沈征宇 18302010009

设计思路

文件传输

可靠传输

peer不断监听输入，并配置信息，初始化config，采取了接收方GBN（回退N步协议）来实现累计确认，发送方将"连续收到3次冗余ACK"和"timeout"定义为丢包发生（计时采用SIGALRM信号实现），然后重新发送丢失分组。采用up_conn_s和down_conn_s来管理外部连接。

上传：当peer收到其它peer发出的WHOHAS分组时，如果检测到自己持有请求的chunk，则通过handle_whohas来回应对方IHAVE分组，当接收到GET分组时，通过HANDLE_GET来开启上传连接，维护到上传连接池中，发送DATA分组并启动定时器，用handle_ACK接收下载方传来的ACK分组，当接收到正常ACK时，更新窗口，继续发送，并重新计时；当接收到冗余ACK时，冗余数目++，若大于等于3，则重发丢失分组。若发生超时，通过handle_TIMEOUT来重发分组。当上传完成，移除该连接。

下载：当peer在命令行收到GET下载请求时，先向其它peer一次或多次发出WHOHAS分组来请求下载源，当持有该目标分组的peers传回IHAVE分组，peer调用handle_ihave方法建立连接开始下载，并将连接维护在down_pool中，再更新目标chunk的下载状态，发送GET分组。当接收到DATA分组，用handle_data发出ACK分组来回应发送方，下载数据并更新任务状态。对于一个chunk的下载结束后，从下载连接池中移除该连接，并用SHA-Hash进行数据比对，判断是否是所需的chunk，如果正确，则将数据写入chunk在GET任务中的缓存，如果不正确，则从其它对等方处下载此chunk。

并发传输

为了提高P2P中的传输效率，需要支持并发传输，因此定义up_pool_s和down_pool_s来管理peer的所有上传和下载连接，在GET任务中，为了利用P2P中更多的对等方，在chunks下载任务中，维护providers管理chunks的来源对等方，并且维护块的下载状态，0代表未开始下载，1代表下载完成，2代表下载中，每个连接互不干扰，最后实现从多个对等方下载多个块，加快传输。例如A下载块1、块2，B、C有1、2，B发送IHAVE 1,2，C发送I HAVE 1,2，A可以选择从B下载1，从C下载2，加快效率。每条连接对应一个peer，记录交互信息如上传了多少文件。上传或者下载在pool中寻找连接，每次收到包通过对应的连接操作

拥塞控制

采用TCP Tahoe策略

当peer为发送方时

1. 慢启动：初始cwnd为1，ssthresh=64，每次收到ACK，窗口大小增加1，当达到阈值或者丢包，进入拥塞避免状态
2. 拥塞避免：此时不再快速增大cwnd，而是每次收到ack后，累加收到ack与期望ack的差值，当累加值大于等于cwnd，说明经过一个rtt（至少），窗口大小++
3. 快速重传：ssthresh设置为max(cwnd/2,2)，并且将窗口大小改为1，按窗口大小重发未确认的包，恢复慢启动。当收到3次重复ACK或者超时后，进行快速重传，重启计时器。当cwnd达到ssthresh，直到上一次cwnd内的所有分组都被确认，才执行cwnd++，否则cwnd不变。
4. 重复ACK：当收到ack=期望ack-1时，说明重复，可能有丢包存在，进行快速重传，cwnd重新设置为1，重新进行慢启动的过程。
5. 超时：线程没有因收到成功ACK或快速重传在1秒内被取消，则表示超时，进行快速重传

当peer为发送方，每一次发送分组或接收到ACK，将cwnd或ACK记录到problem2-peer.txt

程序逻辑

1. 启动后，进行初始化，并等待来自用户的请求或其他对等方的消息
2. 用户输入，调用函数处理输入，根据用户生成WHOHAS包，发送给对等方，等待消息
3. 或者收到对等方消息，进行处理
    - WHOHAS：获取发送方和包的Hash，与已拥有的进行比对，有就封装IHAVE，发送回
    - IHAVE：其他用户的可能回复，创造对等方连接和GET包，加入下载池
    - GET：获取hash值，查找本地chunk，根据cwnd分步发送给对等方
    - DATA：判断发送方是否正确，判断序号是否正确，正确回复ACK，一步一步下载完整个chunk
    - ACK：ack到512：下载完毕，acknum大于期望：改变窗口大小，发送剩余包，acknum等于期望-1：发生重复，大于等于3视作丢包，进入快速重传
4. 阻塞等待新的消息

主要实现

handler

处理接收到的数据包，发送数据包，进行拥塞控制

pkt_ntoh:分组本主机化

new_pkt:传入参数并创建新分组

parse_type:解析对应分组类型

new_whohas_pkt:根据自己的GET任务，生成WHOHAS，注意需要用链表来容纳可能的切片

handle_whohas:处理外部发来的WHOHAS分组，即遍历自己拥有的chunks，得到符合的hash of chunk，打包成IHAVE分组返回（如果有）

handle_get:当收到其他主机的get请求且自己的上传连接池未满时，从chunks的管理部分，根据外部GET请求，获取其需要的chunk的对应数据（512Kbytes），将它拆分成512个DATA分组，根据要发送的分组和接收peer，开启上传连接，启动发送和定时器

handle_ihave:当自己发出去的WHOHAS被响应，且下载连接池未满，根据接收到的IHAVE分组，更新chunks' states and providers(通过update_chunk_provider)，再通过返回的要下载的chunk hash创建下载，生成GET分组请求，返回

handle_data:当自己的get请求被响应，接收到发送方peer发送的数据，且seq是下载连接需要的（seq == down_conn->next_ack），则将数据缓存到chunk_buf中，返回ACK = seq的ACK分组进行响应，否则回复冗余ACK，值是next_ack-1

如果该下载连接的chunk的全部512Kbytes数据都已收到，则检查正确性，更新chunk在GET任务中的status，将获取的数据存入GET的缓存中，移除下载连接，等一个GET任务中的所有chunks都正确下载，则可以按序写入输出文件，否则为status==0，provider存在的chunk继续执行下载任务（建立连接，发送GET）

handle_ack:因为分组为512个，所以当ACK==512,可以认定已收到所有数据并停止计时、释放上传连接。当ACK大于上一次的last_ack，判断处于慢启动还是拥塞避免，更新发送窗口的状态，继续发送DAT分组并重新开始计时。当收到冗余ACK，为ack的dup_times++，当>=3，证明丢包，to_send flag回退到last_ack，更新窗口状态、sshtresh，重新发送DATA分组，重启定时器

get_task_handler

initiate_task:初始化get任务

add_check_data：添加数据，并判断正确性（通过hash）

update_provider：下载指定peer的未下载的块（续传，直到完成整个传输过程）

开发瓶颈

因为此PJ具有一定难度，且我对于C语言不算熟练，因此多次在开发中陷入瓶颈状态，以下记录

- 超时，每次重启定时器撤销上一个定时器线程，当线程1秒内被取消说明没有超时，如果1秒后线程苏醒则代表超时，执行超时操作
- 拥塞状态条件的创造：原本的实现方式是检测上一次改变窗口的时间是否与这次收到ACK的时间相差达到1秒，由于1秒太长，容易收到重复ACK，后来修改，每次发送当前窗口大小的分组数目时，只要它们都被确认，就说明经过一个rtt，增加一个窗口大小

翻译文档

之后附上在理解任务过程中自行的翻译文档，一般情况下不用阅读，谢谢助教

1. 详情

一个可靠的类似BitTorrent的文件传输系统

在UDP上运行，需要实现一个类似TCP的可靠网关

需要能够同时从不同服务器下载文件不同部分“chunks"

推荐32位机器上运行

实现一个类似BitTorrent的路由来寻找同伴和上传/下载文件部分

3.1 背景

这个工程宽松地基于BitTorrent Peer-to-Peer (P2P) file transfer protocol

在传统的文件传输应用中，客户机知道哪个服务器拥有文件，并发送请求给特定的服务器请求这个文件。在P2P文件传输应用中，文件的真实位置是未知的，它可能存在在多个位置。客户首先发送查询来得知它的哪个对等方拥有它想要的文件，接着从一个或多个对等方检索文件。

虽然P2P服务已经司空见惯，BitTorrent引入了一些新的服务概念使它真正流行。首先 BitTorrent将文件分割为不同的chunks，每个chunk可以被独立下载，整个chunks的集合能被组装成文件，本PJ中，chunk大小是固定的**512Kbytes**。

BitTorrent使用中央tracker来追踪每个对等方持有的文件的chunks，一个客户先通过持有.torrent文件来开始下载，这列出了文件**每个chunk的信息（hash value）**。一个chunk是通过cryptographic hash内容来标识的，torrent文件让客户知道需要在网络中向其它peers请求哪些块。在客户下载了chunk后，它必须计算cryptographic hash来决定它是否持有了对的chunk。

BitTorrent持有一个大文件并且将它分成不同分开的小块，它可以从不同的对等方被下载。为了下载特定的chunk，接收方从tracker得到持有这个chunk的peer的列表，并直接择一开始下载。BitTorrent使用“最稀有的块优先”的启发式方法，尝试先获取最稀有的块分块。对等方可以并行下载/上传四个不同的块。

本PJ不同之处

并非实现tracker server，对等方们会蜂拥进网络来寻找哪些对等方拥有哪些块，每个对等方都将知道其它每个对等方的身份，不必实现路由。

简化准备工作，所有文件数据是从主数据文件获取的。对等方被一个文件配置了它们在开始时拥有哪些块，它们也知道。

不用实现BitTorrent’s激励机制

需要基于UDP并且实现可靠性，模仿TCP是可行的

3.2 指导

必须使用C完成，不能使用C++或STL

必须在控制层和传输层使用UDP协议，在unix-based系统上运行

不能使用定制套接字类，使用提供的hashing library，但不提供代码形式的高级功能

3.3 提供的文件

hupsim.pl 这个文件使用topo.map模拟了网络拓扑

sha.[ch] SHA-1哈希生成器

input_buffer.[ch] 处理用户输入

debug.[ch] debug输出的工具

bt_parse.[ch] 解析命令行参数的工具

peer.c 一个骨架对等方文件，为你解决一些准备和处理的工作

nodes.map 提供网络中对等方的列表

topo.map 被hupsim.pl使用的隐藏的网络拓扑，这应该只被hupsim.pl所翻译，不应该被自己的代码读取

make-chunks 根据一个包含chunk-id,hash pairs的输入文件来创建新的文件块的程序，这对创建更大的文件下载方案很有帮助

3.4 术语

主数据文件——这个输入文件包含网络中的所有数据，所有结点都能访问这个文件，但一个对等方应只读自己拥有的块。当一个chunk的id和hash被列在对等方的has-chunk-file中，对等方拥有这个块。

主chunk文件——一个列出了所有主数据文件中的块的块ID和对应的hash的文件

对等方列表文件——一个包含了所有网络中对等方的文件，查看nodes.map来查看样例

has-chunk-file——一个per-node文件包含一个特定node在开始时持有的chunks列表。然而，对等方在从其它对等方下载块以后将能够访问更多的块

get-chunk-file——一个包含一个对等方想要下载的块id和hash列表的文件。这个文件名由用户想要请求新下载时提供

peer identity——当前对等方的标识。对等方应该使用它从对等方列表文件获取其主机名和端口

debug-level——DPRINTF()应该打印的debug语句等级

3.5 文件传输怎样奏效

代码应该产生叫peer的可执行文件，命令行

```
peer -p <peer-list-file> -c <has-chunk-file> -m <maxdownloads>
-i <peer-identity> -f <master-chunk-file> -d <debuglevel>
```

此程序监听命令行的标准输入，唯一指令是GET，用户这个指令应该使程序打开指定chunk文件并下载列在其中的块（假定文件名不占空间）。当程序结束下载，他应通知。不必处理用户的并行请求，测试代码在一个命令结束后才会发送下一个。不同文件的格式在3.7列出

为了找到从哪个主机下载，请求peer发送WHOHAS指令给所有其它的对等方（带参数list），list是想要下载的块的hash列表。整个列表大小可能超过UDP单个包的大小，应假设最大的包大小是1500bytes，这种情况应该分解成多个WHOHAS，hash码大小是固定的20bytes，如果文件太大，客户应反复发送get指令，在收到一个get指令的响应后继续。应该并行发送这些指令。

收到WHOHAS查询后，对等方使用"IHAVE"回复发回它包含的块列表。列表包含它有的块的hash。由于请求被制成适应一个包，回应一定能容在一个包里。

（返回的list是这个对等方所持有的所有块，还是它拥有的在请求列表中的块？）

请求方查看所有ihave回应，决定从哪里拿。接着用GET指令独立下载每一个块，因为用UDP，可以认为get请求混合了应用层get请求和TCP SYN包连接建立函数的功能。

当对等方被请求他一个拥有的chunk，它将会送回多个“data”包给请求的对等方（格式见下）直到请求中指定的chunk被完全转发。对等方可能不一定能满足get请求（如果他已经在服务上限个对等方），这个对等方可以忽略向他的请求或将请求排队列或通知请求方他无法服务。发送通知是可选的，如果发送就用DENIED码。每个对等方只能有从网络中其它每个对等方的一个下载，这意味着UDP包中的IP和端口号唯一决定一个数据包属于哪个下载（因为两方只有一个下载）。每个对等方可以有在多个对等方开展多个下载。

当对等方收到DATA包，它返回ACK包。收包方应该确认知悉所有数据包

3.6 包的格式

UDP是底层协议，包头是确定的

1. Magic Number [2 bytes]
2. Version Number [1 byte]
3. Packet Type [1 byte]
4. Header Length [2 bytes]
5. Total Packet Length [2 bytes]
6. Sequence Number [4 bytes]
7. Acknowledgment Number [4 bytes]

所有多字节整数字段必须按网络字节顺序（幻数、长度和序列号/确认号）进行传输。而且，所有整数都必须是无符号的。

幻数应该是15441，版本号应该是1。对等端应该丢弃没有这些值的包。“Packet Type”字段确定对等方应该期望的负载类型。表1中给出了不同数据包类型的代码

通过更改报头长度，对等端可以为所有数据包提供自定义优化（如果您选择的话）。序列号和确认号用于与TCP类似的可靠传输机制。

Packet Type	Code
WHOHAS	0
IHAVE	1
GET	2
DATA	3
ACK	4
DENIED	5

Table 1: Codes for different packet types.

WHOHAS and IHAVE packets

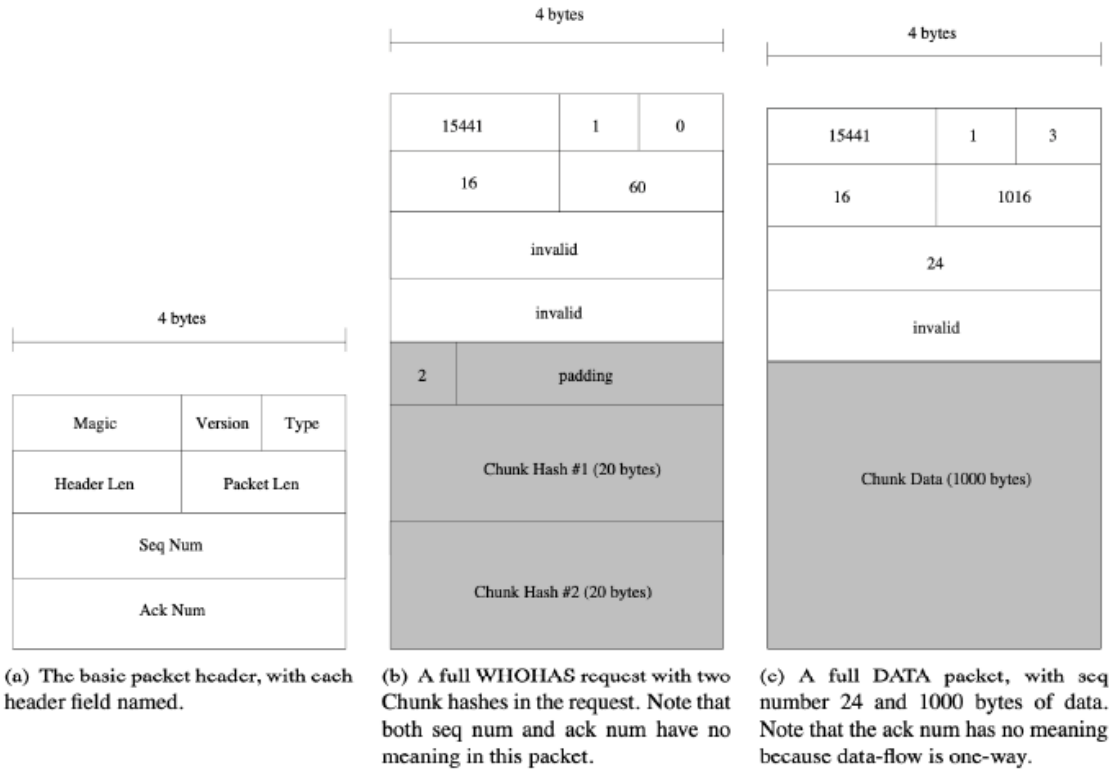
WHOHAS和IHAVE的有效负载包含块散列的数量（1个字节）、3个字节的空白填充空间以保持块32位对齐，以及其中的哈希列表（每个20个字节）。数据包的格式如图2（b）所示。

GET packets

只包括客户想要拿到的块的chunk hash

DATA and ACK packets

图2（c）显示了一个示例DATA包。DATA和ACK包没有已定义有效负载格式；通常它们应该只包含文件数据。



(a) The basic packet header, with each header field named.

(b) A full WHOHAS request with two Chunk hashes in the request. Note that both seq num and ack num have no meaning in this packet.

(c) A full DATA packet, with seq number 24 and 1000 bytes of data. Note that the ack num has no meaning because data-flow is one-way.

Figure 2: Packet headers.

header中的序列号和确认号字段仅在DATA和ACK包中有意义。在这个项目中，对于新的“GET connection”，序列号总是从1开始。接收对等方应发送确认号为1的ACK包，以确认它已接收到序列号为1的数据包，以此类推。

即使在header中同时存在序列号和确认号字段，也不应将data和ACK包组合起来。不要使用DATA包来确认前一个包是否收到，也不要在ACK包中发送数据。这意味着对于任何数据包，ACK num是无效，对于任何ACK分组，SEQ num字段是无效字段。无效字段仍会占用数据包标头中的空间，但它们的值应被接收数据包的对等方忽略。

3.7 文件格式

Chunks File:

File: Chunks: id chunk-hash ....

master-chunks-file有以上格式，第一行指明对等方间需要分享的文件，对等方只应阅读has-chunks-file变量中提供的块，chunks固定大小512KB，不正好的话，合适填充

chunks后的所有行：包含chunk id和对应的hash值，是SHA-1 hash，以十六进制数代表（但不以0x开头），chunk id是十进制数，指示在主文件中的块的偏移量。如果chunk id是i，那么块的内容在主文件中的起始位置是i * 512k

Has Chunk File:

这个文件包含一个对等方拥有的chunk的id和hash，在主块文件中，id是10进制，hash是16进制

同一个块，在has-chunk-file中的id和master-chunks-file中这个块的id是相同的

id chunk-hash id chunk-hash .....

Get Chunk File

文件的格式与has chunk文件完全相同。它包含一个对等方希望下载的ID列表和哈希值。与主块文件一样，十进制格式的id和哈希也是十六进制格式的。对于相同的数据块，get chunk文件中的id可能与主块中该块的id不同-文件。更确切地说，这里的id指的是用户要保存到的文件中块的位置。

Peer List File

此文件包含网络中所有对等方的列表。每行的格式为：

id为十进制数，peer address为点分十进制格式的IP地址。端口号是十进制端口号格式的整数。最简单的方法是在不同的本地主机端口上运行所有主机。

4. 示例

假设有A.gif和B.gif两张图片，在example文件夹下。建议走一遍流程

首先创建两个大小为512K倍数的文件

```
tar cf - A.gif | dd of=/tmp/A.tar bs=512K conv=sync count=2
tar cf - B.gif | dd of=/tmp/B.tar bs=512K conv=sync count=2
```

填充后，A.tar和B.tar都是1MB大小，（2个块的长度）运行两个结点，一个在1111端口，一个在2222端口，A的两个文件Hash是0xDE和0xAD。B的两个文件Hash是0x15和0x441，首先做以下

```
cat /tmp/A.tar /tmp/B.tar > /tmp/C.tar
make-chunks /tmp/C.tar > /tmp/C.chunks
make-chunks /tmp/A.tar > /tmp/A.chunks
make-chunks /tmp/B.tar > /tmp/B.chunks
//将数据文件转成chunks文件，格式是 左块号 右hash
```

这会为C.tar创建master data file，C.chunks的内容是

```
0 00000000000000000000000000000000000000de
1 00000000000000000000000000000000000000ad
2 0000000000000000000000000000000000000015
3 0000000000000000000000000000000000000441
```

A.chunks的内容是

```
0 00000000000000000000000000000000000000de
1 00000000000000000000000000000000000000ad
```

B.chunks的内容是

```
0 0000000000000000000000000000000000000015
1 0000000000000000000000000000000000000441
```

编辑C.chunks文件，增加两行，保存成masterchunks

```
File: /tmp/C.tar Chunks:
0 00000000000000000000000000000000000000de
1 00000000000000000000000000000000000000ad
2 0000000000000000000000000000000000000015
3 0000000000000000000000000000000000000441
//首行指示实文件
```

增加两个对等方，名字叫做/tmp/nodes.map

```
1 127.0.0.1 1111
2 127.0.0.1 2222
```

最后，您需要创建描述每个节点的初始内容的文件。让节点1拥有所有文件A.tar而不包含文件B.tar。让节点2拥有所有的B.tar文件，而不包含A.tar文件。

创建/tmp/A.haschunks，内容是

```
0 00000000000000000000000000000000000000de
1 00000000000000000000000000000000000000ad
```

创建/tmp/B.haschunks，内容是

```
2 0000000000000000000000000000000000000015
3 0000000000000000000000000000000000000441
```

以上id都是从C.masterchunks取的，它又指主数据文件中的偏移量

运行node 1，类型

```
peer -p <peer-list-file> -c <has-chunk-file> -m <maxdownloads>
-i <peer-identity> -f <master-chunk-file> -d <debuglevel>
```

```
peer -p /tmp/nodes.map -c /tmp/A.haschunks -f /tmp/
C.masterchunks -m 4 -i 1
//启动一个结点，读取-p的对等方列表，读取-c它有什么块，-i标识符，在nodes.map中按号读取，-f全部块的文件
```

在不同终端运行node2

```
peer -p /tmp/nodes.map -c /tmp/B.haschunks -f /tmp/
C.masterchunks -m 4 -i 2
```

node1运行起来后

```
GET /tmp/B.chunks /tmp/newB.tar
```

它指明从对等方获取/tmp/B.chunks，并存到/tmp/newB.tar（按/tmp/B.chunks中的value序号）下面说明代码的操作（消息应是二进制的，这里纯文本显示）

```
WHOHAS 2 00…15 00…441
```

向nodes.map中的所有对等方，询问两个块

结点2回复：

```
IHAVE 2 00…15 00…00441
```

结点1向结点2发送：

```
GET 00…15
```

结点2开始发送数据，当结点1收到，发送ACK，在GET完成后，Node1发送

```
GET 00…441
```

总之有三种描述块的格式get-chunks, has-chunks and master-chunks

和peer list格式

5. PJ任务

100%可靠和滑动窗口

DATA包的可靠传输需要100%实现，非数据流量（WHOHAS,IHAVE,GET)不必可靠或用流控制进行传输，对等方应该能够在网络中搜索可用的块，并从拥有它们的对等方下载它们。文件的所有不同部分都应该在请求对等端收集，并且在考虑收到的块之前，应该确保它们的有效性。您可以通过计算SHA-1哈希并将其与指定的块哈希进行比较来检查下载块的有效性。在这项目中，使用8个数据包的固定大小窗口（请注意，TCP使用基于字节的滑动窗口，但您的项目将使用基于数据包的滑动窗口。用包来做比较简单。另外，与TCP不同，你只有一个发送方窗口，这意味着窗口大小不需要在包头中进行通信。）发送方不应该发送超出窗口的数据包。

当发送方收到一个更高的数据包编号的确认时，它会向前滑动窗口。每个数据包都有一个序列号，以下约束对发送方有效（提示：您的对等方可能希望保持与此处显示的状态非常相似的状态）：

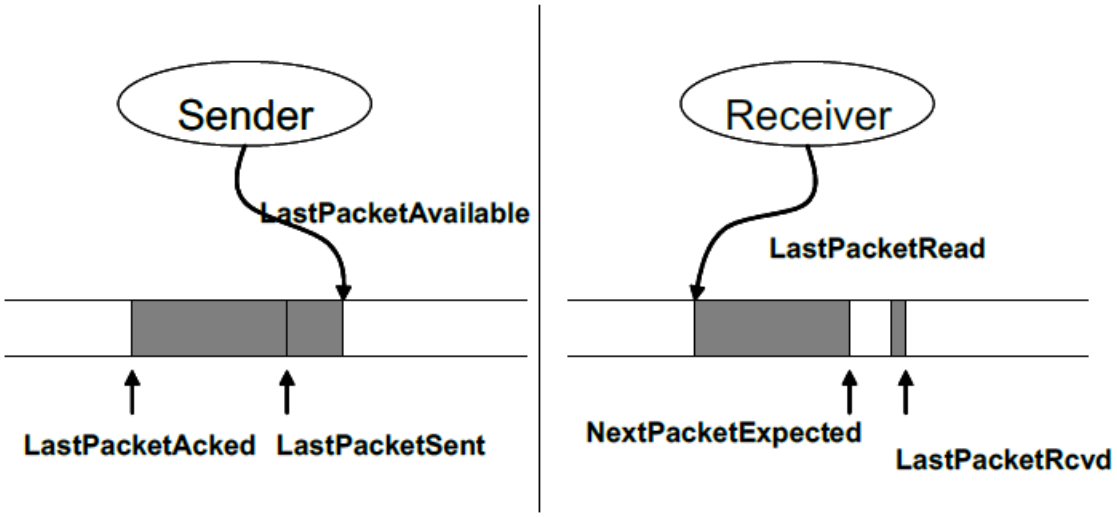


Figure 3: Sliding Window

发送方：

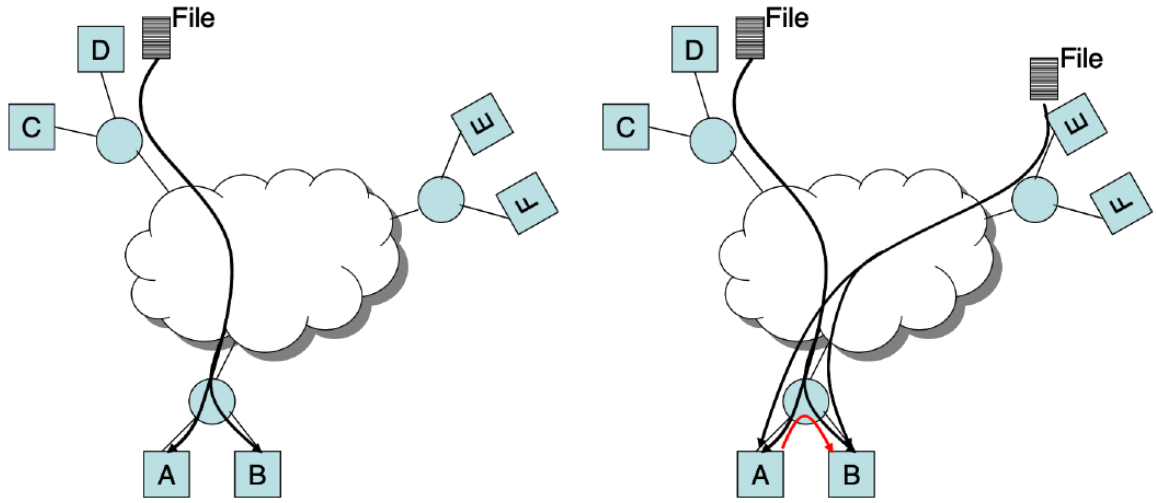
• LastPacketAcked ≤ LastPacketSent • LastPacketSent ≤ LastPacketAvailable • LastPacketAvailable − LastPacketAcked ≤ WindowSize

LastPacketAcked and LastPacketAvailable间的包必须被缓存，可以通过缓存，可以通过有能力用数据文件重新产生

当发送方发送数据包时，它会为它启动一个计时器。然后等待一段固定的时间来获得包的确认。每当接收器收到一个数据包时，它就会发送一个对NextPacketExpected 1的确认。也就是说，在接收到序列号为8的分组时，应答将是“ACK 8，但仅当序列号小于8的所有分组都已被接收时。这些被称为累积确认。发送方有两种方法来知道它发送的数据包是否没有到达接收方：要么发生超时，要么发送方收到“重复的确认”。

如果发送方发送了一个数据包，但在该数据包的计时器过期之前没有收到对它的确认，它将重新发送该数据包。如果发送方发送了一个数据包并收到了重复的确认，它就知道下一个预期的数据包（至少）丢失了。为了避免重新排序造成的混乱，发送方在一行中只有3个重复的ack之后才将它统计为丢失的数据包。

如果请求的客户机从主机接收到一个IHAVE，然后它应该向同一个主机发送一个GET，那么设置一个计时器，在一段时间后（少于5秒）重新传输GET。您的客户机应该有合理的机制来识别数据或获取流量的连续超时何时表明主机可能已崩溃。然后，您的客户机应该尝试从另一个对等机下载该文件（刷新WHOHAS是可以的）。我们将使用类似于图4（a）的网络拓扑来测试您的基本功能。一个更复杂的拓扑，如图4（b）将用于测试并发下载和对崩溃的鲁棒性。正如检查点所建议的，您可以首先用一个完全无损失的虚拟网络来编写基本的流控制代码，以简化开发。



(a) A simple scenario that tests most of the required functionality. Peer D has all the chunks in the file. Peer A wants to get the file from D. In this problem, the file should reach the Peer A, 100% reliably. Peers themselves should not drop valid packets.

(b) An example topology for the speed competition. Peers D and E between them have the entire file. Peers A, B want to get the complete file. The peers should recognize that A and B are close together and transfer more chunks between them rather than getting them from D and E. One test might be to first transfer the file to A, pause, and then have B request the file, to test if A caches the file and offers it. A tougher test might have them request the file at similar times.

Figure 4: Test topologies

6.Spiffy：模拟有损耗的网络

为了测试您的系统，您将需要更多有趣的网络，这些网络可能会有损耗和延迟。为了帮助您实现这一点，我们创建了一个名为“Spiffy”的简单网络模拟器，它完全在您的本地计算机上运行。模拟器由hupsim.pl实现，它在文件指定的节点之间创建一系列带宽和队列大小有限的连接（由topo.map指定），要在虚拟网络上发送数据包，请将sendto（）系统调用更改为spiffy sendto（）。spiffy sendto（）用发送方的id标记每个数据包，然后将其发送到spiffy ROUTER环境变量指定的端口。hupsim.pl侦听该端口（在运行时需要指定该端口运行hupsim.pl），根据发送方的身份，它将通过

topo.map指定的网络发送包到正确的目的地。你交给spiffy sendto（）的包与正常UDP sendto（）调用完全相同。所有数据包都应使用spiffy和spiffy sendto（）发送。

6.1 hupsim.pl

hupsim.pl有四个必须设置的参数

```
hupsim.pl -m <topology file> -n <nodes file> -p <listen
port> -v <verbosity>
```

：包含hupsim.pl会创造的网络配置的文件。给你举个例子todo.map. 文件中的ID应该与中的ID匹配。格式为：

```
src dst bw delay queue-size
```

bw是链路的比特/秒带宽，延迟以毫秒计算，queue-size在包中，不能读文件，只能推断

:网络中所有结点的配置文件，nodes.map是个例子

:hupsim.pl监听的端口，因此，此端口应与网络中节点使用的端口不同。

verbosity：想要从hupsim.pl看到的debug信息，1-4是等级，越高越多

Spiffy Example

```
gcc -c spiffy.c -o spiffy.o
gcc server.c spiffy.o -o
server gcc client.c spiffy.o -o client
```

使用

没有更改nodes.map or topo.map

```
setenv SPIFFY_ROUTER 127.0.0.1:12345
./hupsim.pl -m topo.map -n nodes.map -p 12345 -v 0 &
./server 1 48001 &
./client 2 48002 48001 123
```

The client will print

```
Sent MAGIC: 123
```

and the server will print

```
MAGIC: 123
```

7.评分

此信息可能会更改，但会让您更深入地了解评分时如何分配分数。请注意，许多要点都是用于基本文件传输功能的。在使用更高级的功能或担心边角情况之前，请确保这些功能正常工作。

（50分）搜索chunks并可靠地取回文件：对等方程序应能生成WHOHAS查询并以IHAVE正确响应（有需要的话）为了一个两台主机的简单配置。对等方程序应该能从远程对等方寻找chunks并请求。将会测试输出文件是否和分享的完全一致。记住，除了实现WHOHAS,IHAVE和GET，这部分需要能处理丢包的可靠性。有3个checkpoint来验证正确性（10+20+20）

（20分）支持并效用化并行传输，对等方应该能从多于一个结点同时发送和取回内容（这并不意味着实现线程），你的对等方应该同时利用所有拥有有用数据的结点，而不是简单地同时从一台主机下载一个chunk

（10分）鲁棒性：实现对崩溃的对等方健壮，应该从其它对等方尝试下载被打断的块。一般健壮性：对等方对发送损坏数据的其它对等方应该有弹性。（应先着眼于主要功能，不要老是想边角功能）

（10分）代码风格，包含代码结构，重要功能，实现细节的文档

（10分）阐述如何实现文件传输应用的功能，简要描述实现和设计

8.自动测试

总共有四个脚本（检查点）供您通过。

这个项目的入口点是starter_code中的peer.c。在你实现它之后，你需要make它并将二进制代码复制到每个检查点文件夹中，并检查都通过了。

要点：为了便于分级，请确保可以通过运行从主干目录开始执行以下步骤：

```
% make
% cp peer cp*/
% cd cp*/
% ruby checkpoint*.rb
```

9.提交

- **请在2020.12.20 23:59之前完成Project并提交至Elearning**
- **项目的相关说明及Starter Code请见附件"Project document-2020.pdf"（Project document-2020.pdf）及"Starter Code"（Starter Code.zip）**
- **请务必仔细阅读"Project document-2020.pdf"，确保理解项目要求**
- 提交内容包括：
  - **1）所有源程序代码（可运行）；**
  - **2）Checkpoints.pdf（为便于评分，设计了4个checkpoint，请在此文档中截图展示代码在4个checkpoint上的通过情况）【请注意，我们将再次运行以确保结果一致】**
  - **3）Design.pdf（描述项目的设计和实现）**
  - **4）短视频（演示实现效果并对代码和功能做简要解说）【请尽量控制视频的时长和大小】**
  - **请将以上所有内容打包，并按照"学号-姓名"的方式进行命名，谢谢**
- **具体评分标准见"Project document-2020.pdf"**