POLITECNICO DI TORINO

Management and content delivery for Smart Networks

– Components of the team –

"I declare that what is written in this relation is the result of my work"

Avalle Giorgio **S241834**

Contents

Python code explanation

Introduction

Written code is intended as something more general of what actual requirements are: the basic idea is to start developing something that could be used by myself in the future, if needed. For this reason, you can see that part of the code is not implemented yet (there is just a skeleton) and you can notice that I've tried to keep it as modular as possible, sometimes defining functions who just call other ones, more internally. Anyway, this does not impact on the performances required by the exercises of the laboratories: the code is also well commented, to improve its readability as much as possible.

The code can be divided in **two parts**: the first one refers to classes used to model a generic queue (needed to solve the first laboratory), while the second part refers to classes that model the Cloud storage environment (the other laboratory).

Generally speaking, the code is based on functionalities provided by external libraries (that must be installed in order to test and execute my code):

- **simpy**, to create and run the simulation environment
- **numpy** and **scipy**, to compute particular statistics (such as confidence intervals)
- **matplotlib**, to draw plots on the screen

input_controls.py

This file contains several methods that can be used to control the validity of input arguments

timed_structures.py – TimedData

This data structure refers to data ("data" field) which are characterized also by a **temporal information** ("time" field): according to the "timestamp" Boolean flag, "time" can be a timestamp or a temporal interval.

timed_structures.py – TimedArray

This class represents **arrays formed by "TimedData" objects**, having same "timestamp" value associated (i.e.: it is not possible to mix timestamps with temporal intervals).

Several methods are implemented on this class, in order to retrieve, add, filter, search or update the content of the array.

Cloud Storage

Now, we will analyze the classes and methods I have implemented in order to simulate a cloud storage environment. Some of the previous python files will be recycled, to implement functionalities such as graphs plotting.

cloud_env.py – CloudEnvironment

This class models the cloud storage environment: once instantiated, will be attached to it

- A **simpy** simulation environment
- The **logger** passed as parameter
- A **file manager**, which emulates the central server behavior implementing functionalities for cloud files and shared folders
- A **statistics manager**, to retrieve final statistics about the run simulation

The method "**run**" is used to start a simulation and retrieve statistics: "**generate_network**" creates devices and shared folders, attaching last ones to devices in a random way (this method implements the code given by the professor). Finally, "**look_for_peers**" is useful for P2P applications: it retrieves the list of online devices who have downloaded the file we are looking for.

file_manager.py

Methods "**prepare_input**" and "**get_files**" are used to generate the list of possible files we can find in the cloud. Text file "throughput.txt" contains input data.

Please note: my implementation of the devices' files is not the one suggested by the lab instructions. I was not able to open the text file "ubuntuone_PutContentResponse", because of its size: for this reason, I have decided to consider the sizes stated in the file "throughput.txt" **as sizes of files** (instead of batches). In this way, each line of this text file can be seen as a possible file we can find in the cloud environment, characterized also by a **throughput** value: this value will be used to retrieve the actual simulated throughput value associated to the download/upload operation for the file and, then, to evaluate the download/upload **duration**. This concept will be explained below.

file_manager.py – CloudFile

This class models a file (which has not been shared yet): its "**ID**" is an increasing integer number, while "**size**" and "**throughput**" are the one stated into the text file "throughput.txt"

file_manager.py – SharedFile

This class models a file shared on the cloud: it extends the class "CloudFile". Shared files are characterized also by the **shared folder** in which are placed (the same cloud file can belong to different folders), "**last modified**" timestamp and the ID of the devices who has done the latest update to the file content ("**last_device**", which is usually the owner).

Two shared files can be considered equal if thay have same ID and belong to the same shared folder.

file_manager.py – FileManager

It has the responsibility to create the **list of possible files**, reading the content of "throughput.txt", and **to upload files**. As you can see in the method "**new_upload**", the upload can involve a new file or can be seen as the update of an existing file's content. The file to create/modify is chosen randomly: if the generated random number is higher than the percentage of existing files (according to the list of possible files we can find in the cloud), a non-existing file will be uploaded by the involved device.

Please, note: if all the possible files are already present in the cloud, only updates will be allowed. In this special situation, that can be reached incrementing the simulation time or decreasing the inter-upload time, we may evaluate the possibility to increment the number of possible cloud files, adding lines to the input text file "throughput.txt".

shared_folder.py – SharedFolder

This class model a shared folder, which contains "**SharedFiles**" and is associated to a list of **devices**. Each shared folder is characterized by an **ID**, an integer increasing number.

Once a file is uploaded/updated with the method "**upload_file**", devices associated to the shared folder have to be notified by this operation. The method "**notify_devices**" has this responsibility and allow them to download the file content "on the fly" if they are online.

device.py

In this file we can find methods used to generate time intervals, according to their requested pdf:

- **Inter-session time** is the time interval between two sessions of a device
- **Session duration** is the time interval between the log in and the log out of a device
- **Inter-upload time**, the time interval between two consecutive file uploads/updates
- **Download/upload time**, retrieved starting from the file size and its download rate
- **Download/upload rate**, evaluated according to a uniform pdf starting from the throughput value associated to the file we are going to download from the server/other peers. Possible values go from ½ throughput to 2.5 times of it: this means that the expected transfer rate value **is not** the throughput value reported into the "throughput.txt" input file!

device.py – Device

Devices of the cloud are characterized by an integer **ID** and by the list of **shared folders** they are associated to: when they log in, they chose randomly a shared folder of them and work on it only, for the entire session time. The device can access the cloud environment it works into and to the statistics manager: we can find also the list of **files it has to download** from the server/other peers and the **uploads/updates not completed yet**. Finally, the field "**p2p_contribution**" refers to the quantity of bits the device has sent to other peers, while "**logged_in**" is True if the device is online.

The method "**prepare**" processes "**run**", the method used to involve the device activities into the cloud storage environment simulation: this method generates sessions in an infinite loop, waiting the inter-session time before starting it and choosing the working shared folder.

The method "**session**" realizes a session for the device, given its time duration: first of all, missing file in the working folder are downloaded. Then, if we have enough spare time, we can have uploads/updates of files and parallel downloads (files uploaded in the folder by other devices are downloaded "on the fly").

When we **transfer a file**, we always have to:

- Evaluate the transfer rate and, according also to the file (portion)'s size, compute the transfer duration
- Consider if we have enough time to complete the operation (residual session time)
- Wait for the transfer end before starting a new operation: anyway, uploads/updates and parallel downloads are independent each other and can coexist.

Note that, in case of the **P2P approach**, the file is divided in several parts (downloaded from different devices): each of these portions of file has to be considered as an independent transfer, characterized by their specific rates, durations and data sizes. If the file cannot be entirely downloaded by peers, the residual part of it is downloaded from the server.

Methods "**download**" and "**p2p_download**" are called to download a file from the server/another peer: "**upload**" and "**p2p_upload**" are their counterpart.

Method "**uploads**" allow the device to upload/update files until the end of its session: data transfer is always oriented to the server. The idea is to wait the inter-upload time and then send the file: if there is no enough time to complete the operation, the file is marked as a "missed upload" and will be transferred the next time the device will log in on this specific shared folder. The file to transfer is chosen calling the method "**to_upload**", which internally call the method "new_upload" of the FileManager.

The method "**triggered_downloads**" allow the device to download on the fly fresh file from the cloud server. It is called after the "standard" download phase and it lasts for the residual session duration: the reason why P2P connections are not established at all inside this method is that only one device (the owner of the file) could be an uploading peer. Sending the file from the server to all the online devices working in the specific shared folder is faster and more effective. Anyway, a download is triggered when the device is notified by the method "notify_devices" of the SharedFolder class.

cloud_stats.py – StatsManager

This class is used to compute the **final statistics** at the end of the simulation. Internal data structures are TimedArray, the same class used before for the queues: they are lists of timestamp/time intervals, joined by a data which often is the id of a device or the number of devices/connections at a certain time instant.

Counters:

- **current_online_devices**: number of active devices (logged in)
- **current_downloading**: number of device which are downloading a file from the server
- **current uploading**: number of devices which are uploading a file to the server
- **current_p2p_exchanges**: number of active P2P connections
- **p2p_downloaded_data**: bits downloaded from other peers
- **server_downloaded_data**: bit downloaded from the server

TimedArray:

- **online_devices**: number of devices online at a given timestamp
- **downloading**: number of devices downloading a file from the server, at a given timestamp
- **uploading**: number of devices uploading a file to the server, at a given timestamp
- **p2p_downloading**: number of P2P connections at a given timestamp
- **online_for**: session durations, for a given device's ID
- **download_for**: download duration, for given device's ID and file's ID
- **download_for_succes**: as before, but referred only to correctly downloaded files
- **upload_for**: upload duration, for given device's ID and file's ID
- **upload_for_success**: as before, but referred only to correctly uploaded files

Other arrays (the timestamp is the array index):

- **server_load_in**: bits uploaded by devices to the server, at a given timestamp
- **server_load_out**: bits downloaded by devices from the server, at a given timestamp

Methods of the class StatsManager simply interact with these data structures, updating information stored into them: the method "**stats**" is the one which gives to the user statistical results and print graphs. Methods to retrieve means are very similar (sometimes identical) to the ones used to retrieve statistics for queues.

Second laboratory

The goal of the second laboratory is to create a simulation environment able to model and test a **cloud storage environment**: devices work on shared folder, one per session. When a new file is uploaded into the directory, other devices have to retrieve it, starting the download from the central server and/or (if possible) from other active peers.

The set of possible files that can exist into the cloud storage environment, as said in the code explanation, is reported into the textual input file "*throughput.txt*": each line is a file, whose ID is chosen as an increasing integer number and whose size and average throughput are the one reported into the file.

Intro – Simulations' structure

Simulation duration is always set equal to 360.000 seconds (100 hours): three simulations are done per exercise, setting the **number of devices** to 10, 100 and 500 (300 for the second exercise).

Output of the application consists in graphs, resulting statistics and a log file (with the complete information on what is happening inside the cloud storage environment).

In particular, printed graphs concern temporal evolution of some data of interest. The graph may be missing, if no information are available (ex: P2P connections not allowed). Represented information are:

- The number of **active** (online) **devices**
- The number of **devices downloading** a file from the server
- The number of **devices uploading** a file to the server
- The number of established **P2P connections**
- The aggregate **uploading rate** (server incoming traffic), in bits/s
- The aggregate **downloading rate** (server outgoing traffic), in bits/s

As in the previous exercises graphs are available in the appendixes, or as higher resolution images in the files attached to this report.

Exercise 01

No P2P connections are allowed: files are uploaded and downloaded from the central server. Once uploaded by the owner, other devices have to download the file: downloads can be done on the fly (when the upload finishes) or immediately after the log in.

Obtained graphs can be found in the attached files: as an example, you can find the ones of the second simulation in *"Appendix D – Laboratory 02: Graphs of the exercise 01"*

Test	#1	#2	#3
Devices	10	100	500
Total downloaded data	522 MB	6745 MB	63657 MB
Avg time spent downloading data	5013.8 s	2963.7 s	5467.1 s
Avg time spent uploading data	77524.0 s	128808.1 s	113096.0 s
Avg download duration	1563.0 s	1668.0 s	1525.0 s
Avg upload duration	1684.0 s	1896.0 s	1826.0 s
Avg server incoming traffic	560.8 Kb/s	8210.7 Kb/s	33535.8 Kb/s
Avg server outgoing traffic	11.6 Kb/s	150.1 Kb/s	1416.6 Kb/s

Commenting results

As specified in the code explanation, file sizes the system is working with are not big: the list of possible file size values has been retrieved from a file ("*throughput.txt*") **which is not the suggested one**. The reason is that this file is huge and I simply cannot open it properly.

Anyway, I justify my choice saying that work with smaller file **do not alter the workflow** of the application: in fact, also changing the input file for the cloud storage file list the software can operate. Clearly, I'm aware that working with bigger files have produced different statistics: their transfers would have been longer.

Note that uploads/downloads can fail, if the device disconnects prematurely. In the reported statistics, the up/download "duration" refers to successfully ones, while the "time spent" refers also to failures.

Exercise 02

P2P connections are allowed: files can be uploaded and downloaded from the central server or, if possible, by other active peers. Once a new file is uploaded by the owner into the Cloud, other devices have to download it: downloads can be done on the fly (when the upload finishes) or immediately after the log in.

Obtained graphs can be found in the attached files: as an example, you can find the ones of the second simulation in *"Appendix E – Laboratory 2: Graphs of the exercise 02"*.

Test	#1	#2	#3
Devices	10	100	300
Total downloaded data (server)	70.8 MB	10908 MB	27439 MB
Total P2P exchanged data	295.8 MB	8852 MB	60835 MB
Avg time spent downloading data	683.4 s	2383.25 s	3091.4 s
Avg time spent uploading data	120247.6 s	134005.66 s	125767.8 s
Avg download duration	619.0 s	1519.0 s	1379.0 s
Avg upload duration	1793.0 s	1812.0 s	1829.0 s
Avg server incoming traffic	829.0 Kb/s	8487.8 Kb/s	22634.9 Kb/s
Avg server outgoing traffic	1.6 Kb/s	242.6 Kb/s	610.4 Kb/s

Commenting results

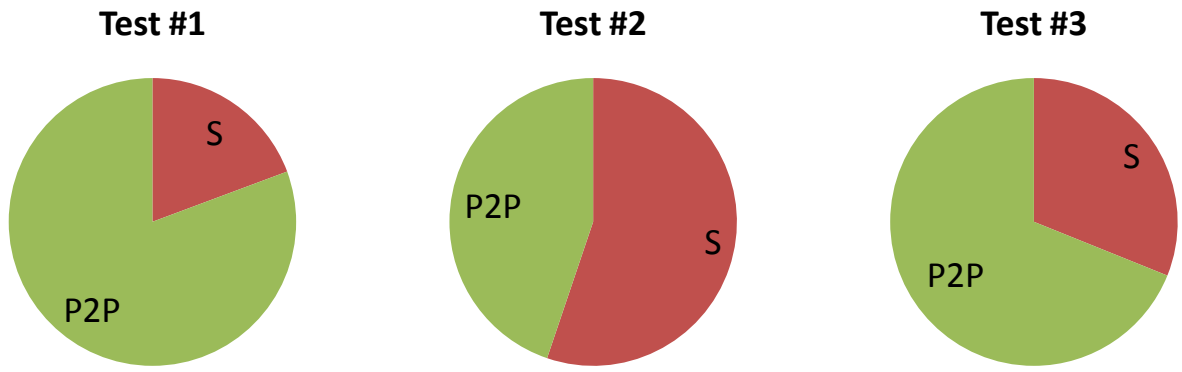
As in the previous exercise, what has been said about file sizes is still valid. Also, in the reported statistics the up/download "duration" refers to successfully ones, while the "time spent" refers also to failures.

As you can see from the table, P2P connections allow to **reduce significantly the server workload**: in fact, download requests can be satisfied by other devices. Comparing server outgoing traffic and server downloaded data with server incoming traffic and P2P downloaded data, we can have an idea of how P2P can help to scale the Cloud service under analysis.

The software gives to the user the amount of data uploaded by the Top-5 P2P contributors: results are reported in the table below.

Although, P2P transfers can succeed only if at least one online uploading peer **exists** (it's active and has the required file) and it **stays active** until the transfer is completed. Devices sessions durations have to coincide often if we want to take the best advantage from P2P: the more the devices in the Cloud environment, the more the matches we can potentially have - that's the reason why in the first simulation we have only two contributors.

In these simulation, the predominant factor who limits the P2P approach is the fact that **peers have to be online simultaneously**. The reason is that they spend a lot of time being offline or downloading files in the post-login session phase: in the case they have the file, they still cannot share it to other devices.



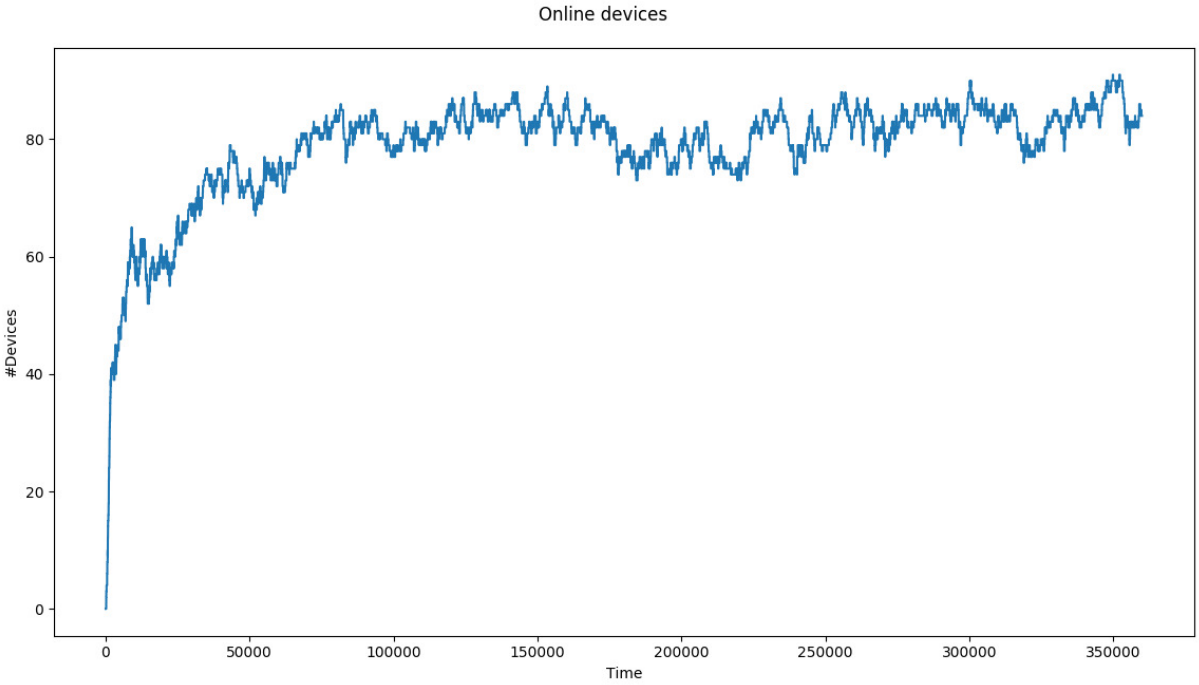
Ratio between total data downloaded from the server (red) and other peers (green)

Test	#1	#2	#3
Top 1: peer ID – Data	8 – 240 MB	90 – 2531 MB	293 – 4143 MB
Top 2: peer ID – Data	3 – 55.9 MB	93 – 2482 MB	248 – 3664 MB
Top 3: peer ID – Data	//	33 – 1280 MB	243 – 3348 MB
Top 4: peer ID – Data	//	38 – 956 MB	11 – 3276 MB
Top 5: peer ID – Data	//	78 – 324 MB	10 – 2849 MB

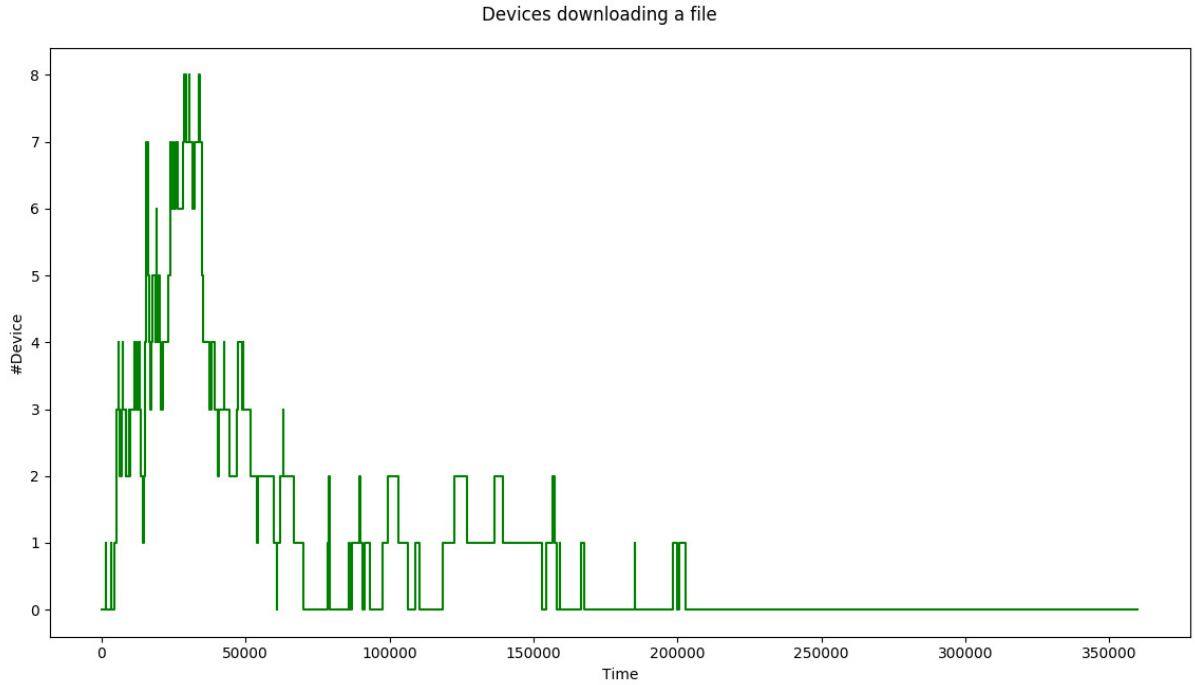
Top-5 P2P contributors: device ID and uploaded data

Appendix D – Laboratory 02: Graphs of the exercise 01

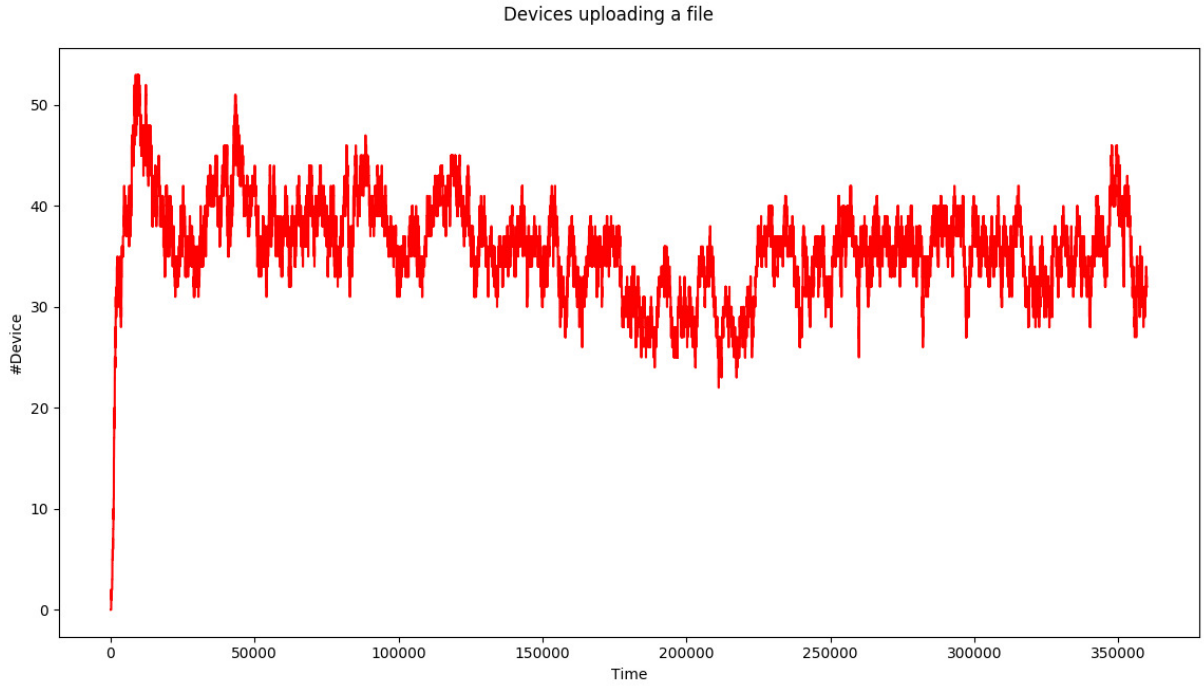
Simulation 02: 100 devices, 100 hours



Number of online devices

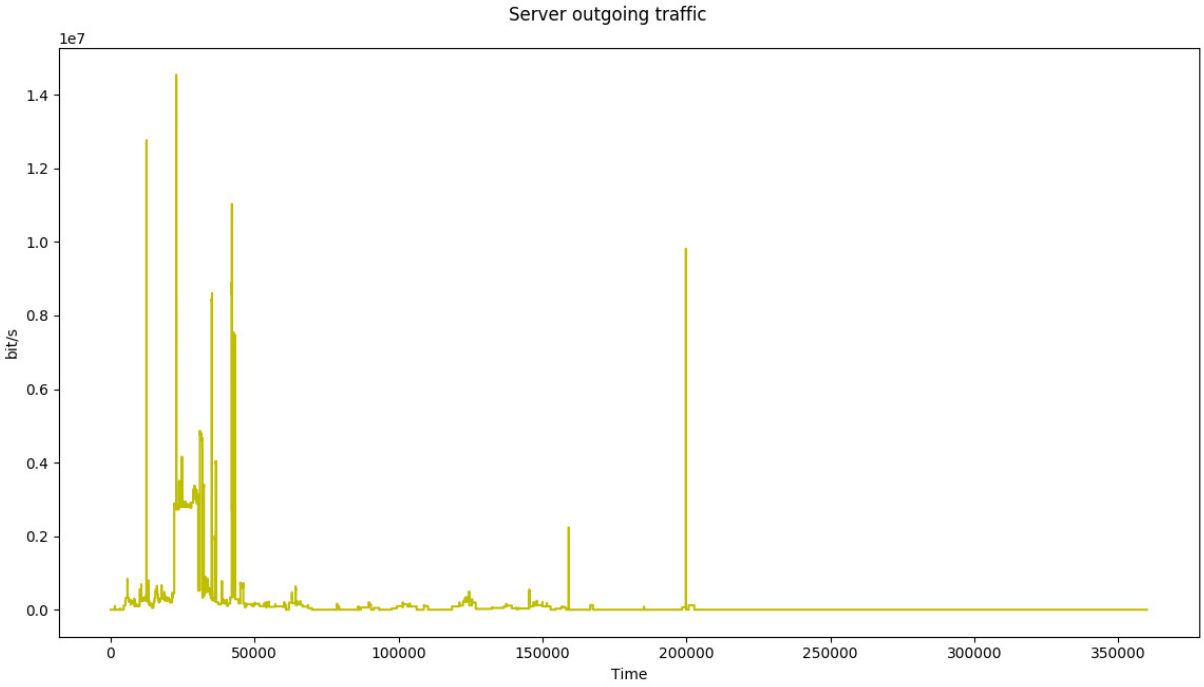


Number of devices downloading a file from the server



Number of devices uploading a file to the server

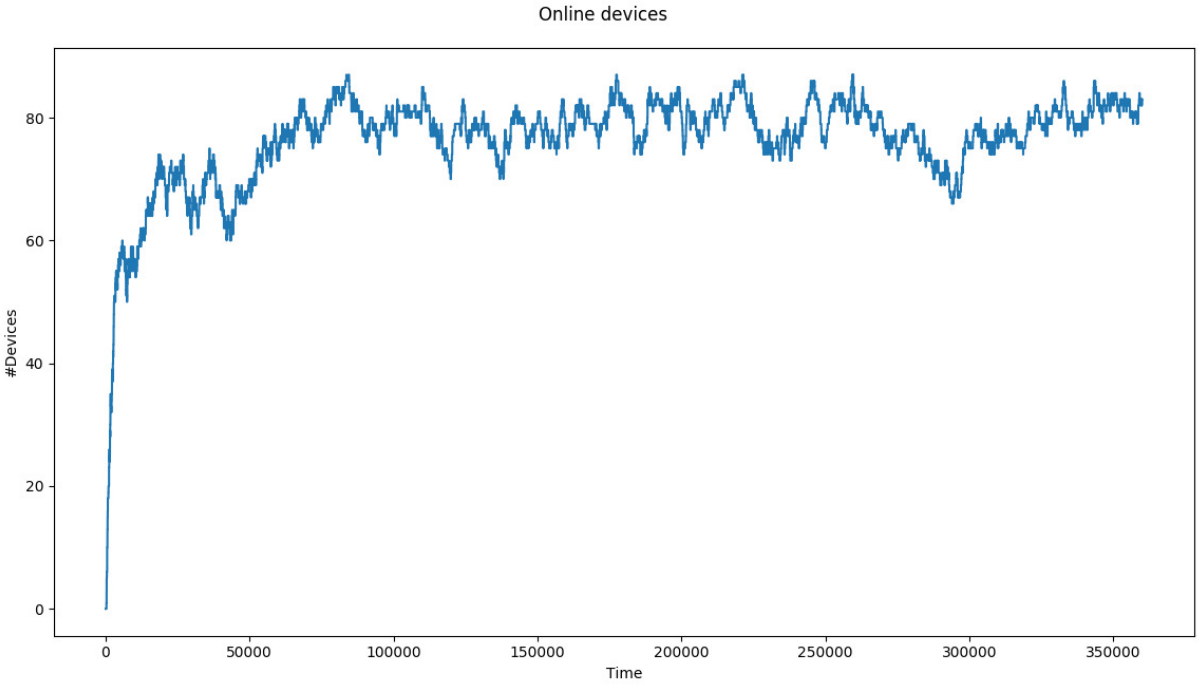
Server incoming traffic rate (b/s)



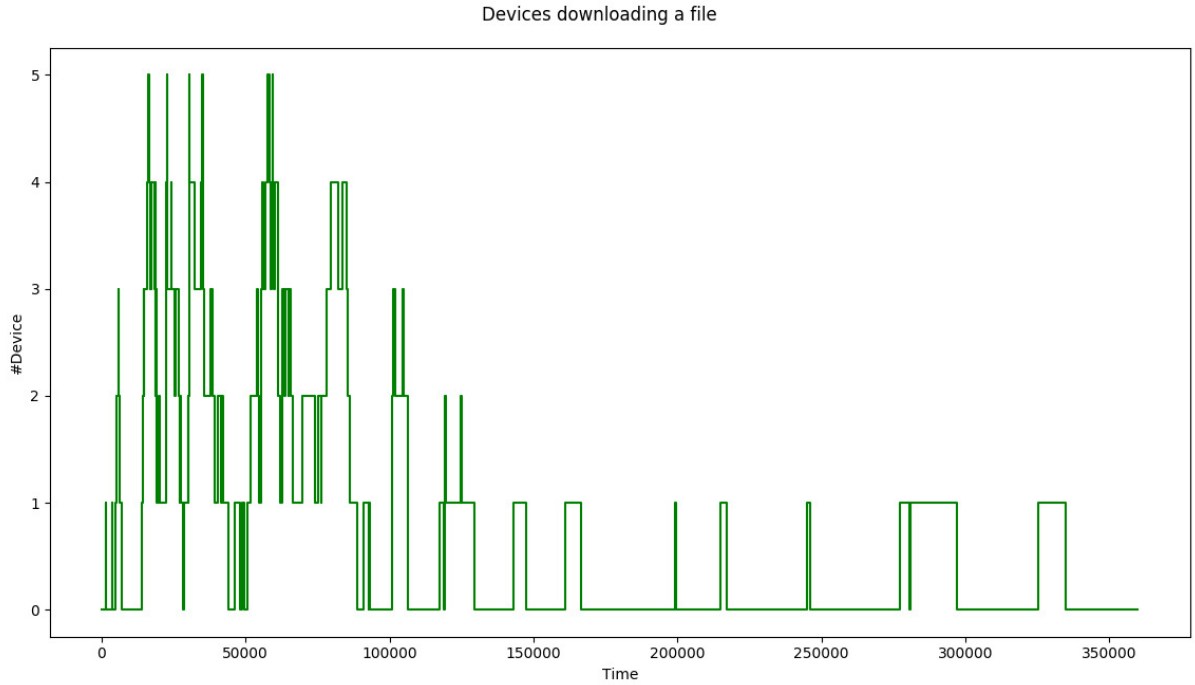
Server outgoing traffic rate (b/s)

Appendix E – Laboratory 2: Graphs of the exercise 02

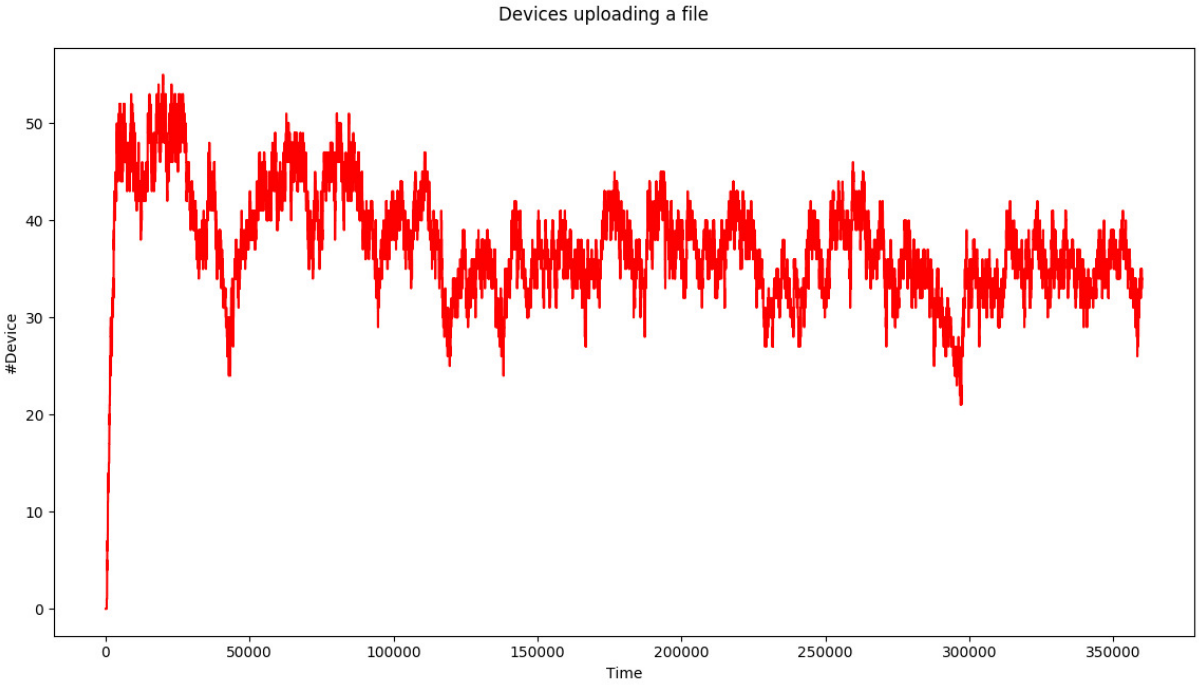
Simulation 02: 100 devices, 100 hours



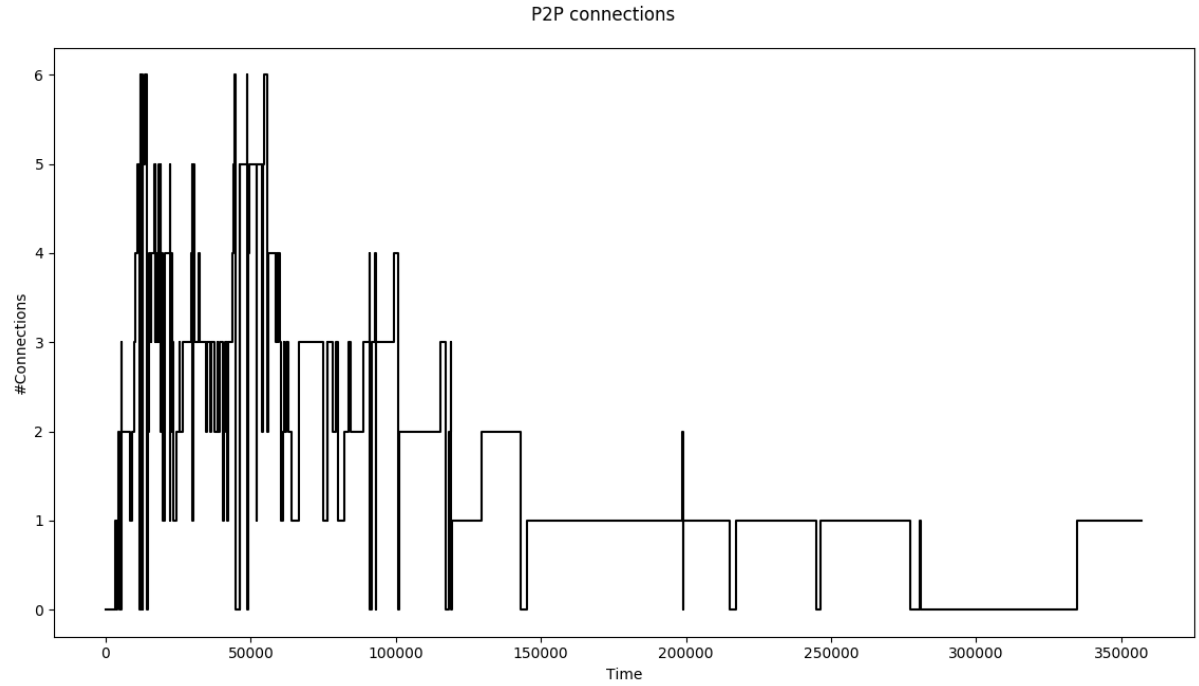
Number of online devices



Number of devices downloading a file from the server

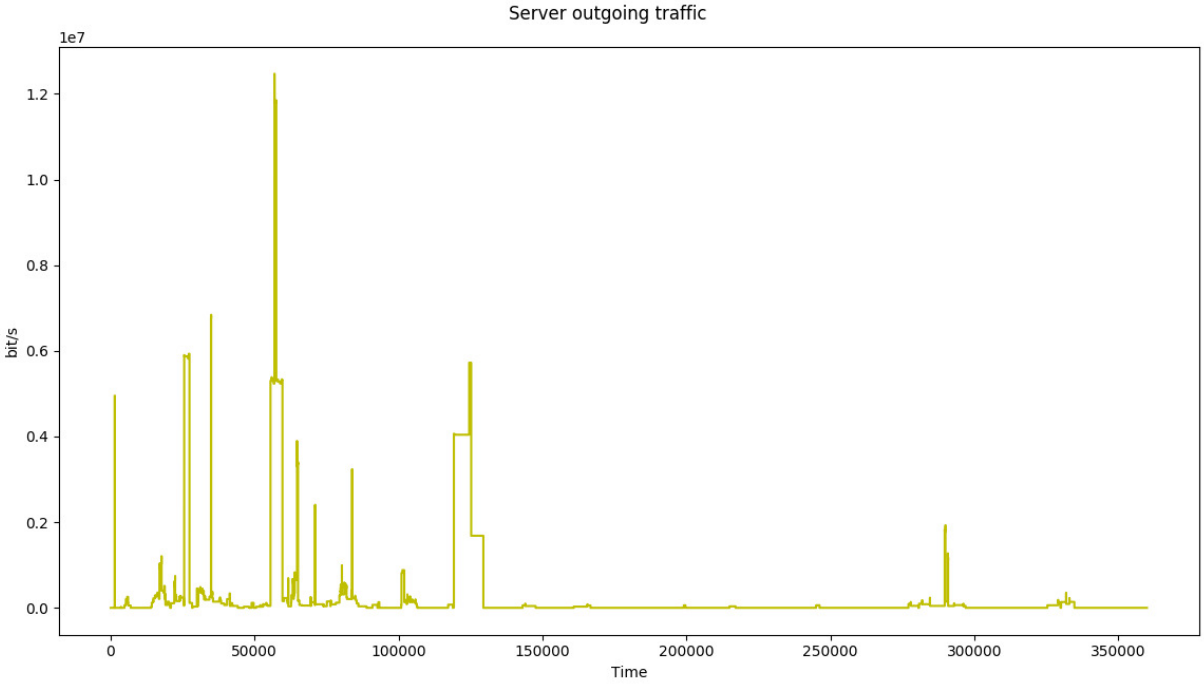


Number of devices uploading a file to the server



Number of active P2P connections

Server incoming traffic (b/s)



Server outgoing traffic (b/s)