Project ArsNet v1.000000001

Usama Ejaz, Ali Hasnat

Introduction:

A system which downloads a large size file from multiple servers simultaneously. The project is divided into two major parts i.e. Client.py and Server.py.

Implementation Details:

Server:

- The servers are set up by running the Server.py. The Server.py file launches the servers indicated by the port numbers given in the arguments.
- Valid port numbers are between 0 and 65535. However, some of the ports in the range 1 through 1023 are used by system-supplied TCP/IP applications. So, to be on the safe side, the program checks for the valid port number between 1023 and 65535.
- The Server.py launches the specified number of servers in threads. Each server opens the file in a bytestream and loads it into a list. Each list in the server is divided into segments using the concept of byte-slicing and concatenation.
- The server start to listen at each port for any incoming connection.
- After connecting to the client, each server sends the header including the file size before hand to make the client ready to receive.
- Since, the packets may not arrive in order, the server sends segment number along the segment.

Client:

- The client is set up by running the Client.py. The Client.py starts tries to connect to each port specified in the arguments. The client creates respective threads to obtain the relevant segments.
- The client handles the scenario of server failure or if server disconnects abruptly.
- The client receives the file size and divides the file size into parts and listens and receives that number of bytes from each server.
- Each segment is obtained in a parallel fashion, So the client sorts the segments into order according to the segment number received.
- After receiving all the segments, the client recombines the segments into a file. The file is then written on the specified location.
- Incase of server failure, the client sends request to the alive servers to get the remaining segments, the segments are divided just like the file was divided to ensure load balancing.
- Incase all the servers fail while sending, the client can resume the downloading process. The client dumps the received bytes every time into a file “dump.txt” and writes the segment numbers in recevied_segments.txt. When the client is run in resume mode, the client reads the already received from the file and sends request for the remaining segments.

Libraries Used:

1. Socket
2. Os
3. Time
4. Threading
5. Argparse

Problems Faced:

1) Load balancing required a lot of thought. I first implemented a recursive version of load balancing, but it didn't work as expected. I later solved the issue by creating another method to get reaming segments in the client.
2) Everything required extra effort due to restriction of using libraries to implement core functionality

Server.py:

Arguments:

The command line syntax for the server is given below:

Server.py -i <status_interval> -n <num_servers> -f <file_location> -p <list_of_server_port_numbers>

- -i (Required) Time interval in seconds between server status reporting
- -n (Required) Total number of virtual servers
- -f (Required) Address pointing to the file location
- -p (Required) List of port numbers('n' port numbers, one for each server)

Usage:

```
usage: TcpServer.py [-h] [-i STATUS_INTERVAL] [-n NUM_SERVERS]
                    [-f FILE_LOCATION] [-p LIST_OF_PORTS [LIST_OF_PORTS
...]]


optional arguments:
  -h, --help            show this help message and exit
  -i STATUS_INTERVAL, --status_interval STATUS_INTERVAL
  -n NUM_SERVERS, --num_servers NUM_SERVERS
  -f FILE_LOCATION, --file_location FILE_LOCATION
  -p LIST_OF_PORTS [LIST_OF_PORTS ...], --list_of_ports LIST_OF_PORTS
[LIST_OF_PORTS ...]
```

Client.py:

Arguments:

Your client should be named client.py. The command line syntax for the client is given below:

Client.py -r -i <metric_interval> -o <output_location> -a <server_ip_address> -p <list_of_server_port_numbers>

- -i (Required) Time interval in seconds between metric reporting (section 3.4)
- -o (Required) path of output directory
- -a (Required) IP address of server
- -p (Required) List of port numbers (one for each server)
- -r (optional) Whether to resume the existing download in progress

Usage:

```
usage: TcpClient.py [-h] [-i METRIC_INTERVAL] [-o OUTPUT_LOCATION]
                    [-a SERVER_IP_ADDRESS]
                    [-p LIST_OF_PORTS [LIST_OF_PORTS ...]] [-r RESUME]

optional arguments:
  -h, --help            show this help message and exit
  -i METRIC_INTERVAL, --metric_interval METRIC_INTERVAL
  -o OUTPUT_LOCATION, --output_location OUTPUT_LOCATION
  -a SERVER_IP_ADDRESS, --server_ip_address SERVER_IP_ADDRESS
  -p LIST_OF_PORTS [LIST_OF_PORTS ...], --list_of_ports LIST_OF_PORTS
[LIST_OF_PORTS ...]
  -r RESUME, --resume RESUME
```