

## 实验室检查点 3: TCP 发送方

到期的: 10 月 22 日星期五下午 5 点

最迟截止日期: 9 月 24 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。

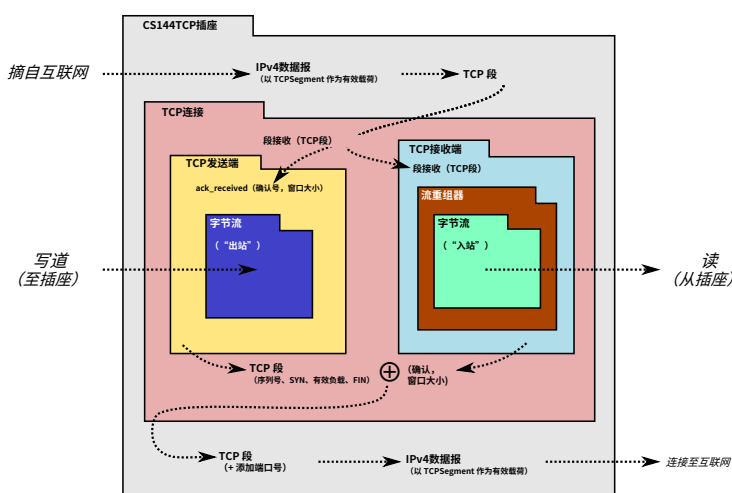
埃德: 请随意向 Ed 提问, 但请不要发布任何源代码。

## 1 概述

*建议: 实施前请阅读整个实验文档。*

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 和实验室 2 中, 您实现了翻译工具从不可靠数据报中携带的段到传入的字节流: 流重组器和 TCP 接收器。

现在, 在实验 3 中, 你将实现连接的另一端。TCP 发送端是一个翻译工具从传出字节流到将成为不可靠数据报有效载荷的段。最后, 在实验 4 中, 你将结合前两个实验中的工作来创建一个有效的 TCP 实现: TCP 连接包含一个 TCP 发送端和 TCP 接收器。您将使用它与互联网上的真实服务器进行对话。



## 2 入门

您实施的TCP发送端将使用您在实验 0-2 中使用的相同 Sponge 库，并添加其他类和测试。开始操作：

1. 确保你已经提交了实验 2 的所有解决方案。请不要修改自由海绵目录，或webget.cc。否则，您可能无法合并实验室 3 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/lab3-startercode`。
4. 几天后，我们将发布一些额外的测试，帮助您避免一些常见错误 - 为了包含它们，您需要重复上述两个步骤。
5. 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）。
6. 外面建造目录，打开并开始编辑writeups/lab3.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 3：TCP 发送方

TCP 是一种通过不可靠的数据报可靠地传输一对流控字节流（每个方向一个）的协议。两方参与 TCP 连接，并且 每一方同时充当“发送方”（其自身传出的字节流）和“接收方”（传入的字节流）。双方被称为连接的“端点”或“对等点”。


本周，你将实现 TCP 的“发送方”部分，负责从 字节流（发送方应用程序创建并写入一个流，并将该流转换为一组传出的 TCP 段。在远程端，TCP 接收方<sup>1</sup>将这些段（那些到达的段 — 它们可能不会全部到达）转换回原始字节流，并将确认和窗口广告发送回发送者。

TCP 发送方和接收方各自负责 TCP 段的一部分。TCP 发送方写道所有领域TCP 段与TCP接收端 在实验 2 中：即序列号、SYN 标志、有效负载和 FIN 标志。但是，TCP 发送方仅读接收方写入的段中的字段：ackno 和窗口大小。以下是 TCP 段的结构，仅突出显示发送方将读取的字段：

---

<sup>1</sup>重要的是要记住，接收者可以任何有效 TCP 接收器的实现——它不一定是你自己实现的TCP接收器。互联网标准的宝贵之处之一在于它们如何在原本行为方式截然不同的端点之间建立一种通用语言。

## TCPSegment

Source Port Number ( sport )					Destination Port Number ( dport )									
Sequence Number ( seqno )														
Acknowledgement Number ( ackno )														
Data Offset ( doff )					URG	ACK	PSH	RST	SYN	FIN	Window Size ( win )			
Checksum ( cksum )						Urgent Pointer ( uptr )								
Options / Padding														
Payload														

这将是你的TCP发送方的责任：

- 跟踪接收方的窗口（处理传入确认窗口大小s）
- 尽可能填充窗口，方法是读取字节流，创建新的TCP段（如果需要，包括SYN和FIN标志），并发送它们。发送方应该继续发送片段直到窗口已满或字节流是空的。
- 跟踪哪些段已发送但尚未被接收方确认我们称之为“未完成”段
- 如果已发送了足够长的时间，但尚未得到确认，则重新发送未完成的段

“我为什么要这么做？”基本原则是发送接收方允许我们发送的任何内容（填充窗口），并不断重新传输，直到接收方确认每个段。这称为“自动重复请求”（ARQ）。发送方将字节流分成多个段并发送，直到接收方的窗口允许为止。感谢您上周的工作，我们知道远程TCP接收方可以重建字节流，只要它至少收到每个索引标记字节一次即可——无论顺序如何。发送方的工作是确保接收方至少收到每个字节一次。

### 3.1 如何TCP发送端知道某个片段是否丢失了？

你的TCP发送端将会发送一堆TCP段s。每个都将包含来自传出的（可能为空的）子字符串字节流，用序列号进行索引来指示其在流中的位置，并在流的开头标有SYN标志，在流的结尾标有FIN标志。

此外发送这些部分，TCP发送端还必须跟踪其杰出

段，直到它们占用的序列号被完全确认。周期性地，段的所有者TCP发送端将调用TCP发送方's打钩方法，表示时间的流逝。TCP发送端负责审查其收集的未偿TCP段并决定最早发送的段是否已经长时间未确认（即没有全部确认其序列号）。如果是，则需要重传（再次发送）。

以下是“拖延太久”的具体规则。<sup>2</sup>您将要实现这个逻辑，它有点复杂，但我们不希望您担心隐藏的测试用例会绊倒您，或者将其视为SAT上的文字问题。本周我们将为您提供一些合理的单元测试，并在您完成整个TCP实现后在实验4中进行更全面的集成测试。只要您100%通过这些测试并且您的实现合理，您就没问题。

“我为什么要这么做？”总体目标是让发送者及时检测出段丢失并需要重新发送的情况。重新发送前等待的时间很重要：您不希望发送者等待太长时间才重新发送段（因为这会延迟流向接收应用程序的字节数），但您也不希望它重新发送一个本来应该被确认的段，如果发送者只是等待了更长时间的话——这会浪费互联网的宝贵容量。

1. 每隔几毫秒，你的TCP发送方's打钩方法将使用一个参数来调用，该参数告诉它自上次调用该方法以来已经过去了多少毫秒。使用它来保持对当前运行的总毫秒数的概念TCP发送端还活着。请不要尝试调用任何“时间”或“时钟”函数来自操作系统或CPU——打钩方法是你的仅有的了解时间的流逝。这让事情具有确定性和可测试性。
2. 当TCP发送端构造时，它会被赋予一个参数，告诉它重传超时（RTO）。RTO是重新发送未完成的TCP段之前要等待的毫秒数。RTO的值会随时间而变化，但“初始值”保持不变。起始代码将RTO的“初始值”保存在名为初始重传超时。
3. 你将实现重传计时器：可以在特定时间启动的警报，一旦RTO过去，警报就会响起（或“过期”）。我们强调，这种时间流逝的概念来自于打钩方法被调用，而不是通过获取实际的时间。
4. 每次发送包含数据的段（序列空间中长度非零）时（无论是第一次还是重传），如果计时器未运行，开始运行这样它将在RTO毫秒后过期（对于RTO的当前值）。

---

<sup>2</sup>这些是基于TCP“真实”规则的简化版本：RFC 6298，建议5.1至5.6。此处的版本略有简化，但您的TCP实现仍将能够与Internet上的真实服务器进行通信。

所谓“过期”，是指时间将在未来一定毫秒数内耗尽。

5. 当所有未完成的数据都得到确认后，停止重传计时器。
6. 如果打钩被调用并且重传计时器已经到期：
  - (a) 重新传输最早（尚未被 TCP 接收方完全确认的（最低序列号）段。您需要将未完成的段存储在某个内部数据结构中，以便执行此操作。
- (二) 如果窗口大小非零：
  - i. 记录连续的重新传输，并增加它，因为你刚刚重新传输了一些东西。你的 TCP 连接将使用此信息来决定连接是否无望（连续重传次数过多）并需要中止。
  - ii. 将 RTO 值加倍。这被称为“指数退避”——它会减慢糟糕网络上的重传速度，以避免进一步阻碍工作。
- (c) 重置重传计时器并启动它，使得它在 RTO 毫秒后过期（考虑到您可能刚刚将 RTO 的值加倍！）。
7. 当接收者给发送者一个确认确认已成功收到新的数据（确认反映比任何先前的序列号都大的绝对序列号 确认）：
  - (a) 将 RTO 重新设置为其“初始值”。
  - (b) 如果发送方有任何未完成的数据，则重新启动重传计时器，以便它在 RTO 毫秒后过期（对于 RTO 的当前值）。
  - (c) 将“连续重传”次数重置为零。

我们建议在单独的类中实现重传计时器的功能，但这取决于您。如果您这样做，请将其添加到现有文件 (tcp发送方.hh和 tcp发送方.cc)。

## 3.2 实现 TCP 发送端

好的！我们已经讨论了什么TCP发送方会这样做（假设有一个传出的字节流，将其分成几段，发送给接收方，如果接收方没有及时确认，则继续重新发送）。我们已经讨论过什么时候得出的结论是，一个未完成的段已丢失，需要重新发送。

现在是时候实现具体的接口了TCP发送端将提供。它需要处理四个重要事件，每个事件最终都可能发送一个TCP段：

1. 空填充窗口 \_()

这TCP发送端被要求*非填满窗户*：它从输入中读取字节流并以以下形式发送尽可能多的字节TCP段，*只要有新的字节需要读取并且窗口中有可用空间*。

你需要确保每一个TCP 段您发送的内容完全适合接收方的窗口。确保每个个人TCP 段尽可能大，但不能大于给出的值TCPConfig::最大有效负载大小（1452字节）。

您可以使用序列空间中的 TCPSegment::length()方法来计算一个段所占用的序列号总数。请记住，SYN 和 FIN 标志也各自占用一个序列号，这意味着 *它们占据了窗户的空间*。

*?如果窗口大小为零该怎么办?* 如果接收方已宣布窗口大小为零，则填充窗口方法应像窗口大小一样一。发送者最终可能会发送一个字节，但该字节会被接收者拒绝（且不予确认），但这也可能促使接收者发送新的确认段，表明其窗口中已腾出更多空间。否则，发送者永远不会知道它被允许再次开始发送。

## 2. void ack 已接收 (const WrappingInt32 ackno, const uint16 t 窗口大小)

从接收方收到一个段，传送新的左边 (=确认和右 (= ackno + 窗口大小)窗口的边缘。TCP发送端应该查看其收集的未完成片段，并删除任何目前已被完全承认的片段（确认大于该段中的所有序列号）。TCP发送端如果有新的空间开放，应该再次填充窗口。

## 3. void tick (const size t自上次滴答以来的毫秒数)：

时间已过去 — 自上次调用此方法以来已过去一定毫秒数。发送方可能需要重新传输未完成的段。

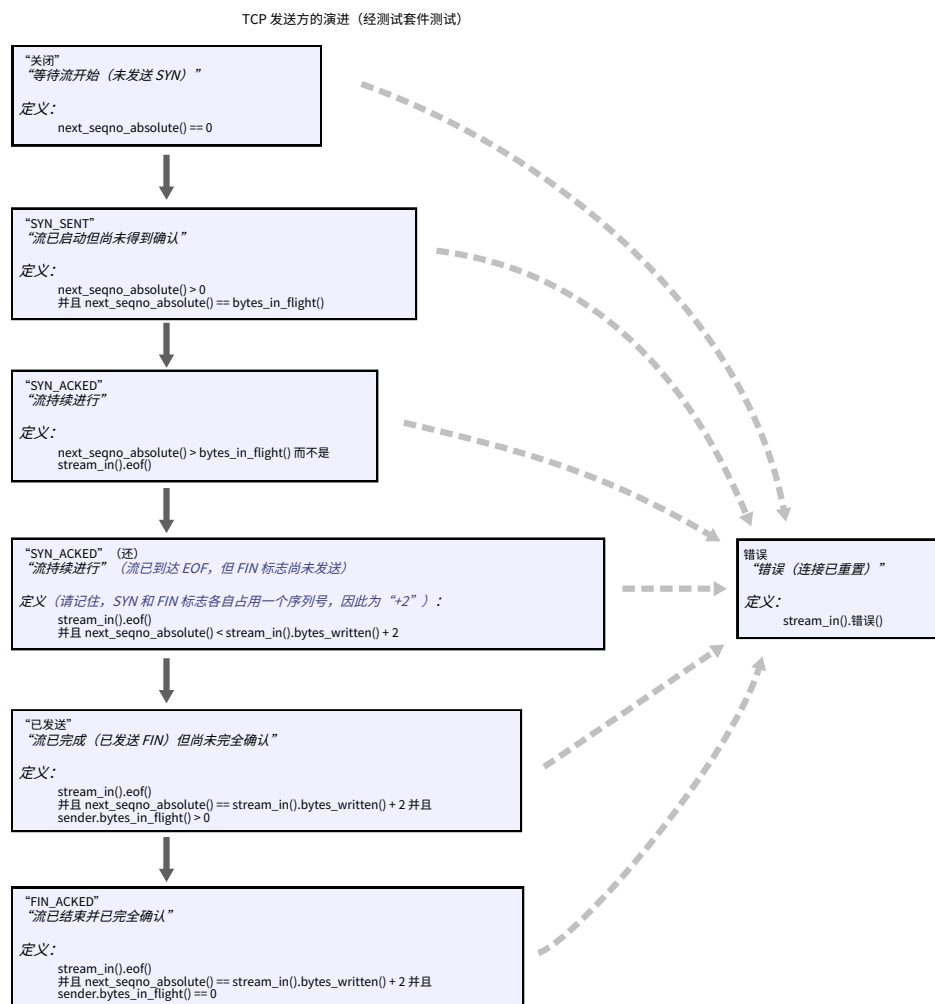
## 4. void 发送空段 ()：这TCP发送端应该生成并发送TCP 段 序列空间长度为零，且序列号设置正确。如果所有者（TCP连接您将在下周实施）想要发送一个空的确认部分。

注意：像这样的段，不占用任何序列号，不需要作为“未完成”段进行跟踪，并且永远不会被重新传输。

要完成实验 3，请查看文档中的完整界面，网址为[https://cs144.github.io/doc/lab3/class\\_tcp\\_sender.html](https://cs144.github.io/doc/lab3/class_tcp_sender.html)并实施完整的TCP发送端公共接口tcp发送方.hh和tcp 发送方.cc files。我们预期您会想要添加私有方法和成员变量，以及可能的辅助类。

### 3.3 测试理论

为了测试您的代码，测试套件将期望它经历一系列情况 - 从发送第一个 SYN 段，到发送所有数据，再到发送 FIN 段，最后得到确认 FIN 段。我们认为你不想创建更多状态变量跟踪这些“状态”——这些状态只是由已经暴露的公共接口定义的TCP发送端类。但为了帮助您理解测试输出，下面是预期演变的图表TCP发送端在流的整个生命周期中。（在实验 4 之前，您不必担心错误状态或 RST 标志。）



### 3.4 常见问题和特殊情况

#### • 我如何“发送”一个片段？

将其推到分段队列。至于你的TCP发送端就而言，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。



- 等一下，我如何才能既“发送”一个段，又能跟踪该段是否未完成，以便知道稍后要重新传输什么？那么我是否必须复制每个段？这很浪费吗？

当你发送一个包含数据的段时，你可能希望将其推送到分段队列和还在内部数据结构中保留一份副本，以便跟踪未完成的段，以便可能重新传输。事实证明，这并不浪费，因为段的有效负载存储为引用计数的只读字符串（缓冲对象）。所以不用担心——它实际上并没有复制有效载荷数据。

- 我的TCP发送端在我从接收器收到ACK之前，假设接收器的窗口大小？

一个字节。

- 如果仅收到确认，我该怎么办部分确认了一些未完成的段？我是否应该尝试截断已确认的字节？

TCP 发送方可以这样做，但就本类的目的而言，没有必要花哨。将每个段视为完全未完成，直到它被完全确认为止——它占用的所有序列号都小于确认。

- 如果我发送了包含“a”、“b”和“c”的三个单独段，但它们从未得到确认，我稍后可以将它们重新传输为一个包含“abc”的大段吗？还是我必须单独重新传输每个段？

再次：TCP 发送方可以这样做，但就本类的目的而言，无需花哨。只需分别跟踪每个未完成的段，当重传计时器到期时，再次发送最早的未完成段。

- 我是否应该将空段存储在“未完成”数据结构中并在必要时重新传输它们？

否 — 唯一应被跟踪为未完成且可能被重新传输的段是那些传输一些数据的段 — 即占用一些序列空间长度的段。不占用序列号（无有效载荷、SYN 或 FIN）的段不需要被记住或重新传输。

- 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。

## 4 开发调试建议

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

2. 您可以使用以下方式测试您的代码（编译后） 。



3. 请重新阅读 Lab 0 文档中关于“使用 Git”的部分，并记住将代码保存在它发布的 Git 存储库中掌握分支。进行小规模提交，使用良好的提交消息来识别更改的内容及其原因。
4. 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，并在可能的情况下将其分解出来。重复的代码块和庞大的函数会使您的代码难以理解。
5. 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。（有时您可能需要使用移动（）函数传递一个不可复制的对象。）
6. 如果遇到分段错误，那一定出了问题！我们希望您能够采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，您可以使用以下命令配置您的构建目录：  

```
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

 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 `valgrind` 工具。您还可以使用 

```
cmake .. -DCMAKE_BUILD_TYPE=Debug
```

 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构建目录并创建一个新目录。

## 5 提交

1. 在您提交的作品中，请仅对 `libspoon` 进行更改。时和 `cc fi` 位于顶层的 `libspoon`。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) `format` (规范化编码风格)
  - (二) `git status` (检查未提交的更改 - 如果有，请提交！)
  - (三) `make` (确保代码可以编译)
  - (四) `check lab3` (确保自动化测试通过)
3. 撰写报告 `writups/lab3.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！