (010)81055316
反盗版热线: (010)81055315
广告经营许可证: 京崇工商广字第 0021 号

Wireshark 可能是世界上最好的开源网络包分析器，能在多种平台上（比如 Windows、Linux 和 Mac）抓取和分析网络包，在 IT 业界有着广泛的应用。

1

本书采用诙谐风趣的手法，由浅入深地用 Wireshark 分析了常见的网络协议，读者在学习 Wireshark 的同时，也会在不知不觉中理解这些协议。作者还通过身边发生的一些真实案例，分享了 Wireshark 的实战技巧。

本书不务虚，不注水，几乎页页干货，篇篇精华，力求为读者提供最佳阅读体验，使读者在一个轻松愉悦的阅读氛围中，潜移默化地掌握 Wireshark 的使用技巧和网络知识，为你的工程师生涯加油助力。

无论你是技术支持工程师、系统管理员、现场工程师、公司 IT 部门的老好人，还是高校网络相关专业的教师，无论你是 CCNA、CCNP、CCIE，还是 MCSE，本书都是迅速了解、掌握 Wireshark 技巧的绝佳读物。

关于作者

关于作者

1

林沛满，2005 年毕业于上海交通大学，现任 EMC 网络存储部门的主任工程师。多年来为多个产品团队提供过技术咨询，范围包括网络、操作系统、文件系统和域等，这就是本书所涵盖的协议如此五花八门的原因。每年临近加薪的日子，他也会组织一些技术培训来提醒上司。本书的部分内容就来自这些培训资料。

平时他也写一些技术博客，你或许还能在 IT168 或者 ChinaUnix 技术社区看到它们。本书也有少数内容来自这些博客。

当林先生不在工作时，大部分时间都花在了园艺花卉上，尤其是欧洲月季。

致 谢

致 谢

感谢很多人对本书的付出。

历城路幼儿园的林小满同学几乎每次邀请爸爸捉迷藏时都失望而归，因为写作本书的时间都被安排在下班后。爸爸会一直记得你期待的眼神和失望时撅起的小嘴。

来自摩根的曹若沈女士帮我弥补了所有的亲子时间，并且在百忙之中检查了每篇草稿。我就算包揽下一年的洗碗工作也难以回报。

来自 EMC、微软和思科等单位的多位朋友审阅了本书的大多数内容，尤其是我的同事赖苏成一个人完成了所有关于 TCP 的部分的审阅。赖先生是一位精通外文的网络专家，在削球技术上也造诣颇深，有意结识这位青年才俊的妹子可以向我索取手机号码。

最后，要感谢 EMC 公司提供了最顶级的网络、VMware 和存储空间，使我能够快速搭建本书所需要的实验环境。

几个月前和老同学聚餐，席间有位经理说，“最近招了个不错的工程师，居然懂 Wireshark。”我刚想科普一下 Wireshark 是什么，就听见另一位表示羡慕，说自己也在寻觅这样的人才。这时候我才意识到，原来 Wireshark 的市场需求已经这么大了。

当然，对我这样的 Wireshark 老粉丝来说，也不会感到很意外。随着互联网的井喷式发展，现代人的生活越来越依赖于网络，很多人开玩笑说 WiFi 也要列入马斯洛需求模型的最底层了。从事网络工作的技术人员自然也承受着从未有过的压力，比如每次促销对于电商都是极大的考验。而 Wireshark 正好是解决网络问题的利器，当我们透过它来看网络时，看到的不再是没有意义的“0”和“1”，而是人人都能理解的语句；由于它支持成百上千的协议，所以我们几乎可以看到网络上的一切，解决起问题自然也更得心应手。不久前我为一家电商做过系统调优，就是基于 Wireshark 的分析结果。

这便是我决定写作本书的原因，这么好的工具应该为更多人所用。本书先带你认识 Wireshark，学会使用它的技巧；然后利用 Wireshark 剖析一些常用的网络协议，相信有一些是你所需要的；最后分享了我用 Wireshark 处理过的几个经典案例，希望对你的工作有所帮助，能起到举一反三的效果。

本书组织结构

有别于网络教材，本书并不从 TCP/IP 的底层讲到顶层，而是采用了从简单到复杂的顺序。全书共分为 3 部分。

第 1 部分“初试锋芒”，先从一道经典的面试题讲起，带你体验 Wireshark 的魅力。接下来两篇是简单的应用实例，分析了服务器失去连接的原因，以及 Excel 程序保存文件的过程。再往下就是该部分最有价值的文章——“你一定会喜欢的技巧”，分享了很多实用窍门。最后的两篇小文章无关技术，分别讲述了 Wireshark 的前世今

生和一位网络高手的故事。这一部分内容相对简单，可以较快阅读。

第 2 部分“庖丁解牛”，用 Wireshark 剖析了很多协议，比如 DNS、TCP、FTP、HTTP 和 NFS 等。有些协议非常复杂，比如用于身份认证的 Kerberos，建议读者学习此类内容时放慢阅读速度，仔细领略其分析技巧。好在应用层协议相对独立，所以当你遇到一个不感兴趣的协议时，直接跳过也无妨。也有些协议相对简单，比如 DNS，可能书中的内容你本来就懂了。不过再简单的协议也有值得研究之处，比如你之前可能没有意识到，DNS 查询在基于 TCP 时效率有多低。这一部分还介绍了 Linux 和 Windows 上的一些小 bug，它们居然在最流行的操作系统上存在了多年而没有被发现。总体而言，这一部分的内容庞杂繁复，需要读者花费最多的时间来阅读。

第 3 部分“举重若轻”，分享了一些用 Wireshark 解决的真实案例，其中大部分是关于性能的，因为性能问题最为棘手。研究这些案例不一定对工作有直接帮助，因为遇到相同症状的概率不高，但是用 Wireshark 解决问题的思路都是相通的，相信读者可以起到触类旁通的效果。我们也许可以在几个小时里学会使用 Wireshark 软件，在几天里学会一个协议，但是思路的养成却需要经年累月的锻炼。最隐蔽的问题往往在网络包中看不到蛛丝马迹，我们不得不用推理、联想甚至发散的思维来寻找原因。希望通过这些案例，有助于读者们形成这种思维习惯。

本书每部分的结尾都有一篇非技术文章，它或者是行业趣闻，或者是本人的工作感触，希望能增加读者的阅读乐趣。

你想知道的一些问题

1. Wireshark 是什么？

Wireshark 是最流行的网络嗅探器之一，能在多种平台上抓取和分析网络包，比如 Windows、Linux 和 Mac 等。它的图形界面非常友好，但如果你觉得鼠标操作不够有腔调，也可以使用它的命令行形式——TShark。

2. 学习 Wireshark 有何意义？

很显然，Wireshark 并不能帮我们变成网络新贵，但它对技术上有所追求的工

程师来说，有着金钱难以衡量的价值。用它来辅助学习，可以更深入地理解网络协议；用它来排查故障，可以更快地发现问题。假如你是团队中唯一掌握 Wireshark 的网络工程师，这个看家本领非常有助于你保持大牛地位。在同事们手足无措时，你可以用最快速度摆平，然后平静地说一句：“问题解决了，我先去泡杯咖啡。”接下来就可以离开座位，让他们一脸崇拜地研究你满是 TShark 命令的屏幕了。

3. 为什么要写作本书？

Wireshark 本身是免费的，在我们心存感激的同时，也注意到一些需要花大钱的地方——Wireshark University 的 5 天培训费为 3395 美金，而且没有在中国开课。对于大多数中国工程师来说，唯一的途径就是自学，这便是我写作本书的原因。

与其他网络图书不同，本书舍弃了公式和协议的条条框框，借助 Wireshark 直观地显示网络细节，让原本拒人千里的协议鲜活地呈现出来。你只需稍加思考，相信很多原来的难点都可以迎刃而解。书中用 Wireshark 解决的几个问题，也全部源于真实案例，很可能在工作中遇到。

4. 本书适合哪些读者？

如果你是公司 IT 部门的老好人，常常有同事咨询各种疑难杂症，那你适合阅读本书。从 ping 不通主机到访问不了共享目录，都有活生生的例子，比如第 1 部分的《从一道面试题开始说起》和《初试牛刀：一个简单的应用实例》。

如果你是技术支持工程师，每天被客户当作出气筒，本书简直就是为你而作。下次就发个 Wireshark 截屏给客户，“看，明明是你们自己的 VLAN 配错了，当然连不上！”

如果你是数据中心的管理员，不时要跟习惯推卸责任的网管吵架，也请阅读本书。它将演示如何通过抓到的包推出网络状况，甚至算出 TCP 拥塞窗口。如果那些网管员问你是怎么算的，你只需低调地掏出本书，让他们看到发黄的纸张和印着咖啡渍的封面即可。

如果你在现场实施项目，常被好客的甲方挽留到深夜，请携带本书。本出第 3 部

分的几篇现场调优案例，说不定会给你带来共鸣。

如果你是高校网络相关专业的一名伟大的人民教师，常因准备课件而发愁，也建议参考本书。上课时打开 Wireshark，也许比精美的课件更受学生欢迎。

其他职业的读者请酌情参考上面内容。但如果你是一名神秘的黑客，我不得不直言相告：虽然 Wireshark 能解析网络包，却不能帮你在肉鸡上抓包，所以本书作用有限。虽然《首席信息安全官必须知道的五大黑客工具》之类的高大上文章会把 Wireshark 列进黑客软件，但是众所周知，头衔上包含“首席”二字的人已经不会亲自使用这些工具。

5. 阅读本书需要什么基础？

要想阅读本书，你需要具备基本的网络知识，比如在学校里上过网络课，或者学习过 CCNA 的培训资料。对于缺乏网络基础的 Wireshark 用户，建议先阅读一本较成系统的教材，个人推荐 Richard Stevens 的《TCP/IP 详解卷 1：协议》。搭上《颈椎病防治一本通》也许还能免运费，前一本有助于你更快地学会 Wireshark，后一本则能在学会 Wireshark 之后治疗职业病。

由于本书涵盖了很多协议，所以每位读者都可能会遇到完全陌生的内容。好在大多协议都相对独立，所以实在看不懂的部分也可以跳过。举个例子，假如你的工作与 Kerberos 毫无关系，那么看不懂也没必要强求，毕竟学起来颇费心血。

6. 对阅读本书有何建议？

本书有别于大部头的网络百科全书，所以无论你在车上还是如厕时皆可轻松阅读。但有部分内容可能需要你放慢速度，甚至多读几遍才能理解。有个实验环境是最好的，可以自己抓些网络包对照学习。技术类知识就是这样，如果你从最简单的地方开始动手操作，接下来就如鱼得水；如果从一开始只依靠冥想，到后面就会走火入魔。

7. 还有什么要对读者说的？

我心目中一本好的技术图书应该是内容准确，表达通俗，容易理解的，本书也尽

量追求这几点（相信本书也做到了）。

为了保证内容的准确性，我邀请了一位 Windows 技术支持、一位网络存储工程师、两位经验丰富的 CCIE 审阅了初稿的大部分文章。如此兴师动众，是因为同时精通 NFS、Kerberos 和 TCP 等协议的工程师并不多见。即便这样，本书仍可能存在纰漏。如果你在阅读过程中发现了任何问题，欢迎反馈到本人邮箱 linpeiman@hotmail.com。

在通俗与精确之间，本书选择了前者。比如“抓包（packet）”一词本身就不够精确，Wireshark 抓到的应该是帧（frame）。但是出于表达习惯，我并没有改成“抓帧”。又比如对同一个网络分层的称呼，工程师们也有不同的习惯，希望读者能够接受这些“混乱”。

容易理解是最难做到的一点。传说白居易写完一首诗，必定先请不识字的老太婆品鉴，一直要修改到老太婆听懂为止。本书的初稿也邀请了我家的“老太婆”进行试读，基本上她看懂后才敢交稿。当然我家这位“老太婆”在本科阶段学习过网络课程。我有时会在书中用图表、类比和 Wireshark 等方式来反复解释同一知识点，就是为了辅助理解。如果让部分读者感到啰嗦，先在此表示歉意。

目 录

初试锋芒/1

从一道面试题开始说起/3	1
小试牛刀：一个简单的应用实例/10	
Excel文件的保存过程/13	
你一定会喜欢的技巧/17	
Patrick的故事/29	
Wireshark的前世今生/32	

庖丁解牛/35

NFS协议的解析/37	
从Wireshark看网络分层/52	
TCP的连接启蒙/57	
快递员的工作策略——TCP窗口/64	
重传的讲究/70	
延迟确认与Nagle算法/80	
百家争鸣/84	
简单的代价——UDP/90	
剖析CIFS协议/93	
网络江湖/104	
DNS小科普/111	
一个古老的协议——FTP/118	
上网的学问——HTTP/126	
无懈可击的Kerberos/132	
TCP/IP的故事/141	
举重若轻/145	
“一小时内给你答复”/147	
午夜铃声/151	

目 录

深藏功与名/157
棋逢对手/162
学无止境/167
一个技术男的自白/174

初试锋芒

1

我每次当面试官，都要伪装成无所不知的大牛。

这当然是无奈的选择——现在每封简历都那么耀眼，不装一下简直镇不住场面。比如尚未毕业的本科生，早就拿下 CCIE 认证；留欧两年的海归，已然精通英、法、德三门外语；最厉害的一位应聘者，研究生阶段就在国际上首次提出了计算机和生物学的跨界理论……可怜我这个老实人在一开场还能装装，到了技术环节就忍不住提问基础知识，一下子把气氛从学术殿堂拉到建筑工地。不过就是这些最基础的问题，却常常把简历精英们难住。本文要介绍的便是其中的一道。

问题：两台服务器 A 和 B 的网络配置如下（见图 1），B 的子网掩码本应该是 255.255.255.0，被不小心配成了 255.255.255.224。它们还能正常通信吗？

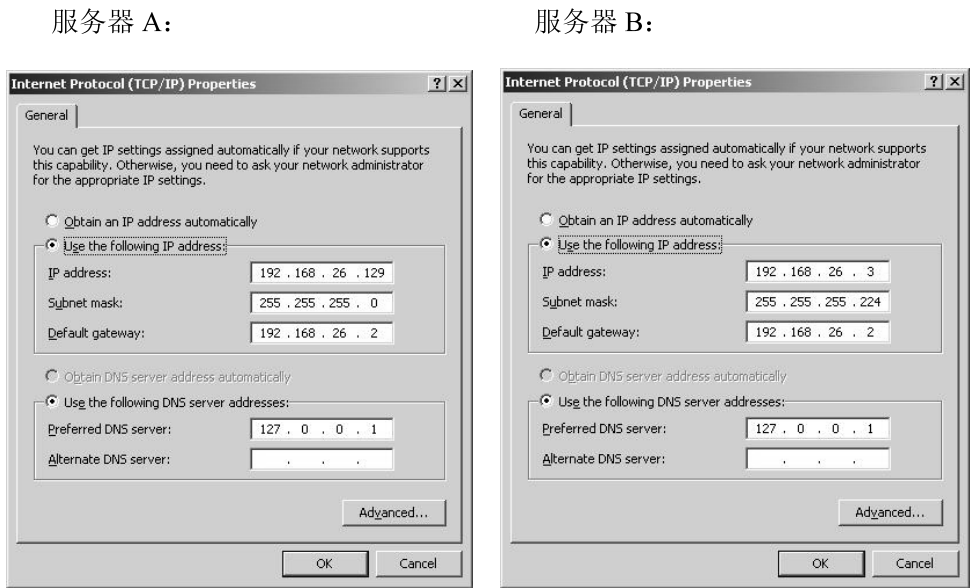


图 1

很多应聘者都会沉思良久（他们一定在心里把我骂了很多遍了），然后给出下面这些形形色色的答案。

答案 1：“A 和 B 不能通信，因为……如果这样都行的话，子网掩码还有什么用？”（这位的反证法听上去很有道理！）

答案 2：“A 和 B 能通信，因为它们可以通过 ARP 广播获得对方的 MAC 地址。”（那子网掩码还有什么用？楼上的反证法用来反驳这位正好。）

答案 3：“A 和 B 能通信，但所有包都要通过默认网关 192.168.26.2 转发。”（请问这么复杂的结果你是怎么想到的？）

答案 4：“A 和 B 不能通信，因为 ARP 不能跨子网。”（这个答案听上去真像是经过认真思考的。）

以上哪个答案是正确的？还是都不正确？如果这是你第一次听到这道题，不妨停下来思考一下。

真相只有一个，应聘者的答案却是五花八门。可见对网络概念的理解不容含糊，否则差之毫厘，谬以千里。要知道，这还只是基本的路由交换知识，假如涉及复杂概念，结果就更不用说了。

问题是即便我们对着教材咬文嚼字，也不一定能悟出正确答案。这个时候，就可以借助 Wireshark 的抓包与分析功能了。我手头就有两台 Windows 服务器，已经按照面试题配好网络。如果你以前没有用过 Wireshark，就开始第一次亲密接触吧。

1. 从 <http://www.wireshark.org/download.html> 免费下载安装包，并在服务器 B 上装好（把所有可选项都装上）。
2. 启动 Wireshark 软件，单击菜单栏上的 Capture，再单击 Interfaces 按钮（见图 2）。

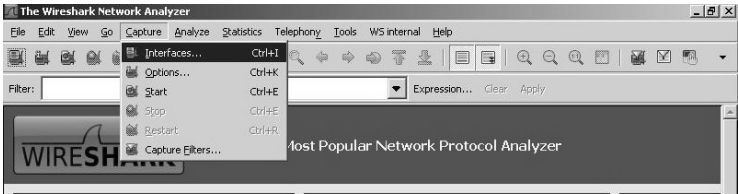


图 2

3. 服务器 B 上的所有网卡都会显示在弹出的新窗口上（见图 3），在要抓包的网卡上单击 Start 按钮。



图 3

4. 在服务器 B 上 ping A 的 IP 地址，结果是通的（见图 4）。该操作产生的网络包已经被 Wireshark 捕获。

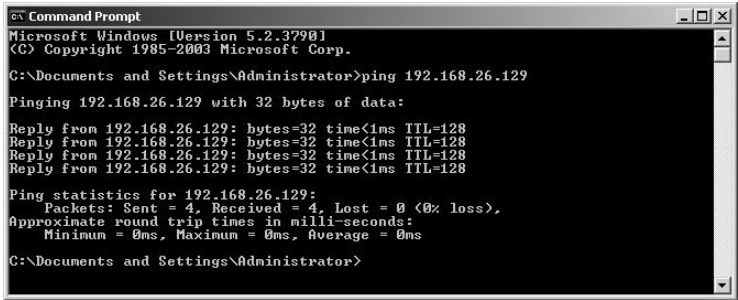


图 4

5. 在 Wireshark 的菜单栏上，再次单击 Capture，然后单击 Stop。
6. 在 Wireshark 的菜单栏上，单击 File，再单击 Save，把网络包保存到硬盘上（这一步并非必需，但存档是个好习惯）。
7. 收集每台设备的 MAC 地址以备分析。
 - 服务器 A: 00:0c:29:0c:22:10

- 服务器 B: 00:0c:29:51:f1:7b
- 默认网关: 00:50:56:e7:2f:88

初试锋芒

从一道面试题开始说起

6

现在可以分析网络包了。如图 5 所示，Wireshark 的界面非常直观。最上面是 Packet List 窗口，它列出了所有网络包。在 Packet List 中选定的网络包会详细地显示在中间的 Packet Details 窗口中。由于我在 Packet List 中选定的是 3 号包，所以图 5 中看到的就是 Frame 3 的详情。最底下是 Packet Bytes Details 窗口，我们一般不会用到它。

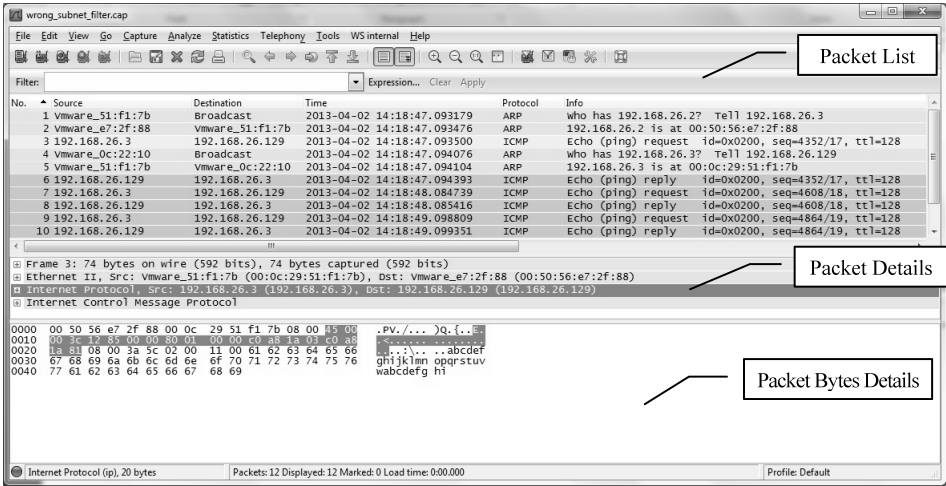


图 5

接下来看看每个包都做了些什么。

1 号包（见图 6）：

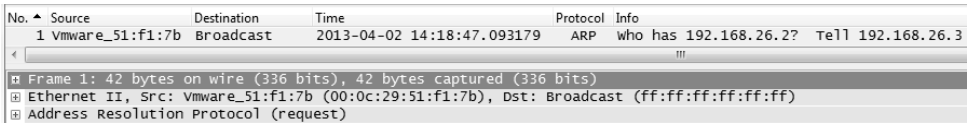


图 6

服务器 B 通过 ARP 广播查询默认网关 192.168.26.2 的 MAC 地址。为什么我 ping 的是服务器 A 的 IP，B 却去查询默认网关的 MAC 地址呢？这是因为 B 根据自己的子网掩码，计算出 A 属于不同子网，跨子网通信需要默认网关的转发。而要和默认网关通信，就需要获得其 MAC 地址。

2 号包（见图 7）：

No.	Source	Destination	Time	Protocol	Info
2	Vmware_e7:2f:88	Vmware_51:f1:7b	2013-04-02 14:18:47.093476	ARP	192.168.26.2 is at 00:50:56:e7:2f:88
!!!					
Frame 2: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)					
Ethernet II, Src: Vmware_e7:2f:88 (00:50:56:e7:2f:88), Dst: Vmware_51:f1:7b (00:0c:29:51:f1:7b)					
Address Resolution Protocol (reply)					

图 7

默认网关 192.168.26.2 向 B 回复了自己的 MAC 地址。为什么这些 MAC 地址的开头明明是“00:50:56”或者“00:0c:29”，Wireshark 上显示出来却都是“Vmware”？这是因为 MAC 地址的前 3 个字节表示厂商。而 00:50:56 和 00:0c:29 都被分配给 Vmware 公司。这是全球统一的标准，所以 Wireshark 干脆显示出厂商名了。

3 号包（见图 8）：

No.	Source	Destination	Time	Protocol	Info
3	192.168.26.3	192.168.26.129	2013-04-02 14:18:47.093500	ICMP	Echo (ping) request id=0x0200, seq=4352/17, ttl=128
!!!					
Frame 3: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)					
Ethernet II, Src: Vmware_51:f1:7b (00:0c:29:51:f1:7b), Dst: Vmware_e7:2f:88 (00:50:56:e7:2f:88)					
Internet Protocol, Src: 192.168.26.3 (192.168.26.3), Dst: 192.168.26.129 (192.168.26.129)					
Internet Control Message Protocol					

图 8

B 发出 ping 包，指定 Destination IP 为 A，即 192.168.26.129。但 Destination MAC 却是默认网关的 00:50:56:e7:2f:88（Destination MAC 可以在图 8 中的 Packet Details 中看到）。这表明 B 希望默认网关把包转发给 A。至于默认网关有没有转发，我们目前无从得知，除非在网关上也抓个包。

4 号包（见图 9）：

No.	Source	Destination	Time	Protocol	Info
4	Vmware_0c:22:10	Broadcast	2013-04-02 14:18:47.094076	ARP	who has 192.168.26.3? Tell 192.168.26.129
!!!					
Frame 4: 60 bytes on wire (480 bits), 60 bytes captured (480 bits)					
Ethernet II, Src: Vmware_0c:22:10 (00:0c:29:0c:22:10), Dst: Broadcast (ff:ff:ff:ff:ff:ff)					
Address Resolution Protocol (request)					

图 9

B 收到了 A 发出的 ARP 广播，这个广播查询的是 B 的 MAC 地址。这是因为在 A 看来，B 属于相同子网，同子网通信无需默认网关的参与，只要通过 ARP 获得对方 MAC 地址就行了。这个包也表明默认网关成功地把 B 发出的 ping 请求转发给 A 了，否则 A 不会无缘无故尝试和 B 通信。

5 号包（见图 10）：

No.	Source	Destination	Time	Protocol	Info
5	Vmware_51:f1:7b	Vmware_0c:22:10	2013-04-02 14:18:47.094104	ARP	192.168.26.3 is at 00:0c:29:51:f1:7b
!!!					
Frame 5: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)					
Ethernet II, Src: Vmware_51:f1:7b (00:0c:29:51:f1:7b), Dst: Vmware_0c:22:10 (00:0c:29:0c:22:10)					
Address Resolution Protocol (reply)					

图 10

B 回复了 A 的 ARP 请求，把自己的 MAC 地址告诉 A。这说明 B 在执行 ARP 回复时并不考虑子网。虽然 ARP 请求来自其他子网的 IP，但也照样回复。

6 号包（见图 11）：

No.	Source	Destination	Time	Protocol	Info
6	192.168.26.129	192.168.26.3	2013-04-02 14:18:47.094393	ICMP	Echo (ping) reply id=0x0200, seq=4352/17, ttl=128
!!!					
Frame 6: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)					
Ethernet II, Src: Vmware_0c:22:10 (00:0c:29:0c:22:10), Dst: Vmware_51:f1:7b (00:0c:29:51:f1:7b)					
Internet Protocol, Src: 192.168.26.129 (192.168.26.129), Dst: 192.168.26.3 (192.168.26.3)					
Internet Control Message Protocol					

图 11

B 终于收到了 A 的 ping 回复。从 MAC 地址 00:0c:29:0c:22:10 可以看出，这个包是从 A 直接过来的，而不是通过默认网关的转发。

7、8、9、10 号包（见图 12）：

No.	Source	Destination	Time	Protocol	Info
7	192.168.26.3	192.168.26.129	2013-04-02 14:18:48.084739	ICMP	Echo (ping) request id=0x0200, seq=4608/18, ttl=128
8	192.168.26.129	192.168.26.3	2013-04-02 14:18:48.085416	ICMP	Echo (ping) reply id=0x0200, seq=4608/18, ttl=128
9	192.168.26.3	192.168.26.129	2013-04-02 14:18:49.098809	ICMP	Echo (ping) request id=0x0200, seq=4864/19, ttl=128
10	192.168.26.129	192.168.26.3	2013-04-02 14:18:49.099351	ICMP	Echo (ping) reply id=0x0200, seq=4864/19, ttl=128
!!!					
Frame 7: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)					
Ethernet II, Src: Vmware_51:f1:7b (00:0c:29:51:f1:7b), Dst: Vmware_e7:2f:88 (00:50:56:e7:2f:88)					
Internet Protocol, Src: 192.168.26.3 (192.168.26.3), Dst: 192.168.26.129 (192.168.26.129)					
Internet Control Message Protocol					

图 12

都是重复的 ping 请求和 ping 回复。因为 A 和 B 都已经知道对方的联系方式，所以就没必要再发 ARP 了。

分析完这几个包，答案出来了。原来通信过程是这样的：B 先把 ping 请求交给默认网关，默认网关再转发给 A。而 A 收到请求后直接把 ping 回复给 B，形成图 13 所示的三角形环路。不知道你答对了吗？

通过这道题，不知道你是否已经感受到了 Wireshark 的神奇？如果有兴趣进

一步练习，不妨也搭个环境，把这道题里 A 和 B 的掩码互换一下。看看这次还能 ping 通吗？如果不能，原因又在哪里？

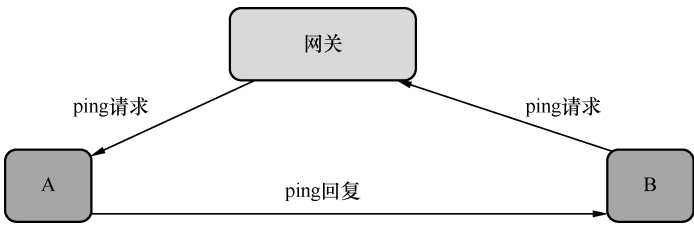


图 13

其实做题对 Wireshark 只是大材小用，它还可以用于学习复杂的协议，或者解决隐蔽的难题。在下文中，我们将体验 Wireshark 在实际工作中的应用。

初试锋芒

从一道面试题开始说起

小试牛刀：一个简单的应用实例

我的老板气宇轩昂，目光笃定，在人群中颇有大将风范（当然是老板娘不在场的时候）。有一年我们在芝加哥流落街头，也没见他皱过眉头。不过前几天，这位气场型领导竟然板着脸跑过来，说赶紧帮忙，有位同事被客户骂惨了。我当然不能拒绝帮（yao）助（qiu）同（jia）事（xin）的机会，立即加入电话会议。

原来事情是这样的：客户不小心重启了服务器 A，然后它就再也无法和服务器 B 通信了。由于这两台服务器之间传输的是关键数据，现场工程师又一时查不出原因，所以客户异常恼火。

问题听起来并不复杂，考虑到起因是服务器 A 的重启，所以我收集了它的网络配置（见图 1）。

```
[root@A ~]# ifconfig |egrep "HWaddr|inet addr"
eth0      Link encap:Ethernet  HWaddr 00:0C:29:CB:74:A9
          inet addr:192.168.26.131  Bcast:192.168.26.255  Mask:255.255.255.0
eth1      Link encap:Ethernet  HWaddr 00:0C:29:CB:74:B3
          inet addr:192.168.174.131  Bcast:192.168.174.255  Mask:255.255.255.0
eth2      Link encap:Ethernet  HWaddr 00:0C:29:CB:74:BD
          inet addr:192.168.186.131  Bcast:192.168.186.255  Mask:255.255.255.0
[root@A ~]# route |grep default
default   192.168.26.2    255.255.255.0   UG    0        0        0 eth0
```

图 1

服务器 B 的网络配置则简单很多，只有一个 IP 地址 192.168.182.131，子网掩码也是 255.255.255.0。

当我们在 A 上 ping B 时，网络包应该怎么走？阅读以下内容之前，读者不妨

先停下来思考一下。

一般情况下，像 A 这类多 IP 的服务器是这样配路由的：假如自身有一个 IP 和对方在同一子网，就从这个 IP 直接发包给对方。假如没有一个 IP 和对方同子网，就走默认网关。在这个环境中，A 的 3 个 IP 显然都与 B 属于不同子网，那就应该走默认网关了。会不会是 A 和默认网关的通信出问题了呢？我从 A 上 ping 了一下网关，结果却是通的。难道是因为网关没有把包转发出去？或者是 ping 请求已经被转发到 B 了，但 ping 回复在路上丢失？我感觉自己已经走进死胡同。每当到了这个时候，我就会想到最值得信赖的队友——Wireshark。

我分别在 eth0、eth1、和 eth2 上抓了包。最先查看的是连接默认网关的 eth0，出乎意料的是，上面竟然一个相关网络包都没有。而在 eth1 上抓的包却是图 2 的表现：A 正通过 ARP 广播查找 B（192.168.182.131）的 MAC 地址，试图绕过默认网关直接与 B 通信。这说明了什么问题呢？

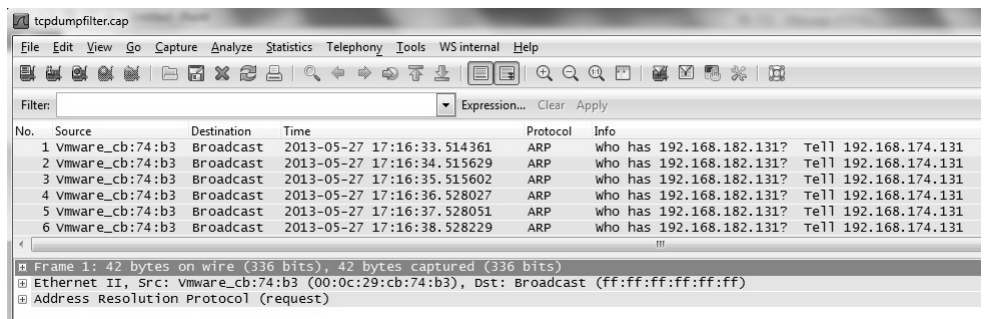


图 2

这说明 A 上存在一项符合 192.168.182.131 的路由，促使 A 通过 eth1 直接与 B 通信。我赶紧逐项检查路由表，果然发现有这么一项（见图 3）：

```
[root@A ~]# route |egrep "Dest|168.182"
```

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.182.0	*	255.255.255.0	U	0	0	0	eth1

图 3

因为 192.168.182.131 属于 192.168.182.0/255.255.255.0，所以就会走这条路由。由于不同子网所配的 VLAN 也不同，所以这些 ARP 请求根本到达不了 B。ping

包就更不用说了，它从来就没发出来过。客户赶紧删除了这条路由，两台服务器的通信也随即恢复。

为什么 A 重启之后会多了这条莫名其妙的路由呢？根据客户回忆，他们以前的确是配过该路由的，后来删掉了，不知道为什么配置文件里还留着。今天重启加载了一遍配置文件，所以这条路由又出现了。你也许会问，为什么不从一开始就仔细检查路由表呢？这样就不至于走错胡同，连抓包和 Wireshark 都省了。我当时也是这样反省的，但现实中要做到并不容易。且不说一开始并没有怀疑到路由表，就算怀疑了也不一定能看出问题来。在这个案例中，系统管理员和现场工程师都检查过路由表，但无一例外地忽略了出问题的一项。这是因为真实环境中的路由表有很多项，在紧张的电话会议上难以注意到多出了异常的一项。而且子网掩码也不是 255.255.255.0 那么直观。假如本文所用的 IP 保持不变，但子网掩码变成 255.255.248.0，路由表就成了图 4 所示的样子。

```
[root@A ~]# netstat -rn
Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	MSS Window	irrt	Iface
192.168.168.0	0.0.0.0	255.255.248.0	U	0 0	0	eth1
192.168.176.0	0.0.0.0	255.255.248.0	U	0 0	0	eth1
192.168.184.0	0.0.0.0	255.255.248.0	U	0 0	0	eth2
192.168.24.0	0.0.0.0	255.255.248.0	U	0 0	0	eth0

图 4

在这个输出中，难以一眼注意到 192.168.176.0 就适用于目标地址 192.168.182.131，至少对我来说是这样的。

我们能从这个案例中学习到什么呢？最直接的启示便是翻出简历，投奔甲方去。这样就可以在搞砸系统的时候，义正词严地要求乙方解决了。假如你固执地想继续当乙方，那就开始学习 Wireshark 吧。再有经验的工程师也有犯迷糊的时候，而 Wireshark 从来不会，它随时随地都能告诉你真相，不偏不倚。

Excel 文件的保存过程

初试锋芒

Excel 文件的保存过程

13

当我们在 Notepad 等文本编辑器上单击 File-->Save 的时候，底层的操作非常简单——编辑器上的内容被直接写入文件了（见图 1）。假如这个文件是被保存到了网络盘上，我们就可以从 Wireshark 抓包上看到这个过程（见图 2）。

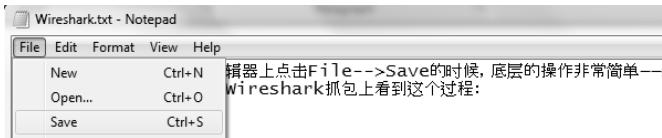


图 1

No.	Source	Destination	Time	Protocol	Info
58	10.32.200.41	10.32.106.50	2014-06-08 12:16:51	SMB2	write Request Len:6 Off:0 File: Temp\wireshark.txt
59	10.32.106.50	10.32.200.41	2014-06-08 12:16:51	SMB2	write Response

图 2

包号 58:

客户端：“我要写 6 个字节到/Temp/wireshark.txt 中”。

包号 59:

服务器：“写好了。”

相比之下，微软 Office 保存文件的过程就没有这么简单了，所以微软的老用户都或多或少经历过保存文件时发生的问题。比如图 3 中的 Excel 提示信息就很常见，它说明该文件被占用，暂时保存不了。这样的问题在 Notepad 上是不会发生的。

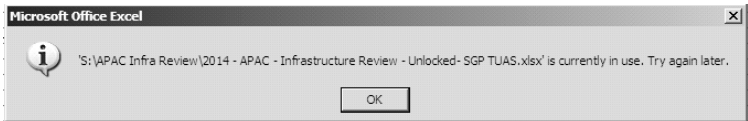


图 3

那么，Excel 究竟是如何保存文件的呢？虽然我的手头没有微软的文档，但只要把文件保存到网络盘上，就可以借助 Wireshark 看到整个过程了。我在实验室中编辑了 Excel 文件“wireshark.xlsx”，然后在保存时抓了个包，我们一起来分析其中比较关键的几个步骤（见图 4）：

No.	Source	Destination	Time	Protocol	Info
24	10.32.200.41	10.32.106.50	2014-	SMB2	Create Request File: Temp\DCD652B.tmp
25	10.32.106.50	10.32.200.41	2014-	SMB2	Create Response, Error: STATUS_OBJECT_NAME_NOT_FOUND
26	10.32.200.41	10.32.106.50	2014-	SMB2	Create Request File: Temp\DCD652B.tmp
27	10.32.106.50	10.32.200.41	2014-	SMB2	Create Response File: Temp\DCD652B.tmp
No.	Source	Destination	Time	Protocol	Info
38	10.32.200.41	10.32.106.50	2014-	SMB2	write Request Len:8184 Off:0 File: Temp\DCD652B.tmp
42	10.32.106.50	10.32.200.41	2014-	SMB2	write Response

图 4

这几个包可以解析为下述过程。

24 号包：

客户端：“/Temp 目录中存在一个叫 DCD652B.tmp 的文件吗？”

25 号包：

服务器：“不存在。”

26 号包：

客户端：“那我要创建一个叫 DCD652B.tmp 的文件。”

27 号包：

服务器：“建好了。”

38 号包：

客户端：“把 Excel 里的内容写到 DCD652B.tmp 里。”

42 号包：

服务器：“写好了。”

从以上过程可见，Excel 并没有直接把文件内容存到 wireshark.xlsx 上，而是存到一个叫 DCD652B.tmp 的临时文件上了。接下来再看（见图 5）。

No.	Source	Destination	Time	Protocol	Info
47	10.32.200.41	10.32.106.50	2014-06-08 13:01:02	SMB2	Create Request File: Temp\6AF04530.tmp
48	10.32.106.50	10.32.200.41	2014-06-08 13:01:02	SMB2	Create Response, Error: STATUS_OBJECT_NAME_NOT_FOUND
No.	Source	Destination	Time	Protocol	Info
97	10.32.200.41	10.32.106.50	2014-	SMB2	SetInfo Request FILE_INFO/SMB2_FILE_RENAME_INFO File: Temp\wireshark.xlsx NewName:Temp\6AF04530.tmp
98	10.32.106.50	10.32.200.41	2014-	SMB2	SetInfo Response
No.	Source	Destination	Time	Protocol	Info
103	10.32.200.41	10.32.106.50	2014-	SMB2	SetInfo Request FILE_INFO/SMB2_FILE_RENAME_INFO File: Temp\DCD652B.tmp NewName:Temp\wireshark.xlsx
104	10.32.106.50	10.32.200.41	2014-	SMB2	SetInfo Response

图 5

47 号包：

客户端：“/Temp 目录里存在一个叫 6AF04530.tmp 的文件吗？”

48 号包：

服务器：“不存在。”

97 号包：

客户端：“那好，把原来的 wireshark.xlsx 重命名成 6AF04530.tmp。”

98 号包：

服务器：“重命名完毕。”

103 号包：

客户端：“再把一开始那个临时文件 DCD652B.tmp 重命名成 wiresahrk.xlsx。”

104 号包：

服务器：“重命名完毕。”

从以上过程可知，原来的 wireshark.xlsx 被重命名成一个临时文件，叫 6AF04530.tmp。而之前创建的那个临时文件 DCD652B.tmp 又被重命名成 wireshark.xlsx。经过以上步骤之后，我们拥有一个包含新内容的 wireshark.xlsx，

还有一个临时文件 6AF04530.tmp（也就是原来那个 wireshark.xlsx）。接着往下看，就发现 6AF04530.tmp 被删除了（见图 6）。

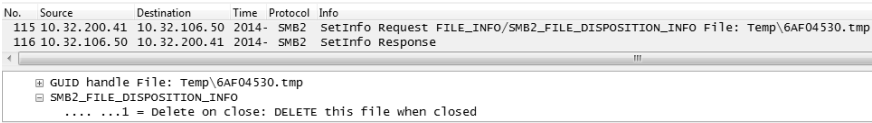


图 6

微软把保存过程设计得如此复杂，自然是有很多好处的。不过复杂的设计往往伴随着更多出问题的概率，因为其中一步出错就意味着保存失败。比如上文提到的报错信息 “...is currently in use. Try again later”，大多数时候的确是文件被占用才触发的，但也有时候是 Excel bug 或者杀毒软件导致的。无论出于何种原因，我们只有理解了保存时发生的细节，才可能探究到真相。

Wireshark 正是获得这些细节的通用法宝，任何经过网络所完成的操作，我们都可以从 Wireshark 中看到。有了这样的利器，还有多少问题能难住你？

你一定会喜欢的技巧

初试锋芒

你一定会喜
欢的技巧

17

我开始学习 Wireshark 的时候，到处碰壁，差点就放弃了。那时最希望的是有前辈能指点迷津，可惜四处求教却鲜有收获。即便多年后的今天，网络上能找到的中文资料还是寥寥无几，少之又少。所以我总结了一些自认为称得上技巧的东西，希望能帮初学者少走一点弯路。

一、抓包

拿到一个网络包时，我们总是希望它尽可能小。因为操作一个大包相当费时，有时甚至会死机。如果让初学者分析 1GB 以上的包，估计会被打击得信心全无。所以抓包时应该尽量只抓必要的部分。有很多方法可以实现这一点。

1. 只抓包头。一般能抓到的每个包（称为“帧”更准确，但是出于表达习惯，本书可能会经常用“包”代替“帧”和“分段”）的最大长度为 1514 字节，启用了 Jumbo Frame（巨型帧）之后可达 9000 字节以上，而大多数时候我们只需要 IP 头或者 TCP 头就足够分析了。在 Wireshark 上可以这样抓到包头：单击菜单栏上的 Capture-->Options，然后在弹出的窗口上定义“Limit each packet to”的值。我一般设个偏大的数字：80 字节，也就是说每个包只抓前 80 字节。这样 TCP 层、网络层和数据链路层的信息都可以包括在内（见图 1）。

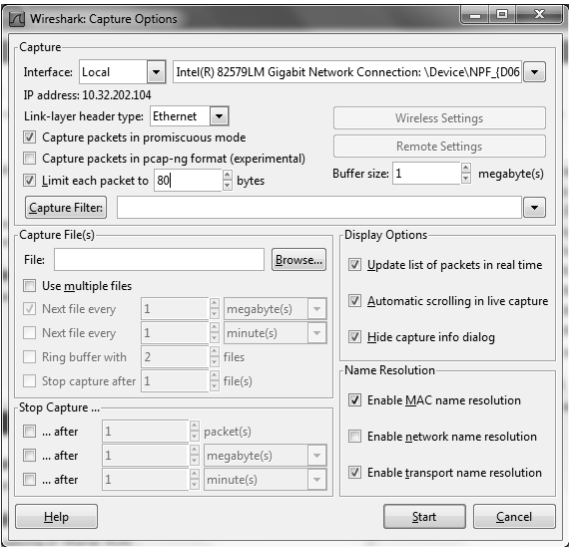


图 1

如果问题涉及应用层，就应该再加上应用层协议头的长度。如果你像我一样经常忘记不同协议头的长度，可以输入一个大点的值。即便设成 200 字节，也比 1514 字节小多了。

以上是使用 Wireshark 抓包时的建议。用 tcpdump 命令抓包时可以用“-s”参数达到相同效果。比如以下命令只抓 eth0 上每个包的前 80 字节，并把结果存到 /tmp/tcpdump.cap 文件中。

```
[root@server_1 /]# tcpdump -i eth0 -s 80 -w /tmp/tcpdump.cap
```

2. 只抓必要的包。服务器上的网络连接可能非常多，而我们只需要其中的一小部分。Wireshark 的 Capture Filter 可以在抓包时过滤掉不需要的包。比如在成百上千的网络连接中，我们只对 IP 为 10.32.200.131 的包感兴趣，那就可以在 Wireshark 上这样设置：单击菜单栏上的 Capture-->Options，然后在 Capture Filter 中输入“host 10.32.200.131”（见图 2）。

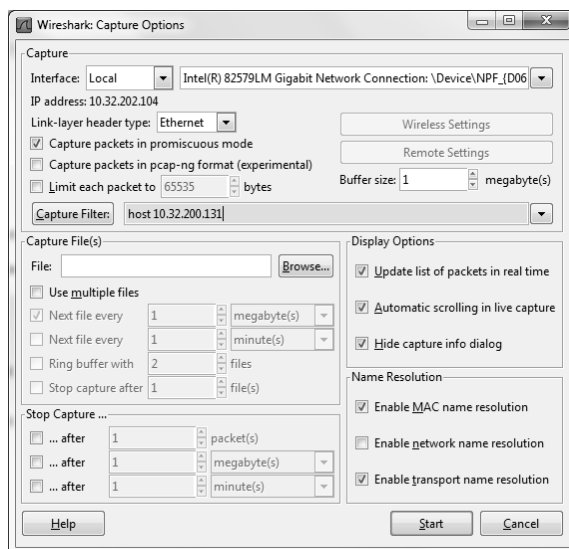


图 2

如果对更多 filter 表达式感兴趣，请参考 <http://wiki.wireshark.org/CaptureFilters>。

用 tcpdump 命令抓包时，也可以用“host”参数达到相同效果。比如以下命令只抓与 10.32.200.131 通信的包，并把结果存到/tmp/tcpdump.cap 文件中。

```
[root@server_1 /]# tcpdump-i eth0 host 10.32.200.131-w /tmp/tcpdump.cap
```

注意：设置 Capture Filter 之前务必三思，以免把有用的包也过滤掉，尤其是容易被忽略的广播包。当然有时候再怎么考虑也会失算，比如我有一次把对方的 IP 地址设为 filter，结果一个包都没抓到。最后只能去掉 filter 再抓，才发现是 NAT（网络地址转换）设备把对方的 IP 地址改掉了。

抓的包除了要小，最好还能每步操作打上标记。这样的包一目了然，赏心悦目。比如要在 Windows 上抓一个包含三步操作的问题，我会这样抓。

(1) ping <IP> -n 1 -l 1

(2) 操作步骤 1

(3) ping <IP> -n 1 -l 2

(4) 操作步骤 2

(5) ping <IP> -n 1 -l 3

(6) 操作步骤 3

如图 3 所示，如果我需要分析步骤 1，则只要看 146~183 之间的包即可。注意到 146 号包最底下的“Data (1 byte)”了吗？byte 的数目表示是第几步，这样就算在步骤很多的情况下也不会混乱。

抓包的技巧还有很多，比如可以写一个脚本来循环抓包，等侦察到某事件时自动停止。一位工程师即便不懂网络分析，但如果能抓得一手好包，也是一项很了不起的技能了。

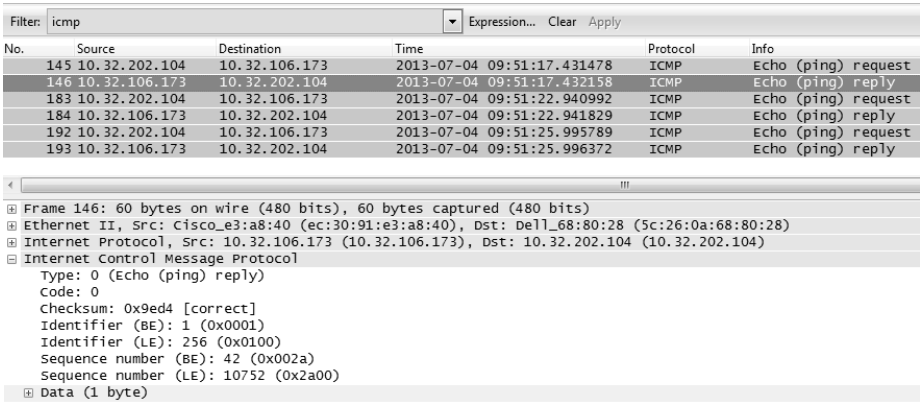


图 3

二、个性化设置

Wireshark 的默认设置堪称友好，但不同用户的从事领域和使用习惯各有不同，所以有时需要根据自己的情况对配置略作修改。

1. 我经常需要参照服务器上的日志时间，找到出问题时的网络包。所以就把 Wireshark 的时间调成跟服务器一样的格式。单击 Wireshark 的 View-->Time Display Format-->Date and Time of Day，就可以实现此设置（见图 4）。

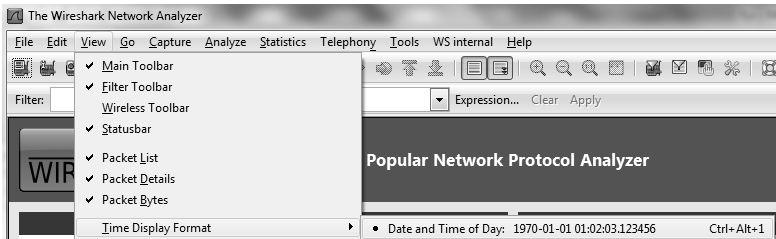


图 4

2. 不同类型的网络包可以自定义颜色，比如网络管理员可能会把 OSPF 等协议或者与 Spanning Tree Protocol（生成树协议）相关的网络包设成最显眼的颜色。而文件服务器的管理员则更关心 FTP、SMB 和 NFS 协议的颜色。我们可以通过 View --> Coloring Rules 来设置颜色。如果同事已经有一套非常适合你工作内容的配色方案，可以请他从 Coloring Rules 窗口导出，然后导入到你的 Wireshark 里（见图 5）。记得下次和他吃饭时主动买单，要知道配一套养眼的颜色可要花不少时间。

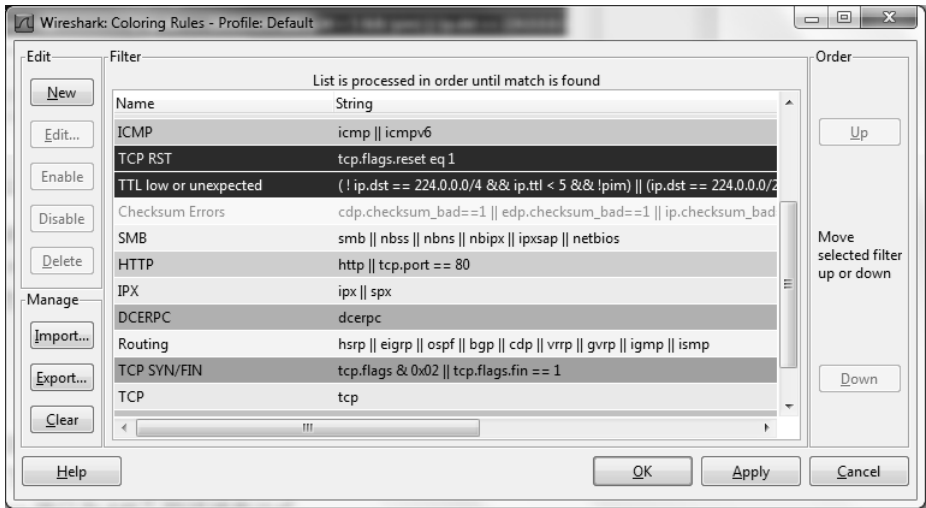


图 5

3. 更多的设置可以在 Edit-->Preferences 窗口中完成。这个窗口的设置精度可以达到一些协议的细节。比如在此窗口单击 Protocols-->TCP 就可以看到多个 TCP 相关选项，将鼠标停在每一项上都会有详细介绍。假如经常要对 Sequence Number 做加减运算，不妨选中 Relative sequence numbers（见图 6），这样会使 Sequence number 看上去比实际小很多。

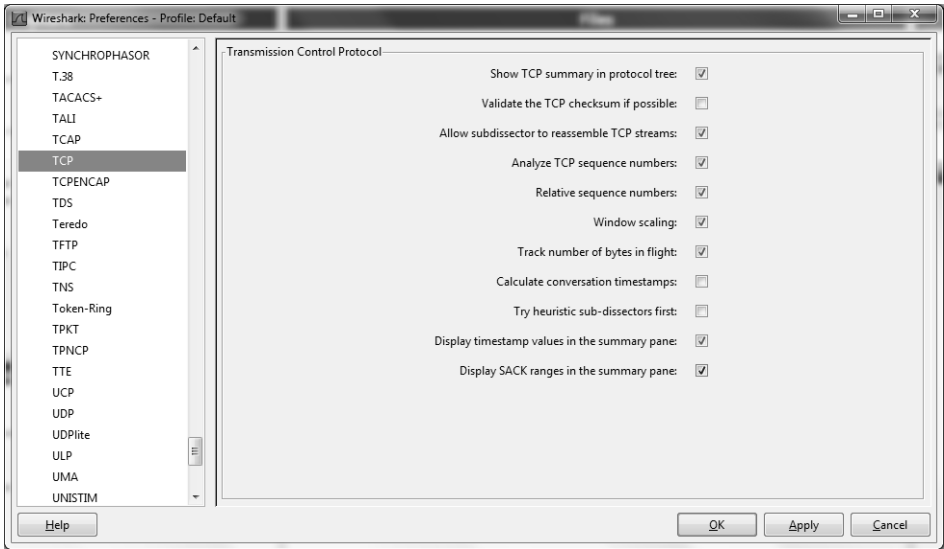


图 6

4. 如果你在其他时区的服务器上抓包，然后下载到自己的电脑上分析，最好把自己电脑的时区设成跟抓包的服务器一样。这样，Wireshark 显示的时间才能匹配服务器上日志的时间。比如说，服务器的日志显示 2/13/2014 13:01:32 有一个错误信息。那我们要在自己电脑上调整时区之后，才能到 Wireshark 上检查 2/13/2014 13:01:32 左右的包，否则就得先换算时间。

三、过滤

很多时候，解决问题的过程就是层层过滤，直至找到关键包。前面已经介绍过抓包时的 Capture Filter 功能了。其实在包抓下来之后，还可以进一步过滤，而且这一层的过滤功能更加强大。图 7 表示一个“IP 为 10.32.106.50，且 TCP 端口为 445”的过滤表达式。

Filter: ip.addr eq 10.32.106.50 && tcp.port eq 445				Expression...	Clear	Apply
No.	Source	Destination	Time	Protocol	Info	
448	10.32.200.48	10.32.106.50	2014-01-27 16:29:14.119762	TCP	49375 > microsoft-ds [SYN] Seq=0 win=8192 Len=0	
449	10.32.106.50	10.32.200.48	2014-01-27 16:29:14.120660	TCP	microsoft-ds > 49375 [SYN, ACK] Seq=0 Ack=1 win=0	
450	10.32.200.48	10.32.106.50	2014-01-27 16:29:14.120757	TCP	49375 > microsoft-ds [ACK] Seq=1 Ack=1 win=65536	
451	10.32.200.48	10.32.106.50	2014-01-27 16:29:14.120817	SMB	Negotiate Protocol Request	

图 7

要说过滤的作用与技巧，就算专门写一本小册子都不为过。篇幅所限，本文只能“过滤”出最适合初学者的部分。

1. 如果已知某个协议发生问题，可以用协议名称过滤一下。以 Windows Domain 的身份验证问题为例，如果已知该域的验证协议是 Kerberos，那么就在 Filter 框输入 Kerberos 作为关键字过滤。除了纯粹的 Kerberos 包，你还将得到 Session Setup 之类包含 Kerberos 的包（见图 8）。

Filter:	kerberos	Expression...	Clear	Apply	
No.	Source	Destination	Time	Protocol	Info
25	10.32.106.116	10.32.106.103	2012-03-14 15:14:07.320947	KRB5	TGS-REQ
26	10.32.106.103	10.32.106.116	2012-03-14 15:14:07.321631	KRB5	TGS-REP
28	10.32.106.116	10.32.106.103	2012-03-14 15:14:07.322618	SMB	Session Setup AndX Request
30	10.32.106.103	10.32.106.116	2012-03-14 15:14:07.323370	SMB	Session Setup AndX Response

图 8

用协议过滤时务必考虑到协议间的依赖性。比如 NFS 共享挂载失败，问题可能发生在挂载时所用的 mount 协议，也可能发生在 mount 之前的 portmap 协议。这种情况下就需要用“portmap || mount”来过滤了（见图 9）。如果不懂协议间的依赖关系怎么办？我也没有好办法，只能暂时放弃这个技巧，等熟悉了该协议后再用。

Filter: portmap mount		☐ Expression...		Clear	Apply
No.	Source	Destination	Time	Protocol	Info
112	10.32.106.159	10.32.106.62	2013-07-15 14:21:18.768742	PortmapV2	GETPORT Call (Reply In 113) NFS(100003) V:3 TCP
113	10.32.106.62	10.32.106.159	2013-07-15 14:21:18.768742	PortmapV2	GETPORT Reply (Call In 112) Port:2049
128	10.32.106.159	10.32.106.62	2013-07-15 14:21:18.772649	PortmapV2	GETPORT Call (Reply In 129) MOUNT(100005) V:3 UDP
129	10.32.106.62	10.32.106.159	2013-07-15 14:21:18.772649	PortmapV2	GETPORT Reply (Call In 128) Port:1234
132	10.32.106.159	10.32.106.62	2013-07-15 14:21:18.772649	MOUNT V3	NULL call (Reply In 133)
133	10.32.106.62	10.32.106.159	2013-07-15 14:21:18.772649	MOUNT V3	NULL Reply (Call In 132)
134	10.32.106.159	10.32.106.62	2013-07-15 14:21:18.772649	MOUNT V3	MNT Call (Reply In 135) /code
135	10.32.106.62	10.32.106.159	2013-07-15 14:21:18.772649	MOUNT V3	MNT Reply (Call In 134)

图 9

2. IP 地址加 port 号是最常用的过滤方式。除了手工输入 ip.addr<IP 地址>&&tcp.port<端口号>之类的过滤表达式，Wireshark 还提供了更快捷的方式：右键单击感兴趣的包，选择 Follow TCP/UDP Stream（选择 TCP 还是 UDP 要视传输层协议而定）就可以自动过滤（见图 10）。而且该 Stream 的对话内容会在新弹出的窗口中显示出来。

10	10.32.106.72	10.32.200.43	2013-10-03 07:34:30.470170	SMB	Session Setup AndX Response
11	10.32.200.43	10.32.106.72	2013-10-03 07:34:30.470888	SMB	Tree Connect AndX Request
12	10.32.106.72	10.32.200.43	2013-10-03 07:34:30.471797	SMB	Tree Connect AndX Response
13	10.32.200.43	10.32.106.72	2013-10-03 07:34:30.472193	SMB	Trans2 Request, QUERY_PATH
14	10.32.106.72	10.32.200.43	2013-10-03 07:34:30.473051	SMB	Trans2 Response, QUERY_PATH
15	10.32.200.43	10.32.106.72	2013-10-03 07:34:30.473177	SMB	Trans2 Request, QUERY_PATH
16	10.32.106.72	10.32.200.43	2013-10-03 07:34:30.473913	SMB	Trans2 Response, QUERY_PATH
17	10.32.200.43	10.32.106.72	2013-10-03 07:34:30.474539	SMB	Trans2 Request, QUERY_PATH
18	10.32.106.72	10.32.200.43	2013-10-03 07:34:30.475274	SMB	Trans2 Response, QUERY_PATH
19	10.32.200.43	10.32.106.72	2013-10-03 07:34:30.475383	SMB	Trans2 Request, QUERY_PATH
20	10.32.106.72	10.32.200.43	2013-10-03 07:34:30.476100	SMB	Trans2 Response, QUERY_PATH
21	10.32.200.43	10.32.106.72	2013-10-03 07:34:30.476377	SMB	Trans2 Request, QUERY_PATH
Frame 10: 156 bytes on wire (1248 bits), 156 bytes captured (1248 bits)					
Ethernet II, Src: cisco_e3:a6:80 (ec:30:91:e3:a6:80), Dst: dell_68:80:28 (5c:26:0a:68:80:28)					
Internet Protocol, Src: 10.32.106.72 (10.32.106.72), Dst: 10.32.200.43 (10.32.200.43)					

Mark Packet (toggle)

Ignore Packet (toggle)

Set Time Reference (toggle)

Manually Resolve Address

Apply as Filter

Prepare a Filter

Conversation Filter

Colorize Conversation

SCTP

Follow TCP Stream

Follow UDP Stream

Follow SSL Stream

图 10

经常有人在论坛上问，Wireshark 是按照什么过滤出一个 TCP/UDP Stream 的？答案就是：两端的 IP 加 port。单击 Wireshark 的 Statistics-->Conversations，

再单击 TCP 或者 UDP 标签就可以看到所有的 Stream（见图 11）。

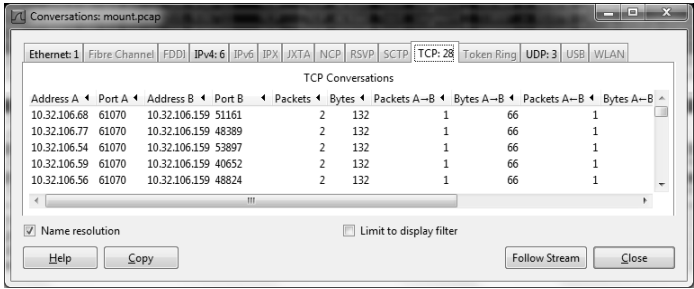


图 11

3. 用鼠标帮助过滤。我们有时因为 Wireshark 而苦恼，并不是因为它功能不够，而是强大到难以驾驭。比如在过滤时，有成千上万的条件可供选择，但怎么写才是合乎语法的？虽然 <http://www.wireshark.org/docs/dfref/> 提供了参考，但经常查找毕竟太费时费力了。Wireshark 考虑到了这个需求，右键单击 Wireshark 上感兴趣的内容，然后选择 Prepare a Filter-->Selected，就会在 Filter 框中自动生成过滤表达式。在有复杂需求的时候，还可以选择 And、Or 等选项来生成一个组合的过滤表达式。

假如右键单击之后选择的不是 Prepare a Filter，而是 Apply as Filter-->Selected，则该过滤表达式生成之后还会自动执行。图 12 显示了一个 SMB 包的 SMB Command: Read AndX 上右键单击，并选择 Selected 之后，所有的 Read 包都会被过滤出来。

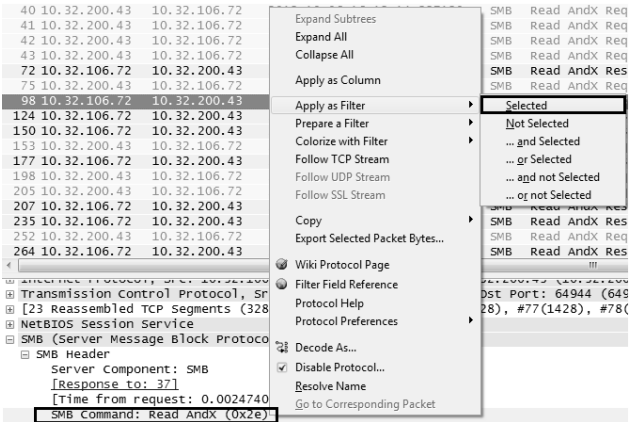


图 12

4. 我们可以把过滤后得到的网络包存在一个新的文件里，因为小文件更方便

操作。单击 Wireshark 的 File-->Save As, 选中 Displayed 单选按钮再保存, 得到的新文件就是过滤后的部分 (见图 13)。

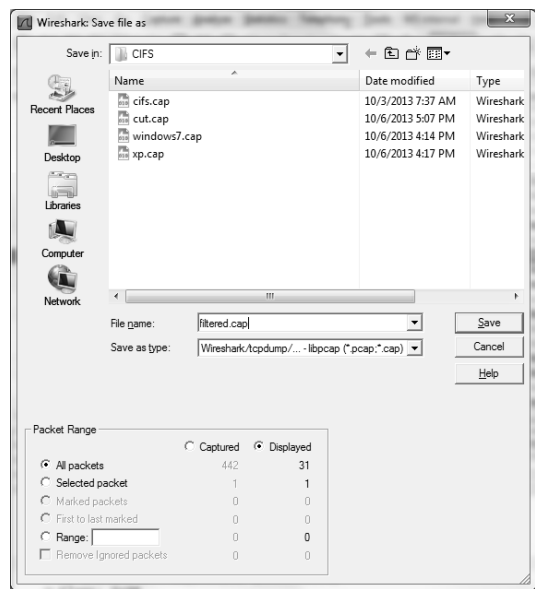


图 13

有时候你会发现, 保存后的文件再打开时会显示很多错误。这是因为过滤后得到的不再是一个完整的 TCP Stream, 就像抓包时漏抓了很多一样。所以选择 Displayed 选项时要慎重考虑。

注意: 有些 Wireshark 版本把这个功能移到了菜单 File-->Export Specified Packets...选项中, 如图 14 所示。

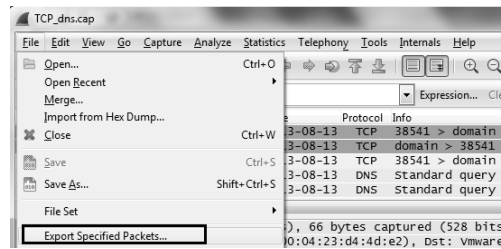


图 14

总体来说, 过滤是 Wireshark 中最有趣, 最难, 也是最有价值之处, 值得我们用心学习。

四、让 Wireshark 自动分析

有些类型的问题，我们根本不需要研究包里的细节，直接交给 Wireshark 分析就行了。

1. 单击 Wireshark 的 Analyze-->Expert Info Composite，可以在不同标签下看到不同级别的提示信息。比如重传的统计、连接的建立和重置统计，等等。在分析网络性能和连接问题时，我们经常需要借助这个功能。图 15 是 TCP 包的重传统计。

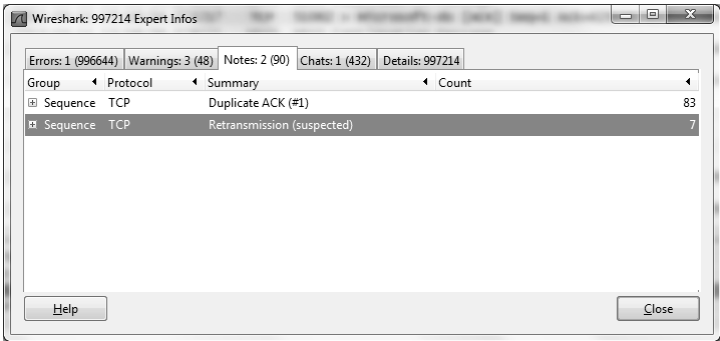


图 15

2. 单击 Statistics-->Service Response Time，再选定协议名称，可以得到响应时间的统计表。我们在衡量服务器性能时经常需要此统计结果。图 16 展示的是 SMB2 读写操作的响应时间。

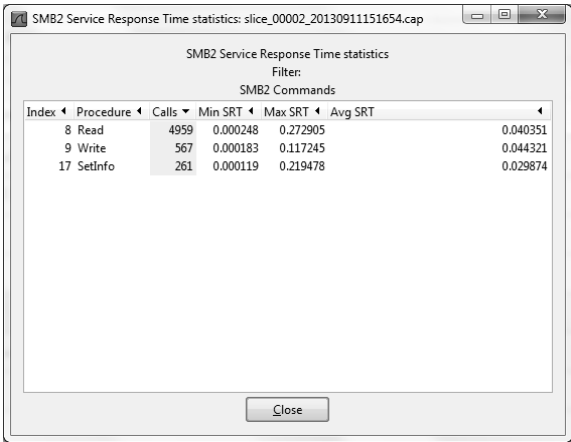


图 16

3. 单击 Statistics-->TCP Stream Graph，可以生成几类统计图。比如我曾经用 Time-Sequence Graph (Stevens)生成了图 17。

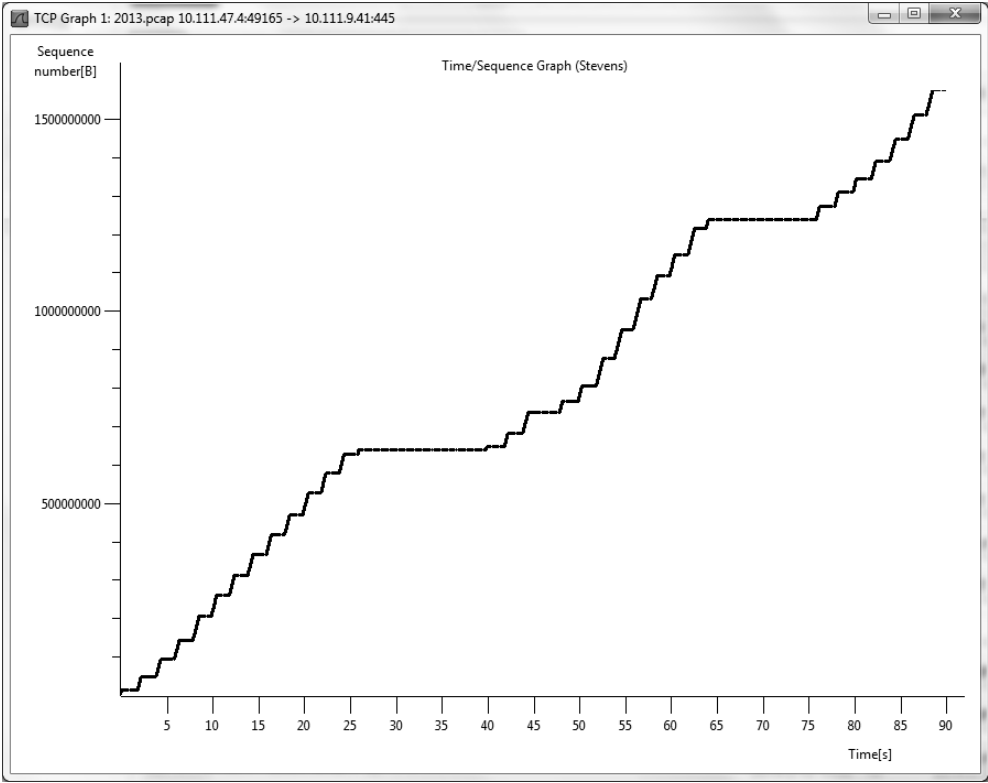


图 17

从图 17 中可以看出 25~40 秒，以及 65~75 秒之间没有传输数据。进一步研究，发现发送方内存不足，所以偶尔出现暂停现象，添加内存后问题就解决了。

为什么 Wireshark 要把这个图称为“Stevens”呢？我猜是为了向《TCP/IP Illustrated》的作者 Richard Stevens 致敬。这也是我非常喜欢的一套书，在此推荐给所有读者。

4. 单击 Statistics-->Summary，可以看到一些统计信息，比如平均流量等，这有助于我们推测负载状况。比如图 18 中的网络包才 1.594Mbit/s，说明流量低得很。

初试锋芒

你一定会喜
欢的技巧

27

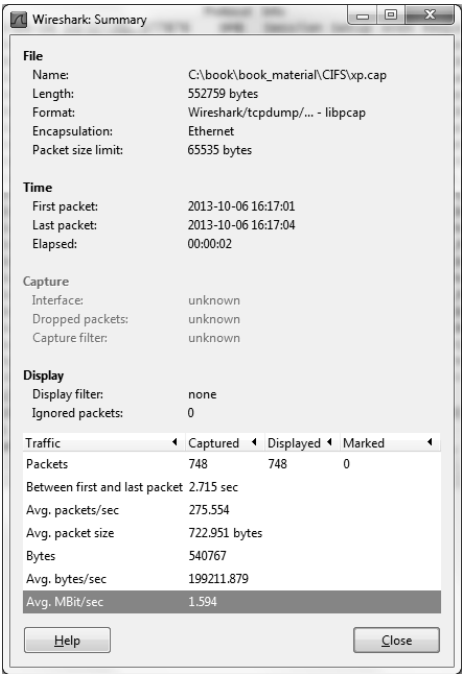


图 18

五、最容易上手的搜索功能

与很多软件一样，Wireshark 也可以通过“Ctrl+F”搜索关键字。假如我们怀疑包里含有“error”一词，就可以按下“Ctrl+F”之后选中“String”单选按钮，然后在 Filter 中输入“error”进行搜索（见图 19）。很多应用层的错误都可以靠这个方法锁定问题包。

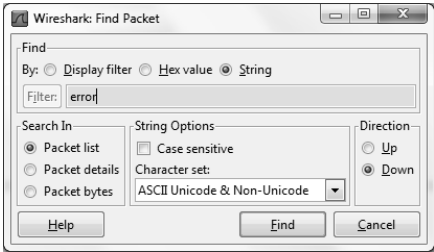


图 19

一篇文章不可能涵盖所有技巧，本文就到此为止。最后要分享的，是我认为最“笨”但也是最重要的一个技巧——勤加练习。只要练到这些技巧都变成习惯，就可以算登堂入室了。

Patrick 的故事

初试锋芒

Patrick 的
故事

29

我还在山脚下的时候，Patrick 已经在山顶了。至今我还只能在山坡仰望他。

第一次听说 Patrick 的名字是在 6 年前。当时我初入存储行业，经常被各类难题所困。有一次，我要把大批文件从 Windows 迁移到文件服务器（NAS）上，不知道为什么有些文件就是过不去，报错信息也没有参考价值。走投无路之际，一位美国同事提了个建议：我司在波士顿有一位很厉害的专家，也许可以请教一下他。我抱着试一试的态度发了一封求助信，没想到十几分钟后就得到回复。专家建议我抓一个网络包，然后用 Wireshark 看看这些文件有没有特殊属性。我立即照办，果然在这些文件上看到 Temporary 属性。知道原因后，问题很快就解决了。那是我第一次接触 Wireshark，而那位专家就是 Patrick。

从此我就喜欢上了 Wireshark，因为它实在很有用，就像是学武之人得到了一把称手好剑。而 Patrick 却渐渐被我淡忘了。直到一年之后，我又遇到这样一个难题：有一台文件服务器的读性能只有 10MB/s，远低于客户的期望。我尝试过很多调优方式，性能却只降不升。徒劳三天之后，我对自己彻底失去信心。这时候我又想起了 Patrick，说不定他能给点意见呢。于是我上传了一个网络包，请他帮忙分析。一小时后奇迹再次出现，我收到了他的回信。信中提到两点建议。

- TCP 超时重传的间隔时间太长，设置一个较小的时间可以减少重传对性能的影响。
- 该网络频繁拥塞，拥塞点大多在 32KB 以上。如果把发送窗口限制在 32KB，就可以避免触碰拥塞点。

我简直不敢相信这些分析，短短一个小时怎么能看出这么深奥的原因？我好歹也用了一年 Wireshark 了，几乎每个菜单都很熟悉，却从来不知道有个地方可

以看出拥塞点。不过有了上次成功经验，我决定还是尝试一下这两条建议。在把超时重传时间减小之后，读性能立即达到 20MB/s，比之前提高了一倍。这个结果实在太振奋人心，一扫三天来的阴霾。我赶紧再设置发送窗口，没想到性能又提高了一倍，达到 40MB/s。现场的工程师和客户都在欢呼，我在电话上也久久不能平静。这时候我才真正被 Patrick 的实力所震撼。觉得自己就像武侠小说中初涉江湖的少年，一年前被深藏不露的大侠所救，却只看到好剑的厉害。一年后再次身陷险境，看到大侠出招，才知道自己有眼不识泰山，恨不得立即磕头拜师。

等到我学会在 Wireshark 上看拥塞窗口，已经是半年后的事了。期间我重读了 Richard Stevens 的《TCP/IP Illustrated》，遇到疑难就请教 Patrick。他每次的回信都极像专业论文，篇幅极长却又字字珠玑，有一次甚至当场写了个程序帮我理解概念。他的严谨、耐心和分享精神都堪称顶级工程师的典范。假如说他是一位老师，那一定是我求学路上碰到过最为出色的老师。我专门在 Outlook 里设了一个 rule，把他的所有邮件放在一起，至今一封都没有删过。在非技术问题上，Patrick 从来惜字如金。我曾经问他：“Have you ever thought about writing a book?” 他很简单地回答：“I am not a good author.” 如果他都不算 good author 的话，有几个人称得上好？即便把我收藏的这些邮件集结起来，也是一本好书了。

我曾经想过，将来某一天能不能学到 Patrick 的水平？现在已经不考虑这个问题了，因为我发现他的技术似乎是没有边界的。有一天，我被一个 Active Directory 的问题难住，微软的工程师也无可奈何，他却精准地解决了。我才知道他对 Windows Domain 也深有研究。当天中午和研发部门的同事一起吃饭时，我向他提起了无所不知的 Patrick。没想到这位同事也很震惊，“他懂这么多啊？我只知道他正在帮我们处理一个操作系统的问题。”从同事转来的邮件上，我果然看到 Patrick 向他讲解了一个操作系统的细节问题。这时我不禁想起他自谦过的一句话“Everybody has his expertise”。可是有什么技术领域不是你的 expertise？我很想当面问问这位素未谋面的老师。

几年后我到波士顿开会，第一个想见的人就是 Patrick。我带了中国的点心，也带着很多感谢去拜访他。可惜他那天没有在办公室里出现。邻座的同事说，“我们也很久没有见到 Patrick 了，他在家办公，而且是在夜里。”听说我是从中国慕

名而来，这位同事滔滔不绝地说起大家对 Patrick 的敬仰，并表示要帮忙联系。我考虑到他在夜里工作，白天肯定要休息，只能放弃登门拜访的念头。回国后收到 Patrick 的邮件，原来他知道后第二天就去了办公室，可惜我那时已经在飞机上了。

所以我至今没有见过 Patrick，但这又有什么关系？在网络时代，有些人就算从来没有机会见面，甚至不知道年龄和种族，也可以是最好的老师。

初试锋芒

Patrick 的
故事

31

Wireshark 的前世今生

初试锋芒

Wireshark
的前世今生

32

这是一个无关技术的小故事。但是作为 Wireshark 爱好者，了解一下这个软件的前世今生也是极好的，谁不想在中午和同（ling）事（dao）一起吃饭的时候讲个业内小故事，显得自己业务精湛又品味不俗呢？

故事要从 20 世纪 90 年代开始说起。那时的 IT 业欣欣向荣：摩托罗拉正野心勃勃地实施铱星计划；Google 的两位创始人还在房东的车库里研究搜索引擎。我们故事的主人公 Gerald Combs 还是默默无闻的青年。和那个时代的很多工程师一样，Gerald 技术精湛，热情上进，动手能力极强。他就职于一家网络提供商，时常需要分析软件来辅助工作。可是这样的软件太少了，而且一个 license 就要 80,000 美金。即便在今天的美国，这也不是一笔小数目。

和我们中的很多人不一样，Gerald 没有下载盗版软件，而是决定自己写一个。他单枪匹马忙碌了几个月。我们今天仍能想见其中的艰辛——即便是从业多年的工程师，对很多网络协议还一知半解，更不要说开发一个能分析协议的软件了。而一位工程师既精通多种协议，又能写好代码，更是常人难以企及的境界。但谦虚的 Gerald 一直对此轻描淡写，“I spent several months doing research and making notes.” 到了 1998 年 7 月，这个软件终于面世了。它带来了这样的功能：当你透过它看到网络时，不再是没有意义的 0 和 1，而是可以理解的简洁文字。有了它的专业解说，我们几乎能直接看懂网络上发生的一切。以前难以排查的问题，在它介入后便显露无遗。它还提供了权威的分析报告，比如重传率统计、响应时间和对话列表等，这解放了原本负担繁重的网络管理员，使他们有更多时间专注其他事务。

Gerald 把这个软件命名为 Ethereal，正对应了它的功能——还原以太网的真相。Ethereal 的代码版权自然属于 Gerald，而他所在的公司 NIS（Network Integration Services）则拥有 Ethereal 商标。当时谁也没有想到，这个归属权会在多年后引起一场风波。由于 Ethereal 写得太好了，而且是以 GNU GPL 开源许可

证发布的，世界各地的开发者纷纷参与到这个项目中。没过多久，它就涵盖了世界上大多数通信协议，成为广受欢迎的网络分析软件。它可以用于教学，如果网络教师用它辅助上课，可以大大提高学生的兴趣。也可以辅助开发和测试，是调试网络程序的好工具。当然它最大的用途还是诊断问题：从数据链路层到应用层的种种协议，几乎涉及网络的地方就有它的用武之地。更难得的是，Gerald 并没有打算从中获利，它至今还是完全免费的，每位愿意学习的工程师都可以受益。

世界的变化总是超乎我们的想象，尤其是在 IT 业。没几年时间，铱星计划彻底破产；Google 却成了最大的网络公司。只有 Gerald 没有变化，一直在兢兢业业地维护 Ethereal。每个月都有新的协议出现，已有的协议也在推出新版本，他永远有忙不完的活。中间仅仅发生过一次改名风波：2006 年他离开 NIS，加入了 CACE。由于和老东家在 Ethereal 的商标问题上无法达成一致，Gerald 把项目改名为 Wireshark。从此 Ethereal 这个风靡多年的项目停止了，只留下 www.ethereal.com 域名。我们至今还能访问它，但是会被重定向到一家叫 AOS 的公司。为什么不是重定向到 NIS 呢？因为 NIS 在 2011 年被 AOS 合并了。

Wireshark 延续了 Ethereal 的成功，现在有成千上万的开发者在追随 Gerald。每年还会召开一次为期 4 天的 Sharkfest 大会。2011 年 Wireshark 在 SecTools 排行第一，2012 年被 Insecure.org 评为 “No. 1 Packet Sniffers”。美国的技术作家们开始为它著书立说，中国的出版社也在引进（比如人民邮电出版社引进出版的《Wireshark 数据包分析实战（第 2 版）》）。值得一提的是，CACE 后来被 Riverbed 收购了，Riverbed 成了 Wireshark 项目的赞助商。很多中国工程师可能觉得 Riverbed 名不见经传，但说到 Linux 里常用的 tcpdump 命令就不会陌生。tcpdump 的开发者之一 Steve McCanne 就是 Riverbed 的 CTO。而 WinPcap 的开发者 Loris Degioanni 也在 Riverbed 工作。似乎冥冥之中自有天意，Riverbed 把网络探测界的先锋们聚到了一起。我们要向 Riverbed 致敬，多亏了这些伟大的工具，我们才得以窥探网络的秘密。

Gerald 不久前在 Twitter 上宣布，“Wireshark is, and will always be open source。”其实 Wireshark 即便不再开源也不会抹杀他的成就。改变世界的 IT 英雄，可以像 Jobs 一样领导一个成功的公司，更可以像 Gerald 一样创造一件传世的作品。他们的成就一样会被镌刻在 IT 历史的丰碑上。