

计算机网络课程项目——C/S与P2P通信

一.实验要求

二. 简介

三.P2P通信

（一）协议

(1). Torrent文件格式

(2)Tracker — Peer协议

Peer发送的Request格式：

Traker发送的Response格式：

（二）实现

(1). 协议的消息传输

(2).Tracker的实现

(3).Peer的实现

(4) Download模块实现

（三）时序图

五、安装部署及实验结果

1运行环境

2. 使用方法

2.1 做种

2.2 启动服务端

2.3 启动客户端

2.4启动下载端

# 计算机网络课程项目——C/S与P2P通信

组员一	杨元昊	16340274	评分：100
组员二	杨赞	16340275	评分：100
组员三	张马良	16340291	评分：100

## 一.实验要求

- C/S通信实现要求：
  1. 两台计算机分别模拟服务器、客户端
  2. 通过编程实现服务器端、客户端程序Socket，Client。
  3. 服务器端程序监听客户端向服务器端发出的请求，并返回数据给客户端。
  4. 不采用方式，自定义通信协议，传输文件要足够大（例如：一个视频文件）
- P2P通信实验要求

1. 为每个peer开发服务器程序、客户端程序
2. 每个peer上线后，向服务器注册自己的通信信息；
3. 假设peer3要下载文件（视频），A与peer1，peer2都拥有A，请设计方案使peer3能够同时从peer1、peer2同时下载该文件，例如：从peer1下载A的前60%、同时从peer2下载后40%。
4. 比较与C/S通信方式的性能指标

## 二. 简介

---

- 本次实验我们使用java和Python3语言，分别在Windows 10和macOS High Sierra环境下开发。
- C/S通信部分，我们实现了一个可以选择文件进行上传下载和断点续传的Android app，主要基于原理是基于套接字编程。
- P2P部分：我们研究并实现了简化版的有tracker服务器的Bittorrent协议。采用消息循环的设计方式，两台对等主机之间建立连接后各自开启一个线程，交换bitfield并初始化自身状态，进入消息循环，根据自身状态和收到的消息决定状态的转换和执行的的操作。各台对等主机，以及对等主机和服务器之间的通信基于了C/S通信部分实现的可靠二进制文件传输模块。
- 下面，我们将详细描述P2P通信的协议和实现。并给出运行结果。

## 三.P2P通信

---

### （一）协议

---

我们实现的P2P通信基本原理是tracker作为总服务器记录在线的各个peer节点的状态，当有peer节点要下载某样东西时，tracker会搜索哪些peer节点有这个文件，然后将peer节点列表返回给需要下载文件的peer节点。然后peer节点就利用下面我们将从三个方面分别介绍我们设计的P2P通信协议

1. Torrent文件格式
2. Tracker — Peer协议

### (1). Torrent文件格式

Torrent文件的作用是：

- 声明了一个P2P网络的tracker服务器地址和端口。
- 声明了在该P2P网路上共享的一个文件的文件名、长度、区块数、各区块哈希值，唯一确定了一个文件。

一个Peer在获取一个Torrent文件后，便可加入该P2P网络并获取该文件。

使用(类)Json的语法描述Torrent文件如下：

```

{
  announce: <str>, #domain name
  port: <int>
  comment: <str>
  info: <dict> {
    piece_length: <int>
    piece_hash: <list<str>>
    file_name: <str>
    file_length: <int>
  }
}

```

## (2)Tracker — Peer协议

这部分协议提供了加入和退出P2P的机制。特别是使得加入P2P的Peer能够获取目前的Peer列表。

### Peer发送的Request格式：

包含：

- Peer的IP
- port, Peer的本地监听端口
- peer\_id, 由peer的ip和port组成
- event, 可能值包括started（用于请求加入网络），stoped（未使用），completed（用于请求退出网络）。

### Traker发送的Response格式：

包含：

- error\_code, 收到的请求有效时为0，非法请求则为1
- message, 包括started ACK和disconnect ACK两种
- num-of-peer, 请求前的peer数
- 请求前P2P网络中的peer的id、端口、地址

```

{
  error_code: <int>
  message: <str>
  num-of-peer: <int>
  peers: <list> [
    {
      peer-id: <str> (peer ip + ':' + peer port)
      peer-port: <int>
      peer-ip: <str>
    }
  ]
}

```

除此之外还借鉴了经典的bittorrent实现，让peer和server传递以下消息（因为我们是要一个客户端向其他多个客户端下载文件，因此暂不考虑文件在多个peer节点互传的情况。）

1. Choke，发送该消息者拒绝向对方发送文件
2. UnChoke，发送该消息者可以向对方发送文件
3. Interested，发送该消息者需要从对方获取文件
4. UnInterested，发送该消息者无需从对方获取文件
5. Have，一方收到一个piece后，发送该消息通知对方已经完成接收该piece
6. Bitfield，一方拥有的文件区块信息
7. Request，一方向另一方请求piece，消息中包含piece编号
8. Piece，一方收到request后回复的文件piece，包含文件内容
9. KeepAlive，保持连接，接收者收到后忽略该消息
10. ServerClose，一方通知另一方自己要关闭了，另一方收到该消息后也会关闭该peer Connection

## (二) 实现

---

### (1). 协议的消息传输

协议中消息的传递是基于C/S通信中的二进制传输代码。

Tracker — Peer协议的消息格式为Json，我们使用Json以下两端代码将Json转换和转换为二进制。

```
def objEncode(obj):
    """ obj, 返回binary对象 """
    return json.dumps(obj, indent=4, sort_keys=True, separators=
        (',', ':')).encode('utf-8')

def objDecode(binary):
    """ binary 返回dict对象 """
    return json.loads(binary.decode('utf-8'))
```

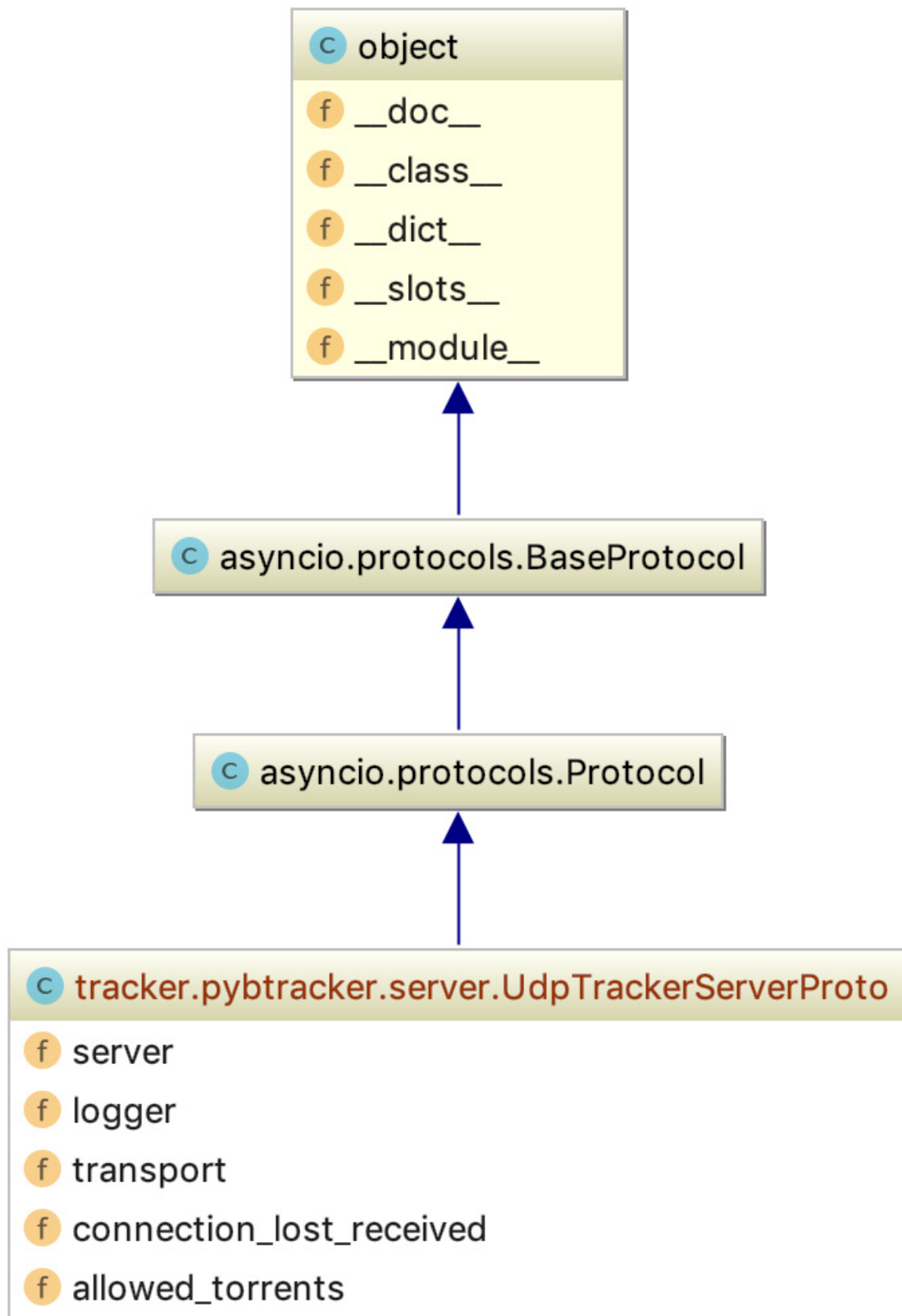
Peer — Peer 协议中协议的原始格式也为Json，但Piece中的原始文件数据在使用objEncode编码时会出错，因此使用Python的Struct类进行转换。

### (2).Tracker的实现

Tracker端的实现主要是利用socket创建一个服务器，其中send\_msg、transport、receive\_msg这几个函数将通过实现threading类的Run函数，将主逻辑运行在线程中。

Server类的run函数监听并Peer呼入的连接，根据消息做出回复。

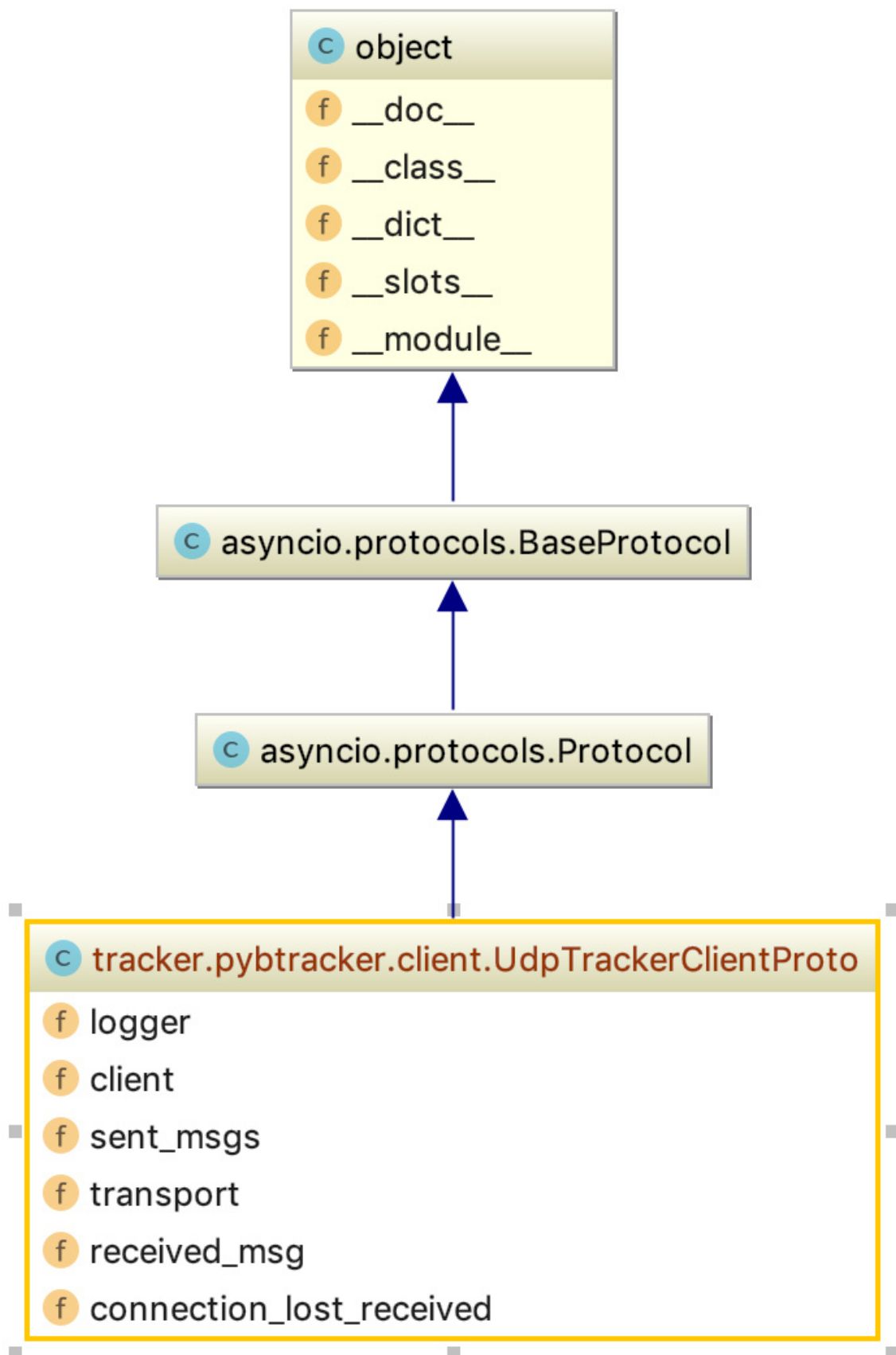
available\_peers\_list函数返回当前Peer列表。START\_ACK/COMPLETE\_ACK两个函数方便地返回了两种Response 消息。



### (3).Peer的实现

这部分是整个实现中的重点。

由`Client`、`send_msg`、`transport`、`receive_msg`这几个函数实现，其中`send_msg`、`transport`、`receive_msg`三个函数将在`run`函数中，被`Thread`类以多线程方式实现主逻辑。



整个工作原理和流程如下：

1. Client类主要完成整个初始化流程，它将接受tracker主机的地址并与它进行连接。
2. send\_msg, receive\_msg来处理与tracker相连和接受tracker发送消息的功能。
3. 之后，Client启动Monitor，该类实现的是被动接收连接功能：它监听本地端口，阻塞循环接受来自其他线程的新连接。得到新的连接new\_socket后，将new\_socket传入并启动一个新的PeerConnection，也即是调用transport函数实现传输功能。

## (4) Download模块实现

这个模块关键是在开启了tracker和peer的情况下，读取某个torrent文件，然后并发的从多个peer处根据torrent文件的列表和bitfield中哪些小块被下载成功，哪些没有下载成功来下载文件的多个小块。

我们开启了一个使用一个全局的线程安全队列来保存缺失的piece的编号。当函数运行时，根据初始bitfield初始化安全队列。然后执行get\_peers\_list向tracker请求peer列表。收到后主动向这些peer发起连接。

```
def get_available_piece_request(self):

    if left_pieces.empty():
        logger.debug('The queue is empty.')
        return 0

    while True:
        if pieces_manager.bitfield == self.peer_bitfield |
pieces_manager.bitfield:

            logger.debug("I don't need this peer:
{}".format(self.socket.s.getpeername()))
            # 如果对面没有我需要的块，直接返回0

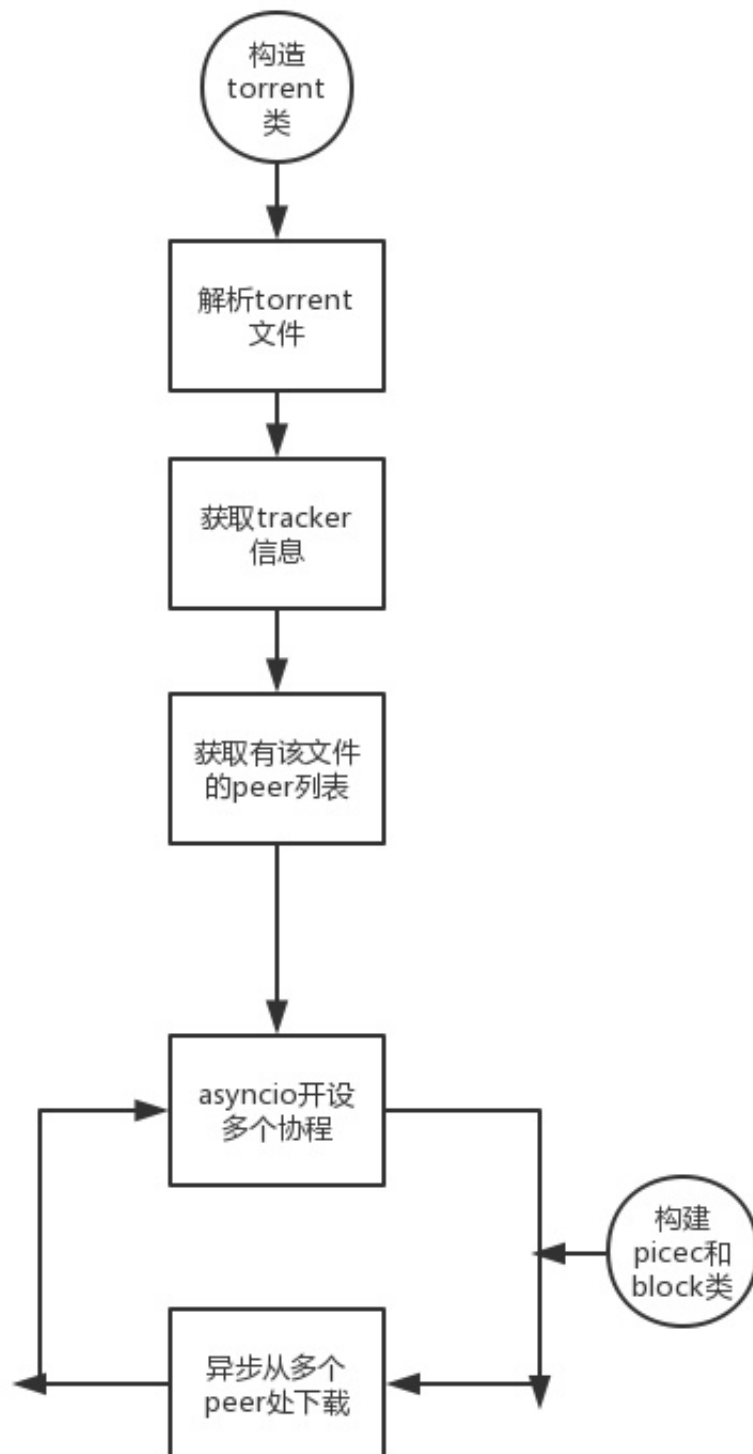
            return 0
            # self.queue_lock.acquire()
            piece_index, piece_hash = left_pieces.get()
            if self.peer_bitfield[piece_index] == 1:
                # 如果对面有这个数据块，就interest，否则就放回队列中
                self.request_piece_index, self.request_piece_hash =
piece_index, piece_hash
                logger.debug("{}: this piece exists in peer:
{}".format(piece_index, self.socket.s.getpeername()))
                return 1
            else:
                # 对面没有这个块，将这个块放回到队列中
                left_pieces.put((piece_index, piece_hash))
                logger.debug("{}: this piece doesn't exist in peer:
{}".format(piece_index, self.socket.s.getpeername()))
                time.sleep(random.random())
                continue
```

当下载了多个小块文件后，我们就讲这些小块文件合并并删除。

```
if not save_file_name:
    save_file_name = 'download_'+self.file_name
if not self.is_completed():
    print('The data is not completed')
    return False
print('The data is completed')
with open(save_file_name, 'wb') as f:
    for i in range(0, self.piece_num):
        f.write(self.pieces_data[i])
if torrent.same_as_torrent(self.torrent_file_name, save_file_name):
    return True
return False
```

逻辑过程如下





```
async def download(torrent_file : str, download_location : str, loop=None):
    torrent = Torrent(torrent_file)
    LOG.info('Torrent: {}'.format(torrent))

    torrent_writer = FileSaver(download_location, torrent)
    session = DownloadSession(torrent,
    torrent_writer.get_received_blocks_queue())
```

```

tracker = Tracker(torrent)

peers_info = await tracker.get_peers()

seen_peers = set()
peers = [
    Peer(session, host, port)
    for host, port in peers_info
]
seen_peers.update([str(p) for p in peers])

LOG.info('[Peers] {}'.format(seen_peers))

await (
    asyncio.gather(*[
        peer.download()
        for peer in peers
    ])
)

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    loop.run_until_complete(download(sys.argv[1], '.', loop=loop))
    loop.close()

```

这里我们关键是利用了 `asyncio` 这个内置了对异步IO的支持的标准库来实现同时从多个peer处下载文件。

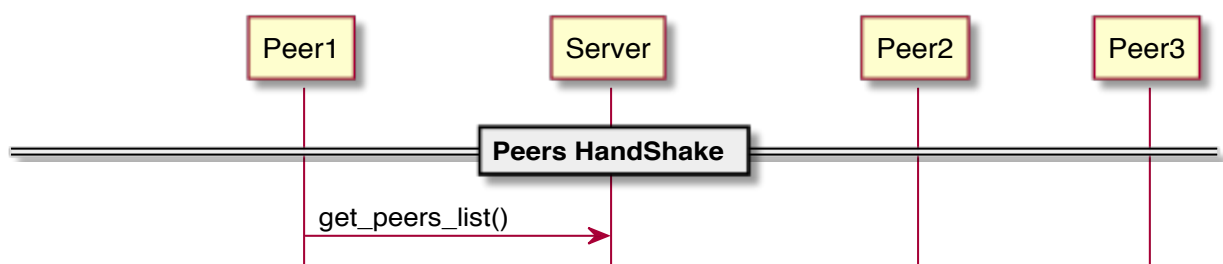
`asyncio` 的编程模型就是一个消息循环。我们从 `asyncio` 模块中直接获取一个 `EventLoop` 的引用，然后把需要执行的协程扔到 `EventLoop` 中执行，就实现了异步IO。它实质上是调用了多个 `goroutine`（协程），并将未来可能提前发生的结果自动保存在了 `future` 模块中，从而极大简化了我们利用 `epoll` 来写多个回调函数的痛苦。

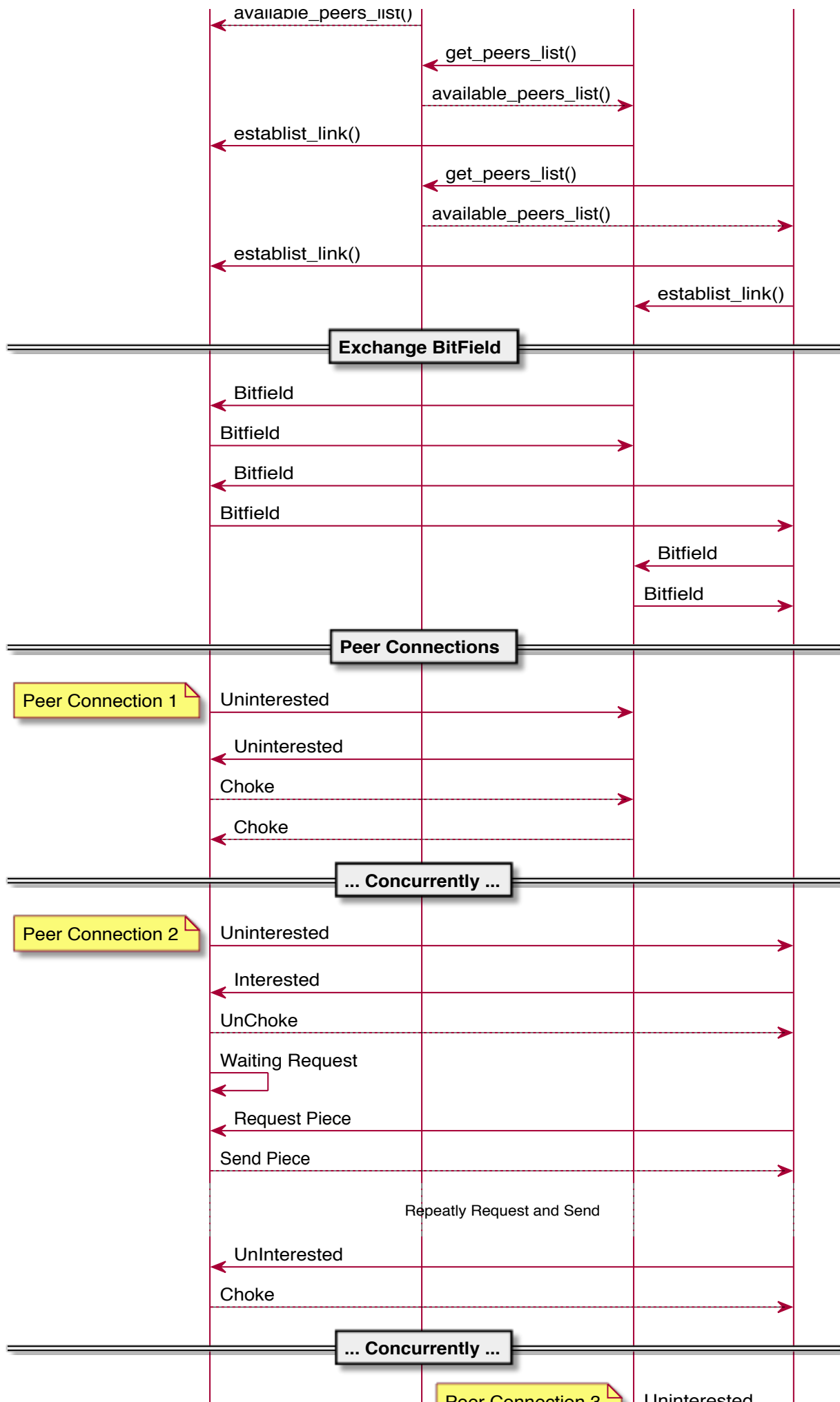
### （三）时序图

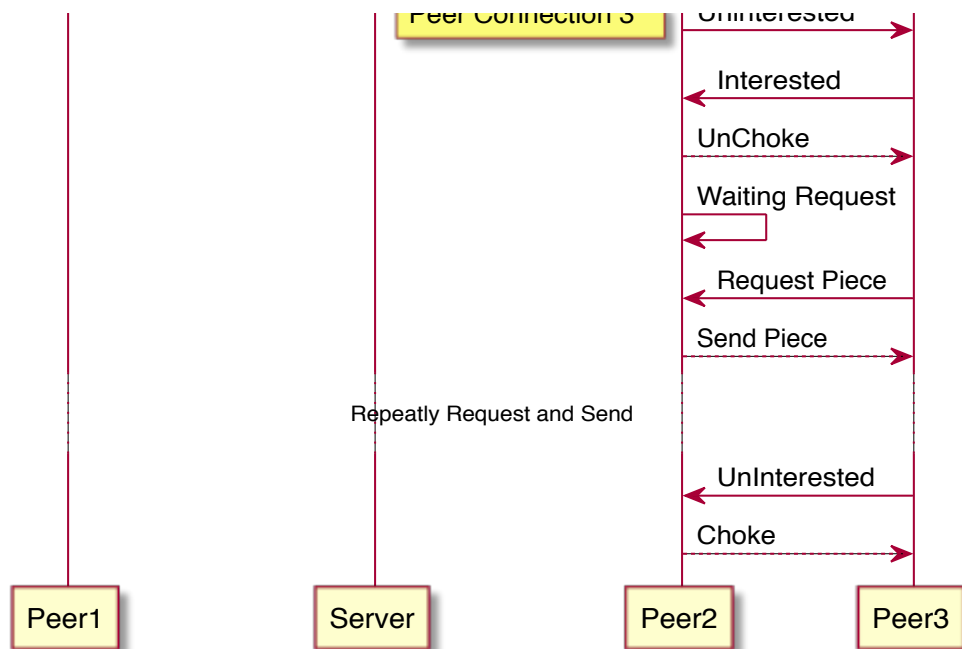
我们以实验要求中的使用场景为例，绘制了整个下载过程的时序图，让整个过程更加清晰易懂。

假设peer3要下载文件（视频），A与peer1，peer2都拥有A，请设计方案使peer3能够同时从peer1、peer2同时下载该文件。

注：图中三个Peer Connection是并行执行的。







## 五、安装部署及实验结果

### 1运行环境

本项目在python3.6环境下开发并测试。

服务器与客户端均运行在同一个内网中。

### 2. 使用方法

这里以demo的使用为例。

- 在demo中，其他客户端都想下载Trying.mov文件。
- 先开启tracker服务器，然后开启其他客户端，这些客户端的代码执行文件目录下有Trying.mov的文件
- 再开启一个下载客户端，它开启后会从tracker得到peer的在线信息，并向多个peer请求得到Trying.mov.

#### 2.1 做种

在命令行下执行以下指令：

```
cd src # 在src文件夹下
python3 seed.py
```

- 需要提前安装好libtorrent库
  - 使用本机IP地址更新种子文件
- 其中代码文件的此处可以更改要做种的文件

```
videoFile = "/path/to/video/XXX.mp4"
workingPath = "/path/to/video"
```


## 2.2 启动服务端

在启动服务端前，确保种子文件已经更新。

启动服务端之后，服务端会接受多个客户端的请求，并通过choked, interested等函数来判断客户端状态：

在命令行执行以下命令：

```
python -m pybtracker.server -b 127.0.0.1:8000 -O
```



```
tracker — python -m pybtracker.server -b 127.0.0.1:8000 -O — 8
/anaconda3/lib/python3.6/runpy.py:125: RuntimeWarning: 'pybtracker.server' fo
in sys.modules after import of package 'pybtracker', but prior to execution o
'pybtracker.server'; this may result in unpredictable behaviour
  warn(RuntimeWarning(msg))
2018-10-31 20:52:52,108 - INFO - Started listening on 127.0.0.1:8000.
^C
2018-10-31 20:52:54,112 - INFO - Tracker stopped.
soras-MacBook-Pro:tracker yangyuanhao$ python -m pybtracker.server -b 127.0.0
8000 -O
/anaconda3/lib/python3.6/runpy.py:125: RuntimeWarning: 'pybtracker.server' fo
in sys.modules after import of package 'pybtracker', but prior to execution o
'pybtracker.server'; this may result in unpredictable behaviour
  warn(RuntimeWarning(msg))
2018-10-31 20:53:56,892 - INFO - Started listening on 127.0.0.1:8000.
^C
2018-10-31 20:55:19,139 - INFO - Tracker stopped.
soras-MacBook-Pro:tracker yangyuanhao$ python -m pybtracker.server -b 127.0.0
8000 -O
/anaconda3/lib/python3.6/runpy.py:125: RuntimeWarning: 'pybtracker.server' fo
in sys.modules after import of package 'pybtracker', but prior to execution o
'pybtracker.server'; this may result in unpredictable behaviour
  warn(RuntimeWarning(msg))
2018-10-31 20:56:38,159 - INFO - Started listening on 127.0.0.1:8000.
```

## 2.3 启动客户端

分别进入1, 2两个文件夹，在这两个文件夹下执行以下命令：

```
python -m pybtracker.client udp://127.0.0.1:8000
```

这样就开启了两个客户端，我们需要保证客户端代码目录下有我们想下载的文件备份

```
tracker — python -m pybtracker.client udp://127.0.0.1:8000 — 80...
warn(RuntimeWarning(msg))
BitTorrent tracker client. Type help or ? to list commands.

(btrc) ç
** Unknown syntax: ç
(btrc)
soras-MacBook-Pro:tracker yangyuanhao$ python -m pybtracker.client udp://127.0.0.1:8000
anaconda3/lib/python3.6/runpy.py:125: RuntimeWarning: 'pybtracker.client' found
in sys.modules after import of package 'pybtracker', but prior to execution of
pybtracker.client'; this may result in unpredictable behaviour
warn(RuntimeWarning(msg))
BitTorrent tracker client. Type help or ? to list commands.

(btrc)
soras-MacBook-Pro:tracker yangyuanhao$ python -m pybtracker.client udp://127.0.0.1:8000
anaconda3/lib/python3.6/runpy.py:125: RuntimeWarning: 'pybtracker.client' found
in sys.modules after import of package 'pybtracker', but prior to execution of
pybtracker.client'; this may result in unpredictable behaviour
warn(RuntimeWarning(msg))
BitTorrent tracker client. Type help or ? to list commands.

(btrc)
```

## 2.4启动下载端

下载端的功能主要是

1. 读取Torrent文件，并将数据初始化到客户端内部数据中。
2. 获知文件名后，检查"data/"文件夹下是否有历史数据块，有则加载，无则不管

进入download文件夹下

```
python torrio.py torrent文件位置
```



❏ bencode.py  
❏ bittorrent.py  
❏ README.markdown  
❏ simplifiedb.py  
❏ tests  
❏ tests.sh  
❏ torrent.py  
❏ tracker.py  
❏ Trying  
❏ util.py



```
b'"\x90\xd4\xa8\x16\x7f\xc9=\xf3\xc0\xf0\xcag\xce\xc4\xa3'  
b'2z\xe4\xff\xb0\xe2\x1d~\x8c\xe2\xe3Y\xe2\x13Z\xe7'  
b'X\xcd<\x93A\x12R%\xf3\xb6\x10k\xb8\x8a\xd7\\xa07\xd9}'  
b'w\x0f\x10\x18z\xe0v\x8d\xd8hn\xc6e\t7rZ\x17\xf7\x9c'  
b'\x7f\xe0\xdb\xdb\x87\xb9\x80\\;g\xe9]\x1d\xc7\x82\xd8'  
b'\xa4\x14\xcd\xab\xea\xe3\x7f\xf0,\xe5\x1d\x14'  
b'\x83\xdb\xd7\xfc\xb6\xbc\x8e\x9etz\x8c\x88\x8c\x13\xee'"  
b'A'\xa8\xd2r\xcd\xa6\x0eB\x17\xc5\x95/\x95\xab\xa4"  
b'l'rZ\x9ac\x96\xf2\x1bo\xc8\xff\xe3\xcf\xa4j65\r\xca\x17'  
b'V\x07\x96\xabj,\xd9\xcc\xa7_\x92mFV\xc2\xbf\4\x1e\x87'  
b'M\xf5\t\xe9\x1d\xf5q\xb4\xf7\xd7\xc2\xc3'  
b'\xda\xa9\xe3\x8f\t\x95\x1f\xccw~\xee<\x07\x99\xc2W'  
b'\xfb\x04\xd4P\xa7\x80Lm\xf2\xd4~\xd6\x00\x88\x90\x19'  
b'\xf2\xf2\xc9ry\xde}}0\xc50\xea0w=\xc9*\xf7\xec'  
b'\xabY\xa1\xcc\xe1\xe2/\xc9X\xc65Z\xe9\xdfY\xa7'  
b'\x87*\xdc\x9b\xe9n\xdbA\xd0\x00\xbe\x7fZ\x06\xbb\xbf'  
b'\xfb\x93\xb9\nI\xbb\x02\xe1\x9b\xeb\x1f5'  
b'\xe1\x99\x9b\xe4@6\xa2\xb4\x9e\xda \xde\xc2z\\xdf'  
b'\xa8\x10r\xd9\xc4\xda\xed\x88\x12^\xde } \x88\xaa\x8f'  
b'l\xbcMh\x8c\x07\xca\xc0\x85\xf6\xab\xe8F)F\x07'  
b'\x92N\xdf\xa5\x04\x91\x85\x824\x04C\xb7\xa0L\xb3,'  
b'\x1a\xa8w\xb6\xfcy\xc7\x10\x02\xe2\x92\xde\xdf\xfe\xfd'"  
b'\xa6\x94p\x1eL\x91\x90\xd9Uy\xf3\xa9w\xef\xfb"'  
b'&k\x8e\x95\xa3\x8f8\x81\xfb)\x1b\xb5N\xbb\xce\xe9'  
b'F&\x92\x92\xb1e\x0ex;\x8fL\xc2\n:.\xf0T)(\xc0\xc7\xb3)%'  
b'\x89*\x04\xff\xf1\xb9\xdb\x14{\x97\xf9\x04'  
b'\x19\x8e\xef\xe5\x12\x14\xe5\xafK\xf1cs\xcd\xc8\xdc,'  
b'\x95\x06\x8d\xfc\x19L[\xa7\n\xf7tcP|\xcd\x9d'  
b'\xfc\xf3\x8b\xf0e\x1b\xd0E\xd8\x01\xf9^\x89Aumy0J\x80'  
b'0K_&K\xfa9J9eV\x1c\xd3\xe1T\x8d\xa54\x89\xed'  
b'\xbb\x15\xc7\xc5)\xe7\xa85\x13\xd9:\x12~\xeaCPF*\xbbA'  
b'A\xf9Rx/\x91'\x91\xc45\xe4" \xf0\xb1\xbc'  
b'\x0f\x86\xfb\xa.\xae\x87E\xa4\xfe\xad\xb7\xd1\x93n\xe5'  
b'6g\xd8\x92\xd1Y\xbdN\x9b\xd5rE\x07KFL\xf0\xcb\xc1['  
b'E\x8a\xa7\x03\x19\xc9\r}\xc5\x86\x12\x94'  
b'\xf2\xb4\x13\xac%\xe1r\xa9\x19\xc9t/!\x96\xe1\xa1'  
b'\x80\xe6H%\x9d\x0b\xfa\x10\x1d\x88v\xa3\xf55A\x19'  
b'\xe1\x1fE\x07\x1dY\xea\xa5<\xab\x85\x10\\]\S'  
b'\x84\x92\x82\x81\x84\xd1\xc5T\xc4\xdd^m\x036`!\xea|\\'  
b'\x1b\xc6\xd2EF\xa5\x0e\xe6\xea\xf8\x89\x1d\x93\xc4\x89.'  
b'\xf1\xfc\xd1\x07y\xc9\xb2\xed)*\xaa;\x1eM\xc5\xc2'  
b'i\x8bc\xf5\x03\x19\xb7\xc9x\x1a\xebx\xcd\x0fk\x05'  
b'\x08$N\xbf\x95?d[\xfeg\x95Y\xfa\xdd\xcb\x9c4\x8e\xcac'  
b'rc\xc9x\xde4t7R\xc2*\x82aT\x9b)\v\xd5[c',  
b'private': 0}}
```