

实验室检查点 2: TCP 接收方

到期的: 10 月 15 日星期五下午 5 点

最迟截止日期: 10 月 17 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 (包括在实验课上), 请在您的文章中注明他们的名字。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。

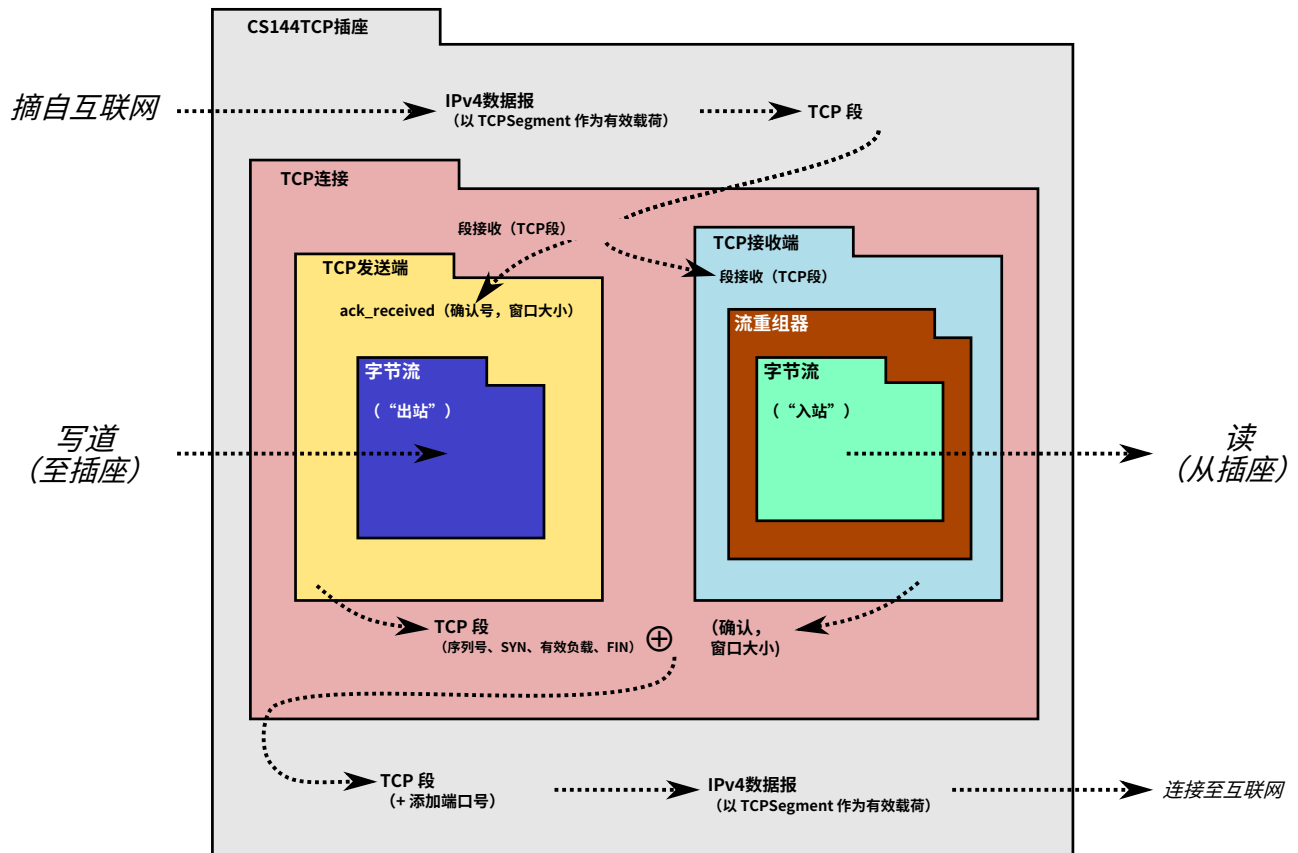
1 概述

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 中, 你创建了一个流重组器它接受一系列子字符串, 这些子字符串均摘录自同一个字节流, 然后将它们重新组合回原始流。

这些模块将在您的 TCP 实现中发挥作用, 但它们中没有任何内容特定于传输控制协议的细节。现在情况发生了变化。在实验 2 中, 您将实现 TCP 接收方, TCP 实现中处理传入字节流的部分。TCP 接收端在传入 TCP 段 (互联网上传输的数据报的有效载荷) 和传入字节流。

这是上一个实验室的图表。TCP 接收端从互联网接收片段 (通过已接收段 () 方法) 并将它们转换为对您的流重组器, 最终写入传入字节流。应用程序从此处读取字节流, 就像你在实验 0 中所做的那样, 通过读取 TCP 套接字。



除了写入传入流之外，TCP接收端负责告诉发送者两件事：

1. “第一个未组装”字节的索引，称为“确认号”或“确认”。这是接收方需要从发送方获取的第一个字节。
2. “第一个未组装”指标与“第一个不可接受”指标之间的距离。这被称为“窗口大小”。

总之，确认和窗口大小describe 描述接收者的窗户：一个指数范围TCP 发送方被允许发送。使用窗口，接收方可以控制传入数据的流量，使发送方限制其发送量，直到接收方准备好接收更多数据。我们有时将确认号称为窗口的“左边缘”（最小索引TCP接收端感兴趣的），并将ackno + 窗口大小作为“右边缘”（刚好超出最大索引TCP接收端感兴趣）。

你已经完成了实现TCP接收端 当你写流重组器和字节流；本实验旨在将这些通用类与 TCP 的细节联系起来。最难的部分将涉及思考 TCP 如何表示流中每个字节的位置（称为“序列号”）。

2 入门

您实施的TCP接收端将使用您在实验 0 和 1 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 1 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 1 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 2 的起始代码 `git 合并 origin/lab2-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/lab2.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 2：TCP 接收方

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，您将实现 TCP 的“接收器”部分，负责接收 TCP 段（实际的数据报有效负载）、重新组合字节流（包括其结束，如果发生），以及确定应发送回发送方进行确认和流量控制的信号。

“我为什么要这么做？” 这些信号对于 TCP 提供在不可靠的数据报网络上提供受流控的、可靠的字节流服务。在 TCP 中，致谢意思是，“下一个接收方需要字节，以便它可以重组更多的“您如何看待‘字节流’？”这告诉发送方需要发送或重新发送哪些字节。流量控制意思是“接收方感兴趣并愿意接收什么范围的索引？”（通常作为其剩余容量的函数）。这告诉发送方它有多少允许发送。

3.1 64 位索引和 32 位 seqno 之间的转换

作为热身，我们需要实现 TCP 表示索引的方式。上周你创建了一个流重组器重新组合子字符串，其中每个字节都有一个 64 位流索引，流中第一个字节的索引始终为零。64 位索引足够大，我们可以将其视为永不溢出。¹然而，在 TCP 标头中，空间非常宝贵，并且流中每个字节的索引不是用 64 位索引表示，而是用 32 位“序列号”或“seqno”表示。这增加了三个复杂性：

1. 您的实现需要规划 32 位整数的环绕。TCP 中的流可以是任意长度——通过 TCP 发送的字节流的长度没有限制。但 2^{32} 字节只有 4 GiB，不算太大。一旦 32 位序列号计数到 $2^{32}-1$ ，流中的下一个字节的序列号为零。
2. TCP 序列号从随机值开始：为了提高安全性并避免被属于同一端点之间较早连接的旧段混淆，TCP 会尝试确保序列号无法猜测且不太可能重复。因此，流的序列号不会从零开始。流中的第一个序列号是随机 32 位数称为初始序列号 (ISN)。这是代表 SYN（流的开始）的序列号。此后，其余序列号的行为正常：数据的第一个字节将具有 $ISN+1$ （模 2^{32} ）的序列号 $\equiv +1$ ），第二个字节将具有 $ISN+2 \pmod{2^{32}}$ ，ETC。
3. 逻辑开始和结束各占用一个序列号：除了确保收到所有数据字节外，TCP 还确保可靠地接收流的开始和结束。因此，在 TCP 中，SYN（流开始）和 FIN（流结束）控制标志被分配了序列号。每个控制标志都占用一序列号。（SYN 标志占用的序列号是 ISN。）流中的每个数据字节也占用一个序列号。请记住，SYN 和 FIN 不是流本身的一部分，也不是“字节”——它们代表字节流本身的开始和结束。

这些序列号（序列号）在每个 TCP 段的报头中传输。（同样，有两个流——每个方向一个。每个流都有单独的序列号和不同的随机 ISN。）有时谈论“绝对序列号”（它总是从零开始，不会换行），以及关于“流指数”（你已经使用过的东西流重组器（StreamReassembler）：流中每个字节的索引（从零开始）。

为了使这些区别具体化，考虑仅包含三个字母的字符串'的字节流猫'。如果 SYN 恰好有 seqno $2^{32}-2$ ，则每个字节的seqnos、绝对seqnos、流索引为：

¹以 100 千兆位/秒的速度传输，需要近 50 年才能达到 2^{64} 字节。相比之下，仅需三分之一秒即可达到 2^{32} 字节。

元素	同步	丙	一个吨	鳍	
序号	2≡+-2	2≡+-1	0	1	2
绝对序列号	0	1	2	3	4
流索引		0	1	2	

该图显示了 TCP 中涉及的三种不同类型的索引：

序列号	绝对序列号	流索引
<ul style="list-style-type: none"> • 从 ISN 开始 • 包括 SYN/FIN • 32 位，包装 • “序号” 	<ul style="list-style-type: none"> • 从 0 开始 • 包括 SYN/FIN • 64 位，不换行 • “绝对序列号” 	<ul style="list-style-type: none"> • 从 0 开始 • 忽略 SYN/FIN • 64 位，不换行 • “流索引”

在绝对序列号和流索引之间进行转换非常简单 - 只需加一或减一即可。不幸的是，在序列号和绝对序列号之间进行转换有点困难，混淆这两者会产生棘手的错误。为了系统地防止这些错误，我们将使用自定义类型表示序列号：WrappingInt32，并写出它与绝对序列号之间的转换（用 uint64_t）。WrappingInt32 复制代码是一个例子 包装类型：包含内部类型的类型（在本例中 uint32 类型但提供了一组不同的函数/运算符。

我们已经为您定义了类型，并提供了一些辅助函数（请参阅[包装整数.hh](#)），但您将在[包装整数.cc](#)：

1. WrappingInt32 裹 (uint64_t n, WrappingInt32 不是)

转换绝对序列号 → 序号。给定绝对序列号 (n) 和初始序列号 (不是)，生成 (相对) 序列号 n 。

2. uint64_t 展开 (WrappingInt32 n, WrappingInt32 isn, uint64_t 检查站)

转换序列号 → 绝对序列号。给定一个序列号 (n)、初始序列号 (不是) 和绝对检查点序列号，计算对应的绝对序列号 n 那是距离检查站最近。

注：检查点是必需的，因为任何给定的 seqno 都对应于许多绝对序列号。例如，ISN 为零时，序列号 “17” 对应于绝对序列号 17，但也对应于 $2^{32} + 17$ 或 $2^{33} + 17$ 或 $2^{34} + 17$ 等。检查点有助于解决歧义：它是一个绝对的 seqno，此类的用户知道它与正确答案 “大致相同”。在您的 TCP 实现中，您将使用最后一个重新组装的字节的索引作为检查点。

暗示：最干净/最简单的实现将使用提供的辅助函数 [包装整数.hh](#)。包装/解包操作应该保留偏移量——两个相差 17 的 seqno 将对应于两个也相差 17 的绝对 seqno。

您可以通过运行 WrappingInt32 测试。从建造 目录，运行


ctest-R 包装。

3.2 实现 TCP 接收方

恭喜您正确掌握了包装和解包逻辑！如果可以的话，我们会和您握手。在本实验的其余部分，您将实现TCP接收器。它将（1）从对端接收数据段，（2）重组字节流使用你的流重组器，（3）计算确认号（ackno）和窗口大小。ackno 和窗口大小最终将通过传出段传回给对端。

首先，请回顾一下 TCP 段的格式。这是两个端点相互发送的消息；它是较低级别数据报的有效载荷。非灰色字段表示本实验中感兴趣的信息：序列号、有效载荷以及 SYN 和 FIN 标志。这些字段由发送方写入，并由接收方读取和执行。

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TCP段类用C++表示此消息。请查看TCP段 (https://cs144.github.io/doc/lab2/class_tcp_seg.html) 和TCP头 (https://cs144.github.io/doc/lab2/struct_tcp_header.html) 您可能对 `序列空间的长度 ()` 方法，计算一个段占用多少个序列号（包括 SYN 和 FIN 标志各自占用一个序列号，以及有效负载的每个字节）。

接下来，我们来谈谈你的TCP接收端将提供：

```
// 构造一个 `TCPReceiver`，它将存储最多 `capacity` 字节TCP接收器（常量size_t
容量）；//在 .hh 文件中为您实现

// 处理传入的 TCP 段
空白已接收段（常量TCP段&段）；
```

```

// 应该发送给对端的确认信息 //

// 如果没有收到 SYN，则返回空 //

// 这是接收器窗口的开始，或者换句话说，// 流中第一个字节的序列号

// 接收方尚未收到。标准::选修的<WrappingInt32>
确认()常量;

// 应发送给对端的窗口大小 //

// 正式来说：这是接收方愿意接受的可接受索引窗口的大小。它是“第一个未组
装”和“第一个不可接受”索引之间的距离。//

// 换句话说：它是容量减去 // TCPReceiver 在字节流中保存的字节数。

size_t窗口大小 () 常量;

// 已存储但尚未重组的字节数
size_t未组装的字节 () 常量; //在 .hh 文件中为您实现

// 访问重新组装的字节流
字节流&流输出 () ; //在 .hh 文件中为您实现

```

这TCP接收端围绕您的StreamReassembler。我们已经实现了构造函数和未组装的字节和流出方法为您提供。響鳴le。以下是您必须为其他人做的事情：

3.2.1 收到的段() _

这是主要的主力方法。TCPReceiver::段已接收()每次从对端接收到新的段时都会被调用。

此方法需要：

- 如果必要的话，设置初始序列号。第一个到达的设置了 SYN 标志的报文段的序列号是初始序列号。您需要跟踪这一点，以便继续在 32 位包装的 seqnos/acknos 和它们的绝对等价物之间进行转换。（请注意，SYN 标志只是一佛罗里达州ag。同一数据段也可以携带数据，甚至可以设置 FIN 标志。
- 将任何数据或流结束标记推送到StreamReassembler。如果 鳍片ag 设置在TCP 段's 标头，这意味着有效负载的最后一个字节是整个流的最后一个字节。请记住流重组器期望流索引从零开始；您必须解开 seqnos 才能生成这些。

3.2.2 ackno()

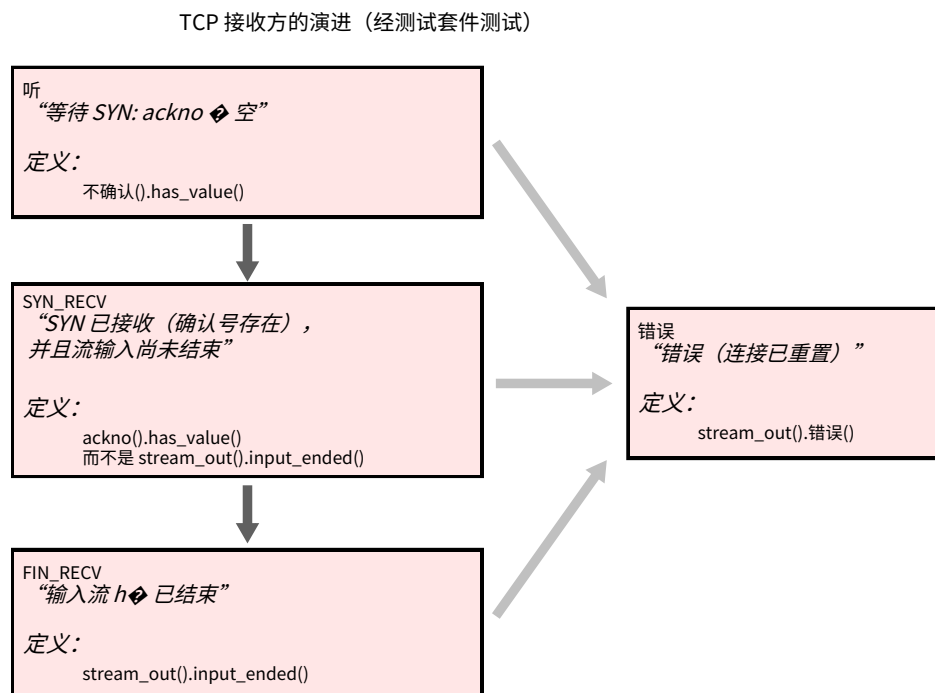
返回可选 `<WrappingInt32>` 包含接收方尚不知道的第一个字节的序列号。这是窗口的左边缘：接收方想要接收的第一个字节。如果尚未设置 ISN，则返回一个空的可选项。

3.2.3 窗口大小()

返回“第一个未组装”索引（与确认相对应的索引）与“第一个不可接受”索引之间的距离。

3.3 演化TCP接收端在连接的整个生命周期内

在 TCP 连接过程中，你的TCP接收端将经历一系列状态：从等待 SYN（带有空的 `ackno`），到正在进行的流，到完成的流，这意味着输入已经在字节流。测试套件将检查您的TCP接收端正确处理传入TCP段s并通过这些状态演变，如下所示。（在实验 4 之前，您不必担心错误状态或 RST 标志。）



4 开发调试建议

1. 实施TCP接收方'文件中的公共接口（以及您想要的任何私有方法或函数）tcp 接收器.cc。您可以将任何您喜欢的私人成员添加到TCP接收端上课tcp 接收器.hh。
2. 编译后，你可以使用以下命令测试代码 `checklab2`。
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更难理解。
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>) 是一项很棒的资源，尽管您不需要任何复杂的 C++ 功能来完成这些实验。
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，你可以使用以下命令配置你的构建目录：
`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。您还可以使用瓦尔格林德工具。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 .两者这些会减慢你的代码速度——完成后别忘了返回到“发布”版本。

5 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)检查实验室2	(确保自动化测试通过)
-----------	-------------

3. 撰写报告writeups/lab2.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. 在您 的报告中，请填写完成作业所花费的小时数以及其他任何评论。
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！