Università degli Studi di Pisa

Peer to Peer and Blockchains

A.Y. 2019/2020

Midterm 1 Report - Monitoring the Interplanetary File System

Author:
Giulia Fois

April 16, 2020

Contents

1 Introduction

This report shows the statistics that have been elaborated on the data that have been gathered by monitoring the IPFS P2P network. In particular, some plots have been drawn from tha analysis of the swarm of peers, and others from the content of the DHT.
More in detail, in **Section 2** the libraries and technical tools that have been used to analyze the network to gather the data, to pre-process the data and to extract the plots are explained. Afterwards, in **Section 3** a variety of plots that concern the peers in the swarm is presented; the data on which the plots are based have been gathered in different spans of time, depending on the nature of the different analyses carried out. Finally, in **Section 4**, some plots related to the churn of the peers contained in the DHT buckets is shown.
The last section is dedicated to the resolution of the exercises assigned about the Kademlia DHT.

2 Technical tools

This section outlines the languages, libraries and tools used to collect, pre-process and analyze the IPFS data.
To gather the data, I used the **IPFS CLI library**[1] installed on my Windows 10 machine. I developed some simple batch scripts, whose source code can be found in the *collect_src* directory. Then, for the pre-processing phase, I wrote some bash scripts, that have been executed on the Ubuntu WSL installed on my machine. In particular, the tools that have been exploited to collect analyses from the IP addresses have been **geoiplookup**[2] to retrieve the country the IP belongs to, curl calls to **whoismyisp**[3] to retrieve the ISP an IP is associated to, and **ping**. The source code of the bash scripts can be found in the *preprocess_src* directory. Lastly, I used python to draw the statistics and elaborate the plots, by mainly exploiting the **pyplot API**[4]. Both the python source code and the Jupyter interactive notebook can be found in the *analysis_src* directory.

3 Swarm statistics

This section contains and explains the plots elaborated on the basis of the data gathered with the CLI command **ipfs swarm peers**. Different statistics have been drawn by analyzing periods of time of different durations.

3.1 Daily statistics

The following statistics have been elaborated on a period of 59 hours, in the range of time going from 07/04/2020 at 21:51 up to 10/04/2020 at 10:51. For simplicity and readability, the timestamps of each command execution have each been rounded up to the succeeding

[1]https://docs.ipfs.io/
[2]https://linux.die.net/man/1/geoiplookup
[3]https://www.whoismyisp.org/
[4]https://matplotlib.org/api/pyplot_api.html

hour, since data was gathered once per hour at *XX:51*. For example, the first swarm of peers retrieved at 21:51 in the plots is shown as related to 22.
The bar plots in **Figure 1** show how the number of peers change, hour by hour, during the days of monitoring. The first and the fourth plot contain less bars because they were not full monitoring days. Moreover, the first bar of the first plot is significantly shorter than the others because it corresponds to the first monitoring, that has been done right after the invocation of the **ipfs daemon** command. The peers in the swarm, in that moment, were only the bootstrap ones.

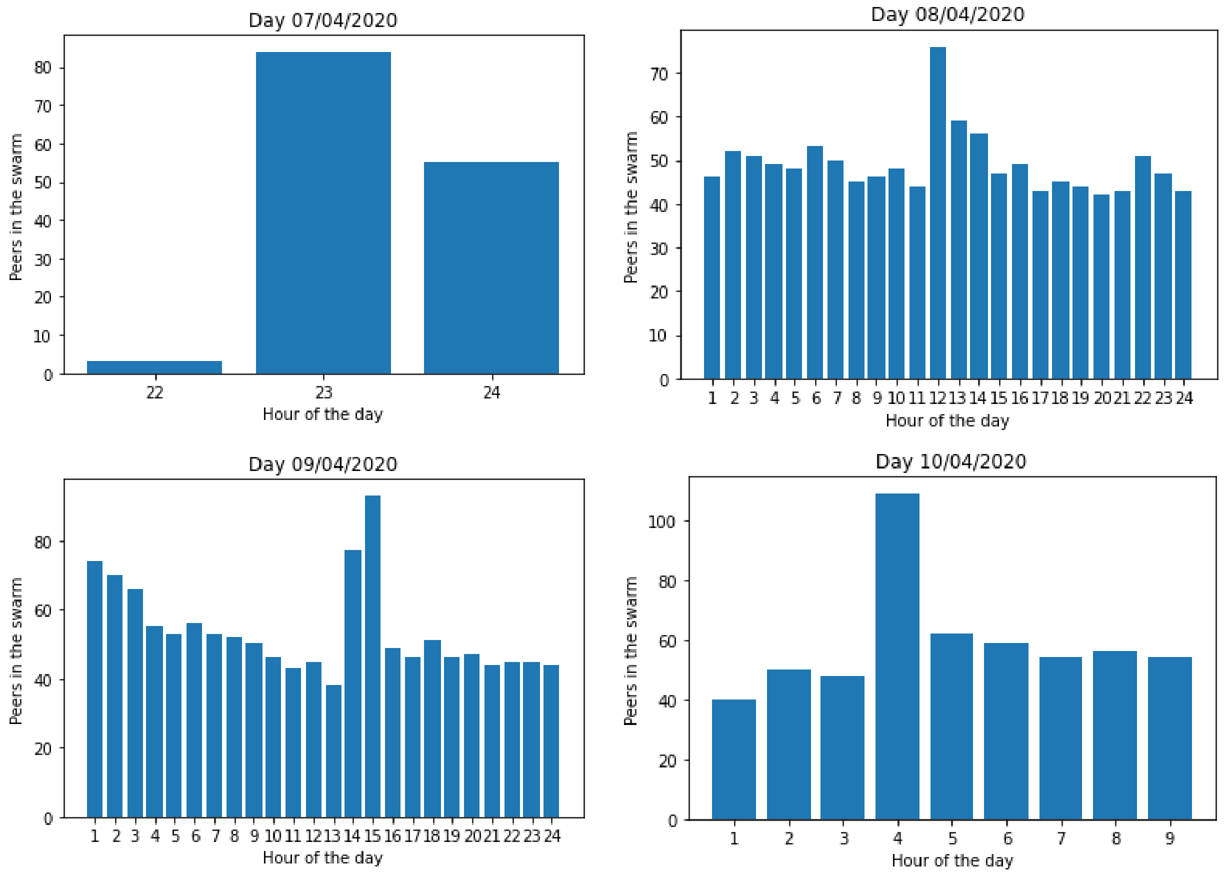

Figure 1: Number of peers in the swarm, hour by hour

Generally, we can say that the number of peers doesn't tend to decrease/increase much from one hour to the succeeding one; in some cases it happens abruptly when there are peaks (that in our monitoring happened at lunchtime of 08/04, during the afternoon of 09/04, and in the middle of the night of 10/04). Moreover, as we can see, the swarm size never goes under the threshold of around 40 peers. **Table 1** contains some aggregate measures concerning this period of monitoring. In particular, we can see how the highest peak of 109 peers is reached at 4:00 in the morning of the 10/04, and the lowest, of 38 peers, is reached at 13:00 of the 09/04. The plots show that generally the number is closer to this lowest peak rather than the highest one, in fact the average number of peers in the swarm is 52.

Day and hour with max no. of peers	10/04 4:00 - 109 peers
Day and hour with min no. of peers	09/04 13:00 - 38 peers
Avg hourly no. of peers	52 peers

Table 1: Aggregate measures for the number of peers in the hours

The number of unique peers retrieved in this period is 463. 24% of them has been connected altogether during the highest hour peak; on average 11% of all different peers retrieved are together in the swarm.
As shown more in detail in **Subsection 3.3**, the vast majority of peers come from China and United States, that are countries whose time zone is very far to the Italian one. Since no Italian peers have been retrieved in any swarm monitored, I decided to check the hourly variation in swarm size for peers coming from European countries, to see if their peak connection times could resemble what would be a standard Italian peak. Our expectation would be to find many European peers connected during the day and less in the night time.

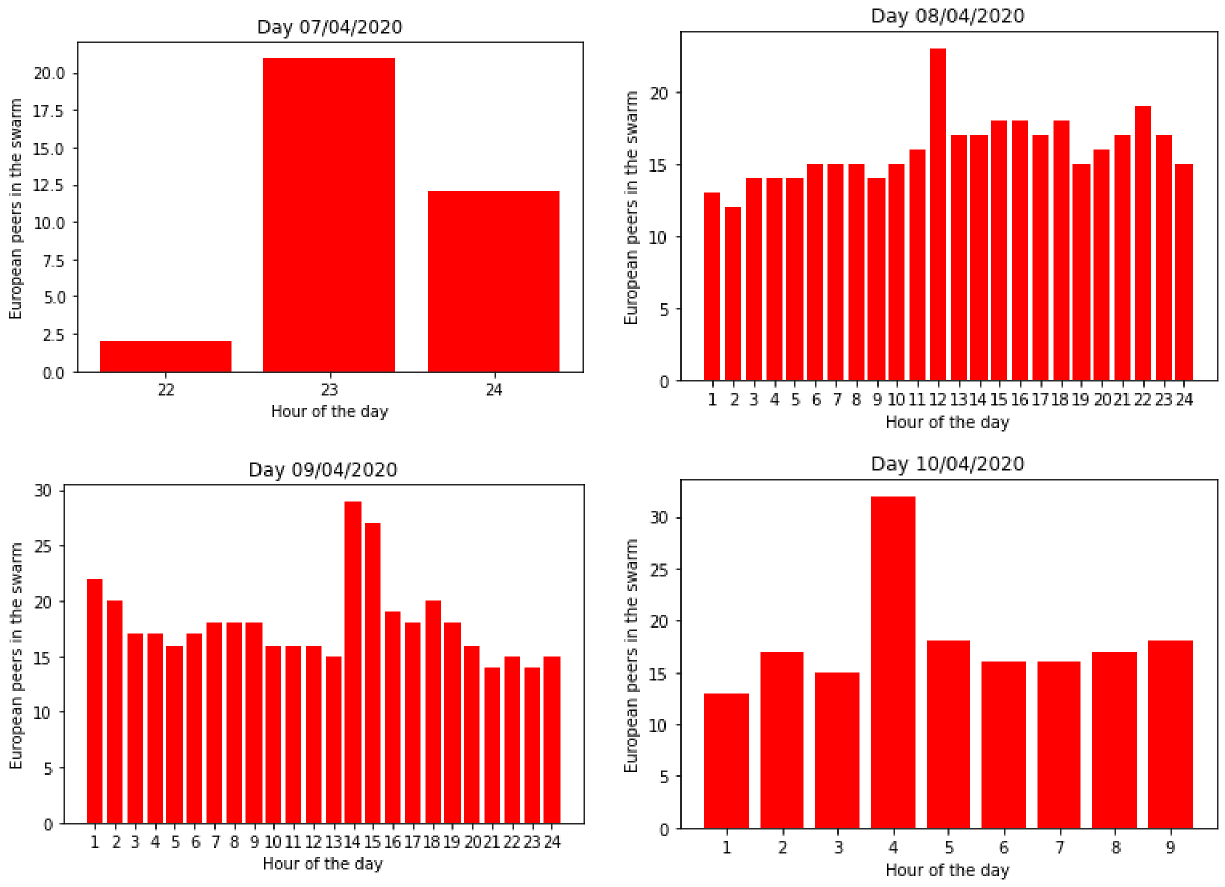

Figure 2: Number of European peers in the swarm, hour by hour

As it is possible to notice in **Figure 2**, the hourly pattern of the European peers actually follows the behaviour of all peers, without provenance distinctions. In fact, apart from the peaks (that could be unrelated to the peer provenance, given that the highest one is at 4 in the morning), there is no relevant difference between the number of peers in the daily hours and the night ones, in which statistically one would expect less network traffic. This could mean that some people may run IPFS constantly, without turning off neither the service nor their machine during the night.

3.2 Hourly statistics

The following statistics have been elaborated on the basis of data collected for 8 hours during the 13/04/2020, in the range of time going from 16:13 up to 22:53. The command **ipfs swarm peers**, in this span of time, has been executed once every 10 minutes, in order to analyze the differences, if any, of the peers flow within an hour with a more restricted interval.

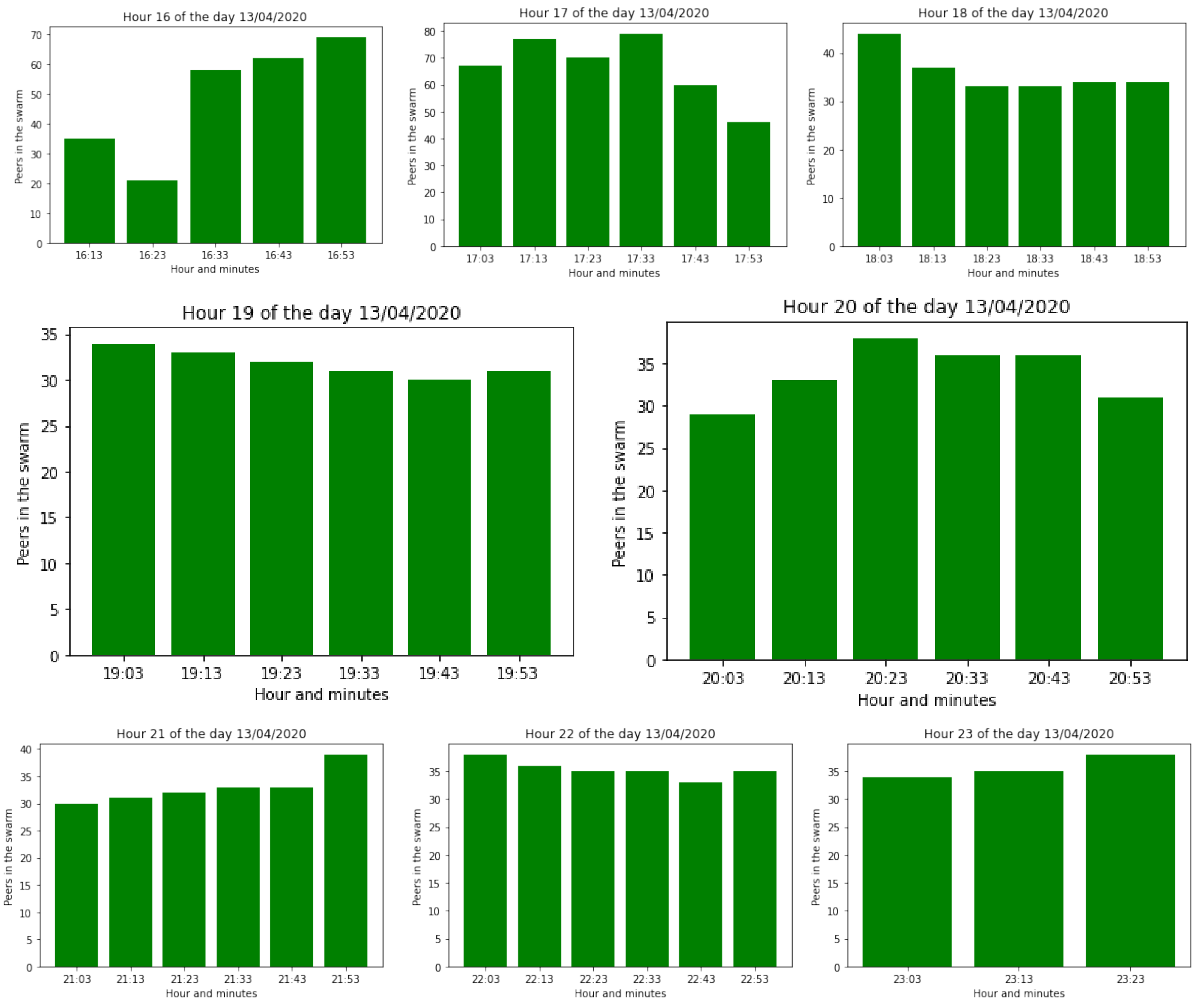

Figure 3: Number of peers in the swarm every 10 minutes

From the bar plots in **Figure 3** we can see that in the afternoon hours (particularly between 16 an 17) the swarm tends to reach higher sizes but at the same time there is more

discontinuity between different time slices of 10 minutes. On the other hand, during the late afternoon and evening, the number of peers tends to stabilize between 30 and 40, without increasing or decreasing too much within the hours.
Moreover, I decided to investigate more deeply the variation of peers in the swarm within just an hour, analyzing the size of the swarm once every minute, that is the lowest possible meaningful interval that could be used (since every command execution was followed by a file write, and this could take several seconds, making it impossible to execute the command every few seconds). The bar plot of **Figure 4** shows the minute by minute variation of peers in the swarm. The data for this plot have been gathered between 9:51 and 10:51 of 14/04/2020. The chart clearly shows the dynamicity of the swarm: for example we can see how from minute 24 to minute 25 the number of peers grows by 20, and then within just another minute it decreases of approximately the same amount.

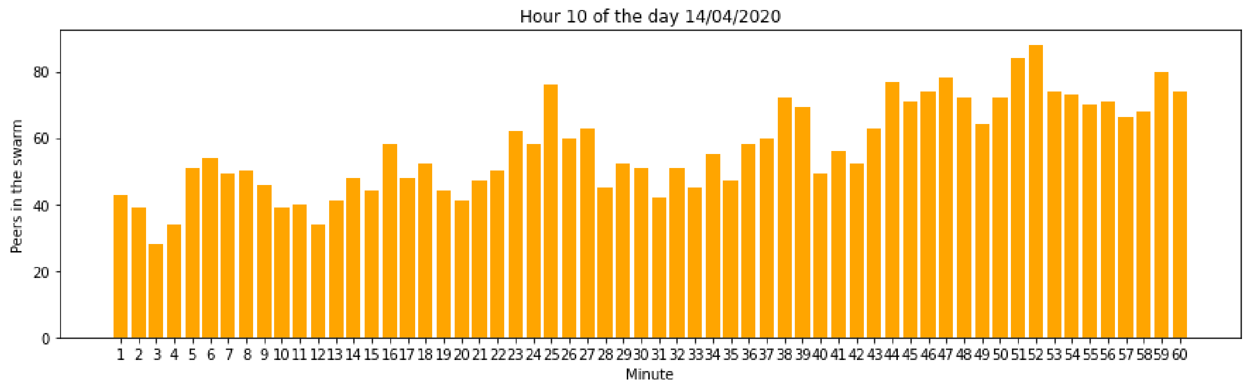

Figure 4: Evolution of the swarm, minute by minute

3.3 Country statistics

In this subsection we evaluate the frequency of the peers' provenance countries in the swarm. The peers that have been taken into consideration are the 463 unique ones collected during the 59 hours of monitoring (as mentioned in **Subsection 3.1**.

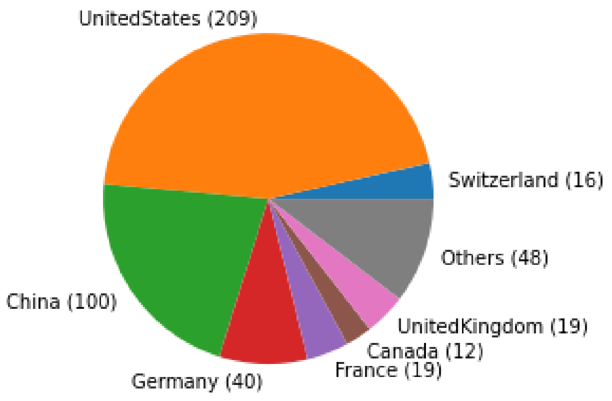

Figure 5: Frequency of each country of peer's provenance

Figure 5 depicts a pie chart, where each slice represents the percentage of peers coming from the labeled specific country. The precise number of peers is mentioned near each country's name. As clear from the chart, the majority of peers in the swarm comes from the United States, followed by China and by Germany in third position. Other countries originate less than 20 peers each. The grey slice, labelled as 'Others', contains all countries whose single contribute is of less than 2% of the total.

Country	**Number of peers**
Singapore	4
Netherlands	9
Russian Federation	6
Japan	3
India	3
Poland	2
Spain	1
Austria	2
Finland	1
Costa Rica	1
Taiwan	1
Greece	2
Lithuania	1
South Africa	1
Sweden	3
Malaysia	2
Czech Republic	2
Australia	2

Table 2: 'Others' slice in detail: number of peers for each country

Table 2 contains all the detailed number of peers in the swarm coming from the countries not shown in the pie chart of the previous figure. Combining the data of the pie chart and the data in the table, we can state that just a small percentage of peers is from Europe, Oceania or South America; most peers come from the United States or from Asian countries. During a monitoring of almost 3 days, no Italian peers have been detected. Moreover, there were 2 peers whose country could not be retrieved, since their IP address was hidden behind a proxy node (probably because of the impossibility to connect directly because of NAT configurations).

3.4 ISP statistics

In this subsection we analyze the information collected about the Internet Service Providers associated to the peers' IP addresses. As was done for the countries, the information analyzed is the one of the 59-hour monitoring phase.

ISP	Number of Peers	Cloud Provider
DigitalOcean	44	✓
PACKET	10	✓
Shenzhen Tencent	58	
Chinanet	21	
Hetzner Online	14	✓
Linode	9	✓
Choopa	50	✓
Contabo	8	✓
Amazon	31	✓
Google	75	✓
OVH	13	✓
Others	130	

Table 3: Frequent ISP of peers

Table 3 contains the names of the 11 more frequent ISP and the number of peers in the swarm that lean on them. Moreover, the third column contains a tick if the ISP of the row is a cloud provider. As we can notice, the majority of providers are cloud-related, and this means that most of the peer client nodes aren't installed directly on the machines of the users, but rather on data centers' server instances. This could be a possible explanation of the fact that, in **Subsection 3.1**, a peak of European peers was retrieved at 4 in the morning: people usually tend to turn off their personal machines during the night hours, while cloud IPFS nodes run constantly.

3.5 RTT statistics

This subsection contains some statistics obtained on the basis of the execution of the **ping** command on the IP addresses of the peers in the swarm. The reference peers are again the ones obtained by the first 59-hours long monitoring. In particular, ping has been set to be executed 5 times on each IP address, to have the most precise values without it requiring too much time. The IP addresses that did not respond to ping haven't been considered.

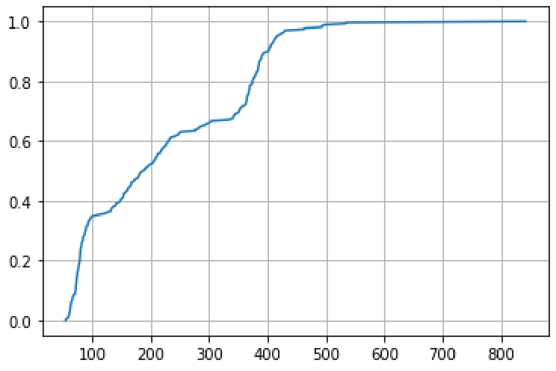

Figure 6: CDF on the RTTs

The average RTT has been extracted from each ping execution, and then the list of all RTTs has been sorted. **Figure 6** shows the cumulative distribution function applied to the list of sorted RTTs. In particular, the x-axis contains a RTT time value (expressed in ms) and the y-axis contains values in [0,1]. The aim of this plot is to show the percentage of IP addresses that, if pinged 5 times, gave an average RTT of a certain value or less. We can see for example that 50% of the IPs are associated to a RTT of 188 ms (which is the median) or less. Almost the totality of IPs are associated to a RTT of less than 800 ms.

Max RTT	842 ms
Min RTT	54 ms
Avg RTT	220 ms
Median RTT	188 ms

Table 4: Aggregate measures for RTT

Table 4 shows some aggregate measures for the RTTs retrieved. As can be also seen from the CDF plot, the maximum RTT slightly exceeds 800 ms, while the minimum is a little above 50 ms. While the median is of 188 ms, the average is of 220 ms: this means that even though half of the IPs have RTTs for sure over the median, "on the right side" of the plot, the average is lowered by the very low RTTs "on the left side": almost 40% of the RTTs are indeed under 100 ms.

4 Churn statistics

In this section some statistics about the level of churn of the peers are shown. More specifically, the content of the DHT has been investigated through the periodic execution of the command **ipfs dht query**. Three sets of data gathering have been carried out by three different processes: the first executed the command once every 10 seconds, the second once every minute, and finally the third once every 5 minutes. All these executions have been conducted within an hour, on 13/04/2020 between 11:37 and 12:37.
The level of churn has been investigated by simply intersecting the set of peers returned by a query with the set of peers returned by the previous one, to see how many nodes remained in the DHT after the given interval of time. The size of the common peers set has been then compared to the actual number of peers in the DHT buckets. Since each query returned some peers more than once, before computing the intersection the duplicates have of course been removed.
Each of the following charts contains two layered bar plots: in particular, the blue one represents the number of peers retrieved by each query execution, while the red one represents the number of peers in common with the result of the previous query. The x-axis represents the flow of time: each bar corresponds to the query at a given time, and distances the previous bar by the specific interval of seconds of that round of executions.

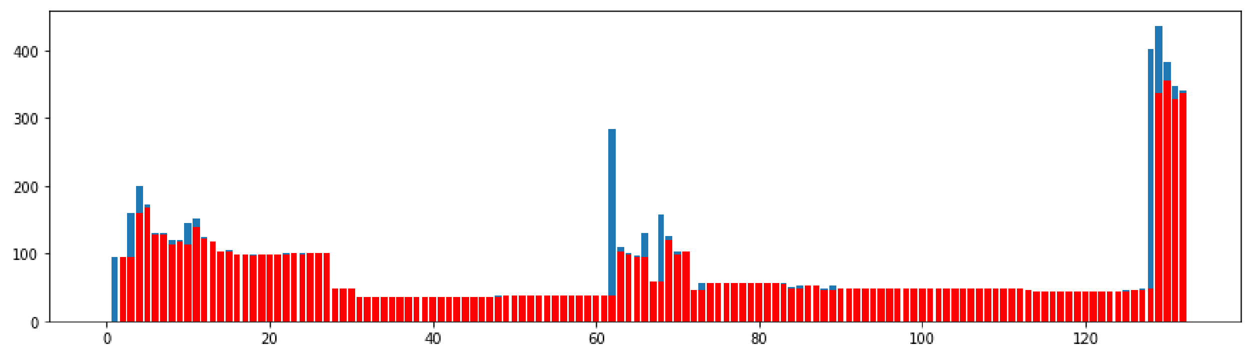

Figure 7: Peers in common between results of queries executed each 10 s

Figure 7 is related to the set of queries executed once every 10 seconds. We can see that for most queries the height difference between the two differently colored bars is almost nil: this shows how the level of churn, if analyzed every 10 seconds, for the most part of the analysis is not high. The parts where the difference is more evident are related to a sudden increase of peers in the DHT, but after little time we see that it becomes stable again.

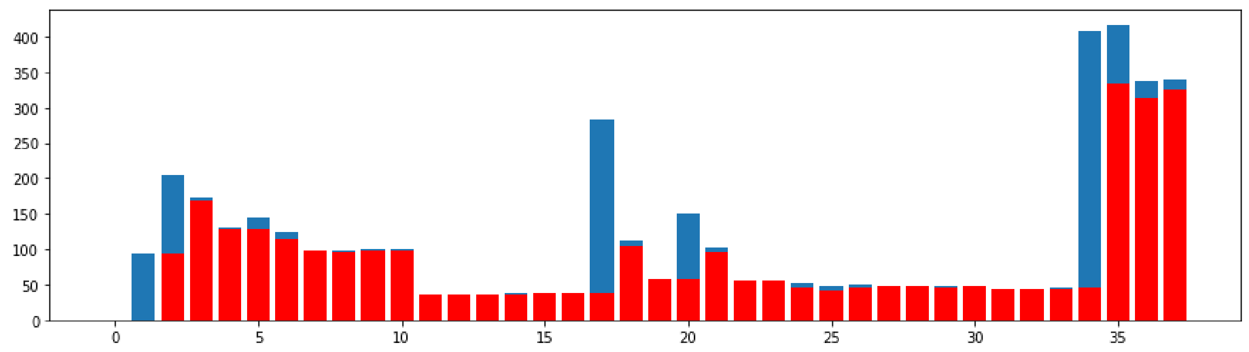

Figure 8: Peers in common between results of queries executed every 60 s

Figure 8 contains the results of the queries executed once every 60 s. The red bars show how the number of peers in common between each query execution is again very high, and constitutes a good percentage of the total number of peers. The situation doesn't change much even when the queries are executed once every 5 minutes, as depicted in **Figure 9**: the only differences seem to be noticeable when the number of peers has an abrupt increment, as happens at the 10th execution.
Given these plots, the peers didn't seem to have a high level of dynamicity in terms of abandoning the network, but there seem to be situations in which many peers join the network and are added to the DHT altogether.

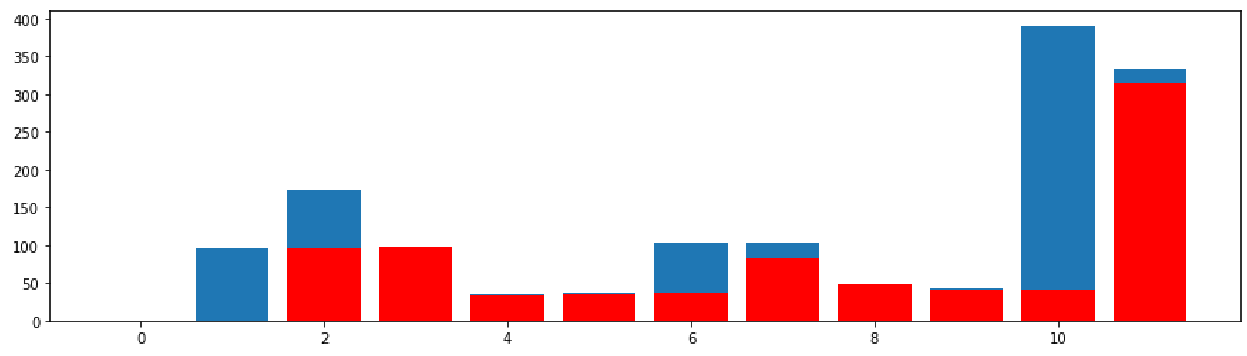

Figure 9: Peers in common between results of queries executed every 300 s

5 The Kademlia DHT: exercise

In this section I explain step by step the reasoning applied to solve the two exercises, based on the DHT illustrated in the midterm text.

5.1 Question 1

For simplicity, I assign letters to the node identifiers I mention in the solution. Let X be the node that inserts the content (X = 11111101), A = 00001101, B = 00110011, C = 01001101, D = 01101000 and E = 01111001. Let's also assume that the routing tables of the just mentioned nodes contain the following information:

Common Prefix	First Node	Second Node
0	0001101	00110011
1	10010001	10110001
2	11000010	-
3	11100101	-

Table 5: Routing table of X

Common Prefix	First Node	Second Node
0	11100101	11000010
1	01001101	01101000
2	00110011	-
3	00011001	-

Table 6: Routing table of A

Common Prefix	First Node	Second Node
0	10000001	10010001
1	01001101	01111001
2	00001101	00001101
3	-	-

Table 7: Routing table of B

Common Prefix	First Node	Second Node
0	11111101	10000001
1	00001101	00110011
2	01101000	01111001
3	-	-

Table 8: Routing table of C

Common Prefix	First Node	Second Node
0	11100101	11000010
1	00011001	00110011
2	01001101	-
3	01111001	-

Table 9: Routing table of D

Now, let's see which steps are taken by X to find the node closest to the key K (K = 01000001) in order to store it.
X starts its lookup by sending simultaneous queries to the $\alpha = 2$ nodes in its routing table that are closer to K. Those two nodes are A and B, contained in the two buckets of the entry 0 of X's routing table. **closestNode** is set to A, because by applying the XOR metrics A results closer to K with respect to B.
Since the query is iterative, A and B in turn look for the 2 closest IDs in their routing tables and send them back to X. A sends the IDs C and D, contained in the entry 1 of its routing table, while B sends back C and E, contained in the entry 1 of its routing table.
X thus receives 3 identifiers: C, D, E, that are inserted in its *k-buckets list* sorted by XOR distance to K: first C, then D, then E. **closestNode** is updated to C.
X then sends the query to C and D. C and D respectively find the closest IDs to K they have in their table and send them back to X. C sends D and E, contained in the entry 2 of its routing table, while D sends C contained in the entry 2 of its routing table and E contained in the entry 3 of its routing table.
X then stops, because none of the IDs that have been sent back are closer to K with respect to **closestNode**, which is C. The key K is thus stored on C = 01001101.

5.2 Question 2

For simplicity, I assign letters to the node identifiers I mention in the solution. Let J = 11010101 be the joining node, and B = 00110011 the bootstrap node. Moreover, let A = 10000001, and C = 10010001.
At the beginning, the routing table of J is composed only by the bootstrap node.

Common Prefix	First Node	Second Node
0	00110011	-
1	-	-
2	-	-
3	-	-

Table 10: Routing table of J - step 0

J then sends a *FIND_NODE(J)* request to B. Let's assume that the routing table of B is the following:

Common Prefix	First Node	Second Node
0	10000001	10010001
1	01001101	01111001
2	00001101	00001101
3	-	-

Table 11: Routing table of B

B sends back the $\alpha = 2$ closest IDs to J it has in its routing table, that are A and C. Those are found in the buckets of the entry 0.
J then updates its routing table as follows:

Common Prefix	First Node	Second Node
0	00110011	-
1	10000001	10010001
2	-	-
3	-	-

Table 12: Routing table of J - step 1

In order to fill out the buckets with indexes 2 and 3, J generates random IDs having respectively 2 bits and 3 bits of common prefix to itself.
Let R1 = 11000100 be the random identifier that has 2 bits in common with J. J sends *FIND_NODE(R1)* to the closest nodes to R1 in its routing table, that are A and C.
Let A and C's routing tables be the following:

Common Prefix	First Node	Second Node
0	00001101	00110011
1	11111101	11100101
2	10110001	10101111
3	10010001	-

Table 13: Routing table of A

Common Prefix	First Node	Second Node
0	00001101	00011001
1	11000010	11100101
2	10101111	10110001
3	10000001	-

Table 14: Routing table of C

A sends back 11111101 and 11100101, while C sends back 11000010 and 11100101. J then proceeds in filling out its routing table:

Common Prefix	First Node	Second Node
0	00110011	-
1	10000001	10010001
2	11111101	11100101
3	11000010	-

Table 15: Routing table of J - step 2

J has now at least one node ID per routing table entry. At the same time, the other nodes have become aware of J's presence in the network and have inserted it in their routing tables. J will continue filling out its routing table through its activity on the network.