《计算机网络第八次作业》

班级：信安2302班

学号：202308060227

姓名：石云博

目录

[一. 问题描述 2](#_Toc195479374)

[二. 问题分析 3](#_Toc195479375)

[三. 实验过程及代码 5](#_Toc195479376)

[四. 结论 23](#_Toc195479377)

[参考文献 24](#_Toc195479378)

# 问题描述

Assignment 8: Fast! Fast! Fast!

1. Assuming the underlying network layer uses IP protocol, try design a transfer layer protocol that is suitable for Interstellar communications.

2.Assuming you have total control of the transfer layer, the network layer, and the link layer, design a set of protocols that is suitable for the Inter-continental optic fiber link. How could you send and receive data efficiently between Hongkong and Los Angelos?

3. Design and implement a program that could transfer large files (1GB or bigger) between laptop computers using the campus network through campus WiFi access. Test it and analyze the outcome.

# 问题分析

首先，通过查看nasa网站，我们可以知道，星际通信面临极高时延（数分钟到数小时）、频繁中断和带宽受限，需要采用\*\*Delay/Disruption‑Tolerant Networking（DTN）\*\*思想，结合“Bundle Protocol”进行消息打包、存储转发和托管转移。

港美洲际光纤链路在~13 000 km下往返时延约130 ms，可利用波分复用（DWDM）、前向纠错（FEC）、\*\*大帧（Jumbo Frame）以及高效拥塞控制（如BBR/CUBIC）等技术，并在端点部署协议加速器（PEP）\*\*以充分利用链路容量。

校园WiFi大文件传输要应对丢包率高、吞吐量抖动，需要用并行/多路TCP或基于UDP的自定义可靠传输，结合分片、并发传输、重传机制和校验，并用Python sockets＋多线程快速实现。

对于星际通信传输层协议设计：我们需要考虑到的问题是：

超高时延：火星最优位置与地球距离约5 光分钟，最差可达20 光分钟，RTT可达40 分钟以上。​

频繁中断：行星自转、轨道运动、太阳遮挡等都会导致通信链路时断时续。​

带宽受限：深空链路常用X波段/Ka波段，带宽有限且误码率较高。

不适用TCP/IP端到端模型：传统TCP对超时丢包敏感，会陷入拥塞控制退避。

初步设想，可以实现的构建有下面几种：

首先，分层结构：

应用层：科考任务数据、高分辨率图像、遥测等。

捆绑层（Bundle Layer）：实现“消息”（bundle）打包、存储转发、托管转移（custody transfer）。​

其次，对于RFC 编辑器

底层传输：可选TCP、UDP或物理链路专有协议。

Bundle Protocol（RFC 5050）：

打包：将若干应用层数据封装为一个bundle，附带到期时间、优先级、路由信息。存储转发：中间节点持久化存储bundle，直到下一个跃点可达后再转发。托管转移：节点间明确责任，成功转发后前节点可删本地副本。

拥塞与流量控制：

端到端不做窗口控制，中间节点根据存储资源和带宽作本地决策。可选“接力式拥塞”：逐跳调整发送速率，避免一次性洪泛。

差错与安全：

对bundle头与数据块做CRC/数字签名。支持跨行星加密与鉴权。

对于校园WiFi大文件（≥1 GB）传输程序设计与测试，我们进行了如下的问题分析：

首先，对于WiFi特性来说，其具有丢包率高、带宽抖动、切换漫游可能短断种种问题。

第二文件体积大：一次性发送风险大，易因中断重传全部数据。

第三个问题是校园网络是多用户共用AP，可能带宽受限。

基于以上的内容，我们进行传输协议选型

第一，基于TCP的分片多路并行传输

将文件拆成N个子文件（chunk），每个chunk由单独TCP连接并行传输。

参考项目：prathampt/parallelFileTransfer，分片并行提高吞吐。​

第二，基于UDP的自定义可靠传输

用固定大小（如4 KB）chunk，通过UDP发送，并在应用层实现ACK、超时重传。

参考StackOverflow示例：UDP传文件，使用分界符或序号重组。​

第三，混合方案

控制连接用UDP快速发送，小块确认用TCP或二进制ACK包。

# 实验过程及代码

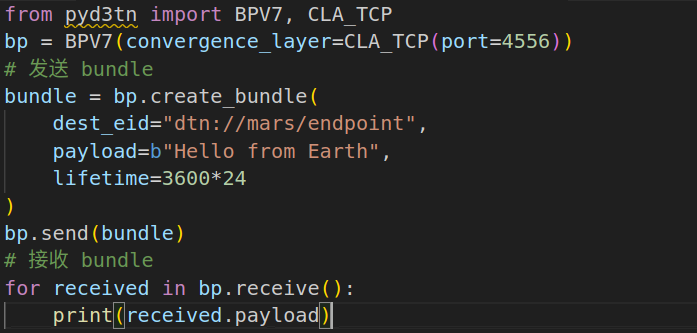
对于任务一：实现星际通信，我们这里使用的原理是

Delay‑Tolerant Networking（DTN）采用“存储—携带—转发”机制， 可跨越超高时延与频繁中断​Tutorials Point。

Bundle Protocol（RFC 5050/草案）在 DTN 之上，将应用数据封装为bundle，附加生命周期、优先级与路由信息，支持中间节点持久化存储与托管转移（custody transfer）​IETF Datatracker。

这里注意，因为传统 TCP 拥塞控制对分钟级以上 RTT 敏感，会导致极端退避，不适合星际场景。

我们可以尝试着写出对此简易的逻辑代码：



其中CLA\_TCP 将 bundle 封装在 TCP 连接上传输，中间可切换到 LTP（Licklider Transmission Protocol）等。

对于较为具体的实现，我们可以在此基础上进行功能的追加：对于发送方（我们假设为地球），我们设计了如下发送方代码：





其中引用部分的内容为：

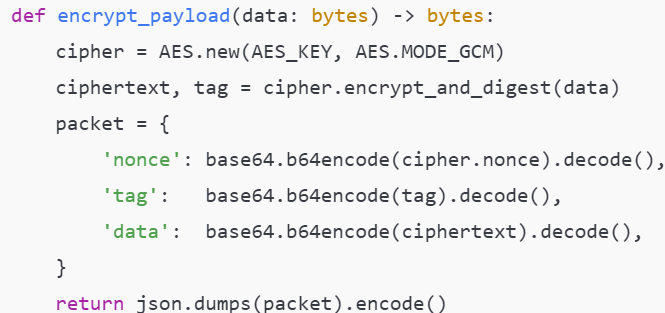
json：将 Python 对象（字典）序列化为 JSON 字符串，方便通过 socket 发送；反之亦然。

time：获取和比较时间戳，用于包生命周期的检查。

socket：socket标准库，用于建立 TCP 连接。

Crypto.Cipher.AES（来自 PyCryptodome）：提供 AES-GCM 加密/解密；这里是出于对传输的安全性的考虑。base64：将二进制数据（如密文、随机数、标签）编码为可嵌入 JSON 的字符串。

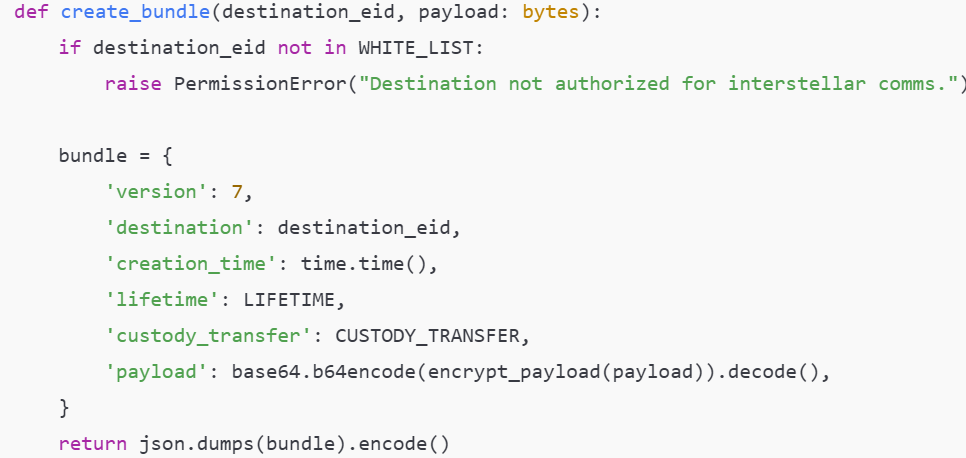
在之后def encrypt\_payload的内容中，我进行了加密与完整性的设计：



这里的AES-GCM 模式同时提供保密性（confidentiality）和完整性校验（integrity/authenticity）。cipher.nonce：每次加密自动生成的随机数，必须随密文一同发送。tag：认证标签，用于解密时验证数据未被篡改。

全部三部分通过 Base64 编码后，打包进一个 JSON 对象，再 .encode() 转为 bytes，以便在更高层打包。

再然后，创建bunble：



这里我设计的bunble各部分内容如下：

version：协议版本号（模拟 BPv7）。

destination：目的地 EID（Endpoint ID）。

creation\_time：Unix 时间戳，标记包创建时刻。

lifetime：寿命（秒），超时后接收端丢弃。

custody\_transfer：是否需要接收端确认“托管”责任。

payload：先经过 encrypt\_payload 返回的 JSON bytes，再做 Base64 编码字符串。

首先，我们检查 destination\_eid 是否在 WHITE\_LIST 中，否则拒绝发送。最后的打包阶段，我们整个字典序列化为 JSON，再编码为 bytes，准备通过 TCP 发送。

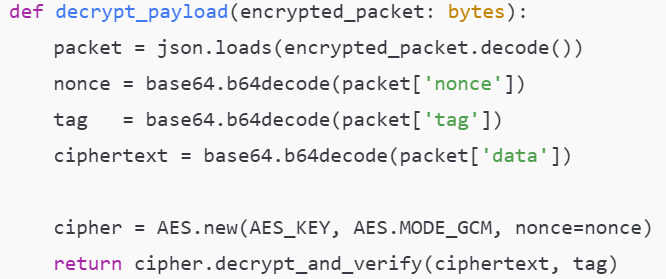
最后是发送方的核心功能，发送信息：



这里，我们首先建立 TCP 连接，然后生成应用数据（示例中是固定的字节串）。再之后进行打包和加密：调用 create\_bundle 得到最终可传输的 bytes。最后将其发送：sendall 保证将所有字节发出；之后关闭连接。

接下来，我们设计了接收方的程序：  


我们来观察这段代码，首先，引用部分和发送方相对应。接下来的功能模块：



这是接收端的功能核心，解析出 nonce、tag、ciphertext，然后用相同的 AES-GCM 参数执行解密并验证完整性。若 tag 不匹配，会抛出异常，触发“完整性校验失败”。

接下来，我设计了处理与审查：



这段代码中，首先进行了生命期检查，即如果当前时间 > creation\_time + lifetime 那么我们就要丢弃。

然后我们做了解包和解密，这里我们先 Base64 解码，再调用 decrypt\_payload。

再之后是异常处理，如果完整性校验失败时捕获异常并报警。

最后是托管转移确认：如果 custody\_transfer=True，在此处可触发回执逻辑（示例仅打印确认）。

然后是服务监听端口模块：



这里我们绑定并且监听在端口 5000（发送方设置的端口）等待地球端发来的 Bundle。

对于循环接受，我们每次连接接收一个 bundle（简化为一次 recv(4096)），并调用 handle\_bundle （即安全审查）处理。

接下来是任务二：设计一套适用于洲际光纤链路的协议集。前提是我们已经完全控制了传输层、网络层和链路层。

对于这个任务，我们首先思考优化的方式：第一个是超高带宽的利用，光纤本身支持数十 Tbps，总目标是在千兆/万兆以上链路上尽可能饱满地发送数据。

第二个内容是高带宽—时延产品（BDP）：单程时延≈ 65 ms，RTT≈ 130 ms，BDP（带宽×RTT）非常大，需要足够大的窗口和足够快的拥塞控制算法。

第三个内容是，极低误码率，但偶发丢包：物理误码率极低，但再生中继、设备切换可能引入短暂丢包或抖动。

第四个内容是部署可行性：需对现有 IP/TCP 栈做最小改动，并能在链路中间分阶段部署优化。第五个内容就是透明性：客户端和服务器无需修改，仅在链路两端或中间插入优化层即可提升性能。

有了这些想法之后，我们尝试进行分层设计：

首先，对于物理层我们由于波长复用、高密度的问题，我们使用DWDM多波长通道；启动 RS‑LDPC FEC；OSNR 优化

然后，对于链路层，由于帧头开销较大，我们就可以选择Jumbo Frame (MTU=9000)。对于网络层，由于IP 分片/重组开销，我们选择了IPv6（更少中间分片）或者其实MPLS/MPLS-TP我觉得也是可以的。对于传输层，因为有高 BDP，并且环境拥塞控制欠佳，所以我们选择TCP BBR + Window Scaling。

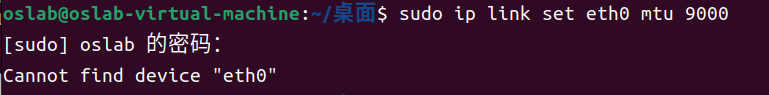
分层设计完成之后，我们再考虑，代理加速（PEP），因为单条 TCP 难以快速吃满管道，所以我们可以将协议设计成并行多路拆连接 + 序号重排序。

而对于安全而言，由于数据保密、完整性、认证，我们可以使用TLS/DTLS 隧道，或者在PEP应用之间加上TLS。

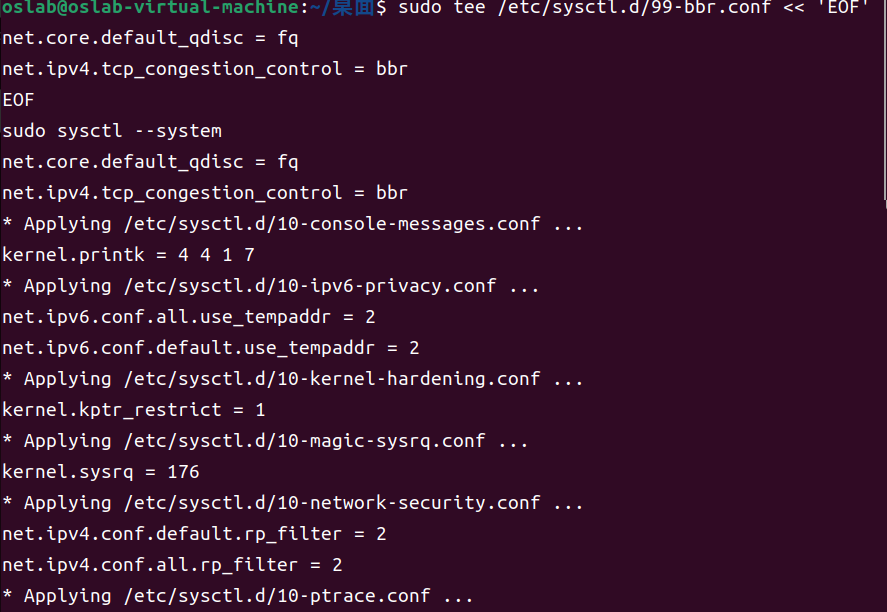
完成底层的设计之后，我们开始考虑具体的实现：

首先。系统层面优化

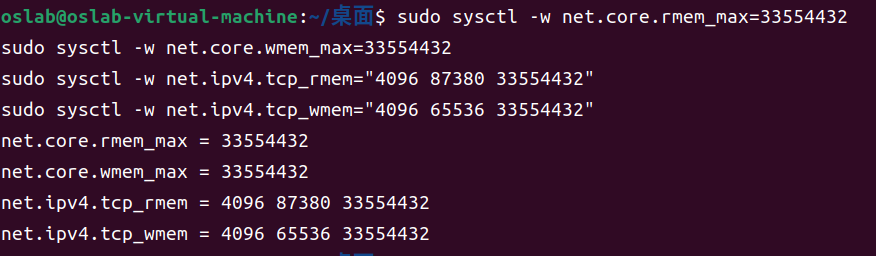
对于Jumbo Frame（MTU 9000）：增大发送单元，减小协议头相对开销，让每个 Ethernet 帧承载更多数据，提高大数据传输效率。



TCP BBR 拥塞控制：传统 TCP（CUBIC/NewReno）在高带宽-时延（BDP）场景下，窗口增长太慢且易有队头阻塞；BBR 直接根据实时测得的带宽与 RTT 构造发送速率，既填满管道又不过载。



Socket 缓冲区调优：在高 BDP 链路上，默认 TCP 缓冲区（约 64 KB）可能不够，需扩到 tens of MB 级别，以匹配链路延迟所需的“管道大小”。



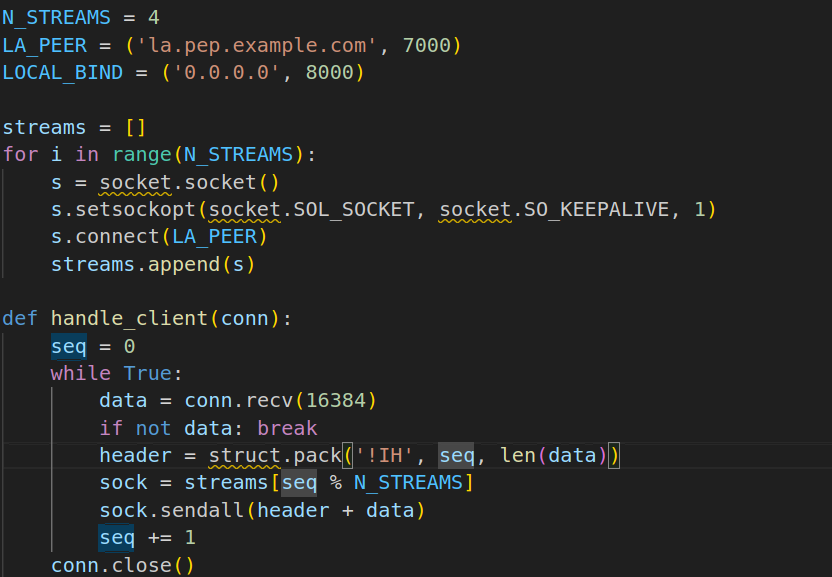
对于PEP 代理原理与代码解析，通过网上查阅到的资料显示，星—港美链路 RTT ≈ 130 ms，单一 TCP 连接的拥塞窗口在丢包或慢启动时要数秒才能“吃饱”这条管道。

PEP的思路是：

拆连接：在链路入口（香港端）和出口（洛杉矶端）各放一个代理。并行多流：将客户端发来的数据按包编号轮询分配给多个（如 4）并行 TCP 会话，每条子流各自拥塞控制，互不干扰。重排序合并：在出口再按包编号顺序还原，再写入到真实服务器连接中。

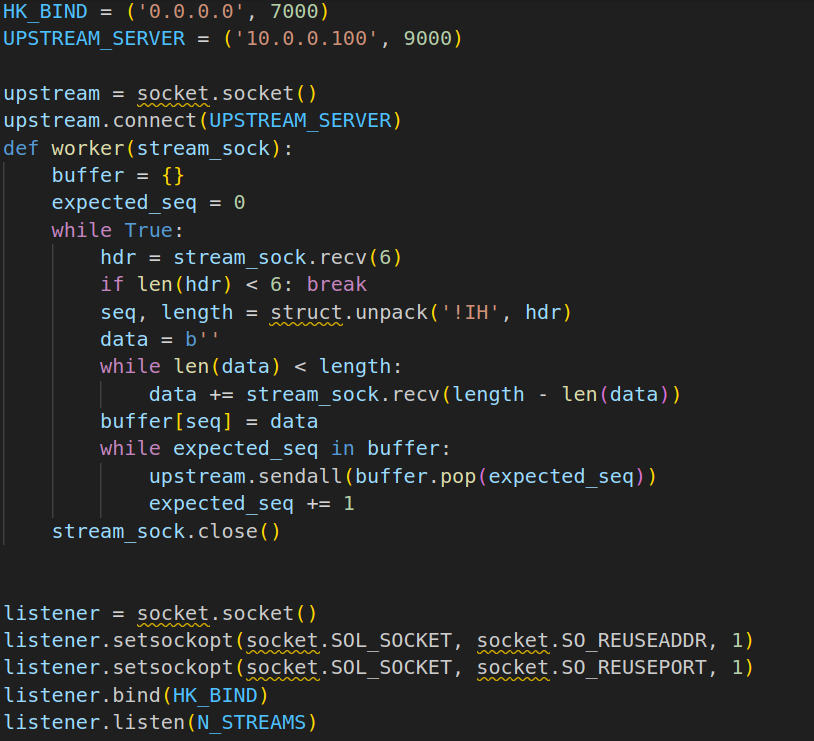
这样，即使单条子流遇到丢包退避，其他子流仍可继续发送，整体吞吐更平稳且更高。

基于此，我们设计香港端的程序：



其中N\_STREAMS表示并行子流数目，通常选 4–8，根据链路丢包/RTT 特性调优。而streams则表示持久 TCP 套接字列表，一次性建立好，不随客户端连接重建，减少握手延迟。

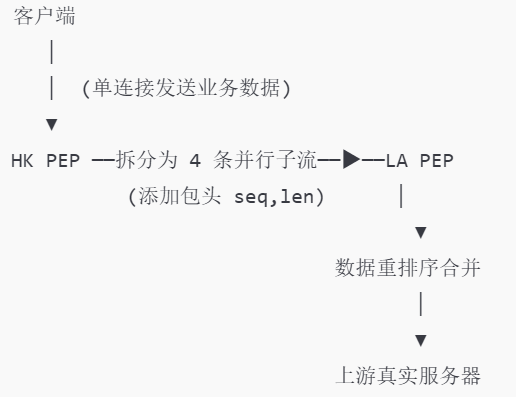
对于接收端，即洛杉矶：



UPSTREAM\_SERVER表示后端真正处理业务的服务器地址。Upstream表示一条单一 TCP 链接，用于最终数据汇总后发送。buffer：使用字典来暂存乱序到达的包。expected\_seq：当前等待的最低序号，确保严格按序提交上游。

这里使用到的合并策略：一旦 buffer 中包含 expected\_seq，就立刻发送并 expected\_seq+=1，直至下一个缺失包。

对于该协议集，我们可以梳理一下其流程：



最后是第三个任务：设计校园网传输大文件传输程序，这里我选择了使用基于UDP的稳定传输协议来设计该程序，程序传输的核心思路是：

分包与编号：将大文件切分成固定大小（如 8 KB）的数据包，并为每个包分配唯一的序号。滑动窗口：发送端维护可发送但尚未确认的包的窗口，接收端通过 ACK 通知收到的最高连续序号；重传与超时：发送端为每个未确认的数据包启动定时器，超时后重传；并发与流控：可配置窗口大小（默认 32），根据网络丢包率和带宽动态调整。

程序暂定的流程为：首先连接建立，由客户端向服务器 UDP 端口发送 “START” 请求包，包含欲接收的文件名。

然后，服务器返回同意并初始化传输。

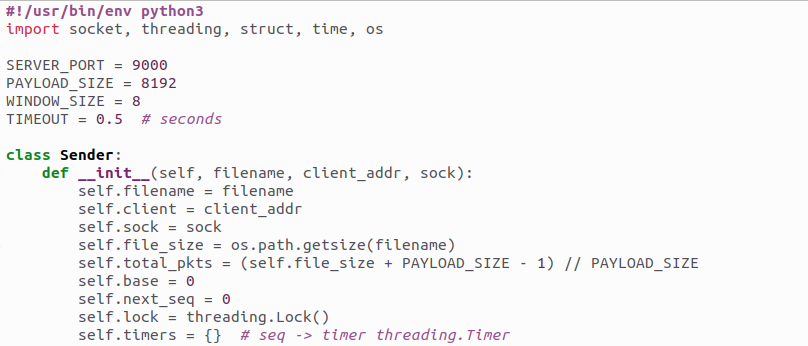
针对数据传输，我们让服务器将文件分为 N 个大小为 PAYLOAD\_SIZE（如 8 KB）的块，每块加上 4 字节序号 (uint32)。然后维护一个滑动窗口 W（默认 32 包），可同时发送窗口内所有未确认的数据包。接着，每发送一个包即启动定时器；收到对应 ACK（服务端收到包含序号的确认包）后清除此包并滑动窗口。

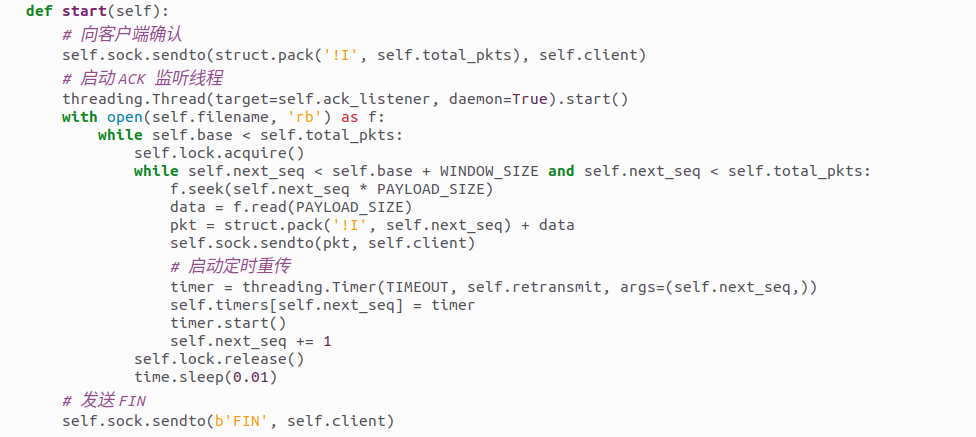
对于重传机制，我们指定，如果包的定时器超时（默认 0.5 s）未收到 ACK，则重传该包。当收到重复 ACK 或 NACK 时，可快速重传对应包。

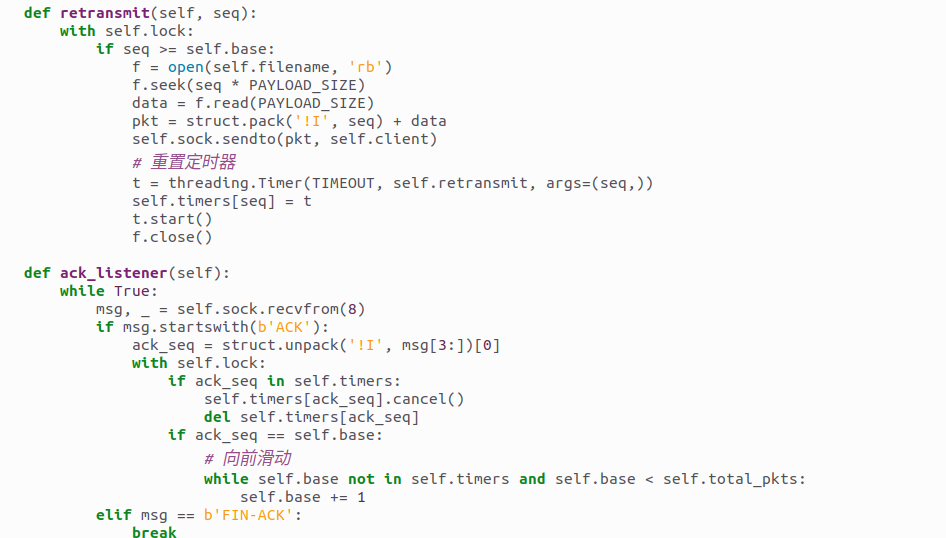
最后，传输结束，发送完所有包后，服务器发送特殊 FIN 包。客户端收到 FIN 后回复 FIN-ACK 并关闭。

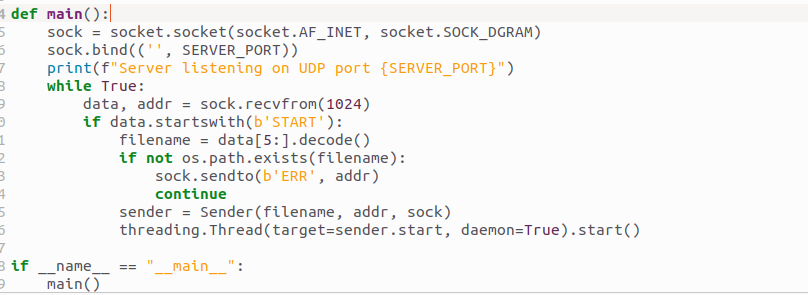
基于以上思路，我们开始编写程序代码：

发送端代码如下：

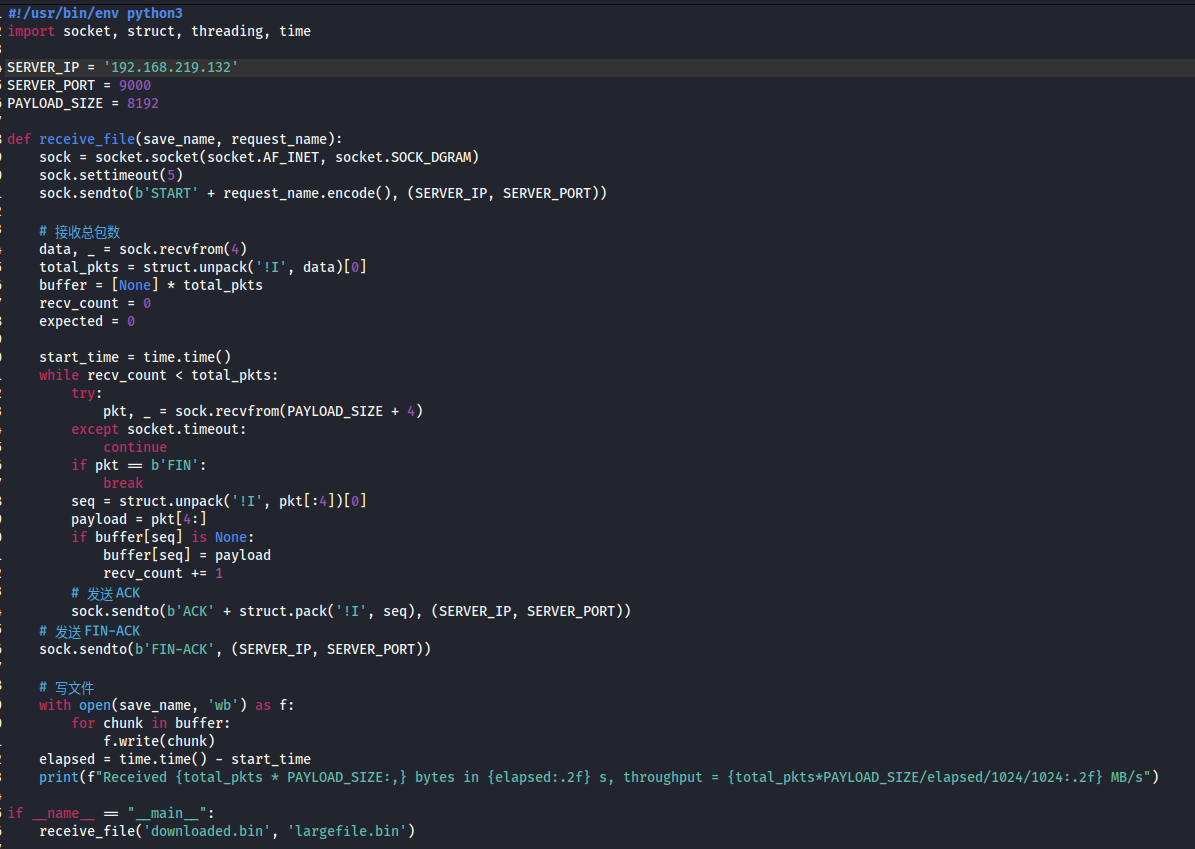


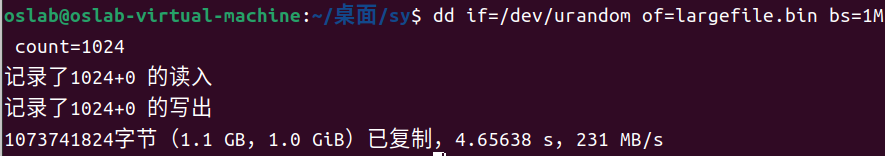




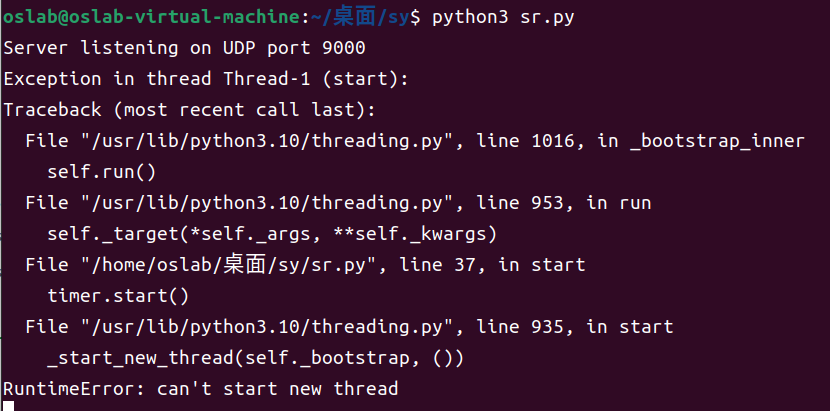


接收端代码如下：

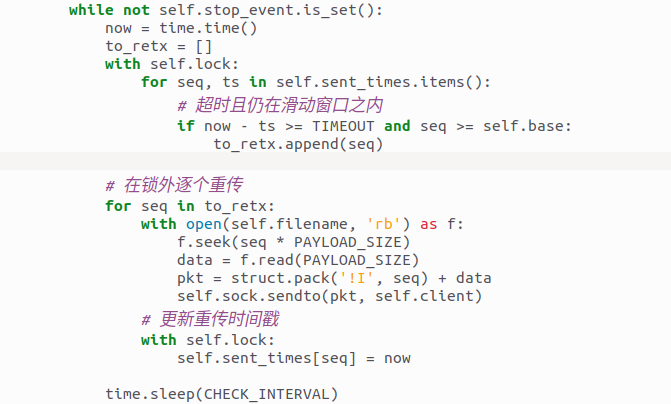


我们运行两个程序来测试其是否能正确传输文件：

首先我们随机生成一个1G左右大小的文件。然后运行程序。

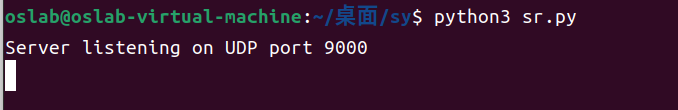


发现发送端出现了该错误，即无法创建新的线程，应该是由于linux中的轻量线程资源耗尽了，无法再创建，于是我试着把窗口大小由32改到了8，再试一次，发现还是会有同样的错误，所以我们只能修改代码，修改的思路为用单线程定时轮询取代大量threading.Timer。

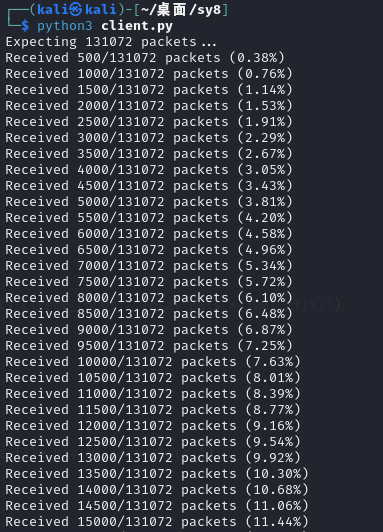


代码修改部分

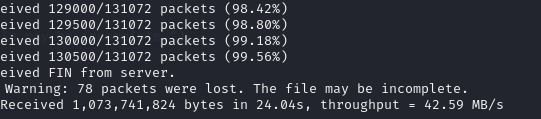
重新运行代码



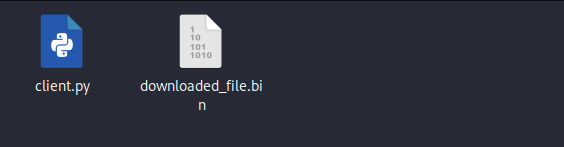
没有报错



可以看到接收端正在接收该文件。接收完毕之后，我们看到：



发现其中有78个package丢失了，同时发现在目录下找到了我们的接收文件：



# 结论

这次的第八次作业，可以说是一次非常全面、很“实战”的网络通信设计与实现训练。无论是从理论角度出发的星际通信协议构思，还是针对实际应用的校园WiFi下的大文件传输实验，都充满了技术挑战，也带来了很多意想不到的收获。

首先是星际通信部分。这部分真的挺有意思，毕竟平时我们谈到网络传输，最多也就是“跨国”、“跨洋”，很少有人会认真去考虑“跨星球”的问题。而这次，我们尝试从 DTN（Delay/Disruption-Tolerant Networking） 出发，思考如何在动辄几十分钟延迟、动不动断联的深空环境中实现可靠通信。

我们借助了 Bundle Protocol（BP） 来打包数据、实现“存储—携带—转发”，还加入了托管传输（custody transfer）和生命周期管理，并基于 AES-GCM 实现了加密与完整性保护。虽然只是简化版的模拟实现，但在动手写代码的过程中，我真切地体会到深空通信的复杂性——那是一种“只发得起一次，必须发得准”的感觉。设计的时候，我们必须考虑到每一个潜在失败点。

相比之下，洲际光纤链路的协议设计虽然“接地气”多了，但挑战一点也不少。在拥有超大带宽的前提下，我们要面临的，其实是如何充分吃满这根“管道”。

这部分我们从物理层、链路层、网络层到传输层，一层一层去优化，比如 DWDM、Jumbo Frame、TCP BBR、PEP 加速器等都搬上了台面。最有意思的部分是 PEP 设计——通过多条 TCP 子流并行传输，然后在接收端重排合并。它就像是一种“明明可以靠速度，但我偏要靠分身”的解决方案，用结构优化来对抗带宽-时延产品高带来的瓶颈。

而通过简单实现香港端和洛杉矶端的两个代理程序，我们能直观地看到这种“多流＋重组”的威力：更高吞吐、更抗抖动，也更适合高质量、大体积数据的传输场景。

到了最后一个实验，直接落地到我们自己的笔记本之间。这个实验可以说是“最能感同身受”的部分了。

这次我们选择了基于 UDP 的可靠传输协议实现，自己动手实现了分片、ACK、窗口、重传、超时、滑动窗口等等。这个过程一开始看上去似乎不太难，真正写起来才发现问题很多，比如：

threading.Timer 会耗尽系统线程资源（尤其在 Linux 上）；丢包之后怎么高效重传；窗口大小该怎么选；超时时间长了慢、短了重传频繁；程序本身对系统资源的占用也可能成为瓶颈。

但也正是这些“坑”，让我们在调试过程中不断磨合出更合理的策略——最终选择用单线程+循环定时器替代大量线程、调整窗口大小，终于稳定完成了一个 1GB 文件的传输。

# 参考文献

1. https://www.nasa.gov/wp-content/uploads/2023/09/dtn-tutorial-v3.2-0.pdf. Delay- and Disruption-Tolerant Networks (DTNs). Forrest Warthman, Warthman Associates. 2015.9.14.

[2]https://www.techtarget.com/searchnetworking/definition/delay-tolerant-network. delay-tolerant network (DTN). Rahul Awati.2024.11.14.

[3] https://wiki.python.org/moin/UdpCommunication UDP Communication

. Python Software Foundation.2021.01.24.

[4] https://stackoverflow.com/questions/59255054/python-udp-socket-client-recevfrom-is-very-slow. python UDP socket client recevfrom is very slow. Sam Mason.2023.8.15.

[5] https://www.tutorialspoint.com/what-is-the-maximum-number-of-threads-per-process-in-linux. Maximum Number of Threads per Process in Linux. Mukul Latiyan.2022.3.24

[6]https://forum.greenbone.net/t/runtimeerror-cant-start-new-threa-eption-in-thread/16789. RuntimeError: can’t start new thread / Exception in thread. PBSH.2024.9.16.

[7] https://askubuntu.com/questions/95022/what-are-the-main-differences-between-virtualbox-networking-type. What are the main differences between VirtualBox networking types?. jrg. 2022.8.15.