UDP File Server

基于 UDP 的可靠快速文件传输

Hackettfan(范深) | Tencent

2013 年 8 月 27 日

§1 引言

人类学家克利福德 · 吉尔兹 (Clifford Geertz) 在其著作《文化的解释》中曾给出了一个朴素而冷静的劝说：“努力在可以应用、可以拓展的地方，应用它、拓展它；在不能应用、不能拓展的地方，就停下来。”先驱们极有创见性地构建了网络的分层架构，使得我们有机会在网络上层拓展下层未完成的功能。

UDP 是一个基于数据报的传输协议，不提供确认、序列号、超时和重传等机制。但 UDP 传输速度快，消耗小，也是我们中心常用的传输协议。本项目尝试在 UDP 上完成可靠文件传输任务，客户端通过 UDP 与服务器交互，支持下载文件、显示服务器文件目录、上传（带秒传验证）的功能。在文件传输过程中，实现了类似 TCP AIMD 的拥塞控制策略，达到了较快的传送速度。

§2 传输模型

§2.1 数据分组结构

我们规定一个数据包 (Packet) 的数据大小为 512 字节，每个数据包还记录了数据包序号 ($DataID$)、数据实际长度 ($DataLength$) 和标志位 $Flag$，如表 1所示。$Flag$ 的作用主要是用于区别接收到的数据包是否是 ACK，如果是 ACK，则标记为 -1，如果是普通数据，则标记为 0。

DataID	DataLength	Flag	Data
int	int	int	char[512]

表 1: 数据分组结构

§2.2 传输流程

进行数据传输之前模拟了 TCP 的三次握手机制，一方面可以检查文件是否存在，另一方面也可以检测网络的通断程度。该过程只是进行了握手而无连接的建立，服务器得到了客户端的地址，没有其他开销。对于上传、下载和显示目录三个操作，其三次握手的过程各有不同：

1. 由客户端发送请求：对于下载和显示目录，则发送接收请求，请求编号写在数据包序号（DataID）中，对于上传任务，则发送需要上传文件的信息（包括文件名、文件大小和文件 md5）。

2. 服务器接到请求后，对于下载请求，则返回文件是否存在，对于上传请求则查询后台数据库中所有文件的 md5，反馈是否启动秒传。

3. 对于许可的下载和显示目录，由客户端发送开始传输的命令；对于上传任务（无秒传），则客户端开始发送第一个文件包，若服务器已存在离线文件，则停止会话。

§2.3 秒传逻辑实现

秒传效果的实现，需要客户端和服务器的协同。在客户端发起三次握手时，应先算出待传文件的 md5，并连同文件名和文件大小发送给服务器。服务器端维护一个简单的数据库文件，存储当前服务器目录下所有文件的文件名、文件大小和 md5，如表 2所示。考虑到服务器工作目录会受到其他操作的影响（删除、添加文件），所以在接收客户端的请求后，服务器首先对数据库作增量更新（添加和删除部分变更了的文件 md5），再核对等待传送的文件信息。对于 md5 相同的待传文件，发出秒传信号，结束客户端的传送。

FileName	FileSize	MD5

表 2: MD5 数据库存储结构

这里需要考虑一个问题，对于文件名不同而内容完全相同的文件，其 md5 也是完全相同的。对于这类情况，服务器的实现是，将源文件复制命名为客户端请求上传的文件名，并回复秒传成功信号。这样就能很好的减少网络的开销。

图 1: 服务器和客户端的模块调用简图

§2.4 批量传输协议

三次握手完成后，开始文件内容的传输。文件传输包括发送和接收两个模块，这两个模块可以独立出来，由客户端和服务端复用。

文件传输中，采取异步 I/O 的方式发送和接收数据包。定义发送窗口大小为 $cwnd$，发送端每次发送 $cwnd$ 个数据分组后进入超时等待，若在等待时限内回收到了所有分组的 ACK，则表示本次发送完成，否则对未收到 ACK 的分组，进行重传，直到本批次所有分组的 ACK 接收完毕，再进行下一轮发送。

由于 UDP 可能出现乱序抵达的问题，在接收端定义一个接收窗口大小为 $rwnd$，每次接收 $rwnd$ 个不同分组。当接收窗口收满时，将窗口内的分组按序写入文件后，清空窗口内容，进行下一轮接收。

§2.5 拥塞控制

遵循从简单入手的原则，最初实现了服务端 (Server)-客户端 (Client) 的可靠传输程序，保证了传输的正确性后，加入中转节点 (Middle)，形成如图 2所示的传输模型。由中转节点实现**带宽变化的调控**和**几率丢包**，在一定程度上模拟实际网络环境。

带宽控制 程序可根据预设置，控制管道带宽每秒的变化。当一秒内中转数据量超过带宽限制后，则丢弃后面所有数据，直到下一秒清零。

几率丢包 几率丢包模拟网络故障，以 0.01% 的概率制造丢包事件。

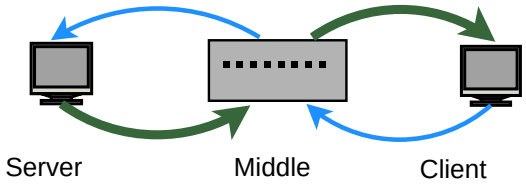


图 2: 传输模型

为了证明拥塞控制机制的有效性，我们尝试了两种不同传输策略：

定量传输 (Fixed Window) 每次固定发送 256 个数据分组。

拥塞避免 (AIMD) 发送窗口初始为 1，每完成一次传输后将窗口大小加 1（加性增），若发生丢包，则将窗口大小减半（乘性减）。最多同时发送 256 个数据分组。

考虑到带宽环境和传输大小对算法的影响，所以分别在两种环境下进行了实验。一种是使用 2.4MB 的文本文件作为传输样本，带宽调控在较小的范围内；另一种是使用 66.4MB 的视频文件作为传输样本，带宽的变化范围和最高值设得很大。

	小带宽平均丢包	小带宽平均耗时	大带宽平均丢包	大带宽平均耗时
Fix Window	5084	21.39s	35971	193.37s
AIMD	317	15.18s	1356	27.46s

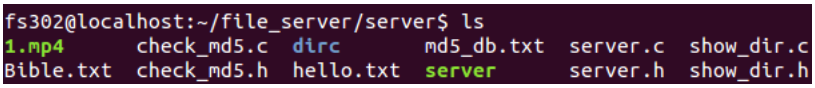
表 3: 丢包数和耗时情况表

表 3显示了两种策略的丢包数和文件传输平均耗时，可以看出 AIMD 的传输效率明显好于 Fixed Window. 于是我们采用了 AIMD 作为拥塞控制策略。

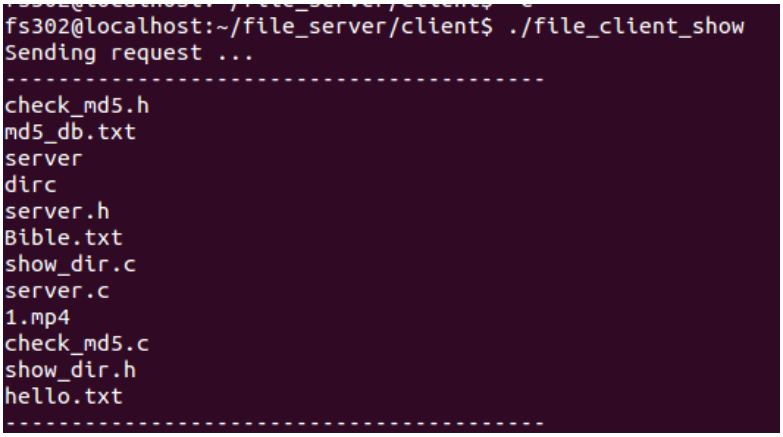
§3 测试用例

§3.1 显示目录

如图 3所示，在服务器端用 ls 列出了工作目录的文件，通过 $./file\_client\_show$，可以读出服务器上的文件列表。



(a) 服务器目录

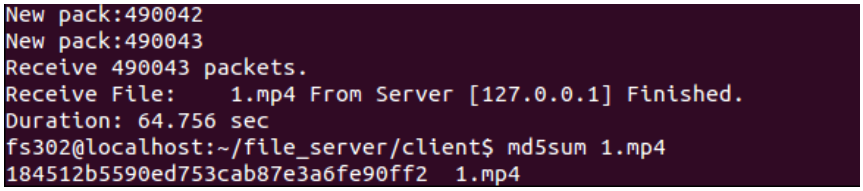


(b) 显示目录命令

图 3: 显示目录

§3.2 下载测试

(a) 服务器传输终端



(b) 客户端接收终端

图 4: 下载

如图 4所示，客户端通过 $./file\_client\_get\ 1.mp4$ 向服务端请求下载名为 $1.mp4$ 的文件，耗时 64.756 秒传输完毕后，比对 md5 值，与服务端文件的一致，说明下载文件顺利完成。

传输文件是一个比较耗时的过程，中间可能出现各种网络异常情况。为了加强系统的友好性和服务器的可持续服务能力，在传输中一方断开连接时，为设置了超时重传次数上限(暂定为 5)，如果重传 5 次依然无法收回 ACK，则自动断开该连接。同样的，接收方等待 5 秒内若没有收到新的数据包，则说明网络故障，也是执行自动断开连接。

§3.3 上传测试

情况	同名	同 md5	应对策略
1	×	×	直接上传
2	×	√	秒传，服务器端复制
3	√	×	上传覆盖
4	√	√	秒传，无须变更

表 4: 上传文件类型

对于客户端请求上传一个新文件来说，服务器已存在的文件与其有 4 种不同的对应情况及其处理策略，如表 4所示。

服务器期望能在不影响客户需求的情况下，最大限度的减小网络传输。下面我们分别对这四种情况进行测试，分别在图 5、6、7、8中演示。

```
Sending 70 packets.
Sending 71 packets.
Sending 72 packets.
Sending 73 packets.
Sending 60 packets.
ReSend 40 times.
Send File:       Bible.txt To Server [127.0.0.1] Finished.
Duration: 1.128 sec
fs302@localhost:~/file_server/client$
```

(a) 客户端发送

```
New pack:4609
New pack:4610
New pack:4611
New pack:4612
New pack:4613
New pack:4614
Receive 4614 packets.
Finished.
Waiting request ...
```

(b) 服务器接收

图 5: [情况 1] 如果服务端不存在待上传的文件，即无同名也无同 md5 的服务器文件，则进行直接上传。如图所示，客户端向服务器上传了一个大小为 $2.4MB$，名为 $Bible.txt$ 的文件。服务器正常接收。

```
fs302@localhost:~/file_server/client$ ls
1.mp4            file_client_get.c   file_client_push.h  hello.txt
BB.txt           file_client_get.h   file_client_show    md5_db.txt
Bible.txt        file_client_push    file_client_show.c
file_client_get  file_client_push.c  file_client_show.h
fs302@localhost:~/file_server/client$ mv 1.mp4 2.mp4
fs302@localhost:~/file_server/client$ ./file_client_push 2.mp4
Sending request ...
File Exist on Server. Using fast copy._
```

(a) 客户端尝试发送

```
fs302@localhost:~/file_server/server$ md5sum 1.mp4
184512b5590ed753cab87e3a6fe90ff2  1.mp4
fs302@localhost:~/file_server/server$ md5sum 2.mp4
184512b5590ed753cab87e3a6fe90ff2  2.mp4
```

(b) 服务器判断后实现秒传

图 6: [情况 2] 文件内容相同，而文件名不同的文件上传，实现重命名秒传。我们将之前从服务器下载的 1.*mp*4 更名为 2.*mp*4，执行上传操作。结果显示，服务器可判断 md5 相同的文件，并实现服务器端的复制，减少了不必要的网络传输消耗。

```
fs302@localhost:~/file_server/client$ md5sum hello.txt
00030833780405f3d9558ae0d17c34d6  hello.txt
fs302@localhost:~/file_server/client$ ./file_client_push hello.txt
Sending request ...
Sending...
Begin transfer.
Sending 1 packets.
Sending 1 packets.
ReSend 0 times.
Send File:       hello.txt To Server [127.0.0.1] Finished.
Duration: 0.000 sec
```

(a) 客户端发送 hello.txt，与服务端的同名文件不同内容

```
fs302@localhost:~/file_server/server$ md5sum hello.txt
f51788276d079b9ce6aa8adb1421e012  hello.txt
fs302@localhost:~/file_server/server$ ./server
Waiting request ...
hello.txt       12      00030833780405f3d9558ae0d17c34d6
Begin Recvfile.
New pack:0
New pack:1
Receive 1 packets.
Finished.
Waiting request ...
^C
fs302@localhost:~/file_server/server$ md5sum hello.txt
00030833780405f3d9558ae0d17c34d6  hello.txt
```

(b) 服务器接收，覆盖原文件

图 7: [情况 3] 虽然服务器存在与待传文件同名的文件，但验证 md5 并不相同后，服务端接收新的文件，覆盖掉原来的文件。

```
fs302@localhost:~/file_server/client$ md5sum Bible.txt
8a7867becf25f0ca62e9580f06746813  Bible.txt
fs302@localhost:~/file_server/client$ ./file_client_push Bible.txt
Sending request ...
File Exist on Server. Using fast copy._
```

(a) 客户端尝试发送

```
Waiting request ...
Bible.txt       2362365 8a7867becf25f0ca62e9580f06746813
Finished.
```

(b) 服务器判断后实现秒传

图 8: [情况 4] 调用 *./file_client_push Bible.txt*。由于 *Bible.txt* 已经在服务器上，服务器检查 md5 后发现文件已存在，执行秒传！

§4 思考和完善

§4.1 增强服务的友好性

服务的每一个细节都是在不断完善的，任何一处不友好的地方都值得我们去改善。虽然此文件服务器实现的功能很简单，但依然有很多细节值得继续改善。由于时间有限而尚未优化的地方，我想写在这边：

1. 对于显示目录的服务，目前尚且实现显示单层目录的功能。对于大型存储系统来说，多层目录是必须的要求，如何让用户自由地切换查看目录，应该是一个值得考虑的问题。另外，用户除了关心文件名，还可能对文件类型、大小以及创建修改的时间也有兴趣，因为这些信息能够帮助用户进行是否下载的决策，很有必要完善。

2. 对于下载服务，目前实现的是在终端实时打印接收的数据包数。实际上用户更关心的是目前完成了多少（百分比），大约还需要多长的时间？以及，能否部分查看尚未下载完成的文件？如果能在信息提示和边传边看上做一些工作，用户体验必然增强。

3. 对于上传的服务，秒传固然是个亮点。但是目前的服务没有过多考虑文件覆盖的问题，对于客户端上传的同名文件，如果内容不同，我们的处理是执行覆盖传输。更友好的处理方式是：提示用户，让用户作选择（覆盖 · 重命名 · 取消上传）。

§4.2 网络服务扩展

当前实现的文件传输系统，是点对点单线程的传输模型。因为 UDP 是无连接的传输协议，服务端绑定端口后就不再区分客户。而 UDP 的优点是其传输速度之快，所以我们可以把它单独用在传输过程中，而用 TCP 进行连接的建立和实现传输前期的通信。这样，TCP 用于连接的确认和创建新的服务套接字，再用 UDP 进行文件传输，就能实现并发且快速的服务了。