根据部分同学比较困惑的地方的反馈,我们更新了拥塞控制项目的文档,作为版本 2,文档如下。这里把上一个版本叫做版本 1。以下是一些说明:

- 版本 2 和版本 1 所有的内容完全一样,文档结构(除了图片的标号以外)也完全一样。如果你已经完整阅读版本 1，请不用担心，你不需要阅读版本 2。

- 版本 2 主要是对版本 1 里一些也许不太好懂的句子进行了改进,对有些字体进行了优化以方便阅读。

- 建议阅读方法:

  -如果你已经阅读了版本 1，并且没有不懂的地方:不需要阅读版本 2。

  ——如果你已经阅读了版本 1,但有些句子还没有理解:可以根据文档结构,在版本 2 中找对应的句子或者段落,看新版本的句子是否能理解,如果仍不能理解,请在讨论上提问。

  ——如果你还没有阅读版本 1:可以直接阅读版本 2(当然也可以选择版本 1,只需要阅读其中一个即可)。

- 再次强调,版本 2 和版本 1 内容完全一样,如果没有任何不懂的地方,只需要阅读其中一个版本即可,版本 2 只是对版本 1 中句子和词语的一个优化的版本,只是为了帮助大家的理解,并没有添加任何新的内容。

项目:P2P 文件传输 UDP 与拥塞控制

2022 年 12 月 14 日

在这个项目中，您需要构建一个可靠的具有拥塞控制的 P2P(P2P)文件传输应用程序。你需要在这个应用程序中实现两个主要部分:

类 p2p 架构下的可靠数据传输(RDT)，包括握手和传输文件块环;

II .类 p2p 文件传输的拥塞控制。

注意本项目采用 UDP 协议作为传输层协议。I 和 II 都是在应用层实现的。第一部分对应一个类似 bittorrent 的协议，即。7 .我们的教科书《计算机网络:一种自上而下的方法》第 2.5 节中介绍的 P2P 文件传输 *th 版在酒井上有售*。第二部分建立在第一部分的基础上，在应用层实现了一种类似 tcp 的 P2P 文件传输协议。可靠的数据传输和拥塞控制的思想可以在第 3.4-3.7 节中找到在我们的课本上。

请仔细阅读本文档，了解所提供的内容以及您希望实现的内容。材料，如文档、入门文件、问答，可在 https: //github.com/SUSTech-CS305-Fall22 获取。我们还为这个项目录制了一段教程。强烈建议你观看它(CS305 计算机网络项目教程- EN)。

本文档组织如下。在第 1 节中，对本项目进行了概述，包括将在第一部分和第二部分中使用的重要术语。在第 2 节中，提供了关于 P2P 文件传输的实现细节。在第 3 节中，我们给出了可靠数据传输和拥塞控制的实现细节。第 4 节给出了设置过程和提供的文件的描述。在第 5 节中列出了一些例子。最后，重要的注意事项、要求、任务、评分标准等在第六节中列出。

1 项目概述

这个项目模拟 BitTorrent 文件传输过程。在这个系统中，有一个文件和多个对等点。每个对等体最初拥有文件的一部分。在传统的 peer-to-peer(P2P)文件传输系统中，节点可能被指示从其他节点下载某些块。

文件分割:将文件分割成一组大小相等的块(见图 1)，每个块的大小为 512 KiB。为了区分这些块，会为每个块计算一个固定大小为 20 字节的加密哈希。每个块可以通过其哈希值唯一标识。

Peers:一个 peer(客户端)是一个运行着固定主机名和端口的程序。最初，每个对等端(客户端)持有多个块，这些块不一定是连续的。对等端所拥有的区块集被称为文件的一个片段。注意，不同对等体持有的片段可能是不同的，也可能是重叠的。

包:一个包是一个块的一小部分，可以在 UDP 包数据区域。有六种不同类型的包，在小节 1.3 中详细描述。

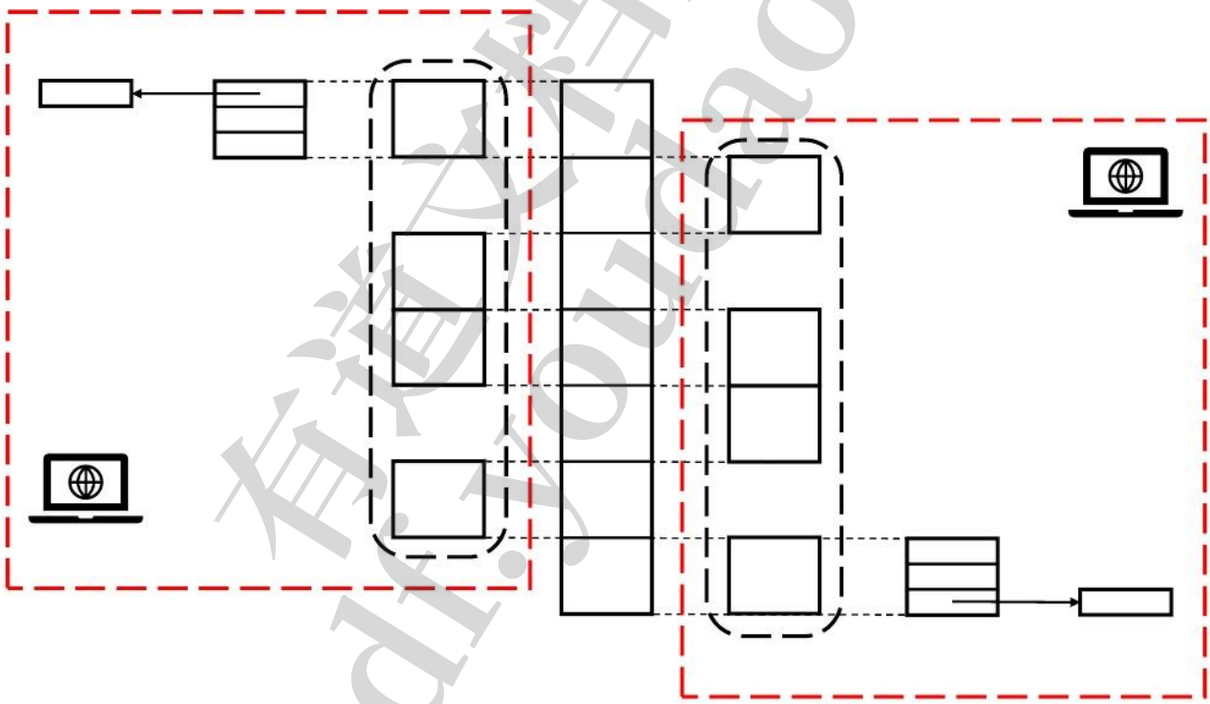

图 1所示。本项目中对等体的文件结构。

文件的重要术语

•*。XXX 表示所有后缀为“XXX”的文件。

一个 chunk 的 chunkdata 是它的 512 KiB 数据字节。

•一个 chunk 的 chunkash 是其 chunkdata 的 20 字节 SHA-1 哈希值。

- *.fragment:形式{chunkash: chunkdata}的序列化字典。它是 peer 的输入文件，运行 peer 时会自动加载到 dictionary 中。参见示例 peer (example/dumbsender. txt)Py, example/dumbreceiver.py)获取详细信息。

- *。Chunkhash:包含 Chunkhash 的文件。的主人。Chunkhash 文件包含了该文件的所有 Chunkhash，以及一个 downloadxxx.;Chunkhash 文件告诉 peer 要下载的内容。

1.1 文件传输流程概述

可以通过 stdin 中的 DOWNLOAD 命令命令对等体下载一个数据块列表，该文件包含要下载的数据块的散列。在收到这样的命令后，对等体检查自己的片段，并尝试向其他对等体请求剩余的块。在这个过程中，会有握手和数据传输的过程。对等体可以并发地(不是并行地)向多个对等体请求 chunk——虽然任何特定的对等体都可以同时向其他对等体请求 chunk，但这些 chunk 的数据包应该被并发地接收。例如，peerA 可能从 peerB 获得 chunk1 的 pkt1，然后从 peerC 获得 chunk2 的 pkt2，然后从 peerA 获得 pkt2。这个过程在 2 中描述。你的程序应该能够正确地将不同的数据包分类到相应的块中，因为在这个项目中只允许单线程实现，详见第 2 节。为了发送一个块，一个对等体需要将它分成多个数据包，然后通过 UDP 传输它们。一旦一个节点接收到所有请求的数据块，它将用它们的散列重新组装到一个新的字典中，并将其序列化到一个二进制文件中，文件名在 DOWNLOAD 命令中给出。

1.2 可靠的数据传输和拥塞控制概述

考虑一对对等体，一个向另一个请求数据块。通过握手建立连接后，捐赠对等体将区块传输给接收对等体。与 TCP 协议一样，本项目要求您通过多种技术实现可靠的数据传输，包括序列号、超时和重传确认(ACK)。为了实现拥塞控制，您应该实现一个拥塞窗口，其窗口大小由超时和 ack 等触发器动态调整。如前所述，可靠的数据传输和拥塞控制都应该在应用层实现，而不是在传输层。详情请阅读第 3 节。

1.3 报文格式

在本项目中，数据包格式定义在 Tab. 1 中。最大数据包长度(头+负载)是 1400 字节，这样你就可以从 UDP 中读取整个数据包(因为一个最大可能长度为 1480 字节的数据包可能由于 UDP 问题而损坏)。报头可以扩展

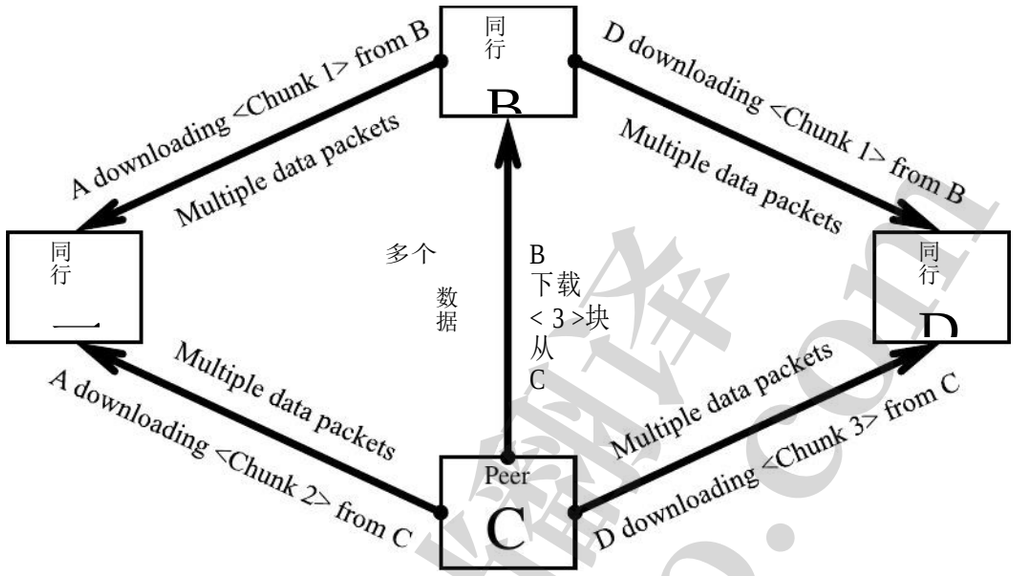

图 2 所示。显示了一个由四个对等点同时处理多个发送和接收过程的系统的示例。

以辅助您的设计，但不能删除现有字段。注意，little-endian 用于所有的包。

数据包由两个主要部分组成——报头和有效载荷，就像我们从课堂上学到的大多数数据包一样。包的简化结构如表 1 所示。

Magic(2 字节)	团队(1 字节)	类型代码(1 字节)
报头长度(2 字节)	Packet Length(2 字节)	
序列号(4 字节)		
ACK 号(4 字节)		
有效载荷		

选项卡。1。 包结构。

1.3.1 报头

头文件字段的含义:

•Magic:魔法数字为 52305。这是一个常数数字，用于识别协议。

•团队:你的团队索引。你的队伍索引是 QQ 文档的第一栏，不能是 305，这个是留作测试用的。

•类型代码:此包的类型。

报头长度:以字节为单位的报头长度。

数据包长度:该数据包的总长度，以字节为单位。

•序列号:数据包的序列号，即。，它计数的是数据包而不是字节。此字段仅对 DATA 包有效。对于其他报文，始终为 0。

•ACK 号:确认号码。只对 ACK 报文有效。对于其他报文，始终为 0。

1.3.2 包类型

数据包总共有六种类型，它们通过头中的类型代码字段进行区分。类型代码如表 2 所示。

包类型	类型代码
情侣一样	0
我	1
得到	2
DATA	3.
ACK	4
否认	5

选项卡。2。数据包类型和对应的类型代码

1.3.3 如何在 python 中制作包

使用结构体。参考我们的示例代码和文档 python struct 文档。

2 第一部分:对等文件传输协议

要完成第一部分-P2P 文件传输，您需要

- 在启动时设置对等体:通过给定的拥有块和其他对等体的位置(即主机名和端口)初始化对等体。

- 实现 P2P 文件传输协议，包括

  —**Listening:通过侦听接收 UDP 报文的 socket 和用户命令获取可下载的 chunk 集合。**

  —**握手:对等体收到用户命令后，需要与其他对等体握手建立连接，才能进行 chunk 传输。**

  —**数据块传输:在与其他对等体建立连接后，对其他对等体进行数据块传输。**此外，还将实现一个拥塞窗口，这将用于第二部分的拥塞控制。

在本节的剩余部分，我们将展示如何首先启动对等体。然后，是 P2P 文件传输过程，包括监听、握手、chunk 传输。

2.1 设置对等体 2.1.1 准备 Chunk 文件

首先，我们需要准备网络中的块文件，并为每个对等体生成分片文件。提供了一个 make_data.py 脚本来执行这样的任务:

Python3 make_data.py <输入文件> <输出文件> < chunk 的 num > <索引> .py

- <input file>:要被分割成块的文件。它可以是任何二进制文件，如*.tar 或*.zip。请注意，大小小于 512 KiB 的文件不能分割为块。

- <输出文件>:A *.fragment 文件，这是一个形式为{chunkash: chunkdata}的序列化字典。其中的 chunkash 由<index>.选择。

- <num of chunks>:分区后要保留的 chunk 数量。如果对于大小为 2051 KiB 的<input file>，该值设置为 3，它将只保留<index>的 4 个块中的前 3 个块。如果它被设置为 5，它将使用 4 而不是 5，因为 5 是越界的。请注意，不能形成块的最后 3 个 KiB 将被丢弃。

- <index>:逗号分隔的索引，表示要选择到<output 文件>中的块。例如，“2,4,6”表示选择 chunk2, chunk4 和 chunk6。索引从 1 开始，而不是 0。

2.1.2 对等体配置

您将需要通过告诉(1)它已经拥有哪些块和(2)其他对等点的位置来配置每个对等点。要启动一个对等点，有几个参数的形式是

Python3 peer.py -p <peer 文件> -c <haschunk 文件> -m <max send> -i <identity> .py .p <peer 文件> -c <haschunk 文件> -m <max send> -i <identity> .p -v < verbose > [-t <timeout>]

- <peer file>:该字段对应对等文件的路径。它包含了所有对等体的身份以及相应的主机名和端口。然后，对等体就知道网络中的所有其他对等体。
- <haschunk file>:这是一个*.fragment 文件。它是一个形式为{chunkash: chunkdata}的序列化字典。该文件由 make_data.py 生成。它会被自动加载。
- <max send>:该对等体能够向其发送数据包的最大对等体数量。如果另一个对等体在这个对等体饱和时从这个对等体请求 chunk，它应该发送回一个 DENIED 包。
- <timeout>:如果没有设置 timeout，你应该估计网络中的 RTT 来设置你的超时值。但是，如果设置了，你应该总是使用这个值。一个预定义的超时值将在测试中使用。
- <identity>:该对等体的身份(ID)。这个身份应该被对等体用来从<peer 文件>中获得自己的位置(即主机名和端口)。然后，它可以使用这个位置来启动一个套接字来侦听数据包。
- <verbosity>:详细级别。从 0 到 3。

对等体设置的详细示例可以在第 5 节中找到。

2.2 听

每个 peer 将继续侦听 UDP 套接字和用户输入，直到终止。如果对等体接收到 UDP 报文或用户输入，应该分别按照如下说明对报文或用户输入进行处理。如果对端在一定时间内没有收到任何报文或用户输入，则返回空消息。

监听用户输入:要下载块，用户会输入

DOWNLOAD<要下载的块> <输出文件名>

- <chunks 下载>:A *.;Chunkhash 文件包含要下载的 chunk 的哈希值。对等体应该下载这个文件中列出的所有块。

•<output filename>: *.fragment 文件的名称。它应该是一个存储{chunkhash: chunkdata}的序列化字典，其中 chunkhash 是<chunkhash 下载>中的哈希，chunkdata 是下载的数据。

然后，当接收到这样的用户输入时，对等端将从<chunks 中的文件中读取，下载命令中给定的>。之后，对等体将需要根据哈希值从其他对等体下载 chunk 数据，并将下载的 chunk 组装到字典中。最后，对等体将序列化后的字典写入<output filename>并打印。

监听套接字(Listening Socket):如果从套接字传输了 UDP 报文，对端应根据报文的类型和内容进行处理。查看我们的框架代码和示例，了解如何进行监听。

2.3 握手

正如 2.2 节中提到的，在接收到用户的 DOWNLOAD 命令后，对等端应该收集所有请求的块数据。将会有两个过程:与其他对等体握手和块传输(将在章节 2.4 中讨论)。

握手过程包括三种类型的消息:WHOHAS、IHAVE 和 GET。具体来说，对等体将通过类似于 TCP 协议的“三向握手”与其他一些对等体建立连接。“三次握手”可以这样描述:

1.节点将 WHOHAS 包发送给网络中之前已知的所有节点，以检查哪些节点拥有请求的块数据。WHOHAS 包包含一个块哈希列表，表示对等体需要哪些块。

2.当其他节点从这个节点接收到 WHOHAS 的数据包时，它们应该查看各自拥有哪些请求的数据块。它们将用 IHAVE 包返回给这个节点。彼此发送 IHAVE 包包含请求块的哈希，它拥有。但是，如果这个节点在接收到一个新的 WHOHAS 时已经发送了<最大发送>数量的其他节点，它应该发送回拒绝。

3.一旦一个节点收到了来自其他节点的所有 IHAVE 报文，它就知道了其他节点所拥有的数据块。然后，对等体将选择特定的对等体，分别从其中下载每个请求的块。它将发送 GET 包，其中包含请求块中的一个的哈希值到每个特定的对等点，用于块下载。例如，如果对等体决定从对等体 1 下载块 A，那么它将发送包含块 A 散列的 GET 包

同行 1。

请注意，在第 3 步中，由于一个节点可以同时接收来自不同节点的多个不同块，因此该节点可以向多个不同的节点发送多个 GET 包。但是，它应该发送在

大多数节点同时 GET 报文。经过“三次握手”，对等体与收到 GET 报文的对等体建立了连接。然后，这些对等体中的每一个都开始向请求对等体传输 chunk 数据。

2.4 数据块传输

任何时候只能有一个数据块从一个对等体传输到另一个对等体。但是一个对等点可以同时从不同的对等点接收多个不同的块，也可以同时向不同的对等点发送多个块。

如图 2 所示的例子，对等体 A 同时从 B 和 C 接收到不同的块。对等体 B 向 A 和 D 发送块数据，同时从 C 接收数据。

3 第二部分:可靠的数据传输和拥塞控制

在完成第 2 节中的 P2P 文件传输后，您将需要实现可靠的数据传输和拥塞控制(RDT)。请注意，RDT 只适用于像 data 这样传输数据包的数据，不需要为 WHOHAS、IHAVE 和 GET 等功能性数据包维护 RDT。

为了实现可靠的数据传输，需要实现超时和三次重复 ack 触发的重传。要实现拥塞控制，可以考虑使用带算法的滑动窗口协议，根据 3.2 节的说明调整窗口大小。请注意，与实际的 TCP 拥塞控制不同，在这个项目中窗口大小是以包为单位的，而不是字节。

3.1 可靠的数据传输

在您的协议中，您需要实现由超时和三个重复的 ack 触发的重传。快速重传)，如在 TCP 协议中。

3.1.1 超时

请记住，在预先指定的时间内未能收到 ack 将被视为超时。当超时发生时，发送方(即。发送数据包的对端需要重传导致超时的数据包。

正如在讲座中介绍的那样，您需要估计 RTT 来确定超时间隔。我们建议您使用教材 3.5.3 节给出的 RTT 公式。也就是说，使用计算 EstimatedRTT

$$EstimatedRTT=(1-\alpha)\cdot EstimatedRTT+\alpha\cdot SampleRTT \quad (1)$$

$\alpha = 0.125$，计算 DevRTT

$$DevRTT=(1-\beta)\cdot DevRTT+\beta\cdot|SampleRTT-EstimatedRTT| \quad (2)$$

用 $\beta = 0.25$。TimeoutInterval 设置为

$$TimeoutInterval=EstimatedRTT+4\cdot DevRTT。 \quad (3)$$

详情请参阅课本第 3.5.3 节。

此外，您还可以自定义确定超时间隔或 RTT 估计(或两者)的方法。但我们强烈建议您使用自适应 RTT 来计算超时间隔。

3.1.2 快速重传:3 个重复的 ack

当发送方收到三个重复的 ack 时，您应该假设序列号= (ACK number + 1)的数据包丢失了，即使没有超时。发送方需要重新发送。关于快速传输的更多细节和重要注意事项将在第 3.2.3 节中介绍。

拥塞控制

您应该在应用层设计一个算法来控制发送端窗口的大小，以实现类似于 TCP 协议的拥塞控制机制。

窗口大小用 cwnd 表示，是根据数据包的数量来定义的。例如，窗口大小为 1 的对等体意味着它在任何时候最多只能发送一个未封装的数据包。主要有两种状态:慢启动和拥塞避免。状态转换图如图 3所示。

3.2.1 慢启动

最初，设置 cwnd 为 1(即一个数据包)。在每收到一个 ACK 时增加 cwnd。发送端不断增加窗口大小，直到检测到第一个丢包，或者窗口大小达到 ssthresh 值，之后进入拥塞避免模式。对于一个新的连接，ssthresh 被设置为一个非常大的值，64 个包。如果一个包在慢启动时丢失，发送方将 ssthresh 设置为 max(cwnd/2 ，2)。

3.2.2 拥塞避免

在收到 ACK 时，增加窗口大小 1/cwnd 包。由于每次发送的报文必须是整数，所以应该使用 cwnd 。类似于慢启动，如果网络中出现了丢失(由于超时或重复 ack 导致)，ssthresh 被设置为 max (cwnd/2, 2)。然后 cwnd 被设置为 1，系统将再次跳转到“慢启动”状态。

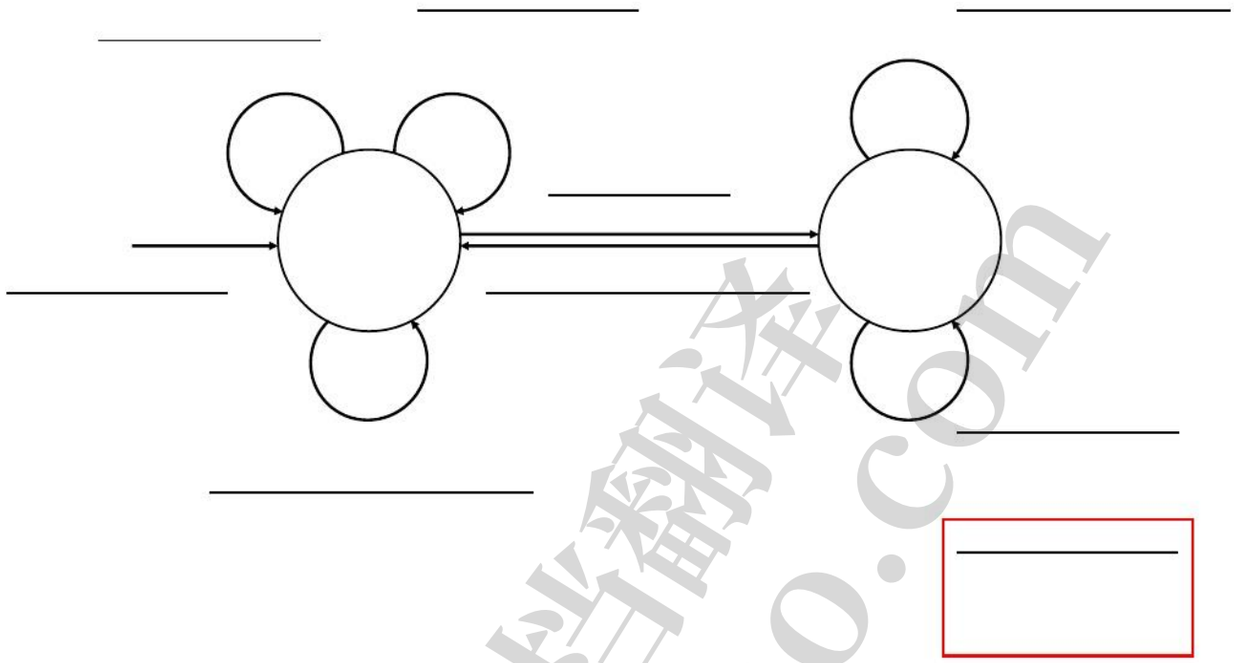

图 3所示。本项目拥塞控制的状态转换图。

3.2.3 快速重传

与慢启动和拥塞避免不同，快速重传并不是一种状态，它只是在接收到 3 个重复 ack 时进行重传而不是等待超时。请注意，每一轮都计算重复 ACK，也就是说，每一轮 SEQ-ACK 都有自己的重复 ACK 计数器。为了简化你的设计，当 duplicateACK==3 时，Fast Retrasnmit 每轮最多被触发一次。

本文不涉及拥塞控制的详细理论，有兴趣的请参考教材 3.6-3.7 节(302-326 页)。

需要注意的是，你的任务是在应用层实现一个“窗口控制算法”来达到类似于 TCP 协议的拥塞控制的效果，而不是 TCP 协议在传输层的拥塞控制机制。

4 设置和提供的文件

4.1 搭建开发环境

本项目在 Ubuntu 20.04 上测试，使用 Python 3.8，不支持 Windows(理论上应该可以在 Mac 上运行)。你可以根据各种教程在本地设置自己的虚拟机或容器。如果你不喜欢设置本地环境，我们也会提供远程容器(每个容器有 2 个 CPU 核心和 2 个 GiB RAM)和适当的环境。虽然我们希望

始终提供可靠的远程容器，我们不能保证远程容器的可靠性，因为当 DDL 接近时，服务器可能会过载。因此，您可能需要频繁地将代码同步到 GitHub。我们会尽量保持它的可靠性。

如何请求远程容器

发送邮件至 12132341@mail.sustech.edu.cn，格式:

标题:Request for remote container-Group{Your-group-number}正文:
你的团队成员，谁将是 sudoer。

注意，每个小组只能请求一个容器，每个容器有 3 个用户。会有一个拥有超级用户权限的用户。我们会尽快回复您。回复中将包括一个简短的文档，说明如何连接容器以及如何使用容器。

4.2 从 GitHub 开始设置存储库

首先，从 GitHub 检索骨架代码:

git 克隆/ github .com/ sustech - cs305 - fall22 /CS305-Project-Skeleton.git
Git 远程重命名原点五线谱

然后在 GitHub 上创建一个私有存储库，并运行
git remote add group <Your_REPO-URL> git
push group main -u

你可能需要配置你的 git 与 SSH 或令牌。更多信息可以从添加 ssh 的 GitHub 中检索。这将使你能够在自己的私有存储库中同步你的代码，同时仍然能够从人员存储库中提取更新。

当新的完整测试在人员存储库上发布时，你可以将它们拉出来

Git 拉人员

4.3 提供的文件

所有提供的文件的结构
为:ProjSkeleton/
|——示例
| |——dumbreceiver.py |
|——dumbsender.py

```
| |——ex_file.tar
| |——
ex_nodes_map . tar
| '——ex_topo.map
| - - -
src
| '——peer.py
|- - -
测试
| |——basic_handshaking_test.py
| |——checkersocket.py | |——
grader.py
| '——tmp1 .
py
” __
—跑
龙套
    |——__init__ .
    py
    |——
    bt_utils.py
    |——
    hupsim.pl
    |——
    make_data.py
    |——
    nodes.map
    |——
    simsocket.py
    ” ——
    topo.map
```

这 4个目录的作用如下:

提供一个简单的可运行的停止和等待实现来演示如何使用所提供的框架。

src/:你需要提交的目录。你需要使用提供的框架文件 peer.py 在这个目录中编写所有的代码。

•test/:包含一些公共测试，用于检查实现的完整性。测试将在稍后发布。

•util/:提供在这个项目中使用的支持模块和脚本。

一些重要文件:

•src /同行。py:一个框架文件，为你处理一些设置和过程。你应该完成这个文件来满足这个项目的要求。你应该只使用提供的 simsocket，不允许使用普通的 socket。

•util / bt_util。py:解析命令行参数的实用程序。您不需要修改该文件。

- util / simsocket。提供一个修改过的套接字类 SimSocket，它可以在使用或不使用模拟器的情况下运行。NOT 修改这个文件。

•util / humsim。pl:一个用 Perl 编写的网络模拟器。它可以模拟路由、排队、拥塞和丢包。

•util /节点。map:网络中对等点及其对应地址的列表。

•util /威尼斯平底渔船。map:向模拟器提供网络拓扑结构。

•util / make_data。py:一个 python 脚本，用于将文件分割成块并生成 chunkash，它的用法将在后面的示例中详细说明。

•测试/年级。py:为测试提供评分会话。

•测试/ basic_handshaking_test。py:测试脚本。可以被 pytest 调用。

•/ dumbreceiver 示例。py:接收端停止等待的简单实现，读取用户输入并处理下载。

•/ dumbsender 示例。py:发送端 stop-and-wait 的简单实现，响应数据包并发送数据。

4.4 网络模拟器

为了测试您的程序，您将需要具有丢失、延迟和许多节点导致拥塞的网络。因此，我们创建了一个简单的网络模拟器“Spiffy”，它可以在您自己的机器上运行。该模拟器由 hupsim 实现。Pl，它在文件拓扑指定的节点之间创建了一系列带宽有限的链接和队列大小。Map(然后可以测试拥塞控制)。要在虚拟网络上运行您的 peer，您需要在之前设置一个环境变量 SIMULATOR

同行:

导出 SIMULATOR="<模拟器 ip>: <模拟器端口>"

然后从另一个 shell 运行模拟器:

Hupsim.pl -m <拓扑文件> -n <节点文件> -p <监听端口> -v < verbose >

•<拓扑文件>:这是包含 hupsim.pl 将创建的网络配置的文件。一个例子是 topo.map。文件中的 IDs 应该与<nodes file>中的 IDs 匹配。每一行用五个属性定义了网络中的一个链接——<src, dst, bw, delay，队列大小>。bw 是链路的带宽，单位是比特/秒。时延是毫秒级的时延。队列大小以包为单位。您的代码 NOT 读取此文件。如果您需要 RTT 等网络特征的值，则必须从网络行为推断它们。您可以使用指数平均计算 RTT。

- <nodes file>:这是包含网络中所有节点的配置信息的文件。一个例子是 nodes.map。
- <listen port>:这是 hupsim.pl 将侦听的端口。因此，该端口应该与网络中节点使用的端口不同。
- <verbosity>:你希望从 hupsim.pl 中看到多少调试消息。这应该是 1 到 4 之间的整数。数值越大，调试输出越多。

运行模拟器并正确设置环境变量后，只要使用 simsocket，你的 peer 就会自动在模拟器上运行。

5 例子

在我们的例子中，一个文件将被分成 4 个块。Peer1 将有 chunk1 和 chunk2，peer2 将有 chunk3 和 chunk4。将调用 Peer1 从 peer2 下载 chunk3。

5.1 准备 chunk 文件

我们首先生成块数据，对于对等体:

```
Python3 ./util/make_data.py ./example/ex_file.tar ./example/data1.fragment 4
    1,2 .
```

该操作将./example/ex_file.tar 拆分为 4 个 512 KiB 的 chunk，并选择 chunk1 和 chunk2 放入。/example/data1.fragment。./example/data1.fragment 将是一个被 pickle 序列化的字典，在运行时将被反序列化。关于 pickle 的更多信息可以在 pickle 文档中找到。

类似地，我们可以使用

```
Python3 ./util/make_data.py ./example/ex_file.tar ./example/data2.fragment 4
    3,4 .
```

这会生成。/example/data2.fragment，其中包含原始文件的 chunk3 和 chunk4。这也会生成一个.chunkhash，其中包含文件的所有 4 个 chunkhash，命名为 master.chunkhash。chunkhash 文件将像

```
1 12 e3340d8b1a692c6580c897c0e26bd1ac0eaadf
2 45 acace8e984465459c893197e593c36daf653db
3 3 b68110847941b84e8d05417a5b2609122a56314
4 4 bec20891a68887eef982e9cda5d02ca8e6d4f57
```

现在创建另一个 chunkhash 文件，告诉 peer1 下载哪些 chunk:

Sed -n “3p” master。Chunkhash > example/download.chunkhash

Sed是一个方便的命令，可以从文件中选择行。该命令的更多信息可以从 sed manpage 中找到。该命令会生成一个新的 chunkash 文件示例/download。只包含 chunk3 哈希的 chunkash。

5.2 使用模拟器运行示例

现在我们已经为示例准备好了数据块。在下面的小节中，你将需要启动多个 shell 来在不同的进程中运行对等体。

在当前 shell 中启动模拟器:

Perl util/hupsim.pl -m example/ex_topo。Map -n example/ex_nodes_map -p 52305 -v 2 .

启动一个新的 shell，设置环境变量 SIMULATOR 并运行 sender:

导出 SIMULATOR="127.0.0.1:52305"

Python3 example/dumbsender.py -p example/ex_nodes_map -c 示例/data2.fragment -m 1 -i 2 -v 3 .

然后再次启动另一个新的 shell 来运行接收器:

导出 SIMULATOR="127.0.0.1:52305"

Python3 example/dumbreceiver.py -p example/ex_nodes_map -c 示例/data1.fragment -m 1 -i 1 -v 3 .

5.3 无模拟器运行示例

不启动模拟器，只运行 dumbsender 和 dumbreceiver。你会发现它更快!

5.4 在 Dumbreceiver 中调用下载

你可以在接收器的 shell 中输入以下命令(有或没有模拟器):DOWNLOAD example/DOWNLOAD。chunkhash 例子/ test.fragment

这将在接收器中启动下载过程，并将下载的文件保存到 example/test.fragment。您将看到对等体正在运行并打印日志。下载将在 4 分钟左右完成，然后打印:

有例子/ test.fragment

预期 chunkhash: 3b68110847941b84e8d05417a5b2609122a56314 收到 chunkhash: 3b68110847941b84e8d05417a5b2609122a56314

成功接收:True
恭喜!你已经完成了这个例子!

重要注意事项

6.1 实施要求

•这个项目是单线程强制的，你不应该使用任何多线程/多处理/ asyncio 技术。

•你不能使用 python 标准库以外的任何库(除了 matplotlib，它将被预安装)。

你可以扩展头文件，但是你不能修改现有的字段。

•代码的最终测试将在 Ubuntu 20.04 和 Python 3.8 上运行。

6.2 任务总结

任务 1(必选)握手和 RDT:实现基本的沟通，包括握手和可靠的数据传输。

任务 2(必选)拥塞控制:在任务 1 的基础上实现拥塞控制算法。

任务 3(必需的)并发性和健壮性:实现一个并发地向多个对等点发送和接收文件的机制。你的实现应该是单线程的。健壮性意味着您的实现应该能够处理对等崩溃或严重拥塞等问题。我们不会针对超级极端情况测试你的代码，但健壮性将有助于通过综合测试。

任务 4(奖励)优化:尽最大努力优化你的实现，以提高传输文件的吞吐量。注意，你的实现仍然应该是单线程的。

6.3 评分

您的实现将从 3 个方面进行评估:基本(健全)测试，综合测试和优化测试。满分为 100 分，还会有 10 分的加分。所有的基础考试都是公开的，也就是说只要通过考试，你就能获得所有的基础分数。但是，我们只会提供有限的综合测试和优化测试的例子，通过这些例子并不能保证你的最终分数。为了让你的进度保持在正轨上，我们会在不同的检查点发布测试脚本:

- Checkpoint0：11 月 29 日，发布握手测试。
- Checkpoint1：12 月 12 日，发布可靠的数据传输和拥塞控制测试。
- Checkpoint2：12 月 17 日，发布并发测试和综合测试样例。
- Checkpoint3：12 月 22 日，发布鲁棒性测试和优化测试样例。

注意，这些检查点不是强制性的，但遵循它们将对你的进度非常有帮助。

6.3.1 运行测试

首先，安装 pytest:

```
Pip3 安装 pytest
```

然后切换到/test 目录，运行 tests 脚本:

```
cd 测试
Pytest basic_handshaking_test.py -v
```

注意，你应该一个一个地运行测试脚本，不要在一个 pytest 会话中运行它们

```
cd 测试
pytest
```

由于流程和文件问题，这将导致所有测试失败。

6.3.2 基础测试(70 分)

我们为你提供了一些基本的(健全)测试脚本来测试你代码的健全程度。然而，当我们的脚本测试所需的行为时，我们确实理解您的创新设计可能与测试脚本相矛盾。即使你没有通过我们的一些基本测试，如果你能证明原因，你也可以赚回你的学分。基本测试将由评分员在 grade .py 中运行。它代理所有数据包，并在中间分析它们。你也可以使用这个框架编写自己的测试。

- 握手测试(12 分):检查你的同伴是否正确洪泛 WHOHAS，以及握手是否在 WHOHAS，IHAVE 和 GET 上执行。你不需要实现数据传输来通过这个测试。
- 可靠的数据传输测试(12 分):检查在网络有丢包的情况下，你的对等端是否能够可靠地传输数据。

- 拥塞控制测试(22 分):这是一个特殊的测试。您需要打印或绘制您的发送窗口大小随时间的变化，以帮助我们评估您的拥塞控制算法。例如，我们期望看到像图 4 这样的图，它清楚地显示在 25 秒发生丢包，并且通过触发发送窗口大小和 ssthresh 的变化来正确处理丢包。从 0 s 到 12 s 有一个“慢启动”过程，从 12 s 到 20 s 有一个“拥塞避免”过程。在 20 秒左右发生丢包后，窗口大小减小到 0，再次启动“慢启动”。然而，第二次“慢启动”结束得更早，因为 ssthresh 已经减半。

- 并发性测试(12 分):检查你是否可以从不同的对等端并发下载，以及当连接满足给定限制时你是否发送 DENIED。

- 鲁棒性测试(12 分):检查你的 peer 是否能够处理 peer 崩溃和严重丢包等极端情况。我们假设如果一个 peer 崩溃，它将永远不会重新启动。

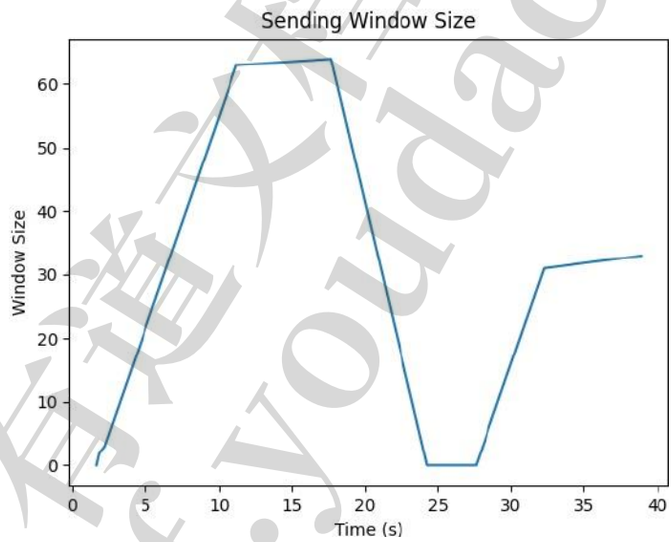

图 4所示。拥塞控制中的发送窗口大小变化

6.3.3 综合测试(30 分)

将有 3 个不同地形的任务。地图，同伴会在模拟器上运行。完成一项任务可以获得 10 分。请注意，综合测试可能运行在大型复杂的网络上，可能会导致对等点崩溃。因此，实现的健壮性将非常重要。我们将发布一些综合测试的例子，你可以在展示你的作品时向我们展示你在这些测试中的表现。你的代码最终会通过类似于发布的隐藏测试进行评估。

6.3.4 优化测试(20 分)

优化测试测试你的拥塞控制实现。评估指标是实现的吞吐量(throughput -put，或 goodput)。你的同伴将运行在一个有瓶颈的网络中。您可以尝试一些技术来提高您的性能，如延迟 ACK 和快速恢复算法。这部分的分数取决于你的性能排名。没能完成优化测试的，这部分会得 0 分。你获得的分数将为:

$$积分= 20 \times rank\_percent$$

例如，如果你的实现优于 80%的学生，你这部分的分数将是 16。

6.4 如何获取帮助

我们在 GitHub 上举办了一个讨论板，请参阅 GitHub 讨论板链接(随时发布您的第一个讨论，向我们问好)。问任何让你困惑的问题，请记住，没有哑巴问题。讨论板的优点是每个人都可以看到你的问题，这样你的经验可能对别人有帮助。你也可以预约助教来讨论你的想法或问题。

6.5 提交和展示什么

你需要提交

- 提交源代码:你应该把你所有的代码文件放在/src 目录下，并将/src 目录压缩到 Team<your Team number>_src.zip 并提交给酒井。
- Presentation:你将做一个简短的 Presentation。你需要给出:

  -幻灯片来描述你的设计概述和一些测试结果(截图)，

  -基本测试，包括你的窗口大小随着时间的推移，

  -综合测试，

  -任何其他可以帮助我们评估你的工作的东西。

6.6 你可以学到的东西

这确实是一个艰难的项目，但完成后，你可能会:
- 对拥塞控制和状态机有更深入的了解。
- 学习如何使用 Python。

•熟悉 Python 中的网络编程。

我们的最终目标是通过适当的培训来帮助你获得知识，而不是让你不知所措。

6.7 审查问题

下面列出了一些问题来帮助你检查你是否跟上了本文的内容。

•什么是 fragment、chunkash 和 chunkdata?*.fragment 文件是什么?*。chunkhash 文件?

•我们有多少种包?

•启动对等体的命令行参数是什么?“-t <timeout>”是什么意思，它如何影响你对超时值的选择?

•快速重传是否会对某个数据包触发两次?

•可以使用哪些库?可以使用多线程吗?

•神奇的数字是什么?

祝你好运，玩得开心!