Peer 2 Peer File Transfer

Release 1.0

Ethan Iannicelli

Apr 11, 2025

CONTENTS:

CHAPTER

ONE

PEER

The `Peer` class represents a Peer entity in a P2P file transfer system that supports broadcast discovery and chunk file shring with other peers in the network:

class `peer.`**`Peer`**(*discovery_port*, *transfer_port*)

Bases: `object`

`start`()

Start peer broadcasting and transfer services. Also start command line input loop for handling commands. - list: output the files and chunks available on this peer - download <file>: requests all available peers for any chunks they have for <file>. use threading for each request, and accumulate in a shared local memory to build the file on the receiver end. - exit: stop the peer and shut down

CHAPTER TWO

DISCOVERY

The `PeerDiscovery` class represents the entity responsible for broadcasting and discovering other peers in a network. This broadcast is ‘parent’ peer, and detects other discovery entities that are serving their respective ‘parent’ peers:

class discovery.**PeerDiscovery**(*transfer_port*, *port*)

Bases: `object`

broadcast_announcement()

Broadcast availability to the network on the broadcast port. send the sender port the distributee port with this broadcast, as well as the ‘start’ itentifier

listen_for_peers()

Listen for incoming peer announcements on the broadcast port. when it receives a broadcast, extract the peer ports that are packaged in the broadcast. if the broadcast is defined as ‘start’, add to known peers - if ‘stop’ remove from known peers

start()

listening for peers and broadcasting itself

stop()

shut down this discovery entity. broadcast itself with ‘stop’ flag, and set running to false

CHAPTER

THREE

TRANSFER

The `FileTransfer` class represents the entity responsible for distributing, receiving, requesting, and sending file chunks to other peers. It is always in a state of distributing and receiving file chunks from other peers, and when it's peer starts a file request it is responsible for getting this data from other peers and saving it to a file:

class `transfer.`**`FileTransfer`**(*port*, *discovery*, *files*)

Bases: `object`

`distribute_files`()

periodically (every 10 seconds) randomly send chunks of files that this peer knows to other peers. all chunks are sent, but the peer is randomly chosen. this is done to simulate peers not having perfect knowledge of the files from the start and to simulate rate limits of data sent.

`handle_client`()

Handle an incoming file request. receive a file name to initialize this process, and then send all chunks that this peer has to the sender address in separate requests.

`receive_distributed_files`()

recieve chunks of files sent to this port. extract the filename, chunk number, and chunk and save it to the files dictionary representing local session storage.

`request_file`(*filename*, *peer_port*, *chunks*, *peer*)

Request a file from a peer. This is called as one of many threads, so we use a global chunks dictionary to store all downloaded chunks. When the chunks dictionary is 'full' (all chunks for file are present), this thread marks as so and attempts to write to a file. If a timeout occurs, the user is notified and an incomplete file is downloaded and written

Parameters

- **`filename`** (*`string`*) – the name of the file to be downloaded
- **`peer_port`** (*`string`*) – the port of the peer that we want to request from
- **`chunks`** (*`map<string, string>`*) – global dictionary for this file's chunks being stored in
- **`peer`** (`Peer`) – the peer that called this function, used to know status of other threads that the peer spawned.

`start_server`()

Start the file transfer server. start threads to handle requests from other peers, distribute chunks to other peers, and receive distributed chunks from other peers

CHAPTER

FOUR

PACKET

To calculate the psuedo udp checksum of the data, use the `udp_checksum()` function. This function is neccessary for maintaining packet integrity:

`packet.`**`udp_checksum`**`(`*data*`)`

perform a psuedo udp checksum by reducing the data to 4 bytes and taking one's complement

Parameters
`data` (*`bitstring`*) – the data that the checksum is created from

Returns
generated checksum

Return type
int

To create a packet from a chunk number and a chunk of data, use the `create_packet()` function:

`packet.`**`create_packet`**`(`*data*, *chunk_num*`)`

create a packet using packet data and a chunk number

Parameters

- **`data`** (*`bitstring`*) – the data to be included in the packet
- **`chunk_num`** (*`int`*) – the chunk number associated with this data

Returns
bitstring representing formed packet

Return type
bitstring

To parse the components of a packet into its different parts, used the `parse_packet()` function. The parts of the packet returned are the checksum, chunk_number, and the chunk data:

`packet.`**`parse_packet`**`(`*packet*`)`

extracts checksum, chunk_num, and data from a packet

Parameters
`packet` (*`bitstring`*) – formatted packet

Returns
3 tuple of check, chunk_num, data

Return type
tuple

CHAPTER FIVE

MAIN

To create and return a parser for the program, use the `parse_arguments()` function:

`main.`**`parse_arguments`**`()`

> create and return argument parser. the parser handles the broadcasting port and the default transfer port. the transfer port requires it, +1, +2, +3 ports to be unused before running the program.

The main function of this program is `main()`. It parsers arguments using a parser, and starts and instance of the peer:

`main.`**`main`**`()`

> main function for P2P program, starts Peer and accepts arguments.

CHAPTER

SIX

LOGGER

To get the logger from the program, use the `get_logger()` function:

`utils.logger.`**`get_logger`**`(`*name='P2P'*`)`

creates and returns a configured logger for the UI of the program

Parameters
`name` (*`string`*) – name of the program/app

Returns
logger

Return type
logger

To create a formatted string output for a in memory stored file dictionary containing chunks, use the `format_file_chunks()` function:

`utils.logger.`**`format_file_chunks`**`(`*files*`)`

helper function to format the output for the files store in local memory on a peer this is not really a logger function, but is used in output

Parameters
`files` (*`map<string, map<string, string>>`*) – files and chunks to output

Returns
files and the number of chunks available in string format

Return type
string