CS144：计算机网络导论 2025年秋季

实验检查点 2: TCP接收方

截止日期：10月12日（周日）晚上11:59

合作政策： 与检查点0相同。请勿查看其他学生的代码或这些作业旧版本的解答。请在你的报告中完全披露任何合作者或任何灰色地带——坦诚是最好的策略。

0 概述

建议：在实现之前阅读整个实验文档。

在检查点0中，你实现了流控字节流（ByteStream）的抽象。在检查点1中，你创建了一个Reassembler，它接受一系列子串，所有子串都摘自同一个字节流，并将它们重新组装回原始流。这些模块在你的TCP实现中将非常有用，但它们中没有任何内容是特定于传输控制协议的细节的。现在情况变了。在检查点2中，你将实现TCPReceiver，即TCP实现中处理传入字节流的部分。

TCPReceiver通过 `receive()` 方法接收来自对端发送方的消息，并将它们转化为对Reassembler的调用，最终写入传入的ByteStream。应用程序从这个ByteStream读取，就像你在Lab 0中从TCPSocket读取一样。

同时，TCPReceiver还通过 `send()` 方法生成返回给对端发送方的消息。这些"接收方消息"负责告诉发送方：

> 第一个"未组装"字节的索引，称为"确认号"或"ackno"。这是接收方从发送方需要的第一个字节。
>
> 输出ByteStream中的可用容量。这称为"窗口大小"。

ackno和窗口大小共同描述了接收方的窗口：TCP发送方被允许发送的索引范围。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ackno称为窗口的"左边缘"（TCPReceiver感兴趣的最小索引），将ackno + 窗口大小称为"右边缘"（刚好超出TCPReceiver感兴趣的最大索引）。

你在编写Reassembler和ByteStream时已经完成了实现TCPReceiver所涉及的大部分算法工作；本实验是关于将这些通用类与TCP的细节连接起来。最困难的部分将涉及思考TCP如何表示每个字节在流中的位置——即所谓的"序列号"。

1 开始

你的TCPReceiver实现将使用与检查点0和1中相同的Minnow库，并增加了额外的类和测试。开始步骤：

> 确保已提交检查点1的所有解决方案。请不要修改src目录顶层之外的任何文件，也不要修改webget.cc。否则你可能无法合并检查点1的起始代码。
>
> 在实验作业仓库内，运行 `git fetch --all` 以获取实验作业的最新版本。
>
> 通过运行 `git merge origin/check2-startercode` 下载检查点2的起始代码。（如果你已将"origin"远程重命名为其他名称，你可能需要在此处使用不同的名称，例如 `git merge upstream/check2-startercode`。）
>
> 确保构建系统已正确设置：`cmake -S . -B build`
>
> `arm64`（UTM）Mac用户注意：过去，g++的"sanitizers"（错误检查器）在arm64上运行非常慢。Minnow使用这些来运行测试。如果你在arm64 Mac上遇到问题，可以配置cmake使用不同的编译器：`cmake -S . -B build -DCMAKE_CXX_COMPILER=clang++`
>
> 编译源代码：`cmake --build build`

打开并开始编辑 `writeups/check2.md` 文件。这是你实验报告的模板，将包含在你的提交中。

2 检查点2：TCP接收方

TCP是一种协议，可通过不可靠数据报可靠地传递一对流控字节流（每个方向一个）。两方或"对等方"参与TCP连接，每个对等方同时充当"发送方"（发送自己的出站字节流）和"接收方"（接收传入字节流）。

本周，你将实现TCP的"接收方"部分，负责接收来自发送方的消息、重新组装字节流（包括其结束时），以及确定应发送回发送方以进行确认和流量控制的消息。

★我为什么要做这个？ 这些信号对于TCP在不可靠数据报网络上提供流控、可靠字节流服务至关重要。在TCP中，确认意味着"接收方需要的下一个字节的索引是什么，以便它可以重新组装更多的ByteStream？"这告诉发送方它需要发送或重新发送哪些字节。流量控制意味着"接收方感兴趣并愿意接收的索引范围是什么？"（这是其可用容量的函数）。这告诉发送方它被允许发送多少。

2.1 在64位索引和32位seqno之间转换

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Reassembler，它重新组装子串，其中每个字节都有一个64位流索引，流中的第一个字节始终索引为零。64位索引足够大，我们可以将其视为永不溢出[1]。然而，在TCP头部中，空间很宝贵，每个字节在流中的索引不是用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：

> 你的实现需要为32位整数回绕做规划。
> TCP中的流可以任意长——通过TCP发送的ByteStream长度没有限制。但2^{32}字节只有4 GiB，并不算太大。一旦32位序列号计数到2^{32}-1，流中的下一个字节将有序列号零。

> TCP序列号从一个随机值开始： 为了提高鲁棒性并避免被属于同一端点之间早期连接的旧段混淆，TCP试图确保序列号无法被猜测且不太可能重复。因此流的序列号不是从零开始。流中的第一个序列号是一个随机的32位数字，称为初始序列号（ISN）。这是代表"零点"或SYN（流的开始）的序列号。其余的序列号在那之后正常表现：第一个数据字节的序列号为ISN+1（mod 2^{32}），第二个字节为ISN+2（mod 2^{32}），依此类推。

> 逻辑开头和结尾各占一个序列号：除了确保收到所有数据字节外，TCP还确保流的开头和结尾被可靠接收。因此，在TCP中，SYN（流开头）和FIN（流结尾）控制标志被分配序列号。每个占用一个序列号。（SYN标志占用的序列号就是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头和结尾。

这些序列号（seqno）在每个TCP段的头部中传输。（而且，同样有两个流——每个方向一个。每个流有单独的序列号和不同的随机ISN。）有时讨论"绝对序列号"的概念（始终从零开始且不回绕）以及"流索引"（你已经在Reassembler中使用的：流中每个字节的索引，从零开始）也很有帮助。

为了使这些区别具体化，考虑包含三字母字符串'cat'的字节流。如果SYN的seqno恰好为2^{32}-2，则每个字节的seqno、绝对seqno和流索引为：

element	syn	c	a	t	fin
seqno	2^32-2	2^32-1	0	1	2
absolute seqno	0	1	2	3	4
stream index		0	1	2	

下图展示了TCP中涉及的三种不同类型的索引：

> 序列号 — 从ISN开始，包括SYN/FIN，32位，回绕，"seqno"

> 绝对序列号 — 从0开始，包括SYN/FIN，64位，不回绕，"absolute seqno"

> 流索引 — 从0开始，不包括SYN/FIN，64位，不回绕，"stream index"

在绝对序列号和流索引之间转换很容易——只需加或减一。不幸的是，在序列号和绝对序列号之间转换有点困难，混淆两者可能会产生棘手的错误。为了系统地防止这些错误，我们将用自定义类型表示序列号：`Wrap32`，并编写它与绝对序列号（用 `uint64_t` 表示）之间的转换。

`Wrap32` 是包装类型的一个例子：一种包含内部类型（在本例中为 `uint32_t`）但提供不同函数/运算符集的类型。

我们已经为你定义了类型并提供了一些辅助函数，但你将在 `wrapping_integers.cc` 中实现转换：

> `static Wrap32 Wrap32::wrap( uint64_t n, Wrap32 zero_point )` — 将绝对seqno转换为seqno。给定一个绝对序列号（n）和一个初始序列号（zero_point），产生n的（相对）序列号。
>
> `uint64_t unwrap( Wrap32 zero_point, uint64_t checkpoint ) const` — 将seqno转换为绝对seqno。给定一个序列号（Wrap32）、初始序列号（zero_point）和一个绝对checkpoint序列号，找到与checkpoint最接近的对应绝对序列号。
>
> 注意：需要checkpoint是因为任何给定的seqno对应多个绝对seqno。例如，当ISN为零时，seqno"17"对应绝对seqno 17，也对应2^{32}+17，或2^{33}+17，等等。checkpoint有助于解决歧义：它是此类的用户知道"在正确答案附近"的一个绝对seqno。在你的TCP实现中，你将使用第一个未组装的索引作为checkpoint。
>
> 提示：最简洁/最简单的实现将使用 `wrapping_integers.hh` 中提供的辅助函数。wrap/unwrap操作应保持偏移量——相差17的两个seqno将对应也相差17的两个绝对seqno。
>
> 提示 #2：我们期望wrap用一行代码，unwrap用不到10行代码。如果你发现自己需要实现更多，可能需要退后一步，尝试思考不同的策略。

你可以通过运行测试来测试你的实现：`cmake --build build --target check2`。（提醒：Mac arm64用户可能希望配置cmake使用"clang++"编译器——见上文。）

2.2 实现TCP接收方

恭喜你搞定了包装和解包逻辑！如果这个胜利发生在实验课上，我们会和你握手（或者，疫情后，碰肘）。在本实验的其余部分，你将实现TCPReceiver。它将(1)接收来自其对端发送方的消息并使用Reassembler重新组装ByteStream，以及(2)发送包含确认号（ackno）和窗口大小的消息回对端的发送方。我们预计这总共需要大约15行代码。

首先，让我们回顾一下TCP"发送方消息"的格式，其中包含有关ByteStream的信息。这些消息从TCPSender发送到其对端的TCPReceiver：

```
struct TCPSenderMessage
{
  Wrap32 seqno { 0 };
  bool SYN {};
  std::string payload {};
  bool FIN {};
  bool RST {};
  // How many sequence numbers does this segment use?
  size_t sequence_length() const { return SYN + payload.size() + FIN; }
};
```

TCPReceiver生成自己返回给对端TCPSender的消息：

```
struct TCPReceiverMessage
{
  std::optional<Wrap32> ackno {};
  uint16_t window_size {};
  bool RST {};
};
```

你的TCPReceiver的工作是接收其中一种消息并发送另一种：

```
class TCPReceiver
{
public:
  explicit TCPReceiver( Reassembler&& reassembler );
  void receive( TCPSenderMessage message );
  TCPReceiverMessage send() const;
  const Reassembler& reassembler() const;
  Reader& reader();
  const Reader& reader() const;
  const Writer& writer() const;
private:
  Reassembler reassembler_;
};
```

2.2.1 `receive()`

每次从对端发送方接收到新段时，都会调用此方法。此方法需要：

设置初始序列号（如果必要）。第一个到达的设置了SYN标志的段的序列号就是初始序列号。你需要跟踪它以便继续在32位包装的seqno/ackno及其绝对等效值之间进行转换。（注意SYN标志只是头部中的一个标志。同一消息也可能携带数据或设置了FIN标志。）

将任何数据推送到`Reassembler`。 如果TCPSegment头部中设置了FIN标志，这意味着有效载荷的最后一个字节是整个流的最后一个字节。请记住Reassembler期望从零开始的流索引；你将不得不解包seqno来产生这些索引。

不重复任何已有的状态或功能。 你已经构建了ByteStream、Reassembler和Wrap32模块，它们做了很多非常有用的事情！TCPReceiver不应跟踪这些模块已处理的任何状态（成员变量），也不应重复它们已处理的任何功能。

3 开发与调试建议

在 `tcp_receiver.cc` 文件中实现TCPReceiver的公共接口（以及你想要的任何私有方法或函数）。你可以在 `tcp_receiver.hh` 中的TCPReceiver类中添加任何你喜欢的私有成员。

你可以用 `cmake --build build --target check2` 测试你的代码。

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分发的Git仓库的main分支上。进行小的提交，使用好的提交消息来标识更改了什么以及为什么。

请努力使你的代码对评分的CA可读。使用合理且清晰的变量命名约定。使用注释来解释复杂或微妙的代码段。使用"防御性编程"——显式检查函数的前置条件或不变量，如果有任何错误则抛出异常。在你的设计中使用模块化——识别共同的抽象和行为，并在可能时将它们提取出来。重复的代码块和巨大的函数会使你的代码更难理解。

请同时遵循检查点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，虽然你不需要使用C++的任何高级特性来完成这些实验。

4 提交

在你的提交中，请仅更改 `src` 目录中的 `.hh` 和 `.cc` 文件。在这些文件中，请随意添加必要的私有成员，但请不要更改任何类的公共接口。

在提交任何作业之前，请按顺序运行以下命令：

(a) 确保已将所有更改提交到Git仓库。你可以运行 `git status` 来确保没有未提交的更改。记住：在编码时进行小的提交。

(b) `cmake --build build --target format`（规范化编码风格）

(c) `cmake --build build --target check2`（确保自动化测试通过）

(d) 可选：`cmake --build build --target tidy`（建议改进以遵循良好的C++编程实践）

在 `writeups/check2.md` 中撰写报告。此文件应为大约20到50行的文档，每行不超过80个字符，以便于阅读。报告应包含以下部分：

（a） 程序结构与设计。 描述你代码中体现的高层结构和设计选择。你不需要详细讨论从起始代码继承的内容。利用这个机会突出重要的设计方面，并为评分的TA提供这些领域的更多细节。我们强烈建议你通过使用子标题和大纲使这份报告尽可能可读。请不要简单地将你的程序翻译成一段英文。

（b） 替代设计选择——你考虑过的或理想情况下在性能、编写难度、阅读难度以及你认为对读者有趣的其他维度方面进行了评估的。如适用，请包括任何测量数据。

（c） 实现挑战。 描述你觉得最棘手的代码部分并解释原因。反思你是如何克服这些挑战的。你是如何调试和测试你的代码的?

（d）残余错误。 尽可能指出并解释代码中仍然存在的任何错误（或未处理的边界情况）。

在你的报告中，请同时填写完成作业所花的小时数和任何其他评论。

如果有任何问题，请尽快在实验课上通知课程工作人员，或在Ed上发帖提问。祝你好运!

5 额外学分

对测试套件的改进将获得额外学分。在 `tests` 目录中的文件（例如 `minnow/tests/recv_connect.cc`）中添加一个测试用例，该用例能捕获某人可能合理犯的、但现有测试套件尚未捕获的真实错误。请将你的测试作为Pull Request提交（公开也没关系），以便我们查看并决定是否将其添加到整体测试套件中。（此机会将持续开放——例如，如果你在第7周找到了一个关于Reassembler的好测试，那也很好。）