中山大學

SUN YAT-SEN UNIVERSITY

# 应用层通信

# 项目报告

# C/S 与 P2P 通信

学院：数据科学与计算机学院

专业：计算机科学与技术

年级：**2016** 级

组长（学号）：王锡淮（**16337236**）

组员（学号）：杨陈泽（**16337271**）

组员（学号）：肖遥（**16337258**）

# 目录

# 1 项目介绍

这是一个应用层的通信应用项目，包括一个服务器-客户端模型和 P2P 模型，两者的功能都是传输文件，项目主页是https://github.com/Leo-xh/C-S-and-P2P-demo。其中，服务器-客户端模型使用的是单服务器多客户端模型，并且单一客户端可以同时请求多个文件，服务器和客户端都使用多线程模型。P2P 模型参考的是 bittorrent 协议，完成了 bittorrent 协议的一个实现（命名为 Compact Bittorrent Protocol/1.0），并且实现了原来的 biitorrent 协议中的几个扩展协议。

## 1.1 设计应用层协议的原则

一个应用层协议应当包含两个主要部分：

1. 编码控制（编码和译码）。
2. 流程控制。

一个协议可以根据多种原则来进行划分，比如按照编码分类、按照协议边界分类。而对于协议的评判来讲，主要有四点评判原则：

1. 协议的高效性，包括打包和解包的效率、数据压缩率等等。关于这一点，应用层协议的效率主要涉及数据方面，因为数据的传输并不是应用层的责任。
2. 协议的简单性。
3. 协议的可扩展性。举个例子，在 Bittorrent 协议中，有许多扩展，本项目中也实现了几个扩展，这些扩展使得 Bittorrent 更加健壮和高效。
4. 协议应当向前和向后兼容。拿上面的 Bittorrent 扩展为例，实现了这些扩展的协议版本兼容了为实现这些扩展的协议版本。

设计协议时，主要需要回答三个基本问题：

1. 这个协议需要完成什么问题。
2. 协议中传输的信息的含义是什么。
3. 该协议的特征是什么。

下面本项目中设计的协议都将围绕这三个问题。

# 2 C/S 通信

本项目中实现的 C/S 通信模型使用 python 实现，主要利用的是 socket，threading，struct，os 等常用库，其中服务器使用多线程，能够支持多个客户端同时请求文件；客户端也使用多线程，能够同时请求多个文件。

提供的服务如下：

1. 原始数据传输。

2. 加密数据传输。

3. 查看服务器的文件目录。

## 2.1 协议设计

这是一个二进制模糊边界和固定边界的协议，即传输的数据以二进制编码，在请求报文中能够确定报文长度，在应答报文中无法明确知道协议报文的长度，需要通过报文中的长度字段知道。

在设计协议格式时，经过小组的讨论，决定在协议头中采用两种（请求和发文件）报文格式，请求报文头中有：类型、服务、版本、序号和文件名字段，而发文件报文头中有：类型、服务、版本、序号、长度、错误码和数据体；每个字段的作用在后面有相应解释。定义好协议格式后，接着定义客户端和服务器之间的行为交互：首先客户端在链接上服务器时，1、可以向服务器发出“查询文件目录”请求，然后服务器查询本地的文件及目录，以明文形式发回去给客户端，客户端拿到后输出回显到屏幕上；2、客户端发出“某文件”请求，并选择是否需要加密，服务器接收请求，若文件不存在，发送一个错误码为 1 的报文给客户端，客户端根据该报文做出响应；若文件存在，服务器将文件分成许多报文向客户端发送（根据加密需要选择加密与否），并且服务器在自己的终端上显示发送文件时的进度条，等文件的所有报文发送完成，服务器再向客户端发送一个数据长度为 0 的报文，告知客户端，文件发送完毕。

接着分析了该应用层协议是用来传输二进制文件（主要功能）和传输明文（次要功能），对于明文来说，就不得不考虑编码转换的问题，因为需要采用了 struct 库的 pack 和 unpack 函数来进行压包和解包的。而且，在明确了该应用层协议是采用 TCP 之后，我们进行了下面的 TCP 缺陷分析：

在实现这个通信模型时会遇到的问题主要是 TCP 协议的分包和粘包问题，TCP 中只有数据流这样的概念，而没有数据包这一类的概念；因为自己设计应用层调用网络编程的 API 接口，一次调用中要发送的报文大小在 API 接口中没有规定，但是在机器的 IP 层，TCP 并不是按照调用者传给 API 的报文大小一次性发出（我们已经知道在以太网中 TCP 报文大小限制在 1500bytes），而是采用自己的算法，将包切割，然后发出，包到达对方的 IP 层时再进行重组（TCP 提供了可靠的服务，不需要考虑丢包问题），但是由于 IP 层的算法设计，导致接收方的 IP 层重组后的传给应用层的报文，并不一定完全等于发送方的应用层传给 IP 层的报文，会可能出现：分包（发送方应用层的一个报文，实际上分成了两个接收方应用层的报文）和粘包（发送方应用层一个报文的部分（或者全部）与另一个报文的部分（或者全部），实际上合成接收方应用层的一个报文）。每次收到的不一定会是一个完整的数据包，所以需要通过某种方法明确当前处理的数据包的大小，然后解析这个大小的数据包。

这类问题的解决方法大致有两种：第一种是，设计一个独特的分隔符，将应用层的每个包后面加上该分隔符，然后接收方根据分隔符来分割出每个包，但是因为我们这次传输的是文件（二进制），无法预测可以采用什么充当分隔符而不会在文件上出现；第二种是：发送方在报文协议头上，加上一个字段，表明该报文数据字段的长度，然后接收方的应用层设置一个报文缓冲区，每个到达的报文存在缓冲区中，首先在缓冲区中提取报文头（固定长度的），然后根据报文头的长度字段再在缓冲区中提取相应长度的数据字段（当缓冲区现有数据长度小于需要提取的长度时，继续收包，直到长度大于等于需要提取的长度），这样便能完全解决由于 TCP 数据流的特点而产生的缺陷问题。我们本次协议采用的是第二种方案。

至于在加密过程中，我们采用的是 AES 对称加密算法，原因是：对称加密，适合数据的加密和加密；而且实现较为简单。

客户端的请求报文设计如表1所示。

| 类型 | 服务 | 版本 | 序号 |
|---|---|---|---|
| 文件名 | | | |

表 1: 客户端请求报文

参数解释如下：类型指的是协议号，服务是服务号（3 种不同的服务），版本是协议版本号，序号是请求序号，大小都是 2 字节，文件名指的是请求的文件名，长度上限为 200 字节。

服务器响应报文设计如表2：

| 类型 | 服务 | 版本 | 序号 | 长度 | 错误码 |
|---|---|---|---|---|---|
| 数据 | | | | | |

表 2: 服务器应答报文

类型、服务、版本、序号字段和请求报文中一样，长度字段指的是数据字段的长度，大小为 2 字节，数据字段是发送往客户端的数据。

对于上面提到的三种服务，对应的服务号分别是 0，1，2。

而错误码字段现在只有两种：0 表示服务器有客户端需要的文件；1 表示服务器没有客户端需要的文件。

服务器和客户端的控制流程如图1和2所示。

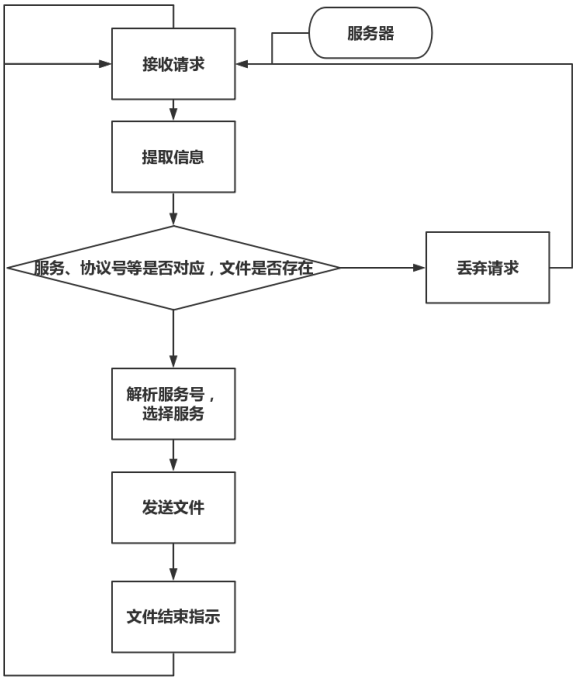


图 1: 服务器流程图

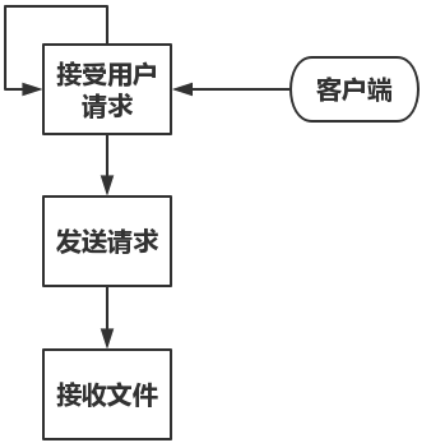


图 2: 客户端流程图

## 2.2 特点

本项目中实现的 C/S 模型的特点是：

1. 实现了支持多文件同时传输文件时的传输进度条，即多个进度条能够同时正常显示。
2. 实现了服务器的文件查找功能。
3. 使用了数据加密技术。
4. 客户端能够同时请求多个文件，服务器能够同时处理多个请求。

# 3 P2P 通信

本项目中完成了 bittorrent 协议的一个实现（命名为 Compact Bittorrent Protocol/1.0），这个实现采用了基本的 Bittorrent 协议规定的内容，同时参考了社区中的建议（比如 Block 大小的选择）。同时我们参考了 Bittorrent 协议网站上列出的的扩展方案中的协议。我们的实现主要包括两个方面，一是客户端（Peer 在和 Tracker 通信时作为客户端）与服务器（Tracker）的通信；二是对等方之间的通信。下面将围绕这两个方面进行介绍。首先一个 Peer 的程序流程和状态转移如图3所示。

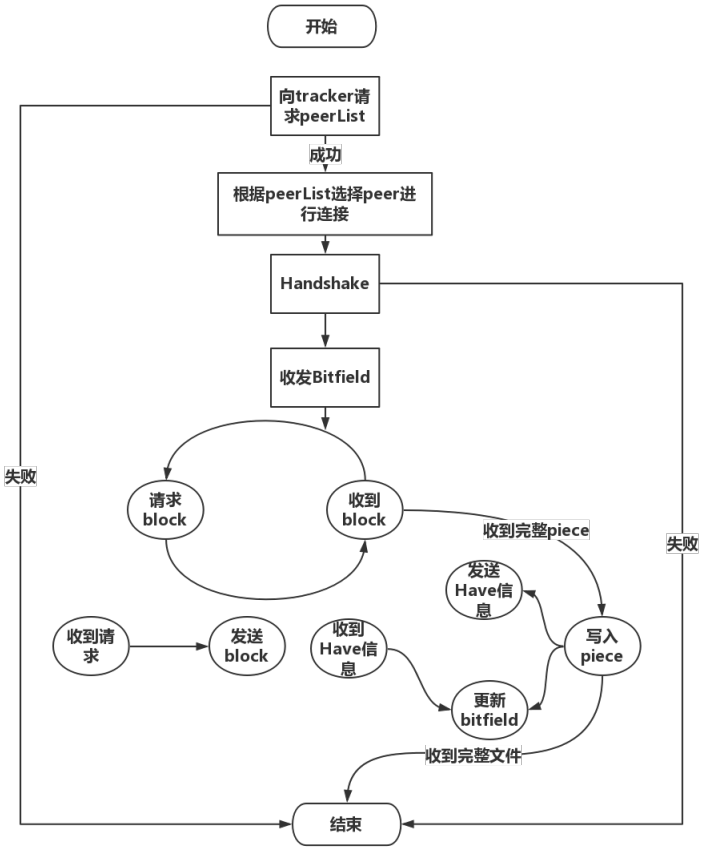


图 3: Peer 程序流程和状态图

## 3.1 Twisted 和异步编程

### 3.1.1 Twisted 简介

Twisted 是用 Python 实现的基于事件驱动的网络引擎框架，Twisted 支持许多常见的传输及应用层协议，包括 TCP、UDP、SSL/TLS、HTTP、IMAP、SSH、IRC 以及 FTP。

在本项目中主要使用的是以下几个类：

1. Transport：Transports 代表网络中两个通信结点之间的连接。Transports 负责描述连接的细节，比如连接是面向流式的还是面向数据报的，流控以及可靠性。
2. Protocol：Protocols 描述了如何以异步的方式处理网络中的事件。
3. Reactor：Twisted 实现了设计模式中的反应堆（reactor）模式，这种模式在单线程环境中调度多个事件源产生的事件到它们各自的事件处理例程中去。Twisted 的核心就是 reactor 事件循环。Reactor 可以感知网络、文件系统以及定时器事件。它等待然后处理这些事件，从特定于平台的行为中抽象出来，并提供统一的接口，使得在网络协议栈的任何位置对事件做出响应都变得简单。
4. Deferred：Deferred 对象以抽象化的方式表达了一种思想，即结果还尚不存在。它同样能够帮助管理产生这个结果所需要的回调链。当从函数中返回时，Deferred 对象承诺在某个时刻函数将产生一个结果。返回的 Deferred 对象中包含所有注册到事件上的回调引用，因此在函数间只需要传递这一个对象即可，跟踪这个对象比单独管理所有的回调要简单的多。

### 3.1.2 异步编程与事件驱动

图是一张区分单线程、多线程、和异步编程模型的示意图。

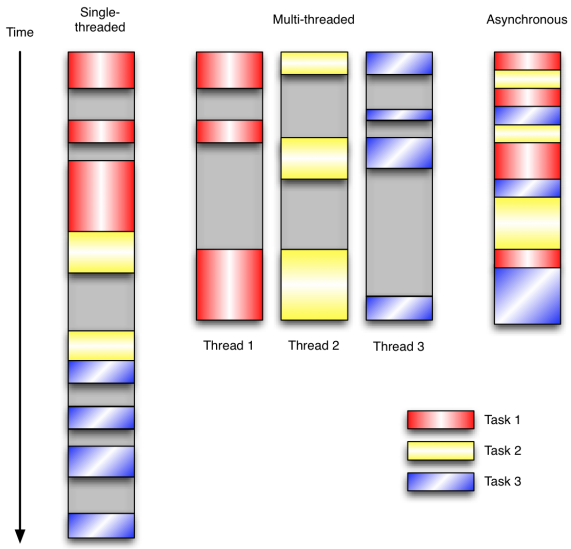


图 4: 示意图

使用异步编程的原因在于网络应用程序通常具有多任务高度独立、某些事件的发生会阻塞另外的事件的特点。

## 3.2 Tracker 和 Client 的通信协议（UDP Tracker Protocol）

### 3.2.1 协议设计

tracker 服务器需要能够维护一个发起种子文件链接请求的对等方列表，将延时过期的对等方从对等方列表中剔除，并响应新的对等方的请求，以及向所有对等方发送实时的对等方列表。

客户端是每个对等方与 tracker 服务器链接的功能部分，向服务器发起种子文件链接请求，拿到 tracker 服务器返回来的对等方列表，交给本地的对等方，并且需要根据服务器设定的时间要求，一定时间再次发起一个请求，告知服务器该对等方还存活以及拿到新对等方列表并更新本地的对等方列表。

tracker 服务器与客户端之间传输的报文包括以下四种:

| protocol_id | action | transaction_id |
|---|---|---|

表 3: 客户端 connect request

参数及作用：action 是表明报文类型（connect 还是 announce），transaction_id 是客户端生成并以此来认证 tracker 返回的报文。

| action | transaction_id | connection_id |
|---|---|---|

表 4: 服务器 connect response

参数及作用：connecttion_id 是 tracker 生成了独一无二的一个验证码，用来验证客户端请求的合法性，并防止遭受大量无效的请求。

| connection_id | action | transaction_id | info_hash |
|---|---|---|---|
| peer_id | downloaded | left | uploaded |
| event | IP address | key | num_want |
| port | | | |

表 5: 客户端 announce request

参数及作用：IP address 是可选选项，port 是客户端对应的对等方监听的端口。

| action | transaction_id | interval |
|---|---|---|
| peer_list | | |

表 6: 服务器 announce response

参数及作用：interval 是服务器自定的时间间隔参数，要求 client 在该时间间隔后必须再次向 tracker 发出一次 announce request，若 tracker 在两倍的 interval 时间后，没有收到 client 发来的 announce request，则认为 client 已失去链接，将它从 peer_list 中剔除出去。

在 tracker 和 client 通信中，为了减轻 tracker 负担，我们选择了 UDP，而在采用 UDP 协议时，就决定了应用层需要去解决丢包、超时的情况，所以在 client 中需要设计超时重传机制（而不是在 tracker 服务器中），而且每发生了超时重传后，下一次的的超时重传间隔设得比现在的间隔要大。

### 3.2.2 协议特点

tracker 和 client 通信设计的特点是：

1. tracker 使用 twisted 事件驱动型编程框架来实现，以 client 的请求这一事件来驱动 tracker 的响应，极大程度上降低 tracker 在没有 client 请求时的资源消耗。
2. 采用 UDP 协议，并为 client 设计了较为完备超时重传的机制，以此有效降低 tracker 的服务负载。
3. tracker 给 client 发送的 peer_list 采用的是紧凑格式的，每个 peer 的信息只需要用到 6 字节，相比于采用 bencoded 字典的列表，数据的压缩率达到了 50% 以上的提高，

## 3.3 Peer 协议（Peer Protocol）

如图3所示，peer 协议主要处理了以下问题：

1. 判断多个 peer 是否在同一个 torrent 中。
2. peer 间请求文件和接受文件，其中涉及选择策略和各个 peer 之间信息交换（主要是 peer 拥有哪些 piece）。
3. 处理文件写入和断点续传。

### 3.3.1 协议设计

**Peer** 间通信

1. 握手。一个 Peer 会在一定时间间隔后定时连接没有连接过的 peer，接受握手后会检测是否应当保持连接。
2. 请求和发送文件。使用一定的策略进行 piece 和 peer 的选择。
3. 报文设计。主要有两种报文，一是 handshake 报文，二是 message 报文。前者用以握手，后者用来进行各种通信。

### 3.3.2 协议特点

1. 支持断点续传。

# 4 安装和部署

# 5 结果

## 5.1 结果展示

首先介绍本项目测试时的网络拓扑图，如图5和6所示。

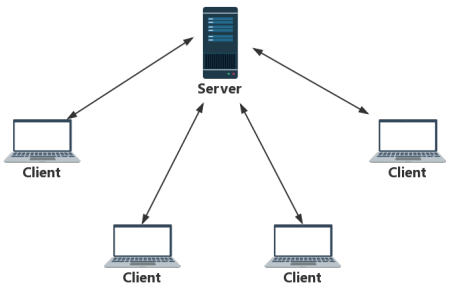


图 5: C/S 模型网络拓扑图

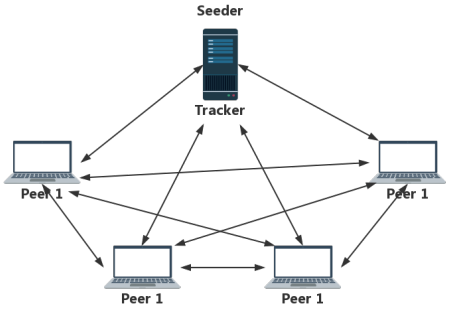


图 6: P2P 模型网络拓扑图

C/S 模式结果展示。

P2P 模式结果展示。

### 5.2 对比

使用 P2P 模式（简化 Bittorrent 协议）在 Peers 数目较少时要比使用 C/S 模式要慢上许多，而在 Peers 数目较多时速度有较大提升。

## 6 总结

协议的设计是相当复杂的，主要在于协议报文主体的设计和协议控制流程的设计。

# 7 项目管理记录

本项目主要使用的是 github 来管理项目，建立了一个仓库，三人同时使用。

# Appendices

## A 参考文献

1. Jonas Fonseca，et al，http://jonas.nitro.dk/bittorrent/bittorrent-rfc.html#anchor17，Bittorrent 协议详细解读。
2. Bram Cohen，http://www.bittorrent.org/beps/bep_0003.html，Bittorrent 官方协议。
3. Jessica McKellar Abe Fettig，Twisted Network Programming Essentials(second edition)。