P2P 可靠文件传输系统说明文档

18302010017 姚鸿韬'

一、 概述

本文档是计算机网络课程 PJ——实现 P2P 可靠文件传输系统的说明文档，主要介绍系统的设计与主要相关具体代码实现。下文将先介绍整体代码结构及各函数功能，并分四部分介绍系统的主要功能，分别是基础文件传输、可靠文件传输、并行文件传输与健壮性。

二、 代码结构

本身出现在 starter_code 中的文件及函数将不再赘述，下文主要介绍新添加的文件及函数的具体功能。

1. process_packets.[h|c]

用于处理与包(packet)相关的操作（收发）。

(1) process_download()

由 peer.c 中的 process_get()函数调用，处理来自用户的下载请求。具体而言，函数将需要下载的数据块 chunks_to_get 制作为（可能多个）WHOHAS 数据包，并将它们发给所有网络中除自己以外的对等方以询问其是否拥有。

(2) process_PACKET()

由 peer.c 中的 process_inbound_udp()函数调用，处理来自套接字（网络）的数据包。具体而言，函数先读取数据包头部，检验其正确性（魔数与版本号），并按数据包类型分别调用下述五个函数。若有回复数据包生成，则向该对等方发出回应。

(3) process_WHOHAS()

处理来自网络的 WHOHAS 数据包。具体而言，函数先从数据包中读出请求的数据块 HASH 值，并依次判断自己是否拥有该数据块。若拥有则添加至回应数据包 IHAVE 中。

(4) process_IHAVE()

处理来自网络的 IHAVE 数据包。具体而言，函数先检查与该对等方是否已建立连接，或当前连接数是否已达上限。若无，则选取数据包中的一个数据块，在本对等方的下载连接池中添加与该对等方下载该数据块的连接，并向该对等方回应数据包 GET。

(5) process_GET()

处理来自网络的 GET 数据包。具体而言，函数先检查当前连接数是否已达上限。若否，则在本对等方的上传连接池中添加与该对等方上传请求数据块的连接，并缓存该数据块在真实文件中的文件内容。最后，发送第一个窗口中的 DATA 数据包。

(6) process_DATA()

处理来自网络的 DATA 数据包。具体而言，函数先检查数据包头部的序列号。若与当前预期序列号一致，则缓存数据包内容（真实文件），并递增预期序列号，回复 ACK 值与序列号一致的 ACK 数据包；若不一致，则直接回复重复 ACK 包。

同时，数据包收取完成后，若整个数据块都已获取，则进行 HASH 值校验，成功则将缓存中的数据块储存。

整个数据块获取完成后，判断是否所有需要下载的数据块都已完成下载。若已完成则将文件内容输出至指定位置，文件传输结束；否则继续发出请求数据块申请。

(7) process_ACK()

处理来自网络的 ACK 数据包。具体而言，函数先检查数据包头部的 ACK 值。若比当前收到的最后 ACK 值大，则为累积确认，更新最后 ACK 值，滑动发送窗口并进行新的 DATA 数据包发送；否则为重复 ACK，重复计数增加。

若重复计数超过三次，则判为数据包丢失，采用 GBN 处理方式滑动窗口至最后 ACK 值处并进行重发。

(8) handle_timeout()

处理计时器超时。具体而言，每当超时触发就检测下载池和上传池中的每个连接，判断

其是否超时。

对于下载池，若超时则重新发送一个重发 ACK 包。若超时次数超过阈值，则舍弃该对等方，并重新向所有其他对等方请求下载数据块。

对于上传池，若超时则采用 GBN 处理方式滑动窗口至最后 ACK 值处，并重发 DATA 数据包。若超时次数超过阈值，则舍弃该对等方，不再向其发送数据包。

(9) find_peer()

辅助函数，从所有对等方中找到数据包来源的对等方。

2. packets.[h|c]

用于制作、检验数据包。

(1) packet_type

自定义枚举类型，六种数据包类型（WHOHAS、IHAVE、GET、DATA、ACK、DENIED）

(2) packet_header

自定义结构，数据包头部。

共 16 字节，按顺序为魔数（15441,2 字节）；版本号（1,1 字节）；数据包类型（0-5,1 字节）；头部长度（16,2 字节）；数据包总长度（2 字节）；序列号（4 字节）；ACK 号（4 字节）

(3) packet

自定义结构，数据包。

包含一个数据包头部和数据包的真实数据（长度为最大长度 1500-头部长度）

(4) init_packet()

初始化数据包的函数。设置正确的魔数与版本号，参数输入类型、数据长度、序列号及 ACK 号。

DATA 包拥有非 0 序列号，ACK 包具有非 0ACK 号，其余序列号及 ACK 号都为 0。

(5) make_WHOHAS/IHAVE/GET/DATA/ACK()

制作与函数名对应的数据包。调用 init_packet()方法并填入正确参数。

(6) get_packet_type()

辅助函数。返回数据包头部中的数据包类型（枚举类型）。

(7) ntoh_packet()/hton_packet()

辅助函数。将数据包头部进行网络字节序与主机字节序的转换。

(8) is_packet_valid()

辅助函数。检查数据包头部，判断其是否有效。

包括：魔数、版本号是否正确，类型是否在范围内，序列号与 ACK 号是否正确置 0。

3. process_chunks.[h|c]

用于处理与数据块(chunk)相关的操作。

(1) chunk_state

自定义枚举类型，要下载的数据块状态。

包括未下载 NOT_DOWNLOADED，正在下载 DOWNLOADING 和下载完成 DONE。

(2) chunk_t

自定义结构，数据块。

包括 id 和其 HASH 值。

(3) chunks_to_get_t

自定义结构，需要下载的数据块。

包括共有多少个需要下载的数据块，每个数据块的信息，每个数据块的真实文件内容，每个数据块的提供方及每个数据块的当前状态。

(4) init_master_chunks()

初始化所有数据块的信息。

由 peer.c 的 main()函数调用，在对等方运行之初初始化总文件信息，获取文件的真实路径，并将所有数据块的信息读至 master_chunks 全局变量中。

(5) init_chunks_IHAVE()

初始化当前对等方拥有的数据块信息。

由 peer.c 的 main()函数调用，在对等方运行之初初始化当前拥有的数据块信息，并读至 chunks_IHAVE 全局变量中。

(6) init_chunks_to_get()

初始化需要下载的数据块信息。

由 peer.c 的 process_get()函数调用。在收到用户的下载请求后从输入文件中读取需要下载的数据块信息至 chunks_to_get 全局变量中。

(7) update_provider()

更新数据块提供方。

由 process_packets.c 的 process_IHAVE()函数调用。在收到 IHAVE 数据包后定位一个需要下载且并未下载的数据块，将其提供方更新为当前对等方，并返回找到数据块的 HASH 值。

(8) get_chunks_from_packet()

从 IHAVE 数据包中读取可提供数据块的 HASH 值并返回。

由 process_packets.c 的 process_IHAVE()函数调用。

(9) make_chunks_to_packet()

将需要下载的数据块 HASH 值制作为 WHOHAS 数据包的数据。

由 process_packets.c 的 process_download()函数调用。调用时确保范围内的数据块 HASH

值能被一个 WHOHAS 数据包容纳。制作数据包时，只有状态为未下载的数据块 HASH 值会被放入，正在下载或已下载的数据块会被过滤。

(10) make_chunk_data_to_packets()

将数据块的真实数据制作为 512 个 DATA 数据包并返回。

由 process_packets.c 的 process_GET()函数调用。具体而言，先定位请求数据包在所有数据包中的 id，找到其在真实文件中的位置，将该块真实内容 512KB 划分为 512 个 1 字节的数据，分别装入 1 个 DATA 数据包的数据部分，最后将这 512 个 DATA 数据包返回。

(11) sava_data()

将数据块的真实内容保存并进行 HASH 值校验。

由 process_packets.c 的 process_DATA()函数在接受完整个数据块的所有数据包后调用。具体而言，先将数据块的真实内容保存于全局变量中，并进行 SHA-1 的 HASH 值校验。若通过则将数据块状态改为完成 DONE，否则改为未下载 NOT_DOWNLOADED。

(12) find_chunk()

辅助函数，从一系列数据块信息中查找 HASH 值与输入参数相同的那一个。

4. connection_pool.[h|c]

用于处理与连接(connection)、连接池(pool)相关的操作。

(1) download_connection_t

自定义结构，下载连接。

包括连接发送方，连接对应的数据块的 HASH 值，下一个期望收到的序列号，下一个数据储存于缓存区的位置，缓存区，上次本连接被使用的时间与本连接的超时计数。

(2) upload_connection_t

自定义结构，上传连接。

包括连接接收方，最后收到的 ACK 号，下一个被发送的序列号，目前最后一个可用的

序列号，重复 ACK 次数，所有将被发出的 DATA 包的缓存，上次本连接被使用的时间与本连接的超时计数。

(3) download/upload_pool_t

自定义结构，下载/上传连接池。

包括连接池中的连接及连接个数。

(4) init_download/upload_connection()

初始化下载/上传连接函数。

对于下载连接，设置下一个期望的序列号为 1，缓存区位置为 0，发送方及 HASH 值从参数读取。

对于上传连接，设置最后收到的 ACK 号为 0，下一个被发送的序列号为 1，目前最后一个可用的序列号为窗口大小 64，重复 ACK 计数为 0，接收方及 DATA 包缓存从参数读取。

对于二者，连接最后使用时间设为当前（创建时间），超时计数为 0。

(5) init_download_upload_pool()

初始化下载/上传连接池。

初始大小都设为 0。

(6) add_to_download/upload_pool()

将一个连接添加进对应的连接池。

遍历连接池，找到一个连接为空的位置，将连接加入。

(7) remove_from_download/upload_pool()

将一个连接从对应连接池中移除。

释放该连接及其下的所有内容，并将连接池中的该位置设为空。

(8) find_download/upload_connection()

在对应连接池中找到发送方/接收方为 peer 的连接。

用于建立连接前确定当前是否与该对等方已有连接。

(9) send_DATAs()

将上传连接窗口中所有可用但还未被发送的 DATA 数据包发送给连接接收方。

5. my_list.[h|c]

自定义列表类，包括自定义结构列表节点及列表，以及将节点加入列表、从列表头取出一个节点、释放列表等函数。

主要用于将各种数据类型结构化组织，便于读写与处理，具体实现较简单不再赘述。

以上即所有新添加文件及主要函数、自定义结构的功能及实现介绍，在真实代码中也有详细注释以供查看。

三、 基础文件传输

基础文件传输即假设网络中不存在丢包、错序、重复的情况存在，仅考虑如何将一个文件从一个对等方传输到另一个对等方的过程。

假设对等方 1 为下载方，对等方 2 为发送方。

基础文件传输的一次流程为：

用户在对等方 1 处键入 GET 指令；

对等方 1 向网络（对等方 2）发送 WHOHAS 包；

对等方 2 向对等方 1 回复 IHAVE 包；

对等方 1 向对等方 2 发送 GET 包；

对等方 2 不断向对等方 1 发送 DATA 包直至结束；

对于每一个 DATA 包，对等方 1 向对等方 2 回复一个对应的 ACK 包。

下文详述一些重要实现细节。

对等方 2 运行初会初始化一个全局变量 chunks_IHAVE 记录其拥有的数据块信息。

收到用户指令后对等方 1 将初始化一个全局变量 chunks_to_get，跟踪所有要求下载的数据块信息（id、HASH 值、文件内容、状态）。

制作 WHOHAS 包时，遍历 chunks_to_get，将每个状态为未下载的数据块 HASH 值填入 WHOHAS 包的数据处。如果要求下载的数据块个数超过一个 WHOHAS 包能容纳的上限（74 个），则将其分割并生成多个 WHOHAS 包，逐个向网络中发送。

收到 WHOHAS 包后，对等方 2 将其中的 HASH 值与 chunks_IHAVE 中的 HASH 值一一比对，找到对等方 1 需要且对等方 2 拥有的数据块，制作成 IHAVE 数据包并回复。

收到 IHAVE 包后，对等方 1 遍历包中所含有的数据块信息，比对 chunks_to_get，确定一个状态为未下载且包含在 IHAVE 中的数据块。此时，建立与对等方 2 的下载连接，准备开始下载这个数据块（发送 GET）。

收到 GET 包后，对等方 2 建立与对等方 1 的上传连接，并开始发送 DATA 包传输数据。对等方 1 不断接受 DATA 包，将内容置于缓存中，直到传输完毕，将缓存保存至 chunks_to_get。此时检查是否仍有需要下载的数据块。若还有，则递归整个流程；否则将文件内容保存至指定位置，传输结束。

四、 可靠文件传输

可靠文件传输即在基础文件传输的基础上添加可靠性检查。即，现在的网络中可能出现丢包、错序、错包、重复等情况。

仍假设对等方 1 为下载方，对等方 2 为上传方。下述可靠性检查的具体实现细节。

在收到任何一个数据包之初，先检查其头部，判断魔数（15441）及版本号（1）是否正确；类型是否符合范围（0-5）；序列号及 ACK 号是否正确置 0（DATA 包序列号非 0，ACK 包 ACK 号非 0，其余都为 0）。若检查不通过直接舍弃该数据包。

建立连接前，先判断是否与该对等方已存在连接。若已存在，则舍弃该建立连接请求。

对等方 1 收到 DATA 数据包时，先检查其头部的序列号。若与期望序列号一致，说明接收到正确数据包，缓存其数据内容，回复递增 ACK 包；否则，说明接收到错序（重复）数据包，忽略其内容，回复重复 ACK 包。

对等方 2 发送 DATA 包时设置发送窗口。对等方 2 收到 ACK 数据包时，先检查其头部

的 ACK 号。若比最后收到的 ACK 值大，说明为累积确认，该 ACK 号之前的数据包都已被正确收到。此时向后滑动窗口至该 ACK 号后一位，并发送可用数据包；若与最后收到的 ACK 值一致说明为重复 ACK，不滑动窗口，重复计数递增。若重复计数已达上限（3 次），认为数据包丢失，采用 GBN 处理办法，窗口前滑至最后收到的 ACK 值后一位，并重发窗口数据包。

还需处理超时。利用一个 SIGALRM 信号作为计时器。如果对等方 1 超时未收到新的 DATA 数据包，则向对等方 2 主动发送一个新的重复 ACK 数据包；如果对等方 2 超时未收到新的 ACK 数据包，则 GBN 窗口前滑并重发窗口数据包。

收到所有 512 个数据包后，对等方 1 将对其进行 SHA-1 校验。若校验通过则保存数据，否则数据包中有内容出错，该数据块状态重设为未下载，并重新向网络请求下载。

五、 并行文件传输

上文所述可靠文件传输仅在两对等方之间单向进行。并行文件传输即在可靠文件传输的基础上引入一对等方同时可以向多方请求数据块，同时一对等方同时可以向多方发送数据块。注意，某特定两对等方之间某方向上同时仅能存在一条传输连接，即对等方 1 同时只能向对等方 2 请求下载一个数据块。

假设对等方 1 为下载方，对等方 2 与 3 为上传方。对等方 1 从对等方 2 处请求块 2，从对等方 3 处请求块 3。更多的上传方及并行上传与之类似。下文详述并行下载的实现细节。

每个对等方运行之初维护两个全局变量 download_pool 下载连接池与 upload_pool 上传连接池，大小由对等方建立时的 max_conn 确定。

建立连接前，先判断连接数是否达到上限，若已达上限则忽略请求。

对等方 1 向对等方 2 发出 GET 数据块 2 数据包后，将一条发送方为对等方 2，数据块为 2 的下载连接加入下载连接池中。对等方 3 同理。

对等方 2 收到 GET 数据包后，将一条接收方为对等方 1，数据块为 2 的上传连接加入上传连接池中，并开始发送 DATA 数据包。对等方 3 同理。

此时，对等方 1 会同时收到来自对等方 2 与对等方 3 的 DATA 数据包。在处理 DATA 数据包前，对等方 1 会先从下载连接池中获取与当前发送方对应的下载连接。接下来对 DATA 包的处理都是基于这条下载连接的。

同理，对等方 2 与对等方 3 在处理 ACK 数据包之前，会先从上传连接池中获取与对等方 1 的上传连接，所有的处理都是基于这条上传连接的。

假设数据块 2 先行下载完毕。此时对等方 1 会试图继续请求下载，但由于数据块 3 此时状态为下载中，因此它不会出现在 WHOHAS 数据包中。直到数据块 3 也下载完成并校验正确，对等方 1 确定整个文件已被取得，将数据写入目标文件，传输完成。

由于多连接并行，因此无法对每个连接应用一个信号进行计时。程序采用的方法是采用一个全局计时器（仍为 SIGALRM），且每个连接维护一个最后使用时间。连接每次被使用，最后使用时间就被更新为当前时间。全局计时器被触发后，对连接池中的每条连接（上传与下载）进行超时检查（当前时间减最后使用时间）。若连接超时，则进行相应超时处理，并更新最后使用时间。当所有连接被释放，意味着文件传输结束，计时器停止。这样就起到了用一个信号控制整个程序所有连接超时情况的效果。

六、 健壮性

最后，上述并行处理并未考虑对等方在传输过程中异常断开，或是 WHOHAS、IHAVE、GET 包丢失的情况。如果出现这些情况，对等方可能会在无限发包/等待中卡死。因此，健壮性即让程序能够在这些情况发生时仍正常运行。

仍假设对等方 1 为下载方，对等方 2 与 3 为上传方。对等方 2 与对等方 3 同时拥有块 2 与块 3。首次请求时，对等方 1 从对等方 2 处请求块 2，从对等方 3 处请求块 3。下文详述健壮性实现细节。

为处理对等方异常断开的情况，在每条连接新维护一个变量超时计数。连接每次被检测超时，超时计数即递增，而连接每次被使用，超时计数就清零。连接触发超时后，检验其超时计数是否超过阈值（5）。若超过，则认为连接已被废弃（对等方断开）。不妨对等方 1 断开，对于对等方 2 与对等方 3，直接将连接从上传连接池中移除即可。而如若对等方 2 断开，对于对等方 1，需将数据块 2 的状态改为未下载，并重新向网络（对等方 3）请求下载数据块 2。对等方 3 会在数据块 3 传输完毕后，继续向对等方 1 传输数据块 2。如此，对等方 1 即从对等方 2 的断开错误中恢复，并最终仍成功下载到了整个文件。

为处理 WHOHAS、IHAVE 包丢失，在发送 WHOHAS 包后就打开计时器。不妨对等方 1 与对等方 2 间的与数据块 2 相关的 WHOHAS 或 IHAVE 包丢失了（二者效果是一样的）。

对等方 1 将无从得知对等方 2 处拥有数据块 2，也就无法建立与它的连接。计时器触发超时后，除了检查每个连接池外，如果没有任何现存的连接，不再直接停下计时器，而是检查 chunks_to_get 中的下载任务是否都已完成。若都已完成，则停止计时器，否则说明 WHOHAS 或 IHAVE 包丢失从而未能建立连接，此时对等方 1 会再次向网络发送 WHOHAS 数据块 2 包以请求下载。此次对等方 2 成功接收到并成功返回了 IHAVE 包，数据块 2 的下载连接由此成功建立。

为处理 GET 包丢失，不妨对等方 1 向对等方 2 发送到 GET 数据块 2 包丢失。此时，对等方 1 已建立与对等方 2 的下载连接，但对等方 2 无法建立与对等方 1 的上传连接。对等方 1 的下载连接触发超时后，处理器检查发现当前缓存位置为 0，得知是 GET 包丢失导致的。此时不再重发 ACK 包，而是重发一个 GET 包。此次对等方 2 成功接收到了 GET 包，并建立了与对等方 1 的上传连接，数据块 2 开始正常传输。

至此，一个基于 UDP 的健壮可靠 P2P 文件传输系统就成功实现了。