实验室检查点 4：山顶（完整的 TCP）

10 月 25 日更新，改进了调试说明

到期的：11 月 12 日（周五），下午 5 点
最迟截止期限：11 月 14 日晚上 11:59（接收反馈的最后一天；风格评分上限为 2/3）

0 合作政策

编程作业必须是你自己的作品：您必须编写您提交的编程作业的所有代码，除了我们作为作业的一部分提供给您的代码。请不要从 Stack Overflow、GitHub 或其他来源复制粘贴代码。如果您根据在 Web 或其他地方找到的示例编写自己的代码，请在您提交的源代码的注释中引用 URL。

与他人合作：您不得向其他人展示您的代码、查看其他人的代码或查看前几年的解决方案。您可以与其他学生讨论作业，但不得抄袭任何人的代码。如果您与其他学生讨论作业，请在您提交的源代码的评论中注明他们的名字。请参阅课程管理讲义了解更多详细信息，如果有任何不清楚的地方，请向 Ed 询问。

埃德：请随意向 Ed 提问，但请不要发布任何源代码。

1 概述

您已到达顶峰。

在实验室 0 中，你实现了*佛罗里达州流控字节流*（字节流（ByteStream）。在实验室 1、2 和 3 中，您实现了以下工具：*双向*— 该抽象与互联网提供的抽象之间：不可靠的数据报。

现在，在实验室 4 中，你将制作一个称为TCP连接，结合你的TCP发送端和TCP接收端并负责处理连接的全局事务。连接的 TCP 段可以封装到用户（TCP-in-UDP）或 Internet（TCP/IP）数据报的有效负载中，从而使您的代码可以与 Internet 上数十亿台使用相同 TCP/IP 语言的其他计算机进行通信。图1再次展示了整体设计。

简短的警告：TCP连接*大多*只是结合了你在之前的实验中实现的发送器和接收器模块——TCP 连接本身可以在不到 100 行代码中实现。如果您的发送方和接收方很强大，这将是一个简短的实验。如果不是，您可能需要花时间调试，并借助测试失败消息。（我们不鼓励您尝试阅读测试源代码，除非这是最后的手段。）根据去年学生的双峰经验，我们强烈建议您早点开始直到截止日期前一天晚上才离开这个实验室。

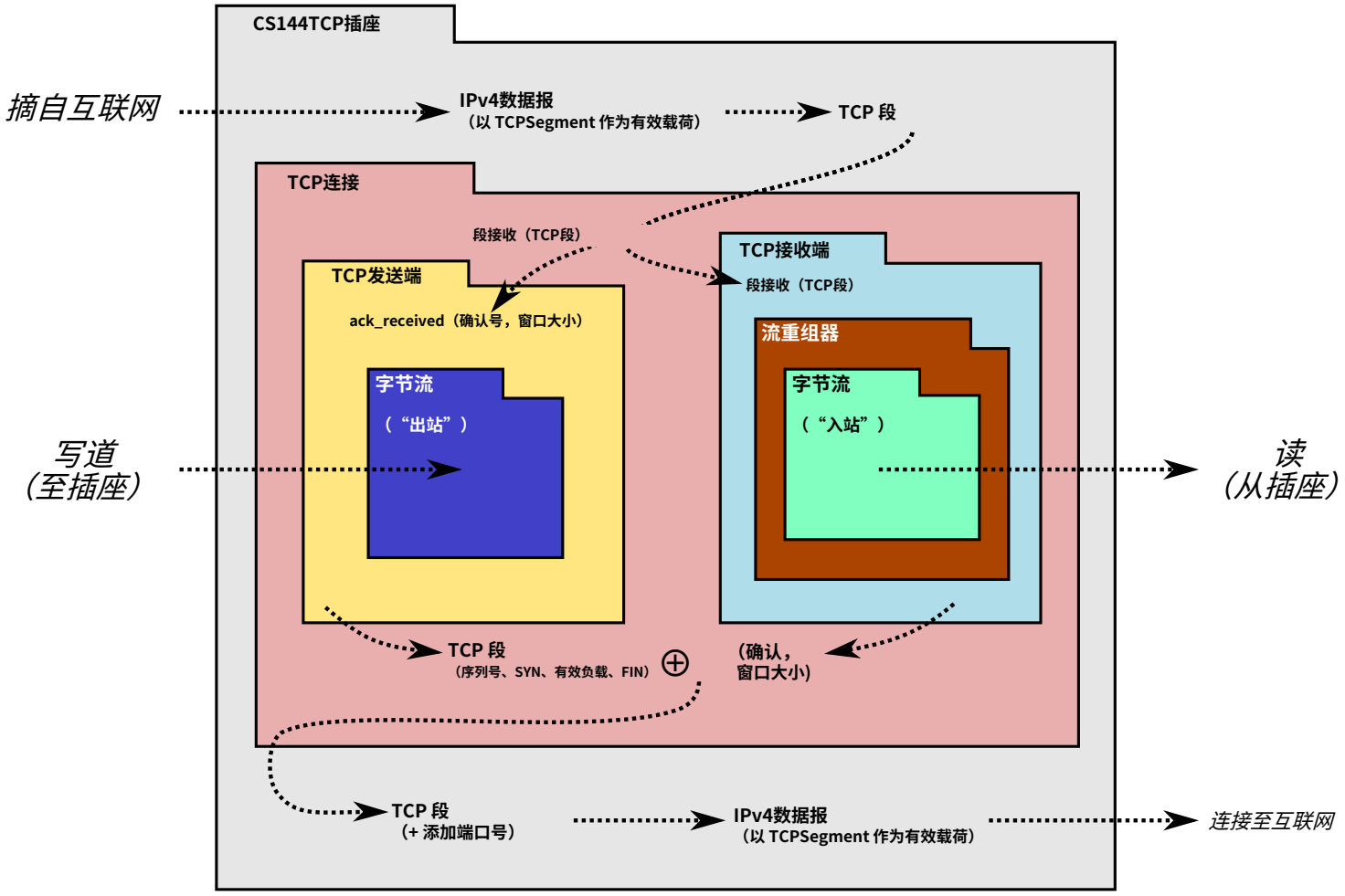


图 1：TCP 实现中的模块和数据流的排列。

2 入门

您实施的TCP连接将使用您在实验 0-3 中使用的相同 Sponge 库，并添加额外的类和测试。我们为您提供了支持代码，用于读取和写入 TCP 段到用户和 Internet 数据报的有效负载中。我们还为您提供了一个类 (CS144TCP套接字)包裹你的TCP连接并使其表现得像一个正常的流套接字，就像TCP套接字你曾经实现过网络获取回到实验 0。在本实验结束时，你将稍微修改你的网络获取使用*你的*TCP 实现—aCS144TCP插座而不是TCP套接字。开始吧：

1. 确保你已经提交了实验 3 的所有解决方案。请不要修改自由海绵目录，或webget.cc。否则，您可能无法合并 Lab 4 启动代码。

2. 在实验室作业存储库中，运行实验室作业的最新版本。 `git 获取` 检索

3. 运行以下命令下载实验 3 的起始代码 `git 合并 origin/lab4-startercode` 。

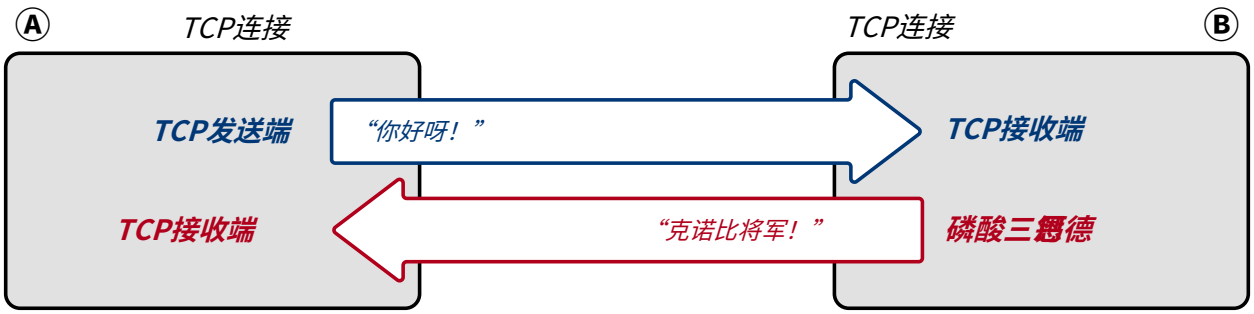
4. 在您 的建造目录，编译源代码： `制作` （例如，你可以运行 `制作-j4` 在编译时使用四个处理器）。

5. 外面建造目录，打开并开始编辑writeups/lab4.md fi这是您的实验室报告模板，将包含在您的提交内容中。

3 实验 4：TCP 连接

本周，您将完成构建一个有效的 TCP 实现。您已经完成了实现该目标的大部分工作：您已经实现了发送方和接收方。本周您的工作是将它们“连接”到一个对象（一个TCP连接)并处理一些与连接相关的全局性管理任务。

回想一下：TCP 可靠地传输一*对*流控字节流，每个方向一个。两方参与 TCP 连接，并且*每*一*方*同时充当“发送者”（其自己的出站字节流）和“接收者”（入站字节流）：



双方（上图中的“A”和“B”）称为连接的“端点”，或“对等体”。您的TCP连接充当一同侪。它负责接收和发送段，确保发送者和接收者了解并有机会做出贡献*他们关心的领域*针对传入和传出段。

以下是基本规则TCP连接必须遵循：

接收片段。如图所示1， 这TCP连接接收TCP 段来自互联网时已接收段方法被调用。发生这种情况时， TCP连接查看该片段并：

- 如果 第一 （重置）标志已设置，将入站和出站流都设置为错误状态并永久终止连接。否则它...
- 将段提供给TCP接收端因此它可以检查传入段中它关心的字段：序列号，同步，有效载荷，和 鳍
- 如果 确认 标志被设置，告诉TCP发送端关于它关心的传入字段段：确认和窗口大小。
- 如果传入的段占用了任何序列号，则TCP连接确保*至少一个*发送段作为答复，以反映确认号和窗口大小的更新。
- 还有一个额外的特殊情况，你必须处理TCP连接's 已接收段（）方法：响应“保持活动”段。对等端*可能* 选择发送一个带有无效序列号的段，以查看您的 TCP 实现是否仍然有效（如果有效，则查看当前窗口是什么）。您的 TCPConnection 应该回复这些“保持有效”即使它们不占用任何序列号。实现这一点的代码如下所示：

```
如果 (_receiver.ackno().has_value() 和 (seg.length_in_sequence_space() == 0)
      并且 seg.header().seqno == _receiver.ackno().value() - 1)
      { _sender.send_empty_segment();
}
```

正在发送段。这TCP连接将发送TCP 段通过互联网：

- 任何时间这TCP发送端已将一个段推送到其传出队列，并在传出段上设置其负责的字段： （序列号， 同步，有效载荷，和 鳍
- 在发送该段之前，TCP连接会问TCP接收端对于其负责传出段的字段：确认和窗口大小. 如果有确认， 它将设置 确认 国旗和在TCP段。

当时间流逝。这TCP连接有一个打钩该方法将由操作系统定期调用。当这种情况发生时，TCP连接需要：

- 告诉TCP发送端关于时间的流逝。
- 中止连接，并向对端发送一个重置段（一个空段，内容为
  第一 标志设置），如果连续重传次数超过上限
  限制TCPConfig::MAX RETX 尝试次数。
- 如果必要的话，彻底结束连接（请参阅第节5）。

因此，每个TCP 段看起来像这样，“发送方写入”和“接收方写入”字段以不同的颜色显示：

完整接口TCP连接是在课程文档中。请花一些时间阅读本文。很多你的实现将涉及“连接”公共 APITCP连接到适当的例程中TCP发送端和TCP接收器。 尽可能地，你希望将所有繁重的工作推迟到发送者和接收者身上，

你已经实施了。话虽如此，但事情不会那么简单，而且
涉及 t 的“全局”行为的一些微妙之处将决定何时完全 整体连接。最难的部分 离子并宣布其
终止TCP连接 不再“活跃”。

以下是一些常见问题和您需要处理的边缘情况的详细信息。

4 常见问题和特殊情况

- *你有多少代码 总体而 你前景胶嗯?*

  言，我们预计 埃我请 耳鼻喉科 （我碳p 连接.cc)将需要
  总共 100–150 行代码。当你用自己的实现实现互ne，测试套件将广泛测试Linux内核的
  操作性时
  TCP 的实现。

- *我该如何开始?*

也许最好的开始方式是将一些“普通”方法连接到适当的调用中TCP发送端和TCP接收器。这可能包括以下内容 剩余出站容量（）、正在传输的字节数（）、和未组装的字节（）。

然后您可以选择实现“writer”方法：连接（），写入（）， 和结束输入流()。其中一些方法可能需要对出站进行一些操作字节流（拥有TCP发送方）和告诉TCP发送端关于它。

您可以选择开始运行测试套件（ 进行检查 ）在你完全实施每种方法；测试失败消息可以为您提供下一步解决的线索或指南。

- *如何申请读来自入站流?*

  TCPConnection::入站流()已在头文件中实现。您无需执行任何其他操作即可支持应用程序读取。

- *是否*TCP连接*需要任何奇特的数据结构或算法吗?*

  不，确实不是。繁重的工作都是由TCP发送端和TCP接收端 您已经实现了。这里的工作实际上只是将所有东西连接起来，并处理一些挥之不去的连接范围的细微差别，这些细微差别无法轻易被发送者和接收者考虑到。

- *如何*TCP连接*实际发送了一个段?*

  类似于TCP发送方—将其推到分段队列。至于你的 TCP连接就而言，只要将其推送到此队列，就认为它已发送。很快所有者就会过来将其弹出（使用公共段输出()我使用访问器方法 (accessor method) 来真正发送它。

- *如何*TCP连接*了解时间的流逝?*

  类似于TCP发送方—这打钩（）方法将被定期调用。请不要使用任何其他方式来表示时间——tick 方法是您了解时间流逝的唯一途径。这可确保事物的确定性和可测试性。

- *什么是*TCP连接*如果传入的段有* 第一 *设置了标志吗?*

  此标志（“重置”）表示立即终止连接。如果您收到带有 的段，则应在入站和出站上设置错误标志字节流s，以及任何后续调用TCP连接::active()应该返回 false。

- *我什么时候应该发送一个片段*第一佛罗里达州农业设置*?*

  有两种情况您需要中止整个连接：

  1. 如果发送方连续重传次数过多而没有成功（超过TCPConfig::最大重新发送尝试次数，即8)。
  2. 如果TCP连接当连接仍处于活动状态时调用析构函数 （积极的（）返回 true)。

发送一个段 第一 设置与接收效果类似：连接是
已不再存在积极的（），并且两者字节流s 应该设置为错误状态。

- *等一下，但是我怎样才能创建一个可以设置序列号的片段 第一 旗帜? 呢?*

  任何传出段都需要有正确的序列号。您可以强制 TCP发送端通过调用其发送空段（）方法。或者你可以让它填满窗口（生成段*如果*它有未完成的信息要发送，例如来自流的字节或SYN/ FIN）通过调用其填充窗口()方法。

- *的目的是什么 确认 国旗? 确认?*

  几乎每一个TCP 段有一个确认，并且具有 确认 标志设置。例外情况正处于连接刚开始的时候，接收方还没有确认任何信息。

  在传出部分，你需要设置确认和 确认 标记时-永远有可能。也就是说，TCP接收方's确认()方法返回一个std::optional<WrappingInt32>有一个值，你可以用它来测试有值()。

  在传入段上，你会想看看确认只有当 确认 字段是集。如果是，则给出确认（和窗口大小）到TCP发送方。

- *我如何解读这些“状态”名称（例如“流已开始”或“流正在进行”）？* 请参阅实验室 2 和实验室 3 讲义中的图表。

  我们想再次强调，“状态”对于测试和调试很有用，但是 我们并不要求您在代码中实现这些状态。您无需创建更多状态变量来跟踪这一点。“状态”只是您的模块已公开的公共接口的一个函数。

- *如果*TCP接收端*想要宣传一个比适合的更大的窗口尺寸*TCPSegment::header().win*菲埃尔德?*

  发送最大的价值。你可能会发现std::numeric 限制课堂很有帮助。

- *TCP 连接什么时候才算“完成”？ 什么时候可以*积极的（）*返回 false?*请参阅下一节。

- *此 PDF 版本发布后，我可以在哪里阅读更多常见问题解答?*

  请查看网站（https://cs144.github.io/lab常见问题.html) 和 Ed 经常见面。

5 TCP 连接的结束：达成共识需要努力

的一项重要功能TCP连接决定 TCP 连接何时完全“完成”。当这种情况发生时，实现将释放其对本地

端口号，停止发送确认以回复传入的段，认为连接已成为历史，并且积极的（）方法返回 false。

有两种方式可以结束一段关系。不正常关机，这TCP连接 发送或接收一个段 第一 标志已设置。在这种情况下，出站和入站字节流s 应该都在错误州和积极的（）可以立即返回 false。

一个干净关机是我们实现“完成”的方法（活跃（）=false），而不会出现错误。这更复杂，但它是一件美妙的事情，因为它尽可能地确保*每个*两者之中字节流已可靠交付*完全地*到接收方。在下一节中（§ §5.1），我们给出了干净关机时的实际结果，因此如果您愿意，可以随意跳过。

太棒了，你还在这里。因为两位将军的问题， 它是*无法保证* 两个对等体都可以实现干净的关闭，但 TCP 非常接近。下面是具体方法。从一方（一方）的角度来看TCP连接，（我们将其称为“本地”对等体）与“远程”对等体之间的连接完全关闭需要四个先决条件：

先决条件 #1这入站流已完全组装并结束。

先决条件 #2这出站本地应用程序已结束流*和*全部发送（包括 事实是它已经结束，即与 的段）到远程对等 。

先决条件 #3这出站流已被远程对等方完全确认。

先决条件 #4这当地的TCP连接相信偏僻的对方可以满足先决条件 # 3. 这是最费脑力的部分。有两种方法可以实现这一点：

- 选项 A：两个流结束后仍逗留。先决条件 1 至 3 均成立，并且远程对等体*似乎*已获得本地对等端对整个流的确认。本地对等端对此并不确定 — TCP 无法可靠地传递确认（它不会确认确认）。但本地对等端非常确信远程对等端已收到其确认，因为远程对等端似乎没有重新传输任何内容，并且本地对等端已等待一段时间以确保这一点。

  具体来说，当满足先决条件 #1 到 #3 时，连接就建立了，并且它至少是初始重传超时的 10 倍（cfg.rt 超时）由于本地对等端已收到*任何*来自远程对等点的段。这称为“延迟”，在两个流完成后，以确保远程对等方不会尝试重新传输我们需要确认的任何内容。这意味着TCP连接需要活一段时间，[1]保持对本地端口号的独占声明，并可能发送确认作为响应

[1]在生产 TCP 实现中，延迟计时器（也称为等待时间定时器或最大段生存期 (MSL) 的两倍）通常为 60 或 120 秒。在连接有效完成后，保留端口号的时间可能很长，尤其是当您要启动绑定到同一端口号的新服务器时——没有人愿意等待两分钟。因此重复使用地址socket 选项本质上使得 Linux 忽略保留，并且可以方便地进行调试或测试。

传入段，即使在TCP发送端和TCP接收端已完全完成工作并且两个流程均已结束。

- 选项B：被动关闭。先决条件 1 到 3 为真，本地对等端 100% 确定远程对等端可以满足先决条件 3。如果 TCP 不确认确认，这怎么可能呢？因为远程对等端第一个结束其流的。

> *?这条规则为什么有效?* 这是个脑筋急转弯，你不需要继续阅读来完成这个实验，但思考一下很有趣，并且可以深入了解两将军问题的深层原因以及不可靠网络中可靠性的固有约束。这种方法之所以有效，是因为 *后*接收并组装远程对等体的
>
> ⌈鳘⌉（先决条件 #1），
> 本地对等体发送了一个序列号比它以前发送的序列号更大的段（至少，它必须发送自己的 ⌈鳘⌉ 满足的段
> 先决条件 #2)，并且该部分也有一个确认确认远程对等体的
> ⌈鳘⌉ 位。远程对等端确认该段（至
> 满足先决条件 #3)，这意味着远程对等体*还必须看到本地对等端对远程对等端的确认* ⌈鳘⌉. 这保证了
> 这远程对等体必须能够满足自己的先决条件 #3。所有这些意味着本地对等体无需停留即可满足先决条件 #4。
>
> 呼！我们说这是个烧脑的难题。*实验室报告中添加额外加分：你能找到更好的方法来解释这一点吗?*

底线是如果TCP连接's 入站流在TCP连接曾经发送过
⌈鳘⌉ 段，然后TCP连接
两个流结束后无需逗留。

5.1 TCP连接的结束（实用总结）

实际上，这一切意味着你的TCP连接有一个成员变量叫做 溪流结束后依然徘徊，通过暴露于 _测试仪器状态 _() 方法。变量开始于真的。如果入站流在TCP连接 在其出站流上已达到 EOF，此变量需要设置为错误的。

只要满足先决条件 1 至 3，连接就“完成”（并且 积极的（）应该返回错误的）如果流结束后徘徊是错误的。否则你需要逗留：_连接只有在足够的时间后才会完成（10 ×cfg.rt 超时）自收到最后一段以来已经过去了。 _ _

6 测试

除了自动化测试之外，我们还鼓励你“尝试”一下你的 TCP 实现，并且使用 wireshark 进行调试。以下是一些说明：

以下是手动运行的示例。您需要打开两个窗口，均位于 sponge/build 目录中。

在一个窗口中运行： `./应用程序/tcp ipv4 -l 169.254.144.9 9090`

这将使您的 TCPConnection 作为服务器运行（其中数据段进入 Internet（IPv4）数据报），监听本地地址（169.254.144.9）和端口 9090。您应该看到：

```
调试：正在监听传入连接...
```

现在你可能想要开始捕捉发送和接收，以便您稍后检查
在 wireshark 中查看它们。在另一个窗口中，运行 `sudo tshark -Pw /tmp/debug.raw -i tun144` 。[2]
这将捕获服务器发送或接收的所有 TCP 段，并将它们保存到名为“/tmp/debug.raw”的文件中。稍后您可以使用 Wireshark（在您的计算机上，不一定在 VM 内）打开此文件以查看每个段的完整详细信息。

在第三个窗口中，运行客户端： `./应用程序/tcp ipv4 -d tun145 -a 169.254.145.9 169.254.144.9 9090`

这将使您的 TCPConnection 作为客户端运行，并连接到服务器正在监听的地址。

在服务器窗口中，您现在应该看到类似以下内容：

```
来自 169.254.144.1:52518 的新连接。
```

在客户端窗口中您现在应该看到：

```
成功连接到 169.254.144.9:9090。
```

现在尝试在任一窗口（客户端或服务器）中输入一些内容并按 ENTER。您应该会看到相同的文本出现在另一个窗口中。

现在尝试结束其中一个流。在客户端窗口中，键入 Ctrl-D。这将结束客户端的出站流。现在客户端窗口应该如下所示：

[2] 您可能需要安装此程序， `sudo apt 安装 tshark` 。

DEBUG：到 169.254.144.9:9090 的出站流已完成（仍有 1 个字节在传输中）。DEBUG：到 169.254.144.9:9090 的出站流已完全确认。

服务器窗口应该包含如下内容：

调试：来自 169.254.144.1:52518 的入站流已干净完成。

最后，在服务器窗口中，再次键入 Ctrl-D 来结束该方向的流。

服务器窗口现在应该打印类似下面的内容并立即返回到命令行（不会停留）：

DEBUG：等待干净关闭... DEBUG：到 169.254.144.1:52518 的出站流已完成 DEBUG：到 169.254.144.1:52518 的出站流已完全确认。
调试：TCP 连接已干净完成。完成。

客户端窗口应该打印如下内容：

调试：来自 169.254.144.9:9090 的入站流已干净完成。
调试：等待来自对等方的延迟段（例如 FIN 的重新传输）...调试：等待干净关闭...

等待 10 秒后，客户端窗口应该会打印其余内容并返回到命令行：

调试：TCP 连接已干净完成。完成。

如果其中某个步骤出错，则表明需要检查终止逻辑（决定何时停止报告 active()=true）。或者，请随时在此处再次发帖，我们可以尝试帮助您进一步调试。

使用小窗口进行测试：如果您担心 TCPSender 在接收方通告零窗口情况时会卡住，请尝试运行上述命令并对 tcp ipv4 程序使用“-w 1”参数。这将使其使用 TCPReceiver 容量 1，这意味着当您在一侧输入“hello”时，只能发送一个字节，然后接收方将通告零窗口。

确保您的实现不会卡在那里！即使接收器容量为 1 个字节，它也应该能够成功发送任意长度的字符串（例如“你好，你好吗”）。

7 性能

完成 TCP 实现并通过所有测试后
进行检查 lab4，请提交！然后，测量系统的性能并
将其提高到至少每秒 100 兆比特。

从建造目录中，运行如下命令：./apps/tcp 基准测试。如果一切顺利，你会看到输出

```
用户@电脑:~/sponge/build$ CPU ./apps/tcp 基准测试
限制的吞吐量                                    ：1.78 千兆位/秒
重新排序后 CPU 限制吞吐量：1.21 Gbit/s
```

要想在实验室中获得满分，你的性能在两条线路上至少需要达到“0.10 Gbit/s”（100 兆比特每秒）。你可能需要分析你的代码，或者找出代码运行缓慢的原因，也可能需要改进某些关键模块的实现（例如，字节流或者流重组器达到这一点。

在你的文章中，请报告您所达到的速度数据（重新排序和不重新排序）。

如果你愿意，你可以尝试尽可能地优化你的代码，但是 *请不要以牺牲 CS144 的其他部分为代价来做到这一点*，包括本实验室的其他部分。我们不会为超过 100 Mbit/s 的性能给予额外分数——您在此最低限度之外所做的任何改进都只是为了您自己的满足和学习。如果您实现了比我们更快的实现[3]在不改变任何公共接口的情况下，我们很乐意向您学习您是如何做到的。

8 重温 webget

是时候庆祝胜利了！记住你的网络获取你在实验 0 中写的内容？它使用 TCP 实现（TCP套接字Linux 内核提供。我们希望您将其切换为使用您自己的 TCP 实现，而无需更改任何其他内容。我们认为您需要做的就是：

- 代替 # 包括“socket.hh” 和 # 包括“tcp 海绵套接字.hh”。
- 更换 TCP套接字 键入 CS144TCP插座。
- 在你的获取 URL()函数，添加调用 socket.wait 直到关闭()。

[3]我们在 2011 Intel Core i7-2600K CPU @ 4.40GHz 上运行了我们的参考实现，操作系统为 Ubuntu 19.04、Linux 5.0.0-31-generic #33-Ubuntu（针对 Meltdown/Spectre/等的默认缓解措施）和 g++ 8.3.0（针对默认（“发布”）版本的默认编译器标志）。CPU 限制吞吐量（第一行）为 7.18 Gbit/s，（第二行）重新排序后为 6.84 Gbit/s。

> *“我为什么要这么做？”* 通常，Linux 内核负责等待 TCP 连接达到“干净关闭”状态（并放弃其端口预留），即使用户进程已经退出。但由于您的 TCP 实现全部在用户空间中，因此除了您的程序之外，没有其他程序可以跟踪连接状态。添加此调用会使套接字等待，直到您的TCP连接 报告活跃（）=错误的。

重新编译并运行 `检查 webget` 确认你已经完成了整个过程：你已经编写了一个基本的 Web 获取器*您自己的完整 TCP 实现*，而且它仍然 成功与真实网络服务器对话。如果你有 卢布，尝试手动运行该程序：`./apps/webget cs144.keithw.org/hasher/xyzzy`. 您将获得一些调试输出可能有帮助的终端。

9 开发和调试建议

1. 实施TCP连接'文件中的公共接口（以及您想要的任何私有方法或函数）tcp 连接.cc。您可以将任何您喜欢的私人成员添加到TCP连接上课tcp 连接.hh。

2. 我们预计总共需要大约 100–150 行代码。您不需要任何花哨的数据结构或算法；

3. 您可以使用以下代码测试代码（编译后） `进行检查`. 这将运行相当综合测试套件（159 项测试）。许多测试都证实，您的 TCP 实现可以在各个方向上的数据包丢失和数据传输的各种组合下，与 Linux 的 TCP 实现或自身一起无错误地传输文件。[4]

4. 请重新阅读 Lab 0 文档中有关“使用 Git”的部分，并在线常见问题解答，并记住将代码保存在 Git 存储库中，它分发在掌握分支。进行小规模的提交，使用良好的提交消息来识别更改的内容及其原因。

5. 请努力使代码易于 CA 理解，CA 将根据代码风格对其进行评分。使用合理、清晰的变量命名约定。使用注释来解释复杂或微妙的代码片段。使用“防御性编程”——明确检查函数或不变量的先决条件，如果出现任何问题，则抛出异常。在设计中使用模块化——识别常见的抽象和行为

[4] 在测试名称中，“c”表示你的代码是客户端（发送第一个 `同步`），“s”表示您的代码是服务器。字母“u”表示正在测试 TCP-over-UDP，“i”表示正在测试 TCP-over-IP (TCP/IP)。字母“n”表示正在尝试与 Linux 的 TCP 实现进行互操作。“S”表示您的代码正在发送数据；“R”表示您的代码正在接收数据，“D”表示数据正在双向发送。在测试名称的末尾，小写的“l”表示接收（传入段）方向有数据包丢失，大写的“L”表示发送（传出段）方向有数据包丢失。

并尽可能将它们分解出来。重复的代码块和庞大的函数会使代码难以理解。

6. 请同时遵守 Lab 0 文档中描述的“现代 C++”风格。cppreference 网站 (https://en.cppreference.com）是一个很好的资源，尽管你不需要任何复杂的 C++ 功能来完成这些实验。（有时你可能需要使用移动（）函数传递一个不可复制的对象。）

7. 如果遇到分段错误，那一定出了问题！我们希望您能够使用安全的编程实践来编写代码，从而让分段错误变得极为罕见（不会malloc()，不新的，没有指针，安全检查会在你不确定的地方抛出异常，等等）。也就是说，为了调试，你可以使用以下命令配置你的构建目录：

   `cmake .. -DCMAKE 构建类型=RelASan` 启用编译器的“清理器”检测内存错误和未定义行为，并在发生时为您提供良好的诊断。您还可以使用瓦尔格林德工具。您还可以使用

   `cmake .. -DCMAKE 构建类型=调试` 并使用 GNU 调试器（您可以使用 gdb 命令来配置 .记住仅使用这些设置进行调试——它们会大大降低程序的编译和执行速度。恢复到“发布”模式的最可靠/最万无一失的方法是删除建造目录并创建一个新目录。

10 提交

1. 在您提交的作品中，请仅对 进行更改。时和 。cc fi位于顶层的libsponge。在这些文件中，请根据需要随意添加私有成员，但请不要更改*民众*任何类的接口。

2. 在提交任何作业之前，请按顺序运行以下内容：

   （一）`制作格式`（规范化编码风格）
   （二）`git 状态`（检查未提交的更改 - 如果有，请提交！）
   （三）`制作`（确保代码可以编译）
   （四）`进行检查`（确保自动化测试通过）

3. 撰写报告writeups/lab4.md。该文件应为大约 20 到 50 行的文档，每行不超过 80 个字符，以方便阅读。报告应包含以下部分：

   （一个）程序结构和设计。描述代码中体现的高级结构和设计选择。您无需详细讨论从起始代码中继承的内容。利用这个机会突出重要的设计方面，并更详细地介绍这些方面，以便您的评分助教理解。强烈建议您使用小标题和大纲使本文尽可能易于阅读。请不要简单地将您的程序翻译成一段英文。

（二）实施挑战。描述您发现最麻烦的代码部分并解释原因。反思您如何克服这些挑战以及是什么帮助您最终理解了让您感到困难的概念。您如何尝试确保您的代码保持您的假设、不变量和先决条件，以及在哪些方面您觉得这很容易或很难？您如何调试和测试您的代码？

（三）餘下的错误。尽可能指出并解释代码中存在的任何错误（或未处理的边缘情况）。

4. 请填写完成该作业所花费的小时数以及其他评论。

5. 准备提交时，请按照以下说明操作https://cs144.github.io/提交。提交前请确保您已完成所有想要完成的工作。

6. 如果在周二晚上的实验课上遇到任何问题，请尽快告知课程工作人员，或者在 Ed 上发布问题。祝你好运！