BT 协议实验项目最终报告

组长: 苏成　　组员: 司雪敏　雷雨豪
学号:111220105,111220103,111220052
邮箱:chengsu.nju@qq.com
计算机科学与技术系 2011 级

1　项目完成情况

• 第 1 周：05 月 16 日 - 05 月 22 日
规划：完成项目规划报告，开始编写基本功能的代码。
实际情况：已完成报告。

• 第二周：05 月 23 日 - 05 月 29 日
规划：编写基本功能的代码。
实际情况：已完成。

• 第三周：05 月 30 日 - 06 月 5 日
规划：完成基本功能的代码。
实际情况：已完成，并验收。

• 第四周：06 月 6 日 - 06 月 12 日
规划：开始编写额外功能的代码。
实际情况：已完成整个 BT 协议程序，并验收。

• 第五周：06 月 13 日 - 06 月 19 日
规划：完成整个 BT 协议程序。
实际情况：已完成整个 BT 协议程序，并验收。

2 摘要

在本次项目中，我们完成了一个小型的 BT 协议，其中具体完成了 BT 协议中 1. peer 与 tracker 之间进行交互，获取 peers 信息。2. peer 与 peer 之间互连，传送数据。3. 在 BT 协议额外功能中，我们完成了 3 个功能：最少优先，最后阶段和断点续传。我们使用 Wildlife.wmv 做种，以及使用了公网的种子测试了我们的 BT 协议的正确性。另外，我们实现了一个小型的 tracker, 我们的多个 peer 都可以像 tools/tracker 一样与我们自己的 tracker 进行交互。

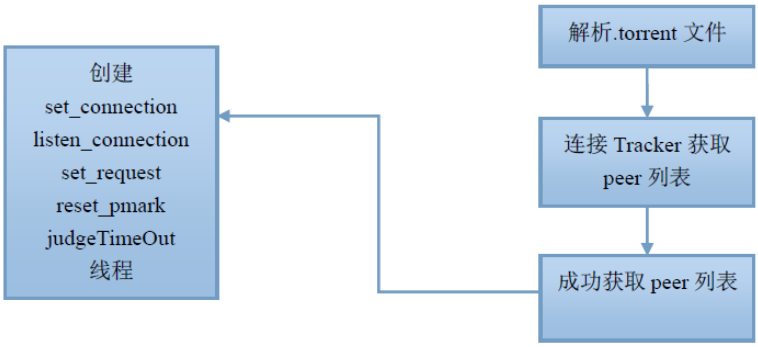
3 需求

	功能实现	成员分工
基本功能	peer 与 tracker 的交互	苏成
	peer 与 peer 之间面向数据消息的处理	雷雨豪
	peer 与 peer 之间面向状态消息的处理	司雪敏
额外功能	tracker 的实现	苏成
	断点续传	
	最少优先	雷雨豪
	最后阶段	司雪敏

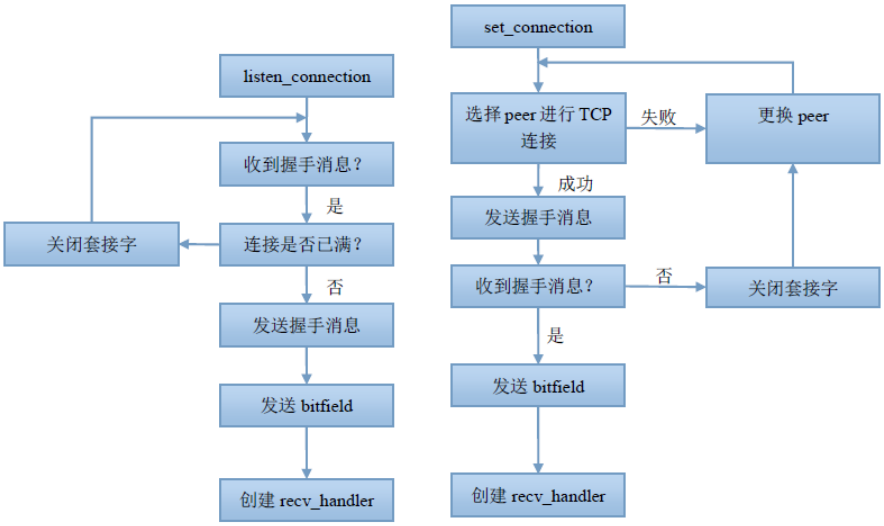
4 设计

4.1 设计流程图

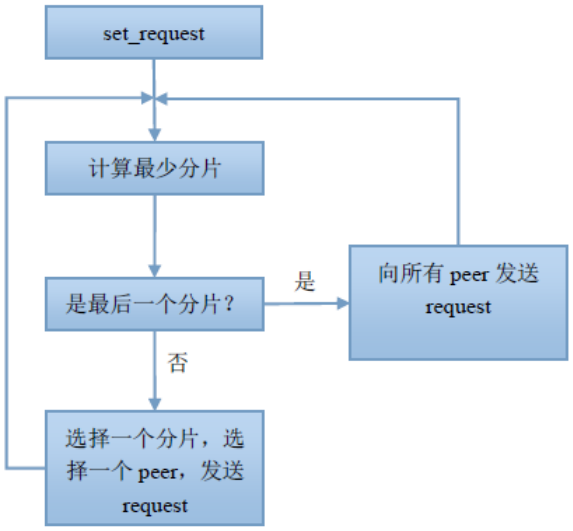
4.1.1 项目总流程图



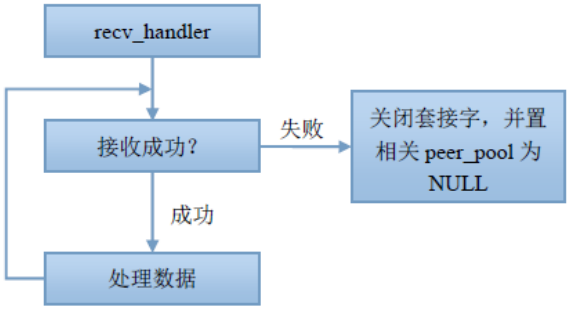
4.1.2 peer 与 peer 交互总流程图



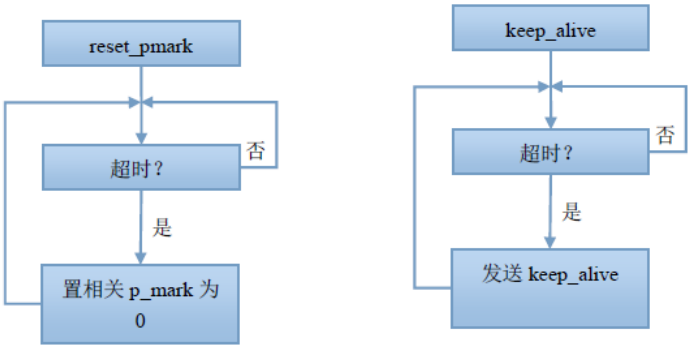
4.1.3 发送 request 函数流程图

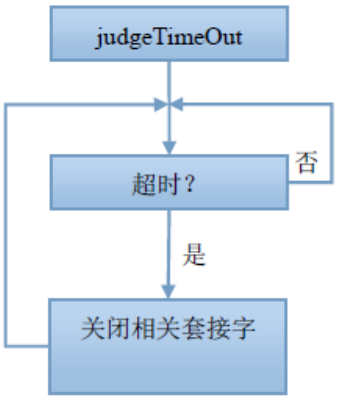


4.1.4 peer 接收数据函数流程图



4.1.5 重置分片接收标志位、keep alive 及判断 peer 连接是否超时函数流程图





4.2 peer 与 tracker 之间进行交互

4.2.1 设计思路

peer 与 tracker 之间进行交互，主要的方式是，BT 客户端通过 make_tracker_request 函数，指定参数为开始、完成、停止或不指定事件类型来生成发给 tracker 的报文，然后通过创建 TCP 套接字来和 tracker 建立连接，发送该报文。再通过 preprocess_tracker_response 函数从 tracker 读到返回的报文。tracker 返回的报文给出了针对相同种子文件的 peer 列表，以及有多少个 peer 拥有完整文件、有多少个 peer 还需下载文件、BT 客户端需要间隔多少时间向 tracker 发送一个不指定事件类型的报文等。

其中需要注意的点是：1.peer 在每隔 tracker 规定的时间内，需要给 tracker 发送一个不指定事件类型的报文，这样的功能类似 keep alive. 2. peer 在用户输入 Ctrl+c 命令终止程序时，需要给 tracker 发送一个 STOPPED 报文，来让 tracker 知道自己已经下线，否则 tracker 不会将该 peer 的信息去掉。3. peer 在由非种子变为 seeder 时，需要给 tracker 发送一个 COMPLETED 报文，来让 tracker 知道自己已经变为种子。

除此之外，peer 与 tracker 之间交互，还涉及到对 tracker 返回报文的 B 编码解码处理，其中主要完成 peers 的二进制模式修改为字典列表模式。在下面的实现方法中，会详细说明。

4.2.2 实现方法

在 BT 客户端和 tracker 交互的这部分，make_tracker_request 不需要改动，最主要是完成 preprocess_tracker_response 中的改动：将 tracker 返回的报文中 peers 字段由字符串的二进制形式改换成字典列表模式。通过阅读 get_peer_data 得知 peers 的字典列表模式应该是 ld...ed...ed...ee，其中每个 d...e 存储一个 peer 的信息，其中存储 peer_id(字符串形式), ip(字符串形式), port(整数形式), 由于 peer_id 在报文中没有出现，这里简单地处理成全 0 的形式，于是得到一个 peer 的字典模式是 d7:peer id20:(20 个 0)2:ip(ip 长度):(ip 的点十进制表示)4:porti(端口号)ee. 由此，只需要遍历 tracker 返回报文的 peers 字段，根据 6 的整数倍，将前 4 位的 ip 的 32 位整数形式，通过 inet_ntoa 函数转换为 ip 的点十进制表示，将后 2 位的 port 通过 ntohs 函数将网络字节序转换为主机字节序，得到端口的整数表示。这些 ip, port 都通过 sprintf 输入到存储修改后的报文的缓冲区字符数

组中。这样，就完成了报文的修改。由 tracker 返回的报文中，从 6:peers 后面开始读取 peer 列表的内容，其中前 4 位为 ip 的 32 位整数形式，后 2 位为端口号，其中需要注意的内容是如何获取到 ip 和 port, 通过一个 struct in_addr 指针指向 ip 所在的位置，然后通过 inet_ntoa 函数来得到 ip 的点十进制表示。对于 port 来说，用一个 16 位整数的指针，即 uint16_t* 来指向 port 所在的位置，然后使用 ntohs 将网络字节序传换为主机字节序就得到了 port 的内容。

除此之外，当 BT 客户端的程序接受到 Ctrl+c 停止命令时，需要断开和 tracker 的连接，以及和其他 peer 断开连接，终止整个程序的运行。具体实现在 client_shutdown 中，首先通过框架代码中提供的 make_tracker_request 生成 STOPPED 报文，并创建套接字，发送给 tracker, 我们将所有连接的 peer 放入一个 peer_t 的数组中去，每个 peer_t 结构体中会记录本客户端与该 peer 连接的 TCP 套接字，在成功发送给 tracker 停止报文以后，给每个 peer 通过上述结构体找到套接字，关闭套接字，完成与其他 peer 的连接断开。

另外，当 BT 客户端接收完整个文件以后，需要给 tracker 发送一个 COMPLETED 报文，实现方式是：我们为该文件的每个分片建立一个标志位，标志为 0 代表客户端没有该分片；标志为 1 代表客户端拥有该分片的一部分 (由于 BT 协议会将分片再分成子分片来请求所致)；标志为 2 代表客户端拥有该分片的全部。因此，在我们的客户端每次接收到一个新的分片时，遍历所有分片的标志位，查看是否拥有所有分片来判断文件是否接受完毕，当发现文件接受完毕后，通过 make_tracker_request 函数给 tracker 发送一个 COMPLETED 报文。我们一开始建立一个字符数组来存储接受的数据，当接受完毕以后，将字符数组写入到一个新的文件中去，代表接受到的文件。

4.3 peer 与 peer 之间面向状态消息的处理

4.3.1 设计思路

在分配任务时我所需要实现的功能是 peer 之间的面向状态消息的处理，因为 peer 之间状态处理与数据处理部分的交集过大，与队友之间的分工不好协调，所以我最终实现的内容为：

（1）peer 之间的 handshake 过程与 handshake 成功后对相关函数（keep_alive, recv_handler 和 send_bitfield）的调用；

（2）keep alive 报文的发送；

（3）判断超时函数的编写。

而其他相关状态队友在实现数据传输时在状态机里直接进行了处理。

【1】handshake 过程 handshake 时本地所需要实现的功能是作为服务器与客户端，作为服务器是接收其他 peer 向自己发送的 handshake 报文，作为客户端是向其他 peer 发送 handshake 报文。在 handshake 成功后会启动两个线程，一个是 keep alive 线程，每隔定时就向其他已经建立连接的 peer 发送 keep alive 报文；另一个是 recv handler 线程用于处理之后的数据和状态的收发过程。同时我们定义了全局变量 peer_pool 与 MAX_PEERS 用来限制最大连接数，本地作为服务器接收其他 peer 连接与作为客户端连接其他 peer 时都共享该 peer_pool。

【2】keep alive 报文发送 keep alive 的实现比较简单，只要定间隔向其他已经建立连接的 peer 发送 keep alive 报文就可以了。

【3】判断超时函数 judgeTimeOut 该函数运行在 main 函数中，由 main 函数创建线程执行。是为了判断某个 peer 是否已经” 消亡”，采取的做法是在每个 peer 中添加成员 send_time，每次该 peer 发送数据或状态时就会更新该成员，而函数 judgeTimeOut 所需要做的事情就是获取当前时间，并用当前时间减去每个 peer 的 send_time，判断结果是否大于某个时间间隔，若大于，则说明该 peer 已经” 消亡”，就断开和它之间的连接。

4.3.2 实现方法

【1】handshake 过程

（1）数据结构

handshake 报文格式：

```
//握手报文格式
typedef struct _handshake
{
        unsigned char len;
        char name[19];
        char reserved[8];
        int info_hash[5];
        char peer_id[20];
} handshake;
```

从 tracker 那儿得到的解析后的 peer 列表

```
// 由tracker返回的peer列表
typedef struct peer_list
{
        peerdata this_peer;
        struct peer_list* next;
}peer_list;

// 由tracker返回的peer列表头指针
extern peer_list* peer_head;
```

peer_pool 的实现：

peer_pool 的实现为指针数组形式，最大限度为 MAX_PEERS，每次有新的连接时寻找 peer_pool 中为 NULL 的项，并置该项为新的连接；当关闭连接时，则将对应的项给释放掉并置其为 NULL。

对 peer_t 结构体的修改：

```
// 针对到一个peer的已建立连接，维护相关数据
typedef struct _peer_t {
  int sockfd;
  int choking;          // 作为上传者，阻塞远端peer
  int interested;       // 远端peer对我们的分片有兴趣
  int choked;           // 作为下载者，我们被远端peer阻塞
  int have_interest;    // 作为下载者，对远端peer的分片有兴趣
  char name[20];
    unsigned char *peer_mark;
    int send_time;
} peer_t;
```

peer_mark 用来对本地需要接收的分片做标记，0 代表该分片未下载，1 代表该分片正在下载，2 代表该分片下载成功，peer_mark 的大小与需要下载的分片大小相同，在 handshake 成功后进行申请。

send_time 是用来记录该 peer 最后一次发送的时间，每次 peer 发送数据时都要进行更新。

（2）主要函数实现 int compare_info_hash(int info_hash1[], int info_hash2[])：该函数用来对接收到的 handshake 报文中的 info_hash 进行比较，判断是否与本地的 info_hash 相同。int find_peer_id(char peer_id[])：该函数用来对接收到的 handshake 报文里的 peer id 进行判断，判断是否属于 tracker 返回给自己的那个 peer 列表中，但后来在于写 tracker 的同学交流后发现实际 tracker 返回的并没有 peer id，或者说 peer id 没有意义，所以最终该函数并没有用处。int handshake(int info_hash[], int sockfd)：该函数用来向套接字为 sockfd 的发送 handshake 报文，发送成功返回 1，发送失败返回 -1。void listen_connection()：在该函数里本地作为服务器不断接收对方的连接，在接收到对方的 handshake 报文时首先需要判断

其 info_hash 是否正确，如果确定与” 我” 连接的 peer 是属于同一” 群组” 的，那么” 我” 就将该 peer 加入到本地列表中去（加入的判断在讲解 peer_pool 时有详解，这里就不再赘述），如果添加成功则向该 peer 返回一个 handshake 报文，否则将该连接关闭。添加成功时申请 peer_mark 空间和置发送时间为当前时间，之后判断” 我” 是否是 seeder，如果是的话就向该 peer 发送 bitfield 报文，如果不是就不发送，同时启动线程 keep_alive 与 recv_handler。void set_connection(peerlist *peer_list)：在该函数中本地作为客户端向服务器返回给本地的 peer 列表中发送 handshake 报文，如果接收到某个 peer 的响应消息则将其加入到本地列表 peer_pool 中，如果添加失败则关闭该套接字。在成功添加之后申请 peer_mark 空间和置发送时间为当前时间，并判断” 我” 是否是 seeder，如果是的话就向该 peer 发送 bitfield 报文，如果不是就不发送，同时启动线程 keep_alive 与 recv_handler。

【2】keep alive 报文发送每隔定时本地就向 peer_pool 里的每个成员发送 keep alive 报文，如果发送失败，则说明该 peer 已” 消亡”，则将该成员从 peer_pool 里给移除并置该项为 NULL。

4.4 peer 与 peer 之间面向数据消息的处理和最少优先功能

4.4.1 设计思路

面向数据消息的处理，按照函数可分为以下几部分：

1.send_bitfield

2.send_request

3.recv_handler

4.send_have

5.p2p_send

6.p2p_recv

4.4.2 实现方法

1. send_bitfield

与中期报告时调用方式稍有不同，每一个连接建立时，在握手协议后，都会给对方发送一个 bitfield，让对方知道自己拥有哪些分片。因为我们已经实现了断点续传功能，因此在开始时将会读取已有文件 .bt，然后通过计算 SHA1 值判断已有哪些分片，在 p_mark 这一变量中进行标记，然后再

发送 bitfield 时判断 p_mark 中相应的值即可知道哪些比特位的应该置为 1。

2. send_request

在中期报告时的代码上进行了修改。为了完成最少分片优先，每隔 RAREST_FIRST_TIME 就会根据当前所有 p_num 对每个分片的计数进行重新排序，选出当前所连接网络中分片数最少的分片，然后再寻找到拥有该分片的用户，发送 request。

3. recv_handler

与中期报告时基本相同，但是增加了部分改进：(a) 收到 have 或者 bitfield 时，会将相应分片的 p_num 计数器增加，以供最少优先排序使用；(b) 接收完一个 piece 时会计算该 piece 的 SHA1 值，判断其和解析 torrent 所获取的该分片的 SHA1 值是否相等，若相等则将其存入缓冲区中，否则丢弃并置该分片的 p_mark 为 0；(c) 当所有分片都已完成时将会写文件，并将该 peer 设置为种子文件。

4. send_have

与中期报告时相同。在 recv_handler 收到 piece 消息并且发现相应分片接收完毕并验证 SHA1 值正确时，将会向其他 peer 发送 have 消息。

5. p2p_send

该函数为发送消息数据接口，与中期时相比进行了部分改进，在其中的 send 函数返回的字节数 n 如果小于所给定的 len，则会循环发送，直至发送的字节数累加之和达到所给定的 len。

6. p2p_recv

该函数为接收消息数据接口，与中期时相比进行了部分该井，因为 recv 实际收到的字节数并不一定等于给定的 len，因此需要循环读取，直至读取的字节数累加之和达到所给定的 len。在中期时因为没有采用循环读的方式，当数据过大时有很大的概率会出现问题，在本次对 p2p_recv 进行改进后该问题得到了解决。

7. 最少分片优先

在全局中加入了新的变量 p_num 动态数组，每当收到 bitfield 或者 have 消息时将会把相应分片的 p_num 计数值增加。在 set_request 阶段，每隔 RAREST_FIRST_TIME 就会计算一次当前连接中哪个分片数最少，并且根据这个排序来选择分片下载。

8. p_mark 重置

实验中拥有一个全局变量 p_mark，记录每个分片当前的状态，0：尚

未开始；1：已开始下载；2：完成/拥有该分片。因为可能存在本地 peer 向某个远端 peer 发送了 request 但是远端 peer 在该分片尚未完全传递完成时与本地 peer 断开了，此时则需要将相应的 p_mark 置为 0，让其重新下载。为了实现这一目标，在本次修改中加入了 reset_pmark 函数，当一定时间后则会启动一次，对所有的 p_mark 进行检查，若为 1 则修改为 0。因为本实验中下载速度较快，因此将测试睡眠时间设置为 30 秒，实际运行时设置为 1 分钟最好（当该分片下载速度达到 4.3KB/s 即可完成），以防因速度过慢而反复重置导致无法完成下载（实际时间如果设置得长一点也许更好）。

4.5 断点续传

4.5.1 设计思路与实现方法

文件的断点续传，需要实现的目标是，当文件下载中，用户按下 Ctrl+c 命令时，需要将文件保存下来，然后当用户再次启动该文件的下载时，已下载的部分无需重复下载，用户可以接着上次未下载的内容进行下载。我们在实现断点续传时，将文件保存下来以后存储为后缀名为.bt 的文件，来表示文件未下载完全，其中实现的关键点在于如何进行续传，我们的实现方式是，一开始开辟一个文件大小的全 0 的字符数组来保存需要下载的文件，在下载过程中，如果下载到了某个分片，就将该分片放置在数组中它应该在的位置，根据索引值和偏移量得出其起始地址，根据其长度得到其终止地址。这样，如果某个分片的所有子分片全部下载完全，也就是说该分片全部下载完全，那么如果我们计算该数组中该分片的 SHA1 值，就应该和 torrent 文件中记录该分片的 SHA1 值相同。这样，我们通过再次启动程序后，读入保存在.bt 文件中的字符数组，也就是未下载完全的文件，分别计算每个分片的 SHA1 值，并和 torrent 文件中每个分片的 SHA1 值进行一一比对，如果相同，代表该分片下载完全，不需再次下载。如果不同，代表该分片未下载完全，我们实行对该分片的重新下载。

除此之外，在实现的细节上，我们需要决定在什么时候生成.bt 文件。在程序刚一启动时，我们通过命令行参数来约定该客户端是否为 seeder, 如果不是，我们再看是否有生成的.bt 文件，如果有，我们读入该文件，进行每个分片是否需要续传的判断。在文件接收完毕时，我们将真正的文件创建，并写入磁盘，此时，如果有生成的.bt 文件，我们将其删除。在用户按下 Ctrl+c 后，我们来判断是否用户的文件接收完毕，这里，由于在用户接

收完文件后，必定会创建文件并写入，此时我们就可以通过判断用户是否创建该文件，即判断磁盘上是否存在用户刚刚创建的文件，如果没有，我们认为用户没有下载完全部文件，需要生成对应的.bt 文件。

4.6 最后阶段

4.6.1 设计思路

之所以要实现最后阶段，是因为有时候 peer 会从一个速率很慢的 peer 那里请求一个分片。这在下载的中间阶段并不是什么问题，但是却可能延迟一个下载的完成。为了避免这种情况的发生，在下载的最后阶段（end game 阶段），peer 会向它的所有 peer 发送所有子分片的请求。一旦收到某些子分片了，那么就会向其它 peer 发送 cancel 消息，取消对这些子分片的请求，以避免带宽的浪费。

这部分的实验思路很直接，就是需要在接收 piece 的时候判断一下是否是倒数第二个 piece（因为具体实现的原因，我们是在发送 request 时进行判断），如果是的话则向所有拥有最后一个 piece 的 peer 发送 request 报文，一旦收到了最后一个 piece 则向非该 peer 的其他 peer 发送 cancel 报文取消之前的请求。

4.6.2 实现方法

【1】数据结构的增加与改变增加了 g_index(int 类型)，用于记录最后需要的分片的下标，如果当前请求的分片非最后一个分片，那么 g_index 的值为 -1。在结构体 peer_t 中增加了状态 cancel，用于标记此 peer 是否被本地 cancel，如果其值为 0 无影响，为 1 的话则不接收 piece。增加了互斥锁 peer_mutex，用于收到最后一个 piece 将其他 peer 的 cancel 位置为 1 时加锁，这样能够避免同时对共同数据域进行修改（多个线程共用 peer 列表）。

【2】具体实现

（1）对函数 set_request() 的修改：set_request() 函数是用于请求一个 piece 的第一个子分片，所以需要在该函数内进行判断当前是否只差一个分片，即是否处于最后阶段，如果是最后阶段则遍历本地存储的 peer 列表 peer_pool，去寻找拥有本地需要的分片的 peer 并向其发送 request 请求；如果不是最后阶段，那么就正常执行 request 操作。

（2）对函数 recv_handler() 的修改：接收 piece 时的操作：此处的主要修改是在接收到 piece 判断一下该 piece 的下标是否是 g_index，如果是的话说明本地已经从当前 peer 那儿得到了最后一个分片，”我” 就立刻将其他” 我” 已经请求的 peer 的 cancel 位置为 1，同时向它们发送 cancel 报文。因为引入了 cancel 这个状态，所以还需要判断当前与本地连接的 peer 的 cancel 状态，如果是 1 的话那么就直接跳出，不做任何操作，注意，该判断在置位与发送 cancel 报文之前，并且在修改其他 peer 的 cancel 时需要加锁（加锁原因已在前面有所提及）。接收 cancel 时的操作：接收到 cancel 报文时将该连接关掉，将对应的 p_mark[i] 位改为 0（表示未下载，因为下载失败），同时释放掉其所占有的 peer_pool 空间，等待新的连接。

（3）增加函数 reset_pmark()：因为有时候会出现方突然将连接断掉的情况，而此时本地可能正与其进行着连接进行下载，所以在碰到本地应该能正确的区分这种情况并且将下载的分片由原来的正在下载标识为没有下载。判断方式是每隔 30 秒检查一次本地的 p_mark[i]，如果其值是 1，那么表示和该分片相关联的 peer 突然关闭，将其值重新置为 0，即重新下载，即使不是上述情况，面对这样慢的下载率，我们也是有理由向其他 peer 重新请求的。

4.7 小型 tracker 的实现

4.7.1 设计思路与实现方法

tracker 和 peer 之间交互，最重要的部分是 tracker 需要维护好现存的 peer 列表，然后正确地给 peer 返回它所需要的 peer 列表，一开始看来，tracker 的内容比较简单，所以打算将其实现。tracker 部分相当于一个服务器，循环监听 peer 与之进行 TCP 的连接，连接上以后接收 peer 发送的 HTTP 报文，从中解析出 peer 的 ip, port 以及 peer 的该文件 info_hash, 下载量，还需下载量等信息。tracker 首先判断该 peer 的 info_hash 是否与自己的 info_hash 相同，相同的话再进行下面的动作。根据 peer 的 HTTP 报文的 event 字段来判断需要进行的操作，如果是 started 事件，需要查看存储 peer 的列表，如果没有该 peer 的信息，需要将其加入到 peer 列表中。并返回一个带有 peer 信息的 HTTP 报文，如何返回该报文会在下一段详细说明。如果是 completed 事件，需要到 peer 列表中找到该 peer, 并用某个标志位来标记该 peer 变为 seeder. 如果是 stopped 事件，需要到 peer 列表中找到该 peer, 并将其删除。如果是不指定事件类型，代表 keep alive，

需要到 peer 列表中找到该 peer, 用某个时间戳来更新 peer 与 tracker 互连的当前时间。

tracker 给 peer 返回 HTTP 报文相对简单，tracker 首先需要去查看 peer 列表，记录所有的 peer 的 ip 和 port, 然后生成对应的 HTTP 报文，这里由于 HTTP 报文需要返回 content-length, complete, incomplete 等字段，content-length 需要再生成后面全部的 content 才能知道，而 complete, incomplete 则需要生成后面的全部 peers 才能知道，所以 HTTP 报文需要从后往前填入，先填入 peers 字段，再填写 complete, incomplete 等字段，然后再填写 HTTP 报文头，填写 content-type, content -length 等内容。

从上面可以看到，由于 tracker 进行最多的是 peer 列表的查找工作，当有多个 peer 连接 tracker 时，则需要提高查找效率。这里，我将 tracker 存储的 peer 列表使用哈希表来实现，哈希函数采用 P.J.Weinberger 提出的对字符串进行移位并异或的哈希函数。哈希键是 peer 的 ip, 当使用 peer 的 ip 来查找 peer 的信息时，通过 ip 来计算哈希函数，得到该 peer 所在的哈希表中的位置，我的哈希表是开散列形式，每个哈希的位置是一个链表，其中保存了哈希键相同的所有 peer 信息，然后遍历该链表得到真正的该 peer 信息。

5 实现细节与问题解决

5.1 peer 与 tracker 之间进行交互

其中遇到的主要的问题在于 HTTP 报文的读取，以及种子文件的解析，由于公网的种子和自己生成的种子不同，需要对公网的种子进行处理，其中公网的第一个种子，在计算 info_hash 时需要忽略末尾的 nodes 字段，否则会计算错误。另外，我们也出现过，连接公网的 tracker 时，只返回了很少的 peer, 并且尝试与它们进行连接时，均连接不上。后来发现是将自己发送给 tracker 的 left 字段没有赋值，是 0，这样 tracker 就以为我是 seeder，从而只给我发送那些很少的未下载完全的 peer. 当修改 left 字段后，成功读取到 50 个 peer 的信息。

在一开始实现时，我们将 peer 与 peer 之间进行 handshake 实现为循环，也就是说使用一个 while 循环，尝试去和每个 peer 进行 handshake，但是，如果实现为一个线程，当遇到一个 peer 不回应时，会在 connect 上一直等待直到超时，这是相当耗时的，于是将 peer 与 peer 之间进行

handshake 改成多个线程之间并发执行，这样就可以同时连接多个不同的 peer, 如果某个 peer 连接很耗时，连接不上也没有关系，只要存在一个 peer 能够连接上，就可以开始文件数据的传输了。

除此之外，在公网的种子中，会有 announce-list 字段来标记备用的 tracker 域名及其端口号，在我们的程序中，我们将这些 tracker 解析出来，以备和主 tracker 连接不上之时使用。

5.2 peer 与 peer 之间面向状态消息的处理

（1）在 listen_connection 函数与 set_connection 函数接收 handshake 报文时采用 while(recv(sockfd, buf, len, 0)>0) 的方式导致在 recv 函数处发生阻塞。在一开始编写代码时担心在函数中接收到的第一个报文并不是 handshake 报文，所以在代码里采取了 while(recv) 的方式，并在循环内部判断 handshake 报文，但这样的话就与数据处理时接收数据发生了冲突，在 handshake 成功后无法发送数据，后来经过思考，如果 peer 之间要进行连接，那么第一个发送到报文肯定是 handshake 报文，所以我的这种担心是没有必要的，于是就将 while 改成了 if，也使得数据能够正常接收与发送。

（2）类型隐式转换导致 if 判断语句判断出错。之前在 peer_t 增加的 send_time 的类型是 unsigned int 类型，而在 judgeTimeOut 函数内部获得的 current_time 确实 int 类型，所以再用 current_time-send_time 时得到的类型是 unsigned int 类型，而恰巧 current_time 的值是小于 send_time 的，那得到的数据在计算机内部就被认为是一个很大的数值，我在进行 current_time-send_time>WAIT_TIME 时得到的就为 true，但按理说此时 current_time-send_time<0 应该返回 false，所以这就导致代码认为 peer 一直超时，会将该 peer 对对应的套接字给关掉，结果就是在数据发送一段时间后就会停止发送。

5.3 peer 与 peer 之间面向数据消息的处理

在 recv 时获取的数据存在问题。

解决：在中期报告中发现了该问题，采用了 sleep(1) 的方式来解决。但是在之后的测试中发现这样并不能完全解决问题，还是有一定的几率会读取到错误的数据，并且该 sleep(1) 也严重影响到了 BitTorrent 的下载速度。经过输出发现，调用 recv(conn, buffer, len, 0) 时，该函数的返回值，假设

为 n，并不一定等于这里的参数 len，因此需要一个循环接收，保证最后获取到的字节数的累加值等于所需字节数 len，此时才返回相应的执行位置。经修改后的代码截图如下：

```
int p2p_recv(char *buffer, int conn, int len){
    int recv_len = 0;
    int left_len = len;
    int n;

    while(recv_len < len){
        if((n = recv(conn, buffer+recv_len, left_len, 0)) <=0 ){
            return -1;
        }
        recv_len += n;
        left_len -= n;
        assert(left_len >= 0);
    }
    assert(recv_len == len);
    return 1;
}
```

5.4 断点续传

断点续传遇到的问题主要是一开始计算已保存文件的分片的 SHA1 值和种子文件中每个分片的 SHA1 值总是不相同，后来将文件全部接收正确以后，再计算保存文件的每个分片的 SHA1 值，并和种子文件中每个分片的 SHA1 值进行比对，发现种子文件的每个分片的 SHA1 值是用网络字节序存储的，这时使用 hnonl 逐一将自己计算的 SHA1 值转换，解决问题。

5.5 最后阶段

（1）对 g_index 值的误判。

对 g_index 赋值的地方为：

```
for(i = 0; i < g_num_pieces; i++)
        if(p_mark[i] == 2)
                ++recv_pieces;          //记录已经下载了多少分片
        else if(p_mark[i] == 0)
                g_index = i;            //记录最后未下载的分片下标
```

这样所产生的问题在于如果当前不是最后一个需要下载的分片，因为此处对 g_index 赋了值，后面用到 g_index 的地方可能就会产生误判，以为此时的 g_index 就是最后所需要的分片。这里采取的做法是如果发现 recv_pieces（已经接收到了的分片个数）不是总数减一那么 g_index 的值就为 -1，后面在用到 g_index 时就可以根据 g_index 的值来判断是否是最后阶段。

（2）会收到对方发送的 cancel 报文。有个很奇怪的现象就是会在接

收数据的时候突然间收到对方的 cancel 报文进而就与对方断开了连接（有时会有一定延时，实验结果中有相应截图），这样的话那么本地接收分片时可能才接收到了一些子分片就无法再继续接收了，而因为向指定 peer 请求了某个分片便不会再向其他 peer 请求了，所以就会导致一直得不到该分片。采取的做法是引入了 reset_pmark() 函数（关于函数原理上面有所介绍，这里不再赘述），很好地解决了这个问题。

（3）无法发送 cancel 报文。cancel 报文的发送是在 set_request() 函数中的，但是该函数内部又调用了 sleep 函数，所以可能会出现因为 sleep 或者所有 peer 同传的缘故，在未能判断是否还差最后一个 piece 之前就收完了所有的 piece，这时就没有所谓的" 最后阶段"，也就没办法发送 cancel 报文。但考虑到这是很正常的情况，所以就没对其进行处理。

5.6 tracker 的实现

tracker 实现遇到的问题是当课程提供的 BT 客户端连接我们的 tracker 时，不会在 HTTP 报文中提供它自己的 ip 地址，后来通过查阅资料，通过 socket 的 accept 的第二个参数会记录连接的对方的 ip 地址和 port, 通过这个 struct sockaddr 结构体转换为 struct sockaddr_in 结构体，从中读出 sin_addr 域，再进行 inet_ntoa 转换，就得到我们需要的 ip 的点十进制表示形式了。

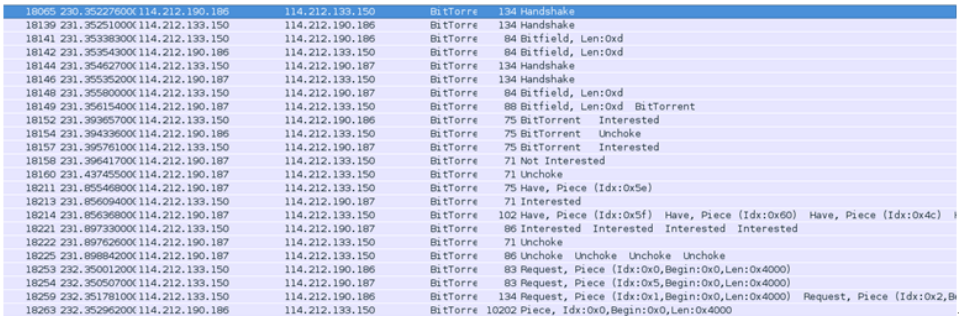
6 实验验证与结果说明

我们通过 wireshark 抓包，并配合程序输出信息，以及最终文件的接收情况来判断自己程序的正确性。以下分别为各个功能的实验验证情况。

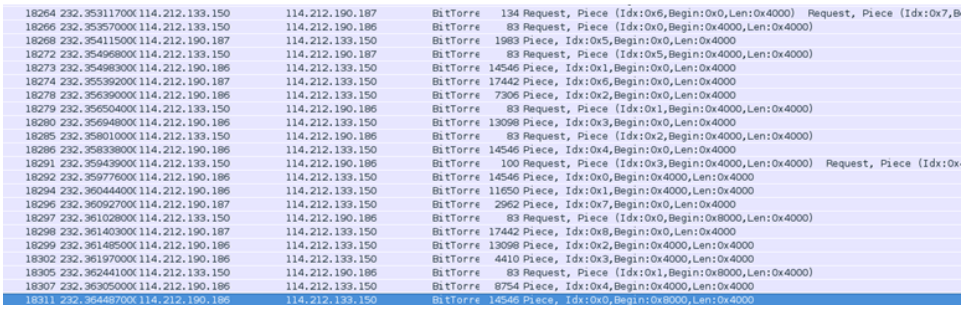
6.1 peer 与 tracker 交互，peer 与 peer 之间交互

114.212.133.150 下载者，本地机器 114.212.190.186 做种者，服务器机器 114.212.190.187 下载者，服务器机器下图中先是 114.212.190.186 与 114.212.133.150（本机）握手，114.212.133.150 回应握手消息，然后两者互发 bitfield 然后是 114.212.133.150 主动向 114.212.190.187 发送握手消息，114.212.190.187 回应握手消息，然后两者互发 bitfield 其中 114.212.133.150 对 186 于 187 的分片感兴趣，发送 interested 消息。186 在接收到 interested 消息后发送了 unchoke 消息给 150。187 也在收到 interested 消息后发送了

unchoke 消息给 150。因为 187 对 150 所拥有的分片不感兴趣，因此发送了一个 not interested 消息给 150。187 下载完一个新分片后，发送 have 消息给 150，150 检查发现本地没有改分片，因此发送 interested 给 187。然后 150 开始向 186 与 187 发送 request，在下图最底部是 186 发送 piece 消息给 150



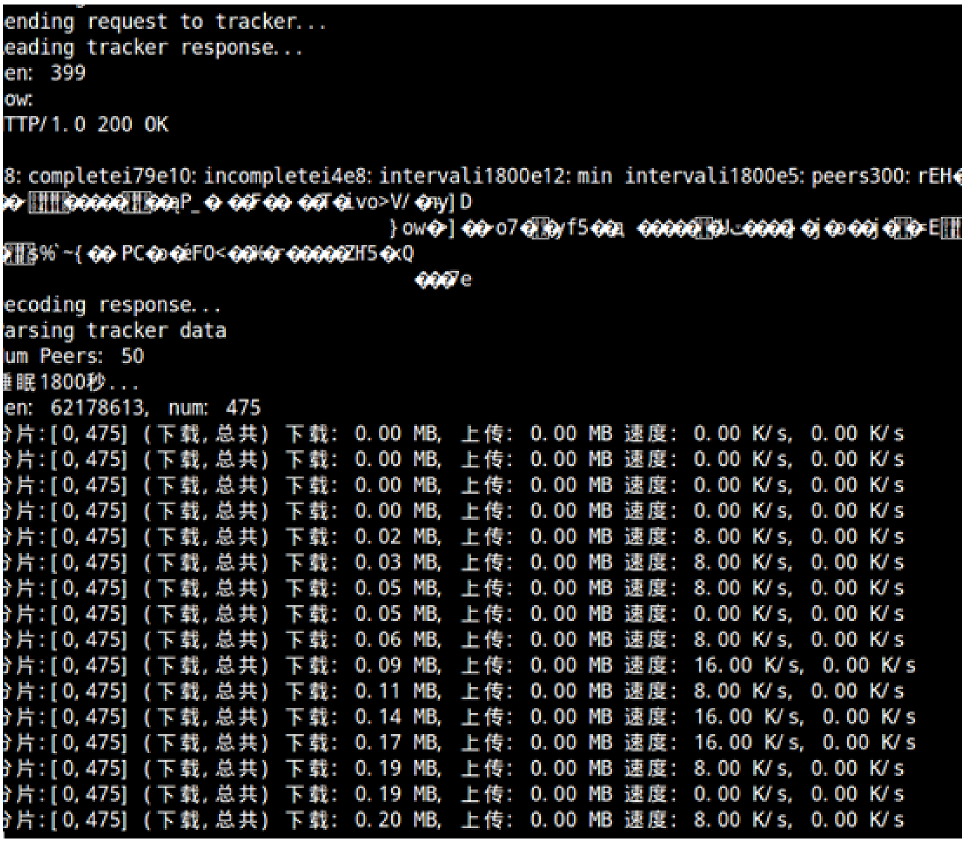
下图中都是一系列的 request 消息与 piece 消息。可以看到 186（做种者）与 187（下载者）都在给 150 发送 piece 消息，证明我们所实现的客户端可在下载的同时完成上传动作。



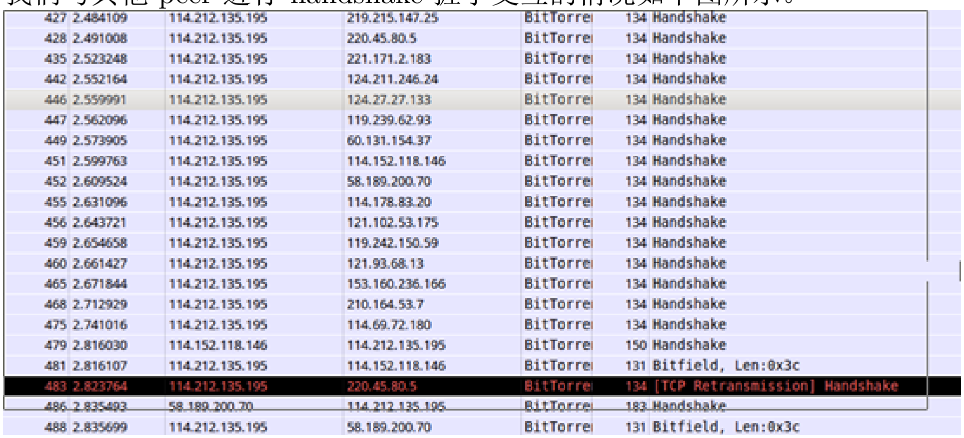
只剩最后一个分片时，向所有拥有该分片的 peer 发送 request 请求后，收到第一个回复的 piece 消息则立刻向其他 peer 发送 cancel 消息，这里显示 150 向 187 发送了 cancel 消息。如下图所示。

6.2 peer 与公网的 tracker 以及 peer 交互

我们选择的是第一个公网种子，如下图所示，我们与 tracker 交互得到的报文，以及下载开始的情况。

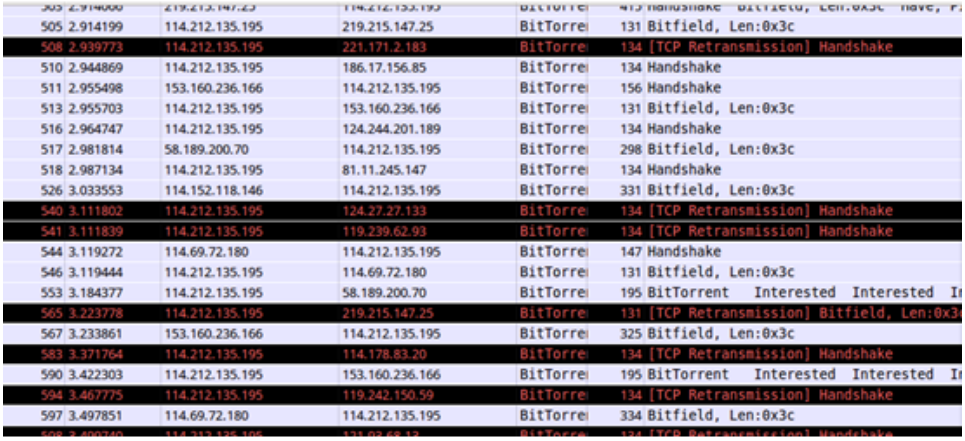


我们与其他 peer 进行 handshake 握手交互的情况如下图所示。



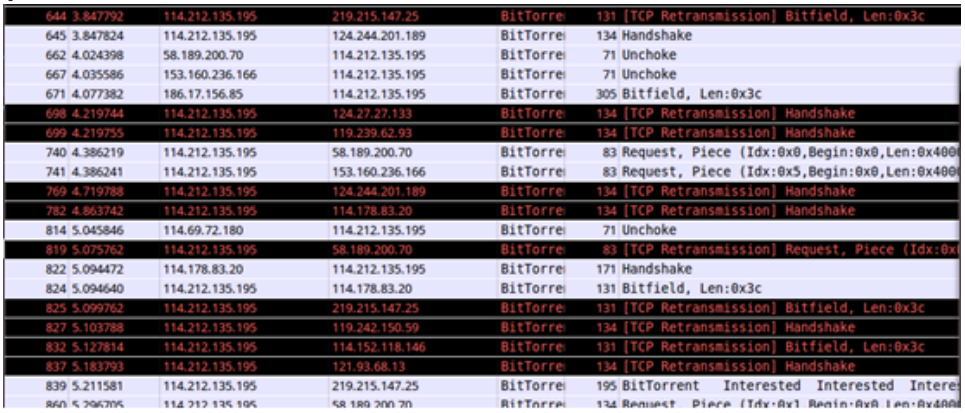
427	2.484109	114.212.135.195	219.215.147.25	BitTorre	134	Handshake
428	2.491008	114.212.135.195	220.45.80.5	BitTorre	134	Handshake
435	2.523248	114.212.135.195	221.171.2.183	BitTorre	134	Handshake
442	2.552164	114.212.135.195	124.211.246.24	BitTorre	134	Handshake
446	2.559991	114.212.135.195	124.27.27.133	BitTorre	134	Handshake
447	2.562096	114.212.135.195	119.239.62.93	BitTorre	134	Handshake
449	2.573905	114.212.135.195	60.131.154.37	BitTorre	134	Handshake
451	2.599763	114.212.135.195	114.152.118.146	BitTorre	134	Handshake
452	2.609524	114.212.135.195	58.189.200.70	BitTorre	134	Handshake
455	2.631096	114.212.135.195	114.178.83.20	BitTorre	134	Handshake
456	2.643721	114.212.135.195	121.102.53.175	BitTorre	134	Handshake
459	2.654658	114.212.135.195	119.242.150.59	BitTorre	134	Handshake
460	2.661427	114.212.135.195	121.93.68.13	BitTorre	134	Handshake
465	2.671844	114.212.135.195	153.160.236.166	BitTorre	134	Handshake
468	2.712929	114.212.135.195	210.164.53.7	BitTorre	134	Handshake
475	2.741016	114.212.135.195	114.69.72.180	BitTorre	134	Handshake
479	2.816030	114.152.118.146	114.212.135.195	BitTorre	150	Handshake
481	2.816107	114.212.135.195	114.152.118.146	BitTorre	131	Bitfield, Len:0x3c
483	2.823764	114.212.135.195	220.45.80.5	BitTorre	134	[TCP Retransmission] Handshake
486	2.835493	58.189.200.70	114.212.135.195	BitTorre	183	Handshake
488	2.835699	114.212.135.195	58.189.200.70	BitTorre	131	Bitfield, Len:0x3c

握手成功后，发送 bitfiled 报文如下图所示。



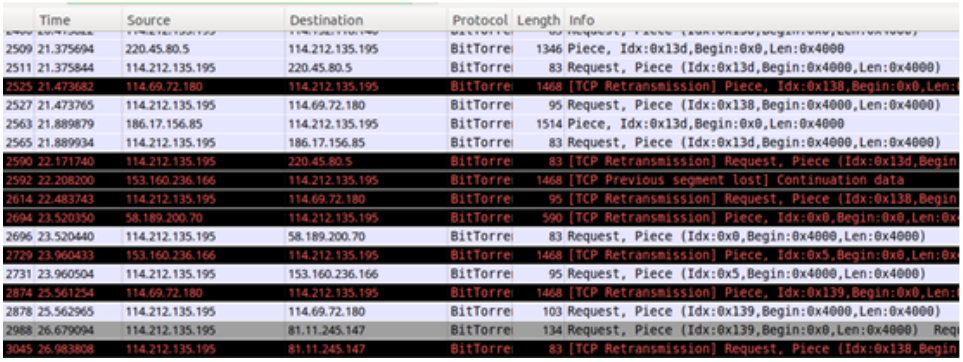
505	2.914199	114.212.135.195	219.215.147.25	BitTorre	131 Bitfield, Len:0x3c
508	2.939773	114.212.135.195	221.171.2.183	BitTorre	134 [TCP Retransmission] Handshake
510	2.944869	114.212.135.195	186.17.156.85	BitTorre	134 Handshake
511	2.955498	153.160.236.166	114.212.135.195	BitTorre	156 Handshake
513	2.955703	114.212.135.195	153.160.236.166	BitTorre	131 Bitfield, Len:0x3c
516	2.964747	114.212.135.195	124.244.201.189	BitTorre	134 Handshake
517	2.981814	58.189.200.70	114.212.135.195	BitTorre	298 Bitfield, Len:0x3c
518	2.987134	114.212.135.195	81.11.245.147	BitTorre	134 Handshake
526	3.033553	114.152.118.146	114.212.135.195	BitTorre	331 Bitfield, Len:0x3c
540	3.111802	114.212.135.195	124.27.27.133	BitTorre	134 [TCP Retransmission] Handshake
541	3.111839	114.212.135.195	119.239.62.93	BitTorre	134 [TCP Retransmission] Handshake
544	3.119272	114.69.72.180	114.212.135.195	BitTorre	147 Handshake
546	3.119444	114.212.135.195	114.69.72.180	BitTorre	131 Bitfield, Len:0x3c
553	3.184377	114.212.135.195	58.189.200.70	BitTorre	195 BitTorrent Interested Interested I
565	3.223778	114.212.135.195	219.215.147.25	BitTorre	131 [TCP Retransmission] Bitfield, Len:0x3
567	3.233861	153.160.236.166	114.212.135.195	BitTorre	325 Bitfield, Len:0x3c
583	3.371764	114.212.135.195	114.178.83.20	BitTorre	134 [TCP Retransmission] Handshake
590	3.422303	114.212.135.195	153.160.236.166	BitTorre	195 BitTorrent Interested Interested I
594	3.467775	114.212.135.195	119.242.150.59	BitTorre	134 [TCP Retransmission] Handshake
597	3.497851	114.69.72.180	114.212.135.195	BitTorre	334 Bitfield, Len:0x3c

当我们的 peer 发送 interest 报文后，对方发送 unchoke 报文，如下图所示。



644	3.847792	114.212.135.195	219.215.147.25	BitTorre	131 [TCP Retransmission] Bitfield, Len:0x3c
645	3.847824	114.212.135.195	124.244.201.189	BitTorre	134 Handshake
662	4.024398	58.189.200.70	114.212.135.195	BitTorre	71 Unchoke
667	4.035586	153.160.236.166	114.212.135.195	BitTorre	71 Unchoke
671	4.077382	186.17.156.85	114.212.135.195	BitTorre	305 Bitfield, Len:0x3c
698	4.219744	114.212.135.195	124.27.27.133	BitTorre	134 [TCP Retransmission] Handshake
699	4.219755	114.212.135.195	119.239.62.93	BitTorre	134 [TCP Retransmission] Handshake
740	4.386219	114.212.135.195	58.189.200.70	BitTorre	83 Request, Piece (Idx:0x0,Begin:0x0,Len:0x400
741	4.386241	114.212.135.195	153.160.236.166	BitTorre	83 Request, Piece (Idx:0x5,Begin:0x0,Len:0x400
769	4.719788	114.212.135.195	124.244.201.189	BitTorre	134 [TCP Retransmission] Handshake
782	4.863742	114.212.135.195	114.178.83.20	BitTorre	134 [TCP Retransmission] Handshake
814	5.045846	114.69.72.180	114.212.135.195	BitTorre	71 Unchoke
819	5.075762	114.212.135.195	58.189.200.70	BitTorre	83 [TCP Retransmission] Request, Piece (Idx:0x
822	5.094472	114.178.83.20	114.212.135.195	BitTorre	171 Handshake
824	5.094640	114.212.135.195	114.178.83.20	BitTorre	131 Bitfield, Len:0x3c
825	5.099762	114.212.135.195	219.215.147.25	BitTorre	131 [TCP Retransmission] Bitfield, Len:0x3c
827	5.103788	114.212.135.195	119.242.150.59	BitTorre	134 [TCP Retransmission] Handshake
832	5.127814	114.212.135.195	114.152.118.146	BitTorre	131 [TCP Retransmission] Bitfield, Len:0x3c
837	5.183793	114.212.135.195	121.93.68.13	BitTorre	134 [TCP Retransmission] Handshake
839	5.211581	114.212.135.195	219.215.147.25	BitTorre	195 BitTorrent Interested Interested Intere

我们发送 request 报文，并且对方成功返回 piece 报文，如下图所示。



	Time	Source	Destination	Protocol	Length	Info
2509	21.375694	220.45.80.5	114.212.135.195	BitTorre	1346	Piece, Idx:0x13d,Begin:0x0,Len:0x4000
2511	21.375844	114.212.135.195	220.45.80.5	BitTorre	83	Request, Piece (Idx:0x13d,Begin:0x4000,Len:0x4000)
2525	21.473682	114.69.72.180	114.212.135.195	BitTorre	1468	[TCP Retransmission] Piece, Idx:0x138,Begin:0x0,Len:
2527	21.473765	114.212.135.195	114.69.72.180	BitTorre	95	Request, Piece (Idx:0x138,Begin:0x4000,Len:0x4000)
2563	21.889879	186.17.156.85	114.212.135.195	BitTorre	1514	Piece, Idx:0x13d,Begin:0x0,Len:0x4000
2565	21.889934	114.212.135.195	186.17.156.85	BitTorre	83	Request, Piece (Idx:0x13d,Begin:0x4000,Len:0x4000)
2590	22.171740	114.212.135.195	220.45.80.5	BitTorre	83	[TCP Retransmission] Request, Piece (Idx:0x13d,Begin
2592	22.208200	153.160.236.166	114.212.135.195	BitTorre	1468	[TCP Previous segment lost] Continuation data
2614	22.483743	114.212.135.195	114.69.72.180	BitTorre	95	[TCP Retransmission] Request, Piece (Idx:0x138,Begin
2694	23.520350	58.189.200.70	114.212.135.195	BitTorre	590	[TCP Retransmission] Piece, Idx:0x0,Begin:0x0,Len:0x
2696	23.520440	114.212.135.195	58.189.200.70	BitTorre	83	Request, Piece (Idx:0x0,Begin:0x4000,Len:0x4000)
2729	23.960433	153.160.236.166	114.212.135.195	BitTorre	1468	[TCP Retransmission] Piece, Idx:0x5,Begin:0x0,Len:0x
2731	23.960504	114.212.135.195	153.160.236.166	BitTorre	95	Request, Piece (Idx:0x5,Begin:0x4000,Len:0x4000)
2874	25.561254	114.69.72.180	114.212.135.195	BitTorre	1468	[TCP Retransmission] Piece, Idx:0x139,Begin:0x0,Len:
2878	25.562965	114.212.135.195	114.69.72.180	BitTorre	103	Request, Piece (Idx:0x139,Begin:0x4000,Len:0x4000)
2988	26.679094	114.212.135.195	81.11.245.147	BitTorre	134	Request, Piece (Idx:0x139,Begin:0x0,Len:0x4000) Req
3045	26.983808	114.212.135.195	81.11.245.147	BitTorre	83	[TCP Retransmission] Request, Piece (Idx:0x138,Begin

当用户按下 Ctrl+c 后，成功给 tracker 发送 stopped 报文，断开和 tracker 的连接，如下图所示。

```
^C##STOP:
GET /announce?info_hash=%D4%2B%D2%7F%B0%DE%BD%A3jLg%1BN%5D%BB%EB%E4%DE%04&peer_id=
5D%EF%D3%04%E7%B8%97%71%35%15%B0%67%19%E5%D8%76%5A%4E%A1%D8&port=6543&ip=114.212.13
.195&uploaded=0&downloaded=229376&left=62178613&event=stopped HTTP/1.1

succeed to send stopped data to tracker
```

6.3 最少优先

最少优先，其中 piece 表明是哪一个分片，number 表明该分片在已连接的组内共有多少个该分片。如下图所示。

```
当前各分片数（从小到大）piece(24): number(1)
piece(23): number(1) piece(26): number(1) piece(25): number(1) piece(82): number(1)
piece(20): number(1) piece(36): number(1) piece(22): number(1) piece(21): number(1)
piece(35): number(1) piece(32): number(1) piece(31): number(1) piece(34): number(1)
piece(33): number(1) piece(27): number(1) piece(28): number(1) piece(30): number(1)
piece(29): number(1) piece(50): number(1) piece(44): number(1) piece(45): number(1)
piece(42): number(1) piece(43): number(1) piece(49): number(1) piece(48): number(1)
piece(46): number(1) piece(47): number(1) piece(83): number(1) piece(86): number(1)
piece(37): number(1) piece(84): number(1) piece(85): number(1) piece(41): number(1)
piece(40): number(1) piece(38): number(1) piece(39): number(1) piece(66): number(2)
piece(71): number(2) piece(73): number(2) piece(72): number(2) piece(67): number(2)
piece(68): number(2) piece(70): number(2) piece(69): number(2) piece(80): number(2)
piece(77): number(2) piece(79): number(2) piece(78): number(2) piece(74): number(2)
piece(75): number(2) piece(76): number(2) piece(100): number(2) piece(95): number(2
piece(93): number(2) piece(94): number(2) piece(99): number(2) piece(98): number(2)
piece(96): number(2) piece(97): number(2) piece(81): number(2) piece(89): number(2)
piece(87): number(2) piece(88): number(2) piece(92): number(2) piece(91): number(2)
piece(90): number(2) piece(0): number(2) piece(12): number(2) piece(13): number(2)
piece(10): number(2) piece(11): number(2) piece(17): number(2) piece(16): number(2)
piece(14): number(2) piece(15): number(2) piece(1): number(2) piece(4): number(2)
piece(5): number(2) piece(2): number(2) piece(3): number(2) piece(9): number(2)
piece(8): number(2) piece(6): number(2) piece(7): number(2) piece(65): number(2)
piece(59): number(2) piece(60): number(2) piece(57): number(2) piece(58): number(2)
piece(64): number(2) piece(63): number(2) piece(61): number(2) piece(62): number(2)
piece(18): number(2) piece(52): number(2) piece(19): number(2) piece(51): number(2)
piece(56): number(2) piece(55): number(2) piece(53): number(2) piece(54): number(2)
```

同时在上传、下载，如下图所示。

```
分片:[65,101]（下载,总共）下载：7.28 MB，上传：2.50 MB 速度：1280.00 K/s, 1280.00 K/s,
```

tools 中所给的 bittorrent 客户端，由截图可看到，上传的总量为 35MB，如下图所示。

```
<1> Wildlife.wmv [26246026] 101/101 (100%)
Total: 25 MB
Seed for others 72 hours
| 0/0/2 [101/101/101] 0MB,35MB | 0,0K/s | 0,4K E:0,2 Connecting
```

最少优先算法中间排序结果如下图所示。最少优先算法在排序时采用了快速排序算法。因为总共有 3 个 peer，除开本 peer 以外其余两个分别是一个做种者和一个已经提前开始了的下载者，因此在这里看到各个 piece 的值可能存在不同，有的分片只有 1 个（排在前排），而有的有 2 个（因为另一个下载者已经下载好了这些分片）。

```
当前各分片数（从小到大）piece(46): number(1)
piece(45): number(1) piece(37): number(1) piece(47): number(1) piece(39): number(1)
piece(42): number(1) piece(40): number(1) piece(44): number(1) piece(43): number(1)
piece(31): number(1) piece(32): number(1) piece(28): number(1) piece(27): number(1)
piece(30): number(1) piece(29): number(1) piece(33): number(1) piece(34): number(1)
piece(72): number(2) piece(73): number(2) piece(71): number(2) piece(59): number(2)
piece(63): number(2) piece(64): number(2) piece(60): number(2) piece(61): number(2)
piece(62): number(2) piece(70): number(2) piece(68): number(2) piece(69): number(2)
piece(65): number(2) piece(66): number(2) piece(67): number(2) piece(100): number(2
piece(90): number(2) piece(92): number(2) piece(91): number(2) piece(87): number(2)
piece(88): number(2) piece(89): number(2) piece(99): number(2) piece(96): number(2)
piece(98): number(2) piece(97): number(2) piece(93): number(2) piece(94): number(2)
piece(95): number(2) piece(74): number(2) piece(78): number(2) piece(80): number(2)
piece(79): number(2) piece(75): number(2) piece(76): number(2) piece(77): number(2)
piece(86): number(2) piece(84): number(2) piece(85): number(2) piece(81): number(2)
piece(82): number(2) piece(83): number(2) piece(0): number(2) piece(14): number(2)
piece(15): number(2) piece(11): number(2) piece(12): number(2) piece(13): number(2)
piece(20): number(2) piece(18): number(2) piece(19): number(2) piece(16): number(2)
piece(17): number(2) piece(1): number(2) piece(4): number(2) piece(5): number(2)
piece(2): number(2) piece(3): number(2) piece(10): number(2) piece(8): number(2)
piece(9): number(2) piece(6): number(2) piece(7): number(2) piece(58): number(2)
piece(51): number(2) piece(52): number(2) piece(48): number(2) piece(49): number(2)
piece(50): number(2) piece(57): number(2) piece(55): number(2) piece(56): number(2)
piece(53): number(2) piece(54): number(2) piece(21): number(2) piece(24): number(2)
piece(25): number(2) piece(22): number(2) piece(23): number(2) piece(41): number(2)
piece(36): number(2) piece(38): number(2) piece(26): number(2) piece(35): number(2)
```

该 peer 在下载的同时也在进行上传操作，为其他 peer 提供本 peer 已有分片的下载。如下图所示。

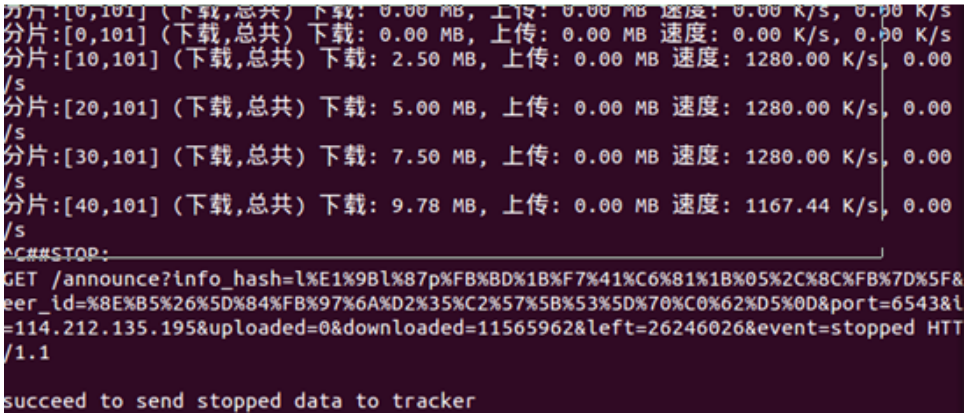
```
分片:[65,101]（下载,总共）下载: 7.28 MB, 上传: 2.50 MB 速度: 1280.00 K/s, 1280.00 K/s,
```

6.4 断点续传

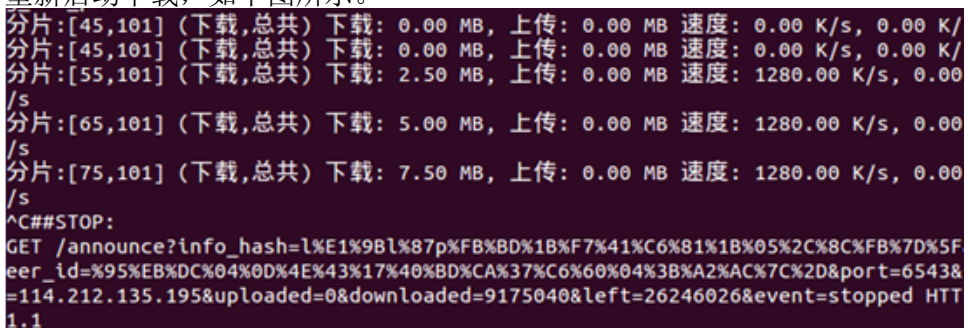
断点续传测试的是我们课程网站上提供的 Wildlife.wmv，初始执行命令如下图所示。

```
chengsu@ubuntu:~/git/final/lab12/bin$ ls
1.txt~              test.torrent.bf
bittorrent          test.txt
core                tracker
last.torrent        tracker.debug
recv_test.txt       tty.log
recv_Wildlife.wmv   Wildlife_64.torrent
simpletorrent       Wildlife.torrent
test3.torrent       (一般コミック) [江口夏実] 鬼灯の冷徹 第14巻.rar.bt
test.torrent
chengsu@ubuntu:~/git/final/lab12/bin$ ./simpletorrent wild.torrent 114.212.135.
95 6543 0 0
```

下载了 45 个分片后停止，下载情况如下图所示。



重新启动下载，如下图所示。



最终下载完毕，如下图所示。

6.5 最后阶段

测试采取的方式是在老师所提供的 186、187、188 上运行 bittorrent 做种，将 114.212.191.43 作为 tracker，本地运行 simpletorrent 下载。

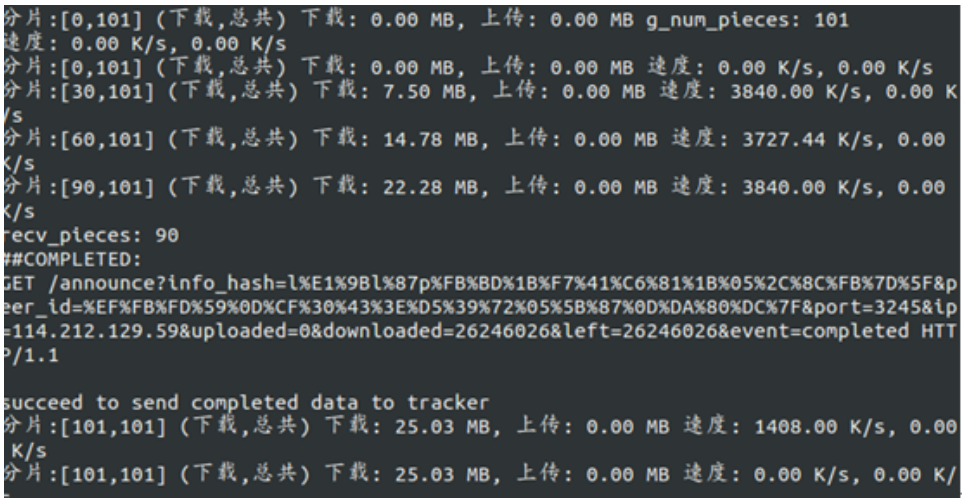
（1）成功下载，并成功发送 cancel 报文，如下图所示。

```
分片:[0,101] (下载,总共) 下载: 0.00 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/s
分片:[0,101] (下载,总共) 下载: 0.00 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/s
分片:[30,101] (下载,总共) 下载: 7.50 MB, 上传: 0.00 MB 速度: 3840.00 K/s, 0.00 K
/s
分片:[50,101] (下载,总共) 下载: 12.50 MB, 上传: 0.00 MB 速度: 2560.00 K/s, 0.00
K/s
分片:[70,101] (下载,总共) 下载: 17.28 MB, 上传: 0.00 MB 速度: 2447.44 K/s, 0.00
K/s
recv_pieces: 80
分片:[90,101] (下载,总共) 下载: 22.28 MB, 上传: 0.00 MB 速度: 2560.00 K/s, 0.00
K/s
recv_pieces: 90
recv_pieces: 100
在最后阶段，要接收的piece为：34
向sockfd为“4”的peer索要最后分片
向sockfd为“7”的peer索要最后分片
从sockfd为“7”的peer索要到了最后一个分片
向sockfd “4” 发送cancel报文
##COMPLETED:
GET /announce?info_hash=l%E1%9Bl%87p%FB%BD%1B%F7%41%C6%81%1B%05%2C%8C%FB%7D%5F&p
eer_id=%4A%FD%E8%67%D8%CF%32%6F%74%2F%6D%35%3D%A0%0B%53%10%82%02%6F&port=3244&ip
=114.212.129.59&uploaded=0&downloaded=26246026&left=26246026&event=completed HTT
P/1.1

succeed to send completed data to tracker
分片:[101,101] (下载,总共) 下载: 25.03 MB, 上传: 0.00 MB 速度: 1408.00 K/s, 0.00
 K/s
分片:[101,101] (下载,总共) 下载: 25.03 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/
```

cancel 报文具体内容如下图所示。

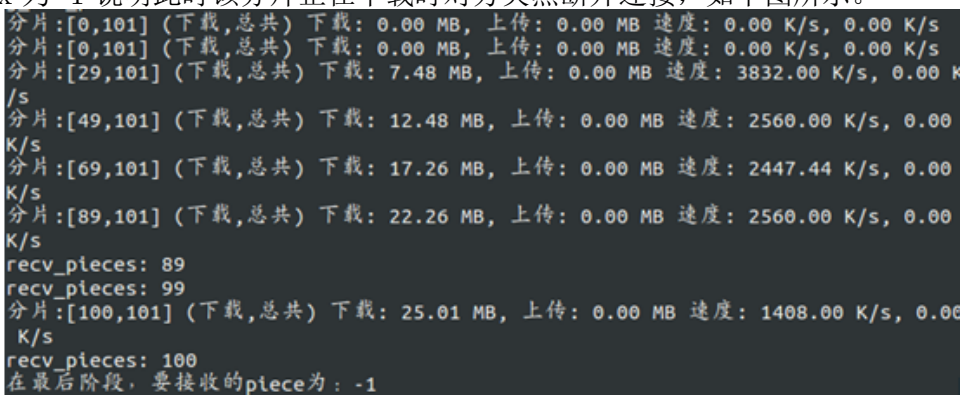
```
50945 37.421192  114.212.129.59   114.212.190.188   BitTorrent   83 Cancel, Piece (Idx:0x22,Begin:0x0,Len:0
▸ Frame 50945: 83 bytes on wire (664 bits), 83 bytes captured (664 bits)
▸ Ethernet II, Src: Vmware_4c:b5:c6 (00:0c:29:4c:b5:c6), Dst: JuniperN_de:d2:9b (00:23:9c:de:d2:9b)
▸ Internet Protocol Version 4, Src: 114.212.129.59 (114.212.129.59), Dst: 114.212.190.188 (114.212.190.188)
▸ Transmission Control Protocol, Src Port: 42279 (42279), Dst Port: ncdmirroring (2706), Seq: 13015, Ack: 11591633, Len
▾ BitTorrent
  ▾ Message: Len:13, Cancel, Piece (Idx:0x22,Begin:0x0,Len:0x4000)
      Message Length: 13
      Message Type: Cancel (8)
      Piece index: 0x00000022
      Begin offset of piece: 0x00000000
      Piece Length: 0x00004000
```

（2）成功下载，但没有发送 cancel 报文（因为这时候分片直接从多个 peer 那里同时拿到，没办法判断最后阶段），如下图所示。



（3）在下载过程中某个已连接的 peer 突然断开连接，这时要重新下载：

到最后阶段时要判断最后一个 piece 的下标 index，如果在最后阶段 index 为 -1 说明此时该分片正在下载时对方突然断开连接，如下图所示。

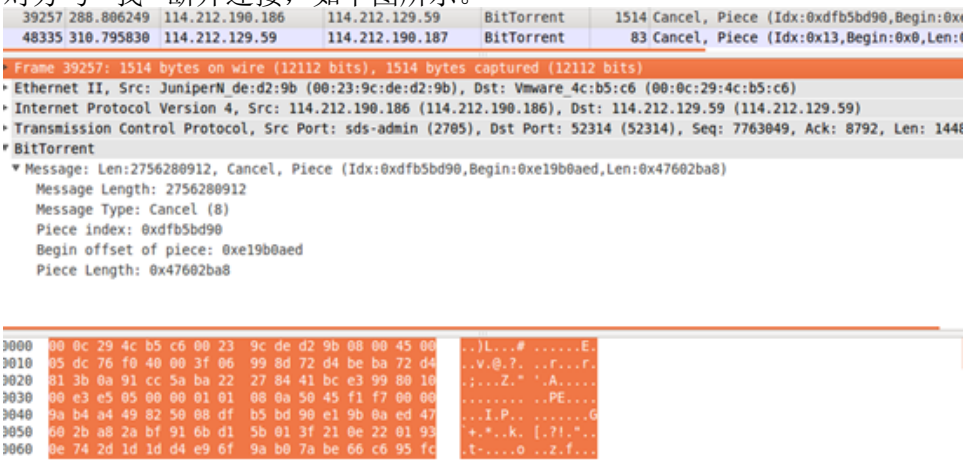


等待一段时间后重新下载该分片并进入最后阶段，如下图所示。

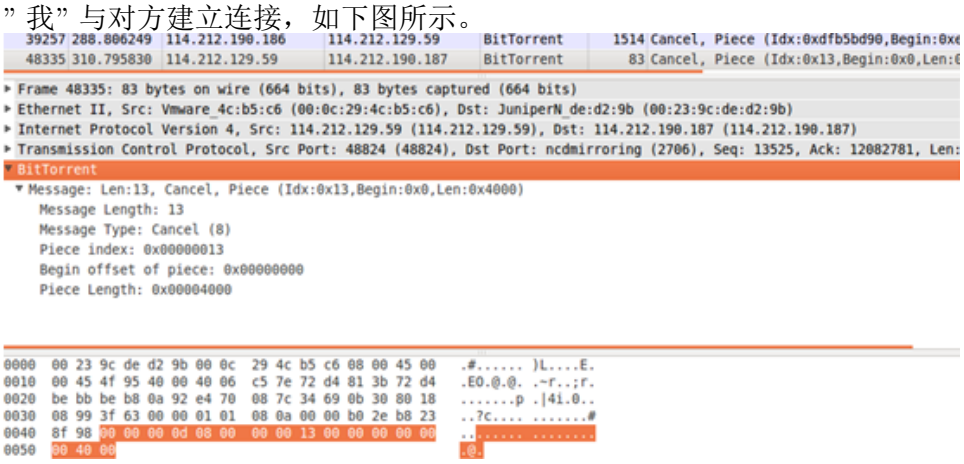
```
recv_pieces: 100
在最后阶段，要接收的piece为： -1
!!!!!!!!!!!!!!!hahahhahaa
分片:[100,101] (下载,总共) 下载: 25.01 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/
s
recv_pieces: 100
在最后阶段，要接收的piece为： 19
向sockfd为“4”的peer索要最后分片
向sockfd为“5”的peer索要最后分片
从sockfd为“5”的peer索要到了最后一个分片
向sockfd “4” 发送cancel报文
##COMPLETED:
GET /announce?info_hash=l%E1%9Bl%87p%FB%BD%1B%F7%41%C6%81%1B%05%2C%8C%FB%7D%5F&p
eer_id=%D7%B9%7E%18%F0%0E%6B%0E%23%1C%D2%4D%C8%AC%9A%00%CC%DD%6E%27&port=3242&ip
=114.212.129.59&uploaded=0&downloaded=26491786&left=26246026&event=completed HTT
P/1.1

succeed to send completed data to tracker
分片:[101,101] (下载,总共) 下载: 25.26 MB, 上传: 0.00 MB 速度: 128.00 K/s, 0.00
K/s
```

这边可以看到对方与”我”断开连接时发送的 cancel 报文，也有我与对方断开连接时发送的 cancel 报文：

对方与”我”断开连接，如下图所示。



”我”与对方建立连接，如下图所示。



（4）这是上一情况的特例，可能在下载过程中突然所有和 peer 的连接都断开了，所以就没办法继续下载，这里的解决方法和上一情况相同。

下载了 98 个分片之后就和所以 peer 都断开连接了，如下图所示。

```
分片:[0,101] (下载,总共) 下载: 0.00 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/s
分片:[0,101] (下载,总共) 下载: 0.00 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/s
分片:[27,101] (下载,总共) 下载: 7.45 MB, 上传: 0.00 MB 速度: 3816.00 K/s, 0.00 K
/s
分片:[47,101] (下载,总共) 下载: 12.45 MB, 上传: 0.00 MB 速度: 2560.00 K/s, 0.00
K/s
分片:[67,101] (下载,总共) 下载: 17.23 MB, 上传: 0.00 MB 速度: 2447.44 K/s, 0.00
K/s
分片:[87,101] (下载,总共) 下载: 22.23 MB, 上传: 0.00 MB 速度: 2560.00 K/s, 0.00
K/s
recv_pieces: 87
recv_pieces: 97
分片:[98,101] (下载,总共) 下载: 24.98 MB, 上传: 0.00 MB 速度: 1408.00 K/s, 0.00
K/s
recv_pieces: 98
recv_pieces: 98
分片:[98,101] (下载,总共) 下载: 24.98 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/s
recv_pieces: 98
```

等待一段时间后重新下载（但因为这时候分片直接从多个 peer 那里同时拿到，没办法判断最后阶段），如下图所示。

```
recv_pieces: 98
分片:[98,101] (下载,总共) 下载: 24.98 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/
recv_pieces: 98
recv_pieces: 98
!!!!!!!!!!!!!!!hahahhahaa
!!!!!!!!!!!!!!!hahahhahaa
!!!!!!!!!!!!!!!hahahhahaa
分片:[98,101] (下载,总共) 下载: 24.98 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K/
recv_pieces: 98
##COMPLETED:
GET /announce?info_hash=l%E1%9Bl%87p%FB%BD%1B%F7%41%C6%81%1B%05%2C%8C%FB%7D%5F&
eer_id=%E4%ED%02%00%52%42%AF%67%F3%86%F9%24%0B%83%17%0D%95%B1%A3%73&port=3243&i
=114.212.129.59&uploaded=0&downloaded=26983306&left=26246026&event=completed HT
P/1.1

succeed to send completed data to tracker
分片:[101,101] (下载,总共) 下载: 25.73 MB, 上传: 0.00 MB 速度: 384.00 K/s, 0.00
K/s
分片:[101,101] (下载,总共) 下载: 25.73 MB, 上传: 0.00 MB 速度: 0.00 K/s, 0.00 K
s
```

对方将我断开连接的报文，如下图所示。