Lab Checkpoint 1: stitching substrings into a byte stream

Due: Sunday, October 5, 11:59 p.m.

Collaboration Policy: Same as checkpoint 0.

0 Overview

For Checkpoint 0, you used an *Internet stream socket* to fetch information from a website and send an email message, using Linux's built-in implementation of the Transmission Control Protocol (TCP). This TCP implementation managed to produce a pair of *reliable in-order byte streams* (one from you to the server, and one in the opposite direction), even though the underlying network only delivers "best-effort" datagrams. By this we mean: short packets of data that can be lost, reordered, altered, or duplicated. You also implemented the byte-stream abstraction yourself, in memory within one computer. Over the coming weeks, you'll implement TCP yourself, to provide the byte-stream abstraction between a pair of computers separated by an unreliable datagram network.

⋆*Why am I doing this?* Providing a service or an abstraction on top of a different less-reliable service accounts for many of the interesting problems in networking. Over the last 40 years, researchers and practitioners have figured out how to convey all kinds of things—messaging and e-mail, hyperlinked documents, search engines, sound and video, virtual worlds, collaborative file sharing, digital currencies—over the Internet. TCP's own role, providing a pair of reliable byte streams using unreliable datagrams, is one of the classic examples of this. A reasonable view has it that TCP implementations count as the **most widely used** nontrivial computer programs on the planet.

The lab assignments will ask you to build up a TCP implementation in a modular way. Remember the `ByteStream` you just implemented in Checkpoint 0? In the coming labs, you'll end up convey two of them across the network: an "outbound" `ByteStream`, for data that a local application writes to a socket and that your TCP will send *to* the peer, and an "inbound" `ByteStream` for data coming *from* the peer that will be read by a local application.

This checkpoint contains a "hands-on" component and an implementation component. You might prefer to start the implementation component before the lab session, and do the hands-on component at the lab session. If you are a CGOE student, please use EdStem to coordinate a time with another student to do the hands-on component.

The hands-on component involves multiple moving parts—so there might be some glitches. Please bear with us at the lab session and we'll do our best to get it working for everybody. If you see an error message from the https://cs144.net website, please report it in a public post on EdStem and we'll take a look.

实验检查点1：将子串拼接成字节流

截止日期：10月5日星期日，晚上11:59

协作政策：与检查点0相同。

0 概述

对于检查点0，你使用了一个互联网流套接字来从网站获取信息并发送电子邮件消息，使用了Linux内置的传输控制协议（TCP）的实现。这个TCP实现成功地产生了一对可靠的有序字节流（一个从你到服务器，另一个方向相反），尽管底层的网络只传输“尽力而为”的数据报。我们的意思是：可以丢失、重新排序、更改或重复的短数据包。你还自己实现了字节流抽象，在内存中的一个计算机内。在接下来的几周里，你将自行实现TCP，以在由不可靠数据报网络隔开的两个计算机之间提供字节流抽象。

⋆我为什么要做这个？在更不可靠的服务之上提供服务或抽象解释了网络中有趣的问题的许多。在过去的40年里，研究人员和从业者已经想出了如何传递各种东西——消息和电子邮件、超链接文档、搜索引擎、声音和视频、虚拟世界、协作文件共享、数字货币——通过互联网。TCP自身的角色，使用不可靠数据报提供一对可靠的字节流，是这个经典例子之一。一个合理的观点认为，TCP实现是世界上使用最广泛的非平凡计算机程序。

实验作业将要求你以模块化的方式构建一个TCP实现。还记得你在检查点0中刚刚实现的 `ByteStream` 吗？在接下来的实验中，你将最终通过网络传输其中的两个：一个“出站” `ByteStream`，用于本地应用程序写入套接字并由你的TCP发送给对端，以及一个“入站” `ByteStream` ，用于从对端接收并由本地应用程序读取的数据。

这个检查点包含一个“动手实践”部分和一个实现部分。你可能更喜欢在实验课前开始实现部分，并在实验课上完成动手实践部分。如果你是CGOE学生，请使用EdStem与另一位学生协调时间完成动手实践部分。

实践环节涉及多个动态组件——因此可能会有一些故障。请在我们实验室期间多包涵，我们会尽力确保所有人都能够正常使用。如果你从 https://cs144.net 网站看到错误信息，请在 EdStem 上发布公开帖子报告，我们会进行查看。

1 Getting started

Your implementation of TCP will use the same Minnow library that you used in Checkpoint 0, with additional classes and tests. To get started:

1. Make sure you have committed all your solutions to Checkpoint 0. Please don't modify any files outside of the `src` directory, or `webget.cc`. You may have trouble merging the Checkpoint 1 starter code otherwise.
2. While inside the repository for the lab assignments, run `git fetch` to retrieve the most recent version of the lab assignments.
3. Download the starter code for Checkpoint 1 by running `git merge origin/check1-startercode`.
4. Make sure your build system is properly set up: `cmake -S . -B build`
5. Compile the source code: `cmake --build build`
6. Open and start editing the `writeups/check1.md` file. This is the template for your lab writeup and will be included in your submission.

2 Hands-on component: a private network for the class

We have created a private network for the CS144 class. This will allow your VM to send datagrams directly to and from the VMs of other students in the class. To make your VM join this network:

1. On your VM, install the "wireguard" and "apparmor-utils" packages by running `sudo apt install wireguard apparmor-utils`
2. Run `sudo aa-complain /etc/apparmor.d/wg` to remove some security constraints on the private-networking software.
3. Visit https://cs144.net/wg and follow the instructions to join the CS144 private network.
4. Once you have joined the network, verify that you can connect by following the "ping" instructions on that page (the instructions appear after you have joined the network).
5. Every time you reboot your VM, you'll have to rejoin the network (if you want to be able to send datagrams to and from other students in this class). You don't have to register a new public key each time, but you do have to rerun the commands on that webpage. The commands will be the same each time.

1 开始使用

你的 TCP 实现将使用你在检查点0中使用的相同 Minnow 库，并增加额外的类和测试。开始操作：

1. 确保你已经将所有解决方案提交到检查点0。请不要修改 `src` 目录或 `webget.cc`之外的任何文件，否则你可能会在合并检查点1启动代码时遇到问题。

2. 在实验作业的代码库中，运行 `git fetch` 以获取实验作业的最新版本。
最新版本的实验作业。

3. 通过运行 `git merge origin/check1-startercode` 下载 Checkpoint 1 的启动代码.

4. 确保您的构建系统已正确设置： `cmake -S . -B build`

5. 编译源代码： `cmake --build build`

6. 打开并开始编辑 `writeups/check1.md` 文件。这是您的实验报告模板，并将包含在您的提交中。

2 实践部分：为课程创建的私有网络

我们为 CS144 课程创建了一个私有网络。这将允许您的虚拟机直接向和从班级中其他学生的虚拟机发送数据报。要让您的虚拟机加入此网络：

1. 在您的虚拟机（VM）上，通过运行命令来安装“wireguard”和“apparmor-utils”软件包 2. 运行
`sudo apt install wireguard apparmor-utils`

3. 运行 `sudo aa-complain /etc/apparmor.d/wg` 以移除私有网络软件的一些安全限制。

3. 访问 https://cs144.net/wg 并按照说明加入 CS144 私有网络。

4. 一旦您已加入网络，请按照该页面上的“ping”
说明进行验证，以确认您能够连接（说明会在您加入网络后出现）。

5. 每次您重启虚拟机时，都需要重新加入网络（如果您希望能够向本课程的其他学生发送和接收数据报）。您不必每次都注册新的公钥，但确实需要重新运行该网页上的命令。每次命令都将是相同的。

2.1 Ping a friend and look at the datagrams

1. On your own computer (e.g. your Mac or Windows machine—not your VM), install the "wireshark" program by following the instructions at https://www.wireshark.org/. (If you are using Debian or Ubuntu GNU/Linux, the command is `sudo apt install wireshark`.
2. Ask a groupmate for their IP address (the one shown on the https://cs144.net/wg webpage **for them**). Using the `ping` command, send some "echo request" datagrams to your friend, and make sure that you get some "echo reply" datagrams back.
3. Tips:
   - You can end the "ping" program by typing `ctrl-C`.)
   - You can make the "ping" command go faster by including the argument `-i 0.2`. This will make it send an "echo request" every 0.2 seconds (5 times per second).
   - You can make the "ping" command print out a summary of the statistics so far (without ending it) by running this command in another terminal: `killall -QUIT ping`.
4. Begin a report in your writeup, including the following information:
   (a) What is the average round-trip delay between when your VM sends an "echo request" and when it receives an "echo reply" from your groupmate's VM?
   (b) What was the delivery rate (what percentage of "echo requests" received a corresponding "echo reply")? What was the loss rate (this is 100% minus the delivery rate)? Send at least 1,000 pings to get a reliable estimate. (This will take about three minutes if using `ping -i 0.2`.
   (c) Did you see any duplicated datagrams (ping will print "DUP")?
   (d) While the ping is running, you and your groupmate can capture some of the raw Internet datagrams by running `sudo rm /tmp/capture.raw; sudo tcpdump -n -w /tmp/capture.raw -i wg0 --print --packet-buffered`. This command will capture the datagrams on the "wg0" interface (the private class network) to a file ("/tmp/capture.raw"), while also printing them out to the screen. Make sure you see some "echo request" and "echo reply" lines printed—that indicates your groupmate is receiving your datagrams and replying to you.
   (e) Use the `wireshark` program to inspect the `/tmp/capture.raw` file on each of your VMs. You probably want to `scp` the `capture.raw` file to your own computer (e.g. a Mac or Windows machine) and then use wireshark to open this file, so you can use its graphical interface. Can you find the fields of the Internet datagram that were discussed in the Jan. 10 (and match the diagram at https://www.rfc-editor.org/rfc/rfc791.html#page-11)?
   (f) Are there any differences between the **same** datagram when it was captured on your VM compared with when they were captured on your friend's VM? What? Make sure you are really analyzing the **same** datagram, captured from two different points of view (sender vs. receiver).

2.1 Pinging朋友并查看数据报

1. 在你自己的计算机（例如你的Mac或Windows机器——不是你的虚拟机）上，按照 https://www.wireshark.org/ 上的说明安装“wireshark”程序。（如果你使用的是Debian或Ubuntu GNU/Linux，命令是 `sudo apt install wireshark`。
2. 向一位组员要他们的IP地址（他们在 https://cs144.net/wg 页面上显示的那个）。使用 `ping` 命令，向你的朋友发送一些“回显请求”数据报，并确保你收到一些“回显应答”数据报。
3. 提示：
   - 您可以通过输入来结束“ping”程序 `ctrl-C`。）
   - 您可以通过包含参数 `-i 0.2` 来让“ping”命令运行得更快。这将使其每隔 0.2 秒（每秒 5 次）发送一个“回显请求”。
   - 您可以让“ping”命令打印出到目前为止的统计摘要（不结束它）通过在另一个终端中运行此命令： `killall -QUIT ping`。
4. 在您的报告中开始一份报告，包括以下信息：
   (a) 您的虚拟机发送“回显请求”到您的组员的虚拟机接收“回显应答”之间的平均往返延迟是多少?
   (b) 传输率是多少（即“回显请求”中有多少百分比收到了相应的“回显应答”）？丢包率是多少（这是传输率的100%减去值）？至少发送1,000个ping以获得可靠的估计。（如果使用 `ping -i 0.2`，这将大约需要三分钟。
   (c) 你看到任何重复的数据报（ping会打印“DUP”）吗?
   (d) 在ping运行时，你和你的组员可以通过运行 `sudo rm /tmp/capture.raw; sudo tcpdump -n -w /tmp/capture.raw -i wg0 --print --packet-buffered` 捕获一些原始的互联网数据报。该命令将“wg0”接口（私有类网络）上的数据报捕获到文件（“/tmp/capture.raw”）中，同时也会将它们打印到屏幕上。确保你看到一些“回显请求”和“回显应答”行被打印出来——这表明你的组员正在接收你的数据报并回复你。
   (e) 使用 `wireshark` 程序来检查你每个虚拟机上的 `/tmp/capture.raw` 文件。你可能需要将 `capture.raw` 文件传输到自己的计算机（例如 Mac 或 Windows 电脑）上，然后使用 wireshark 打开这个文件，这样你就可以使用它的图形界面。你能找到 1 月 10 日讨论的互联网数据报的字段吗（并与 https://www.rfc-editor.org/rfc/rfc791.html#page-11 的图表进行匹配）？
   (f) 在您的虚拟机捕获的相同数据报与在您朋友的虚拟机捕获的相同数据报之间是否有任何差异？是什么？确保您正在真正分析相同的数据报，从两个不同的角度捕获（发送者 vs. 接收者）。

2.2 Send an Internet datagram by hand

In the `apps/ip_raw.cc` file, write a program that sends an Internet datagram to your friend by using a raw socket, using the same method as the Sept. 24 lecture. It's okay to adapt code from this lecture.

1. Send your groupmate an Internet datagram with IP protocol "5" (you'll have to use "sudo" to run the "./build/apps/ip_raw" program), and have your friend use `tcpdump` to make sure they receive the datagram. They can run `sudo tcpdump -n -i wg0 'proto 5'` to print out only datagrams matching protocol "5". Make sure they get it!
2. Send your groupmate a user datagram (with IP protocol "17"), using the "user datagram" header format in https://www.rfc-editor.org/rfc/rfc768. Have your groupmate **receive** this datagram *without using "sudo"*. They can use the "nc -u" program as was done in lecture, or a C++ program using the `UDPSocket` class—whatever they prefer!
3. Include the code for your "ip_raw.cc" in your submission to this checkpoint.
4. Do the same in reverse and receive a datagram from your groupmate.

3 Implementation: putting substrings in sequence

As part of the lab assignment, you are implementing a TCP receiver: the module that receives datagrams and turns them into a reliable byte stream to be read from the socket by the application—just as your `webget` program read the byte stream from the webserver in Checkpoint 0.

The TCP sender is dividing its byte stream up into short *segments* (substrings no more than about 1,460 bytes apiece) so that they each fit inside a datagram. But the network might reorder these datagrams, or drop them, or deliver them more than once. The receiver must reassemble the segments into the contiguous stream of bytes that they started out as.

In this lab you'll write the data structure that will be responsible for this reassembly: a `Reassembler`. It will receive substrings, consisting of a string of bytes, and the index of the first byte of that string within the larger stream. **Each byte of the stream** has its own unique index, starting from zero and counting upwards. As soon as the Reassembler knows the **next** byte of the stream, it will write it to the Writer side of a ByteStream— the same ByteStream you implemented in checkpoint 0. The Reassembler's "customer" can read from the Reader side of the same ByteStream.

Here's what the interface looks like:

```
// Insert a new substring to be reassembled into a ByteStream.
void insert( uint64_t first_index, std::string data, bool is_last_substring );
```

2.2 手动发送互联网数据报

在 `apps/ip raw.cc` 文件中，编写一个程序，使用原始套接字向你的朋友发送互联网数据报，方法与9月24日的讲座相同。可以参考本讲座的代码进行修改。

1. 使用IP协议“5”向你的小组成员发送互联网数据报（你需要使用“sudo”来运行“./build/apps/ip raw”程序），并让你的朋友使用 `tcpdump`来确保他们收到了数据报。他们可以运行 `sudo tcpdump -n -i wg0 'proto 5'`来仅打印出协议为“5”的数据报。确保他们收到了！
2. 使用IP协议“17”向你的小组成员发送用户数据报，使用https://www.rfc-editor.org/rfc/rfc768中定义的“用户数据报”头部格式。让你的小组成员在不使用“sudo”的情况下接收此数据报。他们可以使用讲座中使用的“nc -u”程序，或一个使用 `UDPSocket` 类的C++ 程序——无论他们喜欢哪种方式！
3. 请将您的“ip raw.cc”代码包含在提交给此检查点的文件中。
4. 反向操作，并从您的组员那里接收一个数据报。

3 实现部分：将子串按顺序排列

作为实验作业的一部分，您正在实现一个TCP接收器：该模块接收数据报，并将它们转换为可靠字节流，供应用程序从套接字读取——就像您的 `webget` 程序在检查点0中从Web服务器读取字节流一样。

TCP发送器将其字节流分成较短的段（每个子串不超过约1,460个字节），以便每个段都能放入一个数据报中。但网络可能会重新排序这些数据报，或者丢弃它们，或者多次传输它们。接收器必须将段重新组装成它们最初的样子——一个连续的字节流。

在这个实验中，您将编写负责此重组的数据结构：一个`Reassembler`。它将接收子串，这些子串由一个字节串组成，以及该字节串在较大流中的第一个字节的索引。流中的每个字节都有自己的唯一索引，从零开始并向上计数。一旦重组器知道流的下一个字节，它就会将其写入字节流的写入器端——这就是您在检查点0中实现的相同的字节流。重组器的“客户”可以从同一个字节流的读取器端读取。

这是界面看起来像这样：

```
// Insert a new substring to be reassembled into a ByteStream.
void insert( uint64_t first_index, std::string data, bool is_last_substring );
```

```
// How many bytes are stored in the Reassembler itself?
// This function is for testing only; don't add extra state to support it.
uint64_t count_bytes_pending() const;

// Access output stream reader
Reader& reader();
```

> ⋆*Why am I doing this?* TCP robustness against reordering and duplication comes from its ability to stitch arbitrary excerpts of the byte stream back into the original stream. Implementing this in a discrete testable module will make handling incoming segments easier.

The full (public) interface of the reassembler is described by the `Reassembler` class in the `reassembler.hh` header. Your task is to implement this class. You may add any private members and member functions you desire to the `Reassembler` class, but you cannot change its public interface.

3.1 What should the Reassembler store internally?

The `insert` method informs the `Reassembler` about a new excerpt of the ByteStream, and where it fits in the overall stream (the index of the beginning of the substring).

In principle, then, the `Reassembler` will have to handle three categories of knowledge:

1. Bytes that are the **next bytes** in the stream. The `Reassembler` should push these to the stream (`output_.writer()`) as soon as they are known.
2. Bytes that fit within the stream's available capacity but can't yet be written, because earlier bytes remain unknown. These should be stored internally in the `Reassembler`.
3. Bytes that lie beyond the stream's available capacity. These should be discarded. The `Reassembler`'s will not store any bytes that can't be pushed to the ByteStream either immediately, or as soon as earlier bytes become known.

The goal of this behavior is to **limit the amount of memory** used by the `Reassembler` and `ByteStream`, no matter how the incoming substrings arrive. We've illustrated this in the picture below. The "capacity" is an upper bound on *both*:

1. The number of bytes buffered in the reassembled `ByteStream` (shown in green), and
2. The number of bytes that can be used by "unassembled" substrings (shown in red)

```
// How many bytes are stored in the Reassembler itself?
// This function is for testing only; don't add extra state to support it.
uint64_t count_字节_pending() const;

// Access output stream reader
读取器& reader();
```

> ⋆我为什么要做这个？TCP对重新排序和重复的鲁棒性来自于它将任意字节的字节流片段重新拼接回原始流的能力。在一个离散的可测试模块中实现这一点将使处理传入的段更容易。

重组器的完整（公共）接口由 `Reassembler` 类在`reassembler.hh` 头文件中描述。你的任务是实现这个类。你可以向 `Reassembler` 类添加任何你想要的私有成员和成员函数，但你不能改变它的公共接口。

3.1 重组器应该内部存储什么？

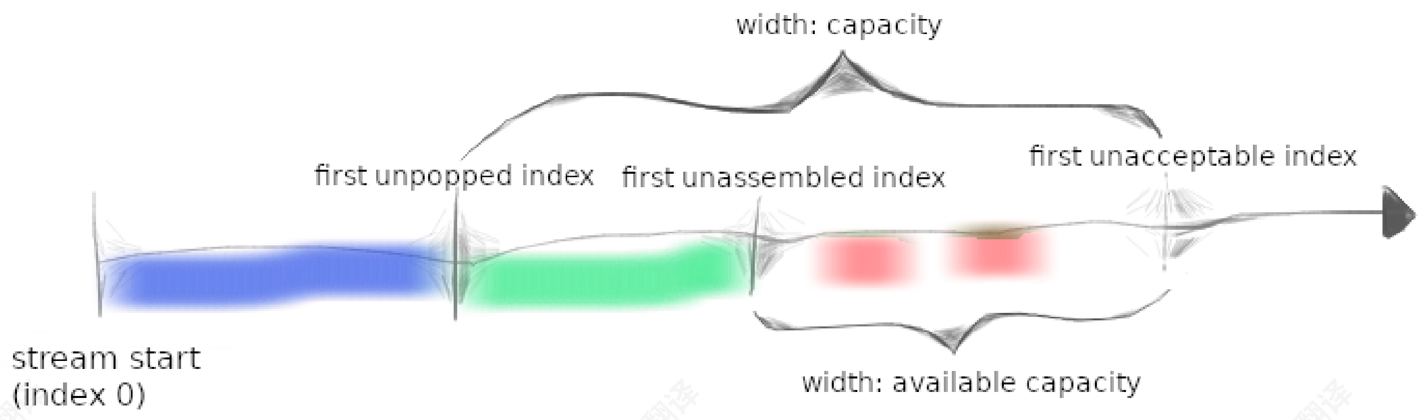
`insert` 方法会通知 `Reassembler`关于一个新的字节流片段，以及它在整个流中的位置（子字符串开始的索引）。

原则上， `Reassembler` 将不得不处理三类知识：

1. 流中下一个字节。 `Reassembler` 应将这些字节推送到流（`output .writer()`），一旦它们被知晓。
2. 可以适应流可用容量但尚不能写入的字节，因为较早的字节仍未知。这些应存储在 `Reassembler`内部。
3. 超出流可用容量的字节。这些应该被丢弃。`Reassembler`不会存储任何无法立即或一旦较早字节已知即可推送到字节流中的字节。

此行为的目的是限制 `Reassembler` 和 `ByteStream` 使用的内存量，无论传入的子串如何到达。我们已在下方图片中说明了这一点。"容量"是两者的上限：

1. 重组 `ByteStream` 中缓存的字节数量（以绿色显示），和
2. "未组装"子串（显示为红色）可使用的字节数



🟥 bytes (substrings) in the Reassembler's internal storage

🟩 bytes buffered in the ByteStream

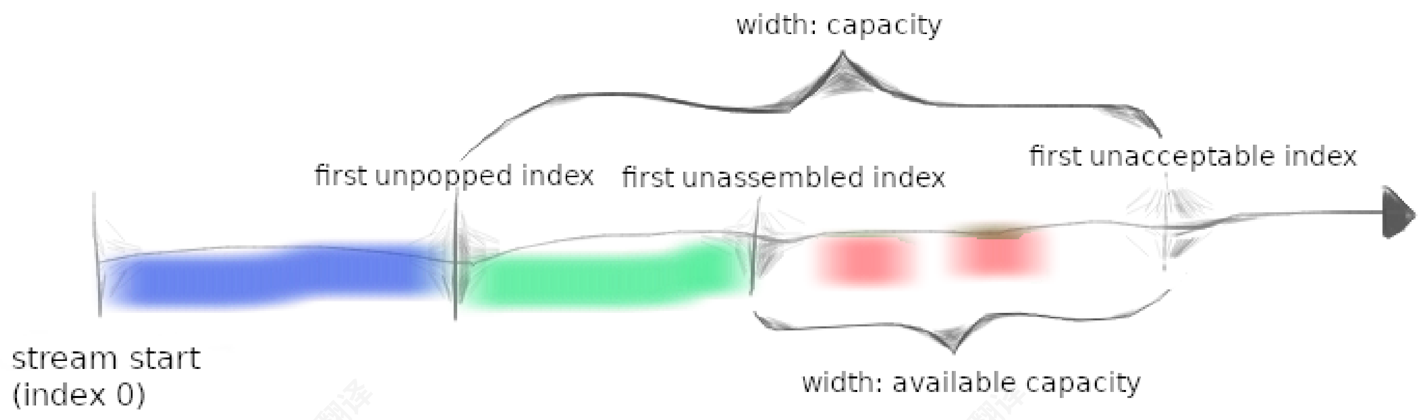
🟦 bytes that have been popped already

You may find this picture useful as you implement the `Reassembler` and work through the tests—it's not always natural what the "right" behavior is.

3.2 FAQs

- *What is the index of the first byte in the whole stream?* Zero.
- *How efficient should my implementation be?* The choice of data structure is again important here. Please don't take this as a challenge to build a grossly space- or time-inefficient data structure—the Reassembler will be the foundation of your TCP implementation. You have a lot of options to choose from.

  We have provided you with a benchmark; anything greater than 0.1 Gbit/s (100 megabits per second) is acceptable. A top-of-the-line Reassembler will achieve 10 Gbit/s.
- *How should inconsistent substrings be handled?* You may assume that they don't exist. That is, you can assume that there is a unique underlying byte-stream, and all substrings are (accurate) slices of it.
- *What may I use?* You may use any part of the standard library you find helpful. In particular, we expect you to use at least one data structure.



🟥 bytes (substrings) in the Reassembler's internal storage

🟩 bytes buffered in the ByteStream

🟦 bytes that have been popped already

在实现 `Reassembler` 并完成测试时，您可能会觉得这张图片很有用——‘正确’的行为并不总是显而易见的。

3.2 常见问题解答

•整个流中第一个字节的索引是什么？零。

• 我的实现应该有多高效？数据结构的选择在这里再次很重要。请不要将其视为构建一个极其空间或时间低效的数据结构的挑战——重组器将是你TCP实现的基础。你有许多选择。

我们为你提供了一个基准；任何大于0.1 Gbit/s（100兆比特每秒）的速率都是可接受的。顶级的重组器将实现10 Gbit/s。

• 如何处理不一致的子串？你可以假设它们不存在。也就是说，你可以假设存在一个唯一的底层字节流，所有子串都是它的（准确的）切片。

•我可以使用什么？你可以使用任何你认为有帮助的标准库部分。特别是，我们希望你至少使用一种数据结构。

- *When should bytes be written to the stream?* As soon as possible. The only situation in which a byte should not be in the stream is that when there is a byte before it that has not been "pushed" yet.
- *May substrings provided to the `insert()` function overlap?* Yes.
- *Will I need to add private members to the Reassembler?* Yes. Substrings may arrive in any order, so your data structure will have to "remember" substrings until they're ready to be put into the stream—that is, until all indices before them have been written.
- *Is it okay for our re-assembly data structure to store overlapping substrings?* No. It is possible to implement an "interface-correct" reassembler that stores overlapping substrings. But allowing the re-assembler to do this undermines the notion of "capacity" as a memory limit. If the caller provides redundant knowledge about the same index, the `Reassembler` should only store one copy of this information.
- *Will the Reassembler ever use the Reader side of the ByteStream?* No—that's for the external customer. The Reassembler uses the Writer side only.
- *How many lines of code are you expecting?* When we run `./scripts/lines-of-code` on the starter code, it prints:

```
ByteStream:     90 lines of code
Reassembler:    26 lines of code
```

  and when we run it on our solutions, it prints:

```
ByteStream:    109 lines of code
Reassembler:    87 lines of code
```

  So a reasonable implementation of the `Reassembler` might be about 50–60 lines of code for the `Reassembler` (on top of the starter code).
- *More FAQs*: For more, please see https://cs144.github.io/lab_faq.html.

4 Development and debugging advice

1. You can test your code (after compiling it) with `cmake --build build --target check1`.
2. Please re-read the section on "using Git" in the Lab 0 document, and remember to keep the code in the Git repository it was distributed in on the `main` branch. Make small commits, using good commit messages that identify what changed and why.
3. Please work to make your code readable to the CA who will be grading it for style and soundness. Use reasonable and clear naming conventions for variables. Use comments to explain complex or subtle pieces of code. Use "defensive programming"—explicitly

- 字节应该何时写入流中？应尽快写入。只有一种情况字节不应在流中，那就是当它前面的字节尚未被“推送”时。
- 提供给 *insert()* 函数的子串可以重叠吗？可以。
- 我需要向重组器添加私有成员吗？是的。子串可能以任何顺序到达，因此你的数据结构必须“记住”子串，直到它们准备好被放入流中——也就是说，直到它们之前的所有索引都已被写入。
- 我们的重组数据结构存储重叠子串可以吗？不可以。可以实现一个“接口正确”的重组器来存储重叠子串。但允许重组器这样做会破坏“容量”作为内存限制的概念。如果调用者提供了关于同一索引的冗余信息， `Reassembler` 应仅存储一份该信息。
- 重组器会使用字节流读取器一侧吗？不会——那是外部客户。重组器仅使用写入器端。
- 您期望多少行代码？当我们在 `./scripts/lines-of-code`上运行启动代码时，它会打印：

```
ByteStream:     90 lines of code
Reassembler:    26 lines of code
```

  而当我们在我们的解决方案上运行它时，它会打印：

```
ByteStream:    109 lines of code
Reassembler:    87 lines of code
```

  因此， `Reassembler` 的一个合理实现可能是大约 50–60 行代码，用于 `Reassembler` （在启动代码之上）。
- 更多常见问题解答：更多内容，请参阅 https://cs144.github.io/lab faq.html。

4 开发和调试建议

1. 你可以测试你的代码（在编译之后）使用 `cmake --build build --target check1`.
2. 请重新阅读实验 0 文档中关于“使用 Git”的部分，并记住要保留代码在分配的 `main` 分支上的 Git 仓库中。进行小规模提交，使用良好的提交信息，说明变更内容和原因。
3. 请努力让你的代码对评分风格和正确性的助教（CA）来说易于阅读。为变量使用合理且清晰的命名规范。使用注释解释复杂或微妙的代码部分。使用“防御性编程”——明确

check preconditions of functions or invariants, and throw an exception if anything is ever wrong. Use modularity in your design—identify common abstractions and behaviors and factor them out when possible. Blocks of repeated code and enormous functions will make it hard to follow your code.

4. Please also keep to the "Modern C++" style described in the Checkpoint 0 document. The cppreference website (https://en.cppreference.com) is a great resource, although you won't need any sophisticated features of C++ to do these labs. (You may sometimes need to use the `move()` function to pass an object that can't be copied.)

5. If you get your builds stuck and aren't sure how to fix them, you can erase your `build` directory (`rm -rf build`—please be careful not to make a typo as this will erase whatever you tell it), and then run `cmake -S . -B build` again.

5 Submit

1. In your submission, please only make changes to the `.hh` and `.cc` files in the `src` directory. Within these files, please feel free to add private members as necessary, but please don't change the *public* interface of any of the classes.

2. Before handing in any assignment, please run these in order:

   (a) Make sure you have committed all of your changes to the Git repository. You can run `git status` to make sure there are no outstanding changes. Remember: make small commits as you code.

   (b) `cmake --build build --target format` (to normalize the coding style)

   (c) `cmake --build build --target check1` (to make sure the automated tests pass)

   (d) Optional: `cmake --build build --target tidy` (suggests improvements to follow good C++ programming practices)

3. Write a report in `writeups/check1.md`. This file should be a roughly 20-to-50-line document with no more than 80 characters per line to make it easier to read. The report should contain the following sections:

   (a) **Structure and Design.** Describe the high-level structure and design choices embodied in your code. You don't need to discuss in detail what you inherited from the starter code. Use this as an opportunity to highlight important design aspects and provide greater detail on those areas for your grading TA to understand. What data structures did you choose in your header file? Are any of them not *strictly* necessary? We'd like you to avoid redundant state if at all possible, unless you think and can justify that there's a serious performance penalty from doing so. You are strongly encouraged to make this writeup as readable as possible by using subheadings and outlines. Please do not simply translate your program into an paragraph of English.

检查函数的前置条件或不变式，如果有任何错误就抛出异常。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它们提取出来。重复的代码块和巨大的函数会使代码难以理解。

4. 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，尽管你不需要用到C++ 的任何复杂特性来做这些实验。（你可能有时需要使用 `move()` 函数来传递一个不能被复制的对象。）

5. 如果你遇到构建卡住且不确定如何修复，你可以删除你的 build目录（ `rm -rf build`——请小心不要输入错误，因为这将删除你告诉它的任何内容），然后再次运行 `cmake -S . -B build` 。

5 提交

1. 在你的提交中，请仅修改 `src`目录中的 `.hh` 和 `.cc` 文件。在这些文件中，请根据需要自由添加私有成员，但请不要更改任何类的公共接口。

2. 提交任何作业前，请按顺序运行以下操作：

   (a) 确保您已将所有更改提交到 Git 仓库。您可运行 `git status` 以确保没有未解决的更改。记住：边编写代码边提交小批量更改。

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

   (c) `cmake --build build --target check1` (以确保自动测试通过)

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

3. 在 `writeups/check1.md`中撰写报告。该文件应是一个大约20至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) 结构与设计。描述代码中体现的高级结构和设计选择。你不需要详细讨论从启动代码继承的内容。利用这个机会突出重要的设计方面，并为你的评分助教提供更详细的说明。你在头文件中选择了哪些数据结构？它们中有任何不是严格必要的吗？我们希望你尽可能避免冗余状态，除非你认为并能够证明这样做会带来严重的性能损失。强烈建议你通过使用子标题和提纲使这份文档尽可能易读。请不要简单地将你的程序翻译成一段英文。

(b) **Alternative design choices** that you considered or ideally evaluated in terms of their performance, difficulty to write (e.g., hours required to produce a bug-free implementation), difficulty to read (e.g., lines of code and their degree of subtlety or nonobvious correctness), and any other dimensions you think are interesting for the reader (or for your own past self before you did this assignment). Include any measurements if applicable. **We really care about helping you grow your ability to imagine and describe the realistic implementation options and weigh their tradeoffs.**

(c) **Implementation Challenges.** Describe the parts of code that you found most troublesome and explain why. Reflect on how you overcame those challenges and what helped you finally understand the concept that was giving you trouble. How did you attempt to ensure that your code maintained your assumptions, invariants, and preconditions, and in what ways did you find this easy or difficult? How did you debug and test your code?

(d) **Remaining Bugs.** Point out and explain as best you can any bugs (or unhandled edge cases) that remain in the code.

4. In your writeup, please also fill in the number of hours the assignment took you and any other comments.

5. The mechanics of “how to turn it in” will be announced before the deadline.

6. Please let the course staff know ASAP of any problems at the lab session, or by posting a question on Ed. Good luck!

(b) 你考虑过或理想中评估过的替代设计方案，包括其性能、编写难度（例如，实现无bug所需的小时数）、可读性（例如，代码行数及其微妙程度或非显而易见的正确性），以及其他你认为对读者（或对你完成此作业前的自己）有趣的维度。如果适用，请包含任何测量结果。我们非常重视帮助你提升想象和描述实际实现方案以及权衡其利弊的能力。

(c) 实现挑战。描述你发现最棘手的部分代码，并解释原因。反思你是如何克服这些挑战的，以及是什么帮助你最终理解了让你困扰的概念。你是如何尝试确保你的代码保持你的假设、不变量和前提条件的，以及你发现这容易还是困难？你是如何调试和测试你的代码的？

(d) 剩余的bug。指出并尽可能解释代码中仍然存在的任何bug（或未处理的边缘情况）。

4. 在你的报告中，请填写完成作业所花费的小时数以及任何其他评论。

5. 提交的“具体操作”将在截止日期前公布。

6. 请尽快告知课程工作人员实验室期间遇到的问题，或在 Ed 上发布问题。祝你好运！