25 | 抓包分析的回顾、拾遗，和提高

2022-04-01 杨胜辉

《网络排查案例课》 课程介绍 >

讲述：杨胜辉
时长 17:51 大小 16.35M

你好，我是胜辉。

在完成了三个实战模块之后，我们已经把三层以上，也就是网络层、传输层、应用层等的排查技术都比较完整地学习了一遍。在二十多节课里，我们学习了对 TCP 的各种行为，比如握手、挥手、拥塞、重传等行为进行排查的技巧，也对 tcpdump 和 Wireshark 进行了深度使用。

除了这些技巧以外，我们更是在 TLS 加解密和应用层 HTTP 协议等理论知识方面，做了不少深入的研究。我想，很多工具的使用技巧、案例的排查思路已经在我们的脑海中回荡，那么如果我们来一次系统性的总结和提高，对我们的学习效果一定是一种更大的促进。

所以在这节课里，我们就来对过去的二十多讲进行一次回顾、总结和提高吧。

tcpdump 和 tshark 等命令行工具的技巧

tcpdump 是我们用得最多的抓包工具，其中的基本技巧可以汇总为以下几个方面。

tcpdump

我们做抓包，一般都需要指定过滤条件，这样才能保证对系统的 CPU、内存、硬盘等资源不产生过大的影响。这里说的过滤条件又分基础参数和真正的过滤条件，我们分别来看一下。

基础参数

这类基础参数可能不算是用来过滤报文本身的参数，它们一般用于指定抓包的网口、报文长度等等。我们来逐个看一下。

要限定网络接口，可以用 -i 参数。比如：

复制代码

```
tcpdump -i eth0
```

> 补充：这里还有个小的注意点。当我们用 `-i any` 的时候，抓到的报文就不是标准以太网帧头格式了，这一点你需要知道一下。

如果我们要查看抓包过程中的详细信息，下面这三种参数任你选择：

复制代码

```
tcpdump -v          #提供TTL等信息
tcpdump -vv         #提供更多信息
tcpdump -vvv        #提供最详细的信息
```

> 补充：如果你还是用 `-w file.pcap` 存入了文件，加上 -v 参数可以每 10 秒打印一次抓取的报文数量。

默认情况下，tcpdump 会对 IP 进行反向域名查询等操作，这些 DNS 查询等行为可能影响抓包的效率。这时候就可以加上 -n 参数：

复制代码

```
tcpdump -n
```

为了节约抓包成本，我们经常会指定要抓取的每个报文的长度。比如，假如 TCP 没有启用扩展头部，那么指定 54 字节即可抓取到从二层帧头部到 TCP 头部的这些信息了。

复制代码

```
tcpdump -s 54
```

tcpdump 过滤条件

抓包的过滤条件当然是重中之重的技巧了，这里我们再复习一下。

要限定 IP，可以用下面的过滤器：

复制代码

```
tcpdump host 10.10.10.10
tcpdump dst host 10.10.10.10
tcpdump src host 10.10.10.11
```

要限定端口，可以用这个：

复制代码

```
tcpdump port 22
```

我们有时候也需要从已有的抓包文件中过滤出报文，然后存入一个新的文件。比如下面的例子：

复制代码

```
tcpdump -r file.pcap 'tcp[tcpflags] & (tcp-rst) != 0' -w rst.pcap
```

tshark

tshark 是一个跟随 Wireshark 一起安装进系统的工具，类似情况的工具还有 editcap、mergecap、capinfos 等等，算是 Wireshark 大礼包了。这些工具都有很强大的功能，这个大礼包真是物超所值。

我们先看 tshark。在课程中，我们主要用 tshark 实现了三个不同的需求场景。

第一个场景是**统计文件中的 HTTP 状态码**。我们可以到Gitee里找到相关的抓包示例文件，然后运行下面的命令：

```
tshark -r lesson17-in-shorten.pcap -T fields -e http.response.code | grep -v ^
```

输出如下：

```
2931 200
 704 502
 227 304
 141 400
  45 301
  41 302
  16 408
  14 403
   6 503
   6 404
   2 206
```

可见，HTTP 状态码都非常整体地统计展现出来了。所以我们做抓包分析，不仅要熟练掌握 Wireshark，同时也应该多了解这些命令行工具，它们经常可以提供 Wireshark 以外的一种良好的补充。

第二个场景是**解析文件中的 HTTP 事务的耗时**。比如我们可以用下面的命令：

```
tshark -r 文件名 -T fields -e http.time | grep -v ^$
```

第三个场景，是在找到 HTTP 耗时最高的事务，然后找这个**事务所在的整个 TCP 流的所有报文**，我们可以用下面的命令：

```
1 tshark -r 文件名 -T fields -e frame.number -e http.time -e tcp.stream | sort -k
```

这个命令比较长，用到了多次管道符，特别是最后一个管道符后面，使用了 `tshark -Y "tcp.stream eq xx"`，而这个过滤器就是把某个特定的 TCP 流给完整地展示出来，这样的话我们不仅可以找到 HTTP 耗时最高的事务，而且这个事务的完整的 TCP 流里的报文都展示了出来，方便我们做进一步的分析。

tshark 的功能有很多，比如它还可以查看 TCP 初始往返时间：

```
1 tshark -r file.cap -T fields -e tcp.analysis.initial_rtt | grep -v ^$
```

也可以统计 TCP 重传包的数量：

```
1 tshark -n -q -r file.pcap -z "io,stat,0,tcp.analysis.retransmission"
```

最后，tshark 其实也经常用来直接抓包，这时候它跟 tcpdump 是差不多的。比如下面这样：

```
1  tshark -i "Wireless Network Connection" -w file.pcap host 123.45.66.77
```

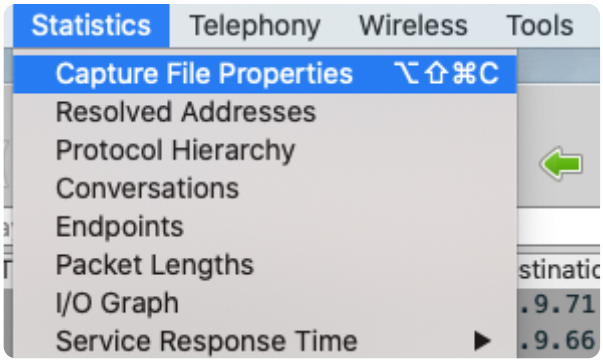
capinfos

这个工具主要是用来获取抓包文件的整体信息，比如我们使用下面这个命令，就获取到了抓包文件的报文数、文件大小、抓包时长等丰富的信息。

```
$ capinfos lesson17-in-shorten.pcap
File name:           lesson17-in-shorten.pcap
File type:           Wireshark/tcpdump/... - pcap
File encapsulation:  Ethernet
File timestamp precision:  microseconds (6)
Packet size limit:   file hdr: 65535 bytes
Packet size limit:   inferred: 150 bytes
Number of packets:   88 k
File size:           9769 kB
Data size:           115 MB
Capture duration:    297.564908 seconds
......
```

复制代码

它的功能跟 Wireshark 里 Statistics 下拉菜单里的 Capture File Properties 是类似的：



editcap

有时候我们需要传递抓包文件，同时又不想透露应用层数据等敏感信息，那么一个简便的做法，就是直接修改抓包文件中每个报文的长度。前面说过，`tcpdump -s` 是可以指定抓包大小的，而即使文件已经生成了，我们还是可以**通过 editcap 名把它改小**。是不是挺方便的？

比如要把原始文件 `old.pcap` 的每个报文截短为 54 字节，可以执行下面的命令：

```
editcap -s 54 old.pcap new.pcap
```

复制代码

另外，editcap 也可以用来把 TLS key 文件跟抓包文件合并在一起，这样你在分享文件的时候，就不需要传送两个文件了，而是一个合并过的文件就可以。比如下面的命令：

```
editcap --inject-secrets tls,key.log in.pcap out.pcap
```

复制代码

> 注意：命令中的 `tls` 跟后面的 `key.log` 文件名之间有一个逗号，虽然略有点奇怪，但这就是它的格式。

Wireshark 的技巧

Wireshark 可以算是我们课程中最闪亮的明珠了。不过很多同学在学习这门课之前，可能对 Wireshark 的了解很少，一打开文件就晕了，心里只有一个念头“不明觉厉”。当然，通过课程的学习，相信你对这方面已经积累了相当不错的经验。如果你现在看到 Wireshark 窗口甚至有“亲切”的感觉，那就对了，说明你有了长足的进步，已经真正跟 Wireshark 成为密友了。

这里我们再来简单回顾一下相关的技巧。

Wireshark 的解读技巧

在解读抓包文件时，第一步一般可以查看 Expert Information。这个信息的解读方法是这样的：

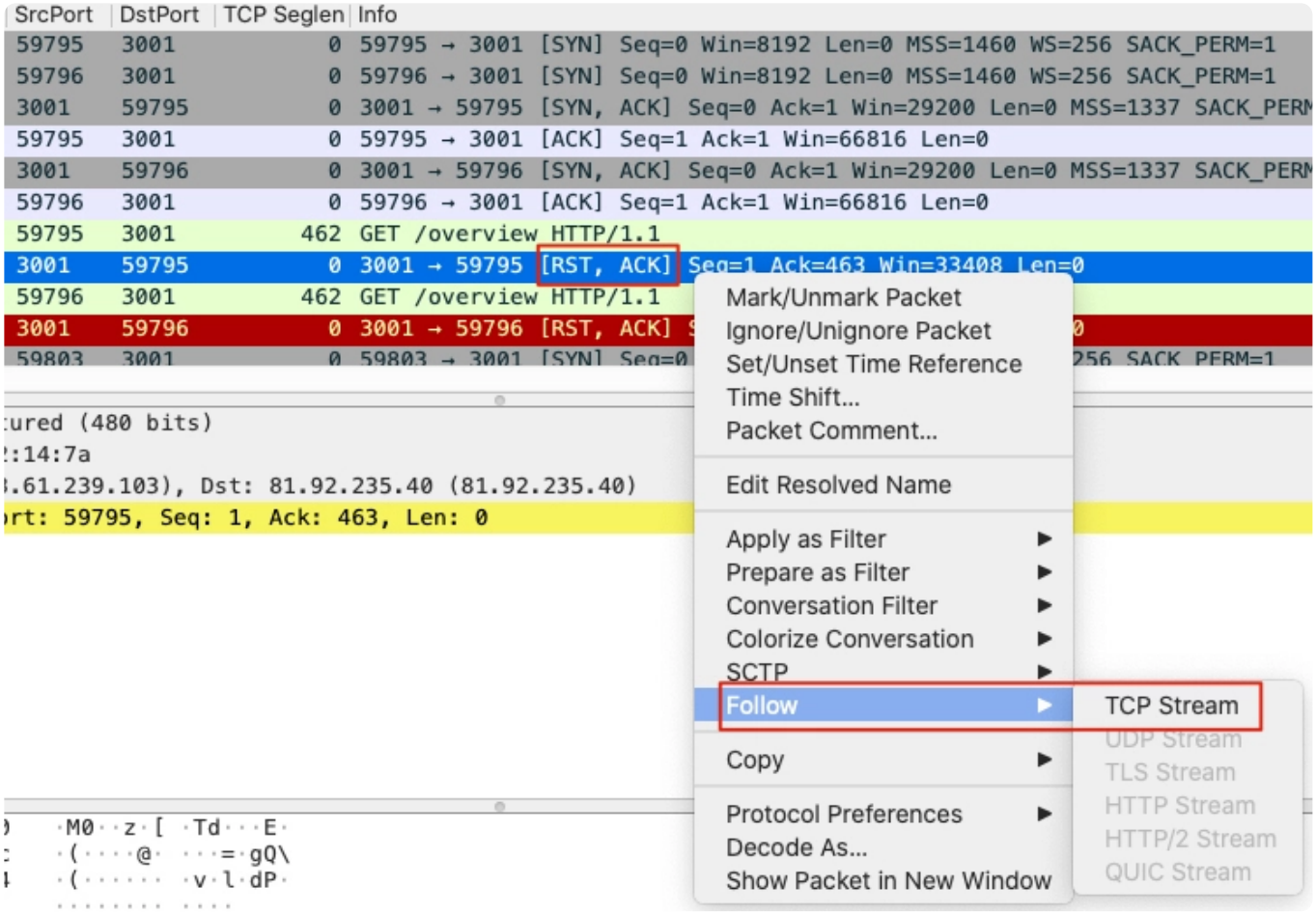
Warning 条目的底色是黄色，意味着可能有问题，应重点关注。

Note 条目的底色是浅蓝色，是在允许范围内的小问题，也要关注。

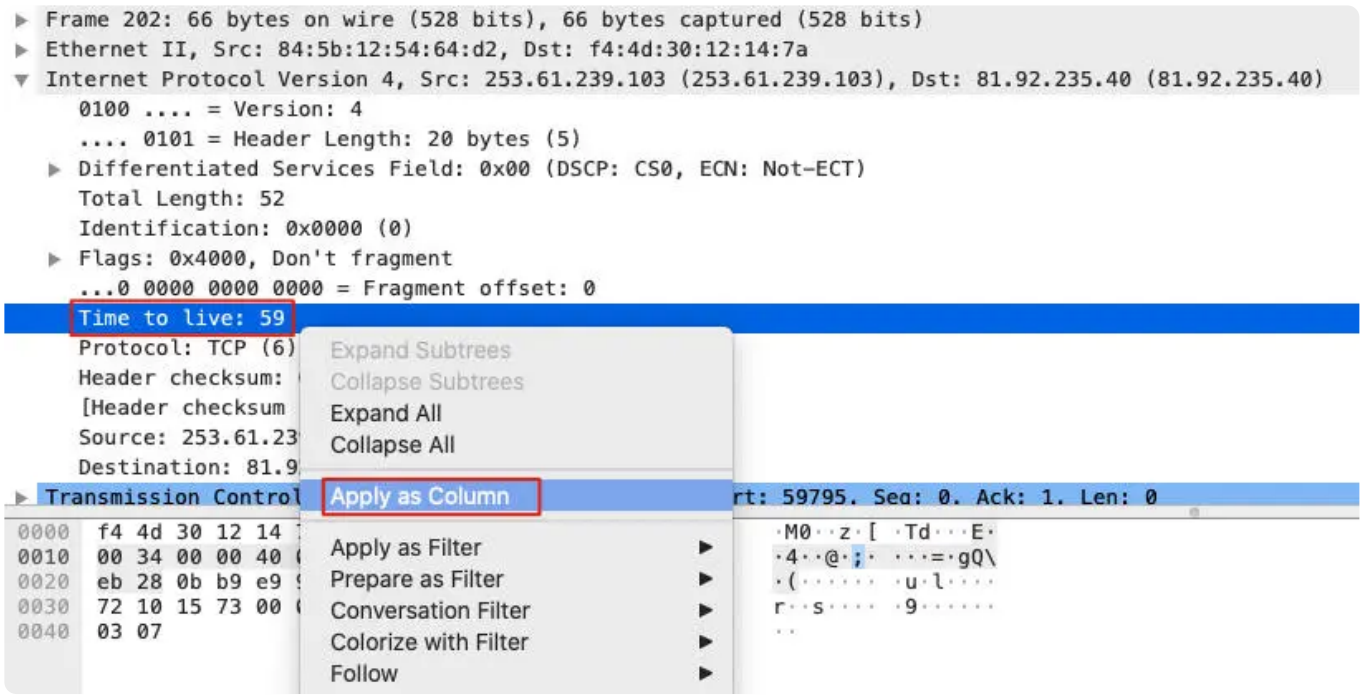
Chat 条目的底色是正常蓝色，属于 TCP/UDP 的正常行为，比如这次通信里 TCP 握手和挥手分别有多少次等，可以作为参考。

一般来说，乱序可以重点关注。

在解读完 Expert Information 之后，我们会找到可疑的报文，展开进一步的分析。此时我们用得最多的技巧，可能就是 **Follow TCP Stream** 了。我们可以选中一个报文后右单击，然后选择 Follow，在二级菜单中选择 TCP Stream，这样就来到了这个报文所在的 TCP 流。基本上每次做抓包分析都会用到这一步的操作。



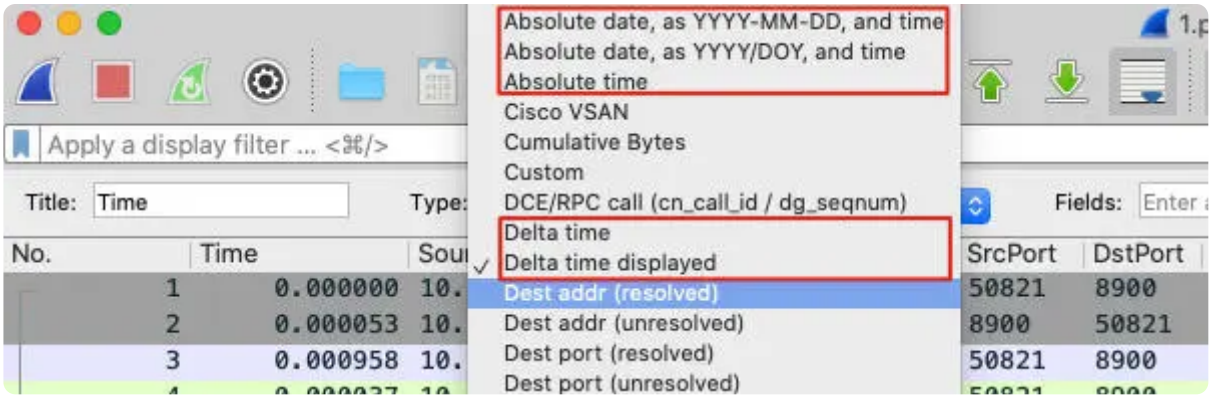
另外，Wireshark 默认展示的列经常不能满足我们的需求，所以我们也要学会**添加自定义列**的方法。简单来说，就是选择一个报文后，进入它的详情部分（界面下方），然后选中某一个属性，右单击，选择 Apply as Column 即可。比如下面这样：



上图中，我们通过添加 TTL 自定义列，就可以让每个报文的 TTL 值都在主视图中展现，极大地方便了对这些 TTL 的比较。所以我们除了掌握协议知识以外，也要挖掘各类工具的使用技巧。所谓“工欲善其事，必先利其器”也。

另外，在排查中，**时间**这个信息也是十分重要的。根据场景的不同，我们需要不同的 Time 列的表示方式。比如：

> 当我们关注**前后报文间的间隔时间**时，我们会选择 Delta time 或者 Delta time displayed。
>
> 当我们关注**跨度比较大的报文间隔**时，最好选择 Absolute time 或者另外两个 Absolute date 的形式。



对 TCP 提示的解读

Wireshark 很方便也很强大，但如果我们脱离了对网络协议的理解，那其实还是无从下手。比如我们做得最多的 TCP 排查，就需要结合自己对 TCP 的掌握才能做好解读。

举个例子。同样是 Wireshark 关于 TCP 窗口方面的提示，TCP Window Full 跟 TCP Zero Window 都是什么含义，有什么差别呢？像这样的例子很多，如果你真正地从 Wireshark 给你的众多信息中获取了有效的线索，那你的排查能力必定有一个明显的提升了。

TCP Window 相关的信息

就 TCP Window Full 而言，它的意思是**一端的在途字节数等于另一端的接收窗口**。Wireshark 会根据发送出去的数据量和已经确认的数量，计算出在途字节数，然后跟另一端的接收窗口进行比较。如果两者相等，说明另一端的接收窗口已经被用满了。这个对于排查 TCP 传输速度相关的问题是很有帮助的。

然后是 TCP Zero Window，这个比较简单，就是**接收窗口为零**。这是主动通告对端“自己的接收窗口已经为零，不要再发送数据了”。所以你能看出来它跟 TCP Window Full 的区别了吗？

它们的**区别**是：TCP Zero Window 是一个体现在 TCP 报文里的信息，算是主动“投降”。而 TCP Window Full 不是在报文中的信息，它是 Wireshark 自己分析出来的结论。知道了两者的区别，相信你也知道在相应的场景下，该如何结合这种解读来帮助排查了。

TCP 传输状况信息

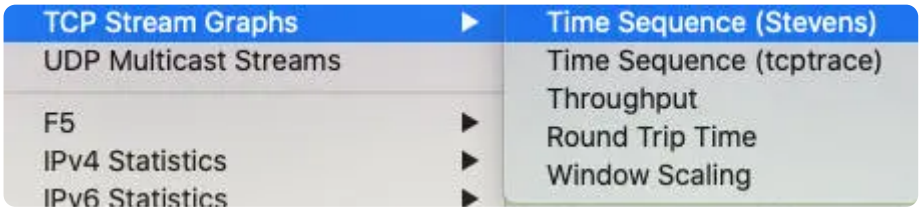
关于 TCP 传输速度也是一个热门话题，我们在第 9~11 讲都做了比较深入的探讨。在 Wireshark 里，我们可以利用下面几个图表来帮助排查。

I/O Graph：这个可以看到 IP 报文级别的传输速度。

TCP Stream Graphs：这里又分了几种子图表，分别是：

- Time Sequence (Stevens)：查看 TCP 序列号随着时间的变化趋势。
- Throughput：查看传输数据量的变化趋势。
- Round Trip Time：查看传输中 RTT 的变化。
- Window Scaling：查看接收窗口在传输中的变化，还可以直观地找到 TCP Window Full 事件。

你可以在 Statistics 下拉菜单中找到上面的这些工具。



Wireshark 过滤器

过滤器这一块也是很丰富且很有价值的内容。有时候，能否写出一个高效准确的过滤器，几乎可以当做是区分我们抓包分析水准的量尺，所以对过滤器再重视也不为过。下面我们来回顾几个典型的过滤器。

在**匹配字符串**的时候，我们可以有以下这样的好几种选择，分别在不同的分层上实现了文本过滤。

```
1 tcp contains "abc"
2 ip contains "abc"
3 http contains "abc"
```

那如果要**匹配二进制数据**该如何做到呢？我们知道，每个字节是 8 位，Wireshark 用两个十六进制数字表示一个字节，我们在每个字节之间加上破折号就可以了。比如下面这样也可以搜到带“abc”的报文：

```
1 tcp.payload contains 61-62-63
```

当我们要根据 TCP 报文的标志位进行过滤的时候，就可以用下面这种过滤器：

```
1 tcp.flags.reset eq 1      #找到RST报文
2 tcp.flags.syn eq 1        #找到握手的SYN报文
3 tcp.flags.fin eq 1        #找到挥手的FIN报文
```

我们还可以根据时间匹配来过滤报文，比如：

```
1 frame.time >="Mar 28, 2022 00:00:00"
```

tcp.analysis 类过滤器

此外，Wireshark 的过滤器中还有一个非常重要的部分，是 TCP 分析器 tcp.analysis，当你在 Wiresahrk 过滤器栏输入 tcp.analysis 后，就自动有很多的过滤器提示出来。这些过滤器经常能起到很大的帮助，因为它们大多不是直接的报文内容，而是从报文中推导出来的信息，已经包含了一定的分析价值。

比如下面这个过滤器，可以找到超过 200ms 才发回的确认报文：

```
1 tcp.analysis.ack_rtt >0.2 and tcp.len == 0
```

复制代码

当我们要找到所有的 TCP Window Full 的报文的时候，就可以输入：

```
1 tcp.analysis.window_full
```

复制代码

更多的 tcp.analysis 过滤器你可以自己去 Wireshark 里摸索一下，相信你会有很大的收获。

排查技巧

其实，网络排查很多人都在做，也都会抓包，那为什么不是每个人都可以从抓包文件中分析出有效的信息来呢？可能关键就在于我课程里提到过的**两大鸿沟**。

> 应用现象跟网络现象之间的鸿沟：你可能看得懂应用层的日志，但是不知道网络上具体发生了什么。
>
> 工具提示跟协议理解之间的鸿沟：你看得懂 Wireshark、tcpdump 这类工具的输出信息的含义，但就是无法真正地把它们跟你对协议的理解对应起来。

为了突破第一个鸿沟，我们需要熟悉抓包技术，知道什么时候抓包、抓哪些包，然后也有能力把抓包文件中的信息，跟应用层日志中的信息对应起来。

为了突破第二个鸿沟，我们需要深刻理解网络协议，知道 TCP、IP、HTTP 这些协议的行为和脾性，然后结合 Wireshark 的各种技巧，把网络协议跟实际现象结合起来，从而推进排查工作。

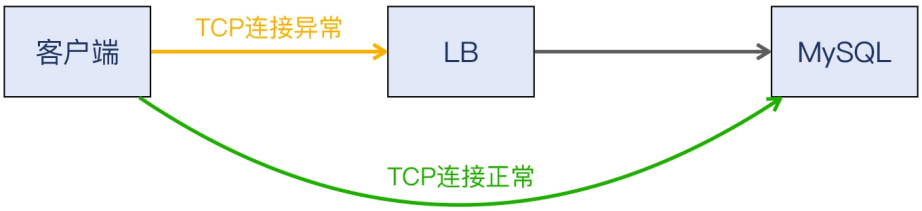
当然，填平这两个鸿沟只是我们的目标，而手段就是这里要讨论的排查技巧了。这里的内容也比较多，我选了几个典型场景和思路，希望对你有所启发。

逐段测试法

我们经常会遇到一个请求从开始到结束会经过很多环节的情况，比如：

> 客户端 -> 第四层 LB -> 第七层 LB -> 服务端

在这种时候，我们可以做逐段测试，比如从客户端直接访问服务端，这样可以很快地确定中间这两个环节是否跟问题有直接关系了。比如像下面这样：



极客时间

抓包分析思路

在进入抓包分析阶段后，从经验性的排查思路这个角度来说，一般的抓包分析可以用我们在第 5 讲“定位防火墙（一）”中提到过的步骤，也就是：

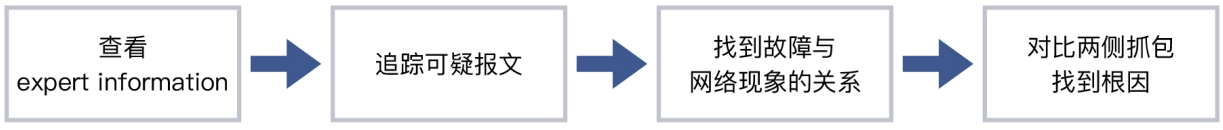
查看 Expert Information；

重点关注可疑报文（比如 Warining 级别），追踪其整个 TCP 流；

深入分析这个可疑 TCP 流的第二到四层的报文，再结合应用层表象，综合分析其根因；

结合两侧的抓包文件，找到属于同一个 TCP 流的数据包，对比分析，得出结论。

下图就概括了这个过程，你可以来参考一下：



极客时间

整数值

在有整数值出现的时候，我们需要提起精神，务必重点调查这个整数值背后的原因。因为一般来说，这往往意味着**有人为的设置**在里面，比如某种客户端超时设置。这些设置经常会跟问题有直接或间接的关系。

比如在第 15 讲，我们就发现客户端有 5 秒超时的机制，并在 Wireshark 里体现了这一点。

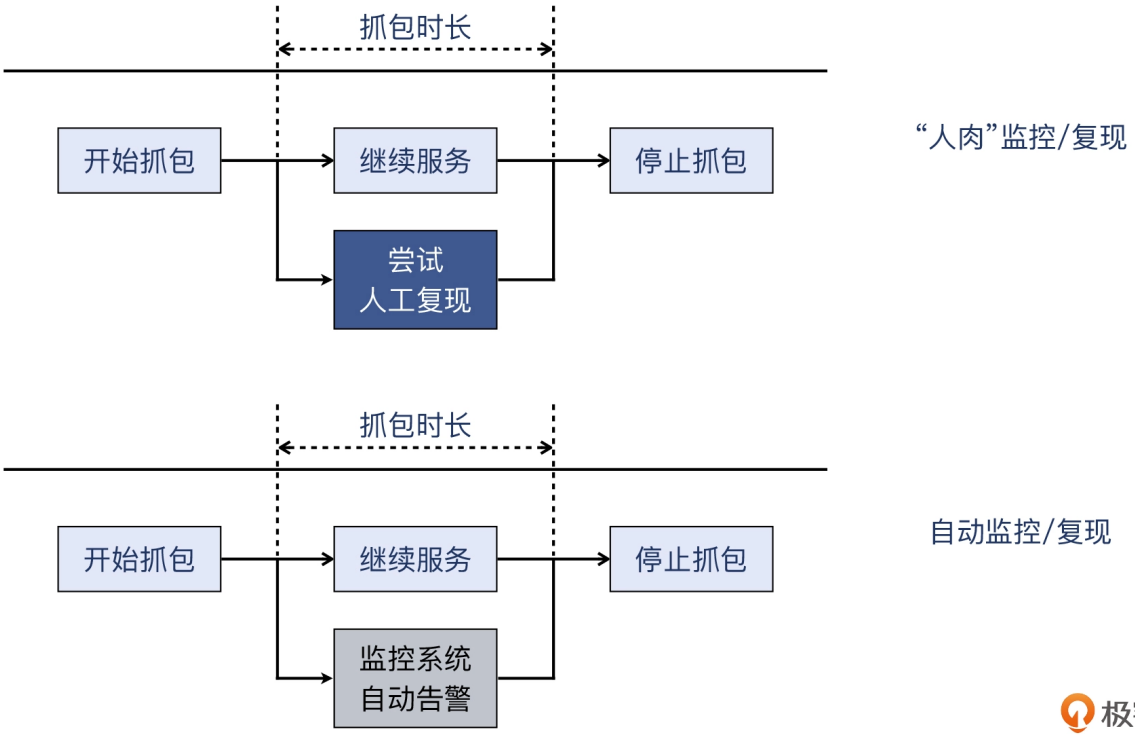
No.	Time	Source	Destination	SrcPort	DstPort	TTL	TCP Seglen	Frame length	Fr
1	0.000000	47.103.47.87	27.125.239.114	14492	80	47	0	66	
2	0.000014	27.125.239.114	47.103.47.87	80	14492	64	0	66	
3	0.030001	47.103.47.87	27.125.239.114	14492	80	47	0	60	
4	2.997629	47.103.47.87	27.125.239.114	14492	80	47	308	362	
5 5秒	0.000081	27.125.239.114	47.103.47.87	80	14492	64	0	54	
6	2.000016	47.103.47.87	27.125.239.114	14492	80	47	0	60	
7	0.000012	27.125.239.114	47.103.47.87	80	14492	64	0	66	
8	16.028791	47.103.47.87	27.125.239.114	14492	80	47	468	522	
9	0.000010	27.125.239.114	47.103.47.87	80	14492	64	0	54	
10	0.000538	27.125.239.114	47.103.47.87	80	14492	64	68	122	
11	0.030135	47.103.47.87	27.125.239.114	14492	80	47	0	60	

偶发性问题

对于偶发性问题，我们可以采用这样的策略：

> 预估 -> 开始抓包和观察 -> 问题重现 -> 停止抓包 -> 分析

下图概括了这个过程，供你参考：



对比分析

还有一个很关键的技术是“对比分析”。我们做抓包分析，经常要在多处抓包后做对比分析。比如以下两种典型场景：

同一个 TCP 流在客户端和服务端的抓包文件；

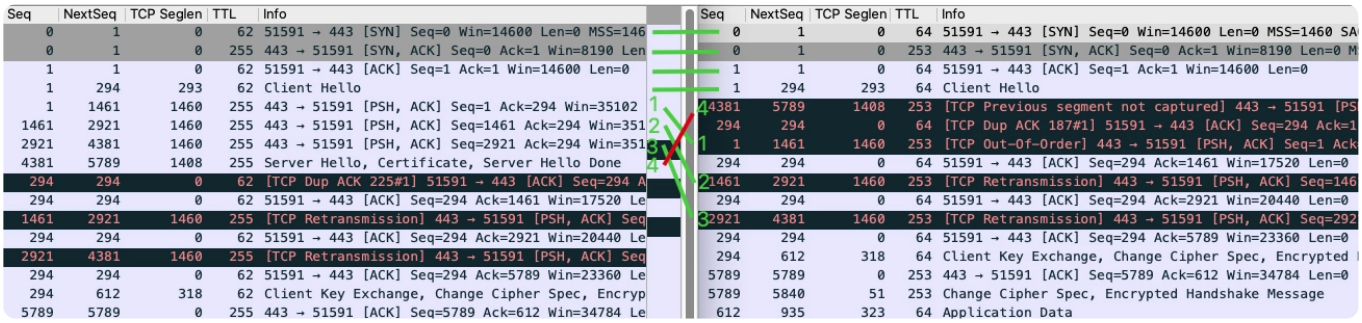
LB 前后不同 TCP 连接的抓包文件。

对于第一种场景，我们可以**用报文本身的字段**作为匹配条件，因为属于同一个 TCP 流，它们的这些属性字段值一定不会变化。比如用裸序列号（原始序列号），像下面的过滤器：

复制代码

```
1 tcp.seq_raw eq 123456
```

找到同一个 TCP 流之后，就可以把两个 Wireshark 窗口都打开，放在一起进行比较。你可以回顾下第 5 讲的案例，参考我们是如何做对比分析的。



Seq	NextSeq	TCP Seglen	TTL	Info
0	1	0	62	51591 → 443 [SYN] Seq=0 Win=14600 Len=0 MSS=146
0	1	0	255	443 → 51591 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len
1	1	0	62	51591 → 443 [ACK] Seq=1 Ack=1 Win=14600 Len=0
1	294	293	62	Client Hello
1	1461	1460	255	443 → 51591 [PSH, ACK] Seq=1 Ack=294 Win=35102
1461	2921	1460	255	443 → 51591 [PSH, ACK] Seq=1461 Ack=294 Win=351
2921	4381	1460	255	443 → 51591 [PSH, ACK] Seq=2921 Ack=294 Win=351
4381	5789	1408	255	Server Hello, Certificate, Server Hello Done
294	294	0	62	[TCP Dup ACK 225#1] 51591 → 443 [ACK] Seq=294 A
294	294	0	62	51591 → 443 [ACK] Seq=294 Ack=1461 Win=17520 Le
1461	2921	1460	255	[TCP Retransmission] 443 → 51591 [PSH, ACK] Seq
294	294	0	62	51591 → 443 [ACK] Seq=294 Ack=2921 Win=20440 Le
2921	4381	1460	255	[TCP Retransmission] 443 → 51591 [PSH, ACK] Seq
294	294	0	62	51591 → 443 [ACK] Seq=294 Ack=5789 Win=23360 Le
294	612	318	62	Client Key Exchange, Change Cipher Spec, Encryp
5789	5789	0	255	443 → 51591 [ACK] Seq=5789 Ack=612 Win=34784 Le

Seq	NextSeq	TCP Seglen	TTL	Info
0	1	0	64	51591 → 443 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SA
0	1	0	253	443 → 51591 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 M
1	1	0	64	51591 → 443 [ACK] Seq=1 Ack=1 Win=14600 Len=0
1	294	293	64	Client Hello
4381	5789	1408	253	[TCP Previous segment not captured] 443 → 51591 [PS
294	294	0	64	[TCP Dup ACK 187#1] 51591 → 443 [ACK] Seq=294 Ack=1
1	1461	1460	253	[TCP Out-Of-Order] 443 → 51591 [PSH, ACK] Seq=1 Ack
294	294	0	64	51591 → 443 [ACK] Seq=294 Ack=1461 Win=17520 Len=0
1461	2921	1460	253	[TCP Retransmission] 443 → 51591 [PSH, ACK] Seq=146
294	294	0	64	51591 → 443 [ACK] Seq=294 Ack=2921 Win=20440 Len=0
2921	4381	1460	253	[TCP Retransmission] 443 → 51591 [PSH, ACK] Seq=292
294	294	0	64	51591 → 443 [ACK] Seq=294 Ack=5789 Win=23360 Len=0
294	612	318	64	Client Key Exchange, Change Cipher Spec, Encrypted
5789	5789	0	253	443 → 51591 [ACK] Seq=5789 Ack=612 Win=34784 Len=0
5789	5840	51	253	Change Cipher Spec, Encrypted Handshake Message
612	935	323	64	Application Data

而对于第二种场景，TCP 连接就是完全不同的了，所以无法像上面第一种场景那样，把 TCP 报文里的字段值作为关联映射的条件。我们一般可以**寻找应用层的特征**，比如某个 uuid，然后运用 Wireshark 过滤器，在两侧不同 TCP 连接的抓包文件中，找到同样包含这个 uuid 的报文。比如这些过滤器：

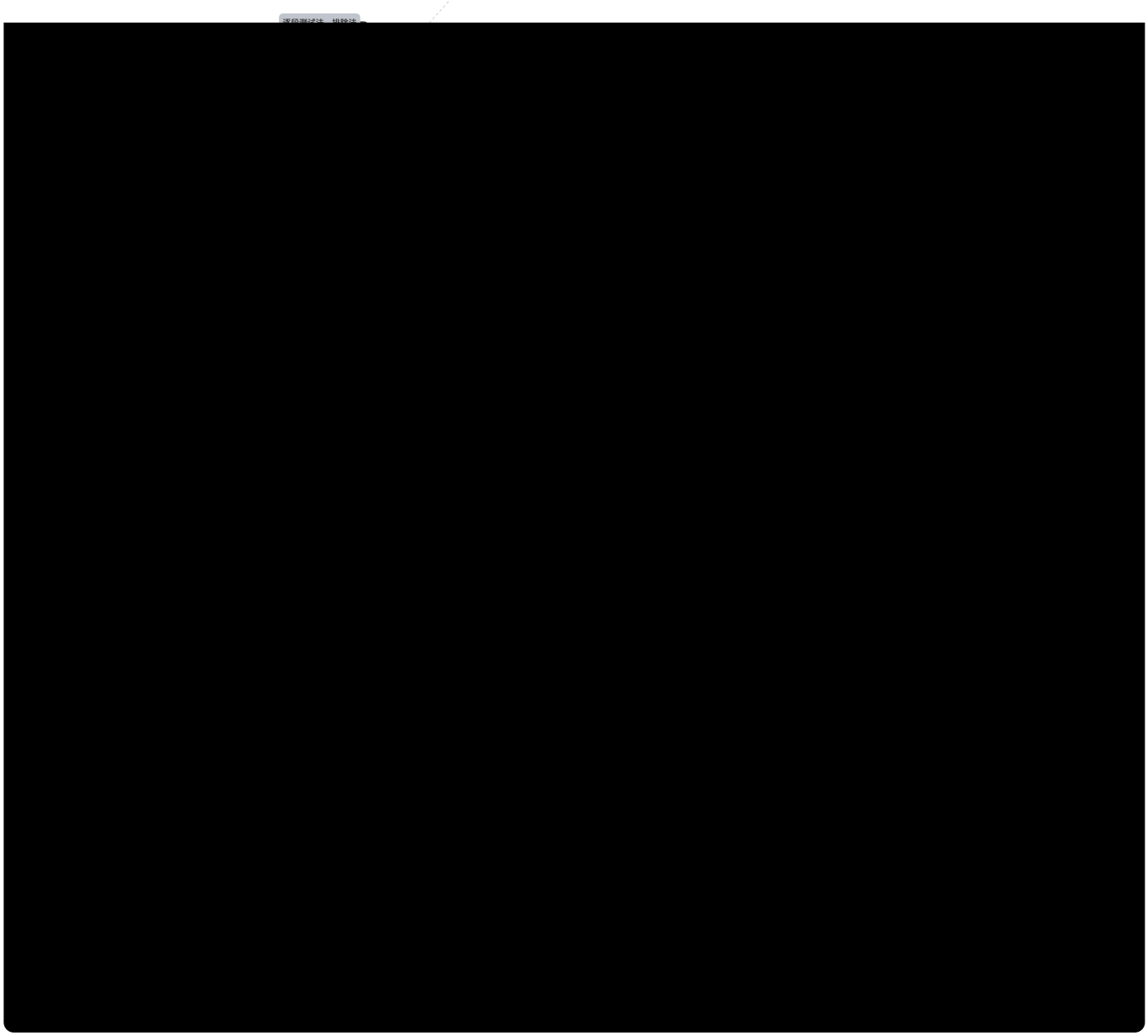
复制代码

```
1 http contains "uuidxxxxx"
2 tcp contains "uuidxxxxx"
3 ip contains "uuidxxxxx"
4 frame contains "uuidxxxxx"
```

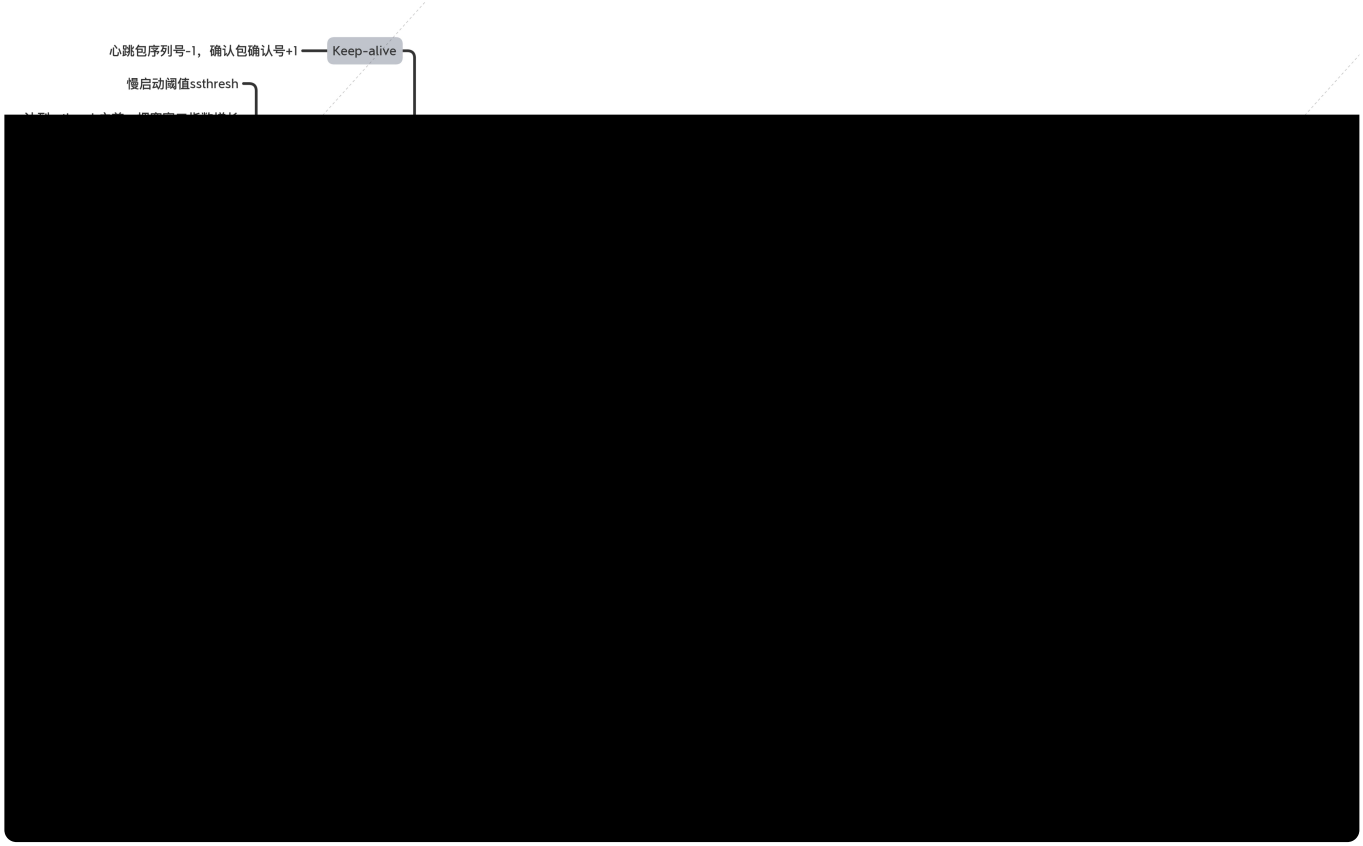
小结

这节课，我们是对整个课程中，不少抓包分析的技术细节进行了回顾。我在春节特别放送（三）里其实提到过，思维导图是一个很好的学习工具，所以我就汇总整理了专栏中的核心内容。因为内容也比较多，我分为了 4 个部分来分别展示，你可以通过它们快速找到你需要的知识点。

排查技术



协议



系统相关

抓包分析

思考题

给你留两道思考题：

> 假设通信的一端发送了 TCP Zero Window 报文，并且后续没有更多新的报文过来，那么另一端是不是一直不能发送新的数据了呢？
>
> 请写一个 tshark 命令，用来过滤出抓包文件中的 DupAck。

欢迎你把答案分享到留言区，我们一同进步、成长。

上一篇 用户故事 | 王未：网络排查能力是一名合格运维工程师的必备技能

下一篇

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。