Peer to Peer Systems and Blockchains
Academic Year 2018/2019

Midterm assignment

Analysing the Kademlia DHT

Report

Leonardo Frioli
580611

Index

Introduction

The aim of this report is to explain the main design choices done during the assignment's implementation and to show the results of the topology analisys of the obtained graph.

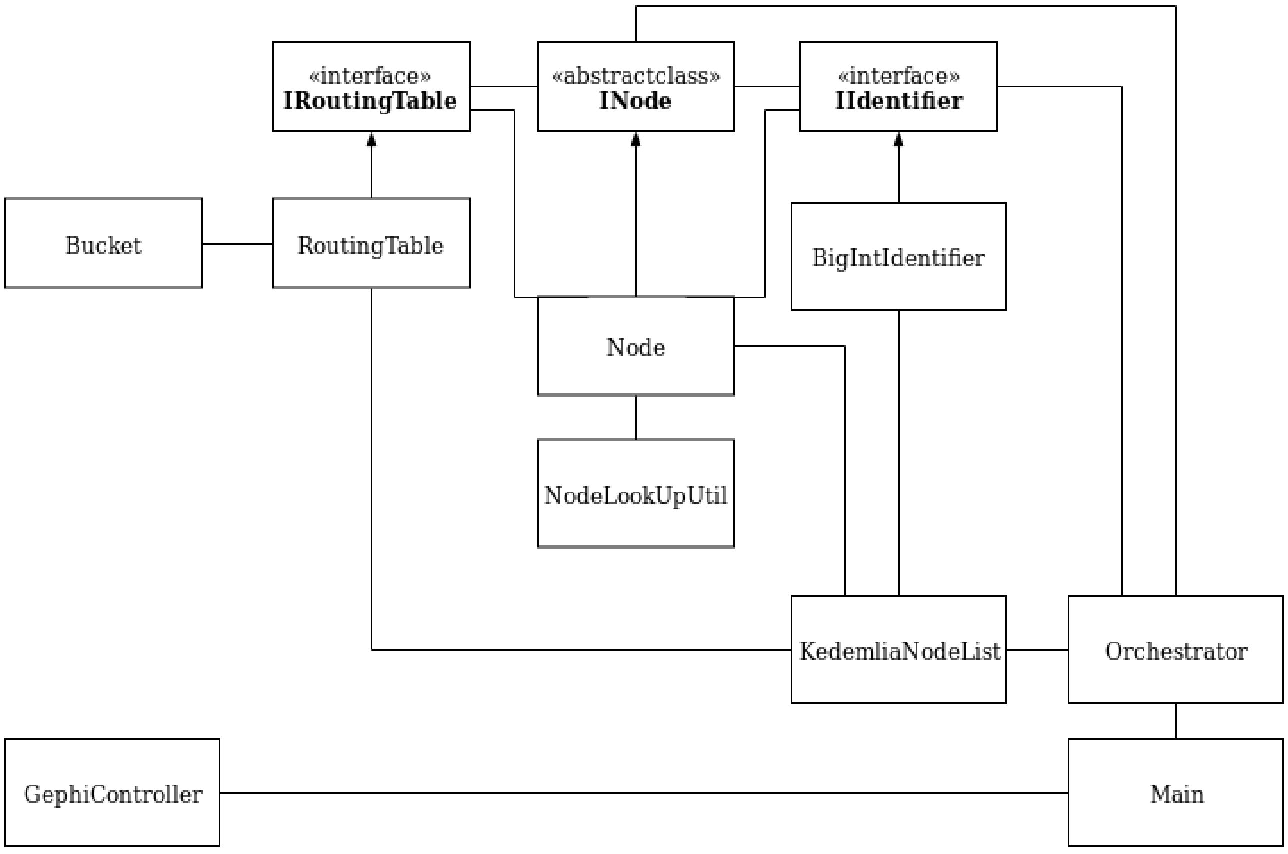
We will describe the general structure of the code, going in depth discussing some design choices like the identifiers' generation and the routing table construction during the bootstrap phase.

In the second part of the report we will discuss the code for the topology analisys and the obtained data.

Design Choices

The code has been written in Java using NetBeans as IDE. The implementation is made of two macro sections: the first section constructs the Kademlia structure and enriches the routing tables of the added nodes; the second one performs the graph analisys.
In this chapter we will talk about the first part because is the more important, the second part of the code will be discussed later.

We tried to write a code as clear as possible. Each class has its own well defined role and responsibility. The use of some Interfaces allows to avoid strong dependencies and reduces logic errors due to strange interactions.



The above figures shows with a UML-like diagram the relations between the classes. It’s useless to describe each class implementation. It’s worthy instead to just discuss the main choices done during the coding of the BigIntIdentifier class and the NodeLookUpUtil.
Before talking about this two classes, it’s important to mention the PING method in the Node class. This method return always true, because we are in a non-distributed environment and there is no churn. With this implementation, once a bucket is full there is no chance that a new node is inserted in it, because noone of the nodes can crash (there isn’t need of an algorith to maintain consistent the state of the network).
We can claim that, if we do not consider the above fact, the remaining implementation reflects that one explained in the Kademlia’s paper and slides.

Identifiers

The identifiers are managed in the whole program through an interface so that the entire code is note dependent on their real implementation. If we want to modify the identifiers implementation, this can be done changing just few lines of code without distrupting the semantic of the program.

The class BigIntIdentifier holds a BigInteger to rapresent the Node id. The Type BigInteger is really essential because it allows to represent integer of big dimensions (longs and doubles are not good for this purpose).

The identifier constructor takes directly a BigInteger that is randomly generated using a constructor of the BigInteger class which takes a Random object.
We choose to assign the ids in that way because using a hashing function doesn’t make sense in this particular case. In the distributed version of Kademlia, as the paper says, they use 160 bit id gerated with SHA1 applied to some node informations like IP adress and port. In our case, since everything is local, doesn’t make sense to first generate some random information and then hashing it. It’s really more simple and elegant to just get a random BigInteger.
We are not saying that the hashing function and the random generator function have the same properties, they are really different indeed, but in our case using the first one is not the best choice. Lets suppose that we randomly generate an IP and port number and then we hash them to get a valid id. If, in a second moment, our random function generate a collision we will have the same collision in the hashed identifier.
The proposed implementation conforms better to the specifications. We are required to test the program with different values of m. If we choose to use a hash we will be obliged to use fixed size id (160,256,512) because trunking an hash will easily produce lot of collisions.

In conclusion, we decided to take a random BigInteger and to keep a list of the already choosen ids. If we get a collision (the probability is small if the random function guarantees a uniform output distribution), we just retry the random generation.

FindNode e LookUp

As required the FIND_NODE function keeps track of the traversed nodes. The method takes as input the id to find and a list of nodes. The FindNode receiver will add to its table the node traversed by the method and will enqueue itself in that list. The FINDO_NODE returns both the list of the node close to the id and the list of the traversed nodes. It is the duty of the functions calling the FIND_NODE to pass the list of traversed node to the calls.

Also the LOOK_UP procedure has been implemented. The coding of this function wasn't explicitly required in the assignment. Unfortunately there had been a misunderstanding that has lead to the implementation of this function. After a discussion with the professor we decide to keep it anyway because the aim of the point 2b in the specification was to implement a function to populate the routing tables of the new node.
The proposed LookUp does it perfectly, because, every time a FIND_NODE is called inside, the list of traversed node is kept consistent and added to the routing table of the node executing the function (which is the new node). The proposed LookUp tries every time to add the new discovered nodes to the routing table of the caller, so that the just insert node can join the network.

The LookUp has been implemented according to the specifications in the kademlia's paper and slides. Since the assignment required to use this function to populate the routing table, we inserted in the code some calls to a procedure that adds the discovered nodes.

Analisys of the Topology

The modules discussed, beside creating the nodes and populating the routing table, write on a .csv file every time a node is linked to another one. This file will be used to build the graph exploiting Gephi-toolkit. All the analisys of the topology is managed by the GephiController class. The main idea was to automate as much as possible the graph generation and the computation of the various statistics so that it's easy to perform more experiments.
The Main class makes the directory 'statistics/' in which the file.csv will be saved (it is actually saved in a subdirectory). This is the direcoty where the images and the statistics will go once they are computed. For each parameters combination there is a directory containing the results. If we do more than one experiment with the same parameters a specific direcotry (called 'run') will be created to allow different results.
We create also a file 'allstats.csv' that contains all the results of all the different runs, this file can be used to generate charts. A small python script is also provided. The script allows to execute automatically all the experiments trying with diffetrent combinations of parameters.

Gephi and Gephi-toolkit

To better perfom the experiments, we used the .jar library provided by Gephi. This library allows to use all the functionalities provided by Gephi 's GUI just writing a Java program. The library is used to load the .csv file, compute the statistics (average degree, clustering coefficient etc..) and save the image of the graph. We decided to used the library because it avoids to compute the result via UI with Gephi, so it's easier and faster to perform lots of experiments.

Results and Charts

Dataset

Thanks to the python script it has been possible to do a lot of experiments (151 to be precise). The choice of data has been made seeking a compromise between the necesity of covering an acceptable range of values and the constraints given by the computer on which we performed the expriments.
As we can see from the script the parameters' values have been choosen in this ranges:

k in {2, 8, 20}
n in {250, 500, 1000, 2000, 3000}
m in {16, 32, 64}

We tried all the possible combination of this values, and for each of the combination three runs have been executed to be more reliable.

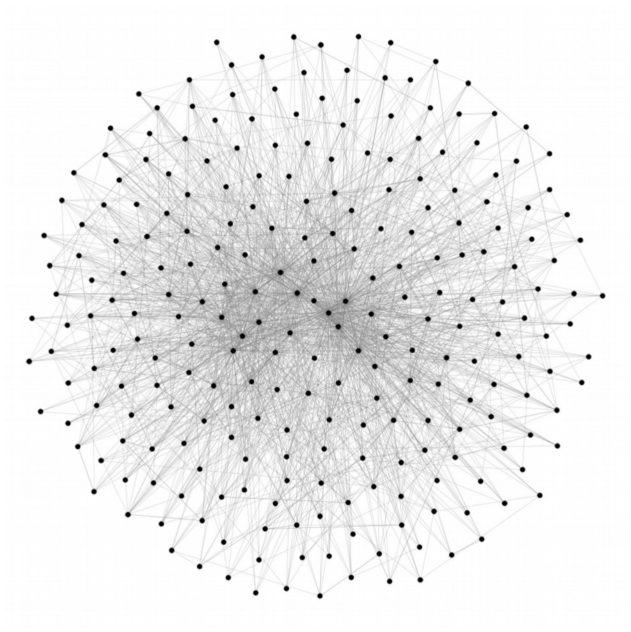
We chose k both to try very low values and to experiment with more realistic ones.
The value of n has been influenced by the environment: previous experiments showed that high values of n can bring to a long computing time (about 10 minutes); so we preferred to perform more experimets with medium-low values.
Anticipating the results, we can say that the values of m seem not to influence a lot the trends of the various metrics. If, in the charts showed below, some values of m are missing this means that the trend described is confirmed also by the missing value.

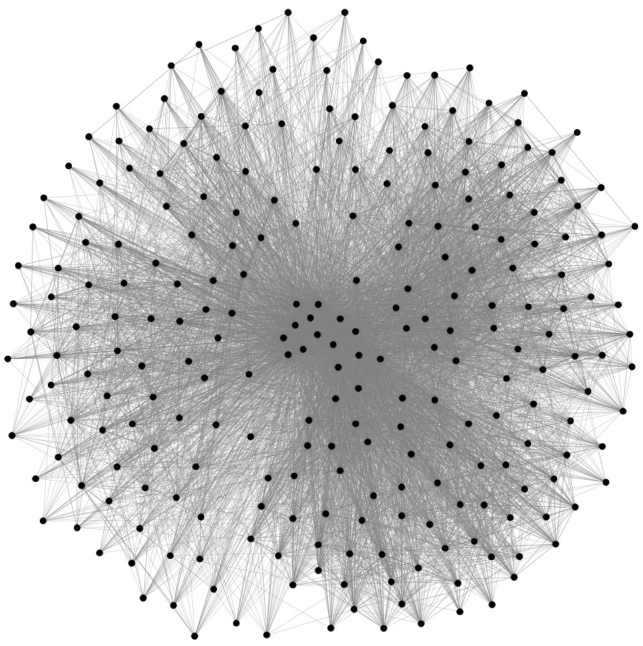
At the end we tried also some experiments with more realistic values such as m = 160 and k = 20, the settings recommended for a real implementation of Kademlia.
We performed a last experiment with values: n = 10000, m = 160, k = 20 .

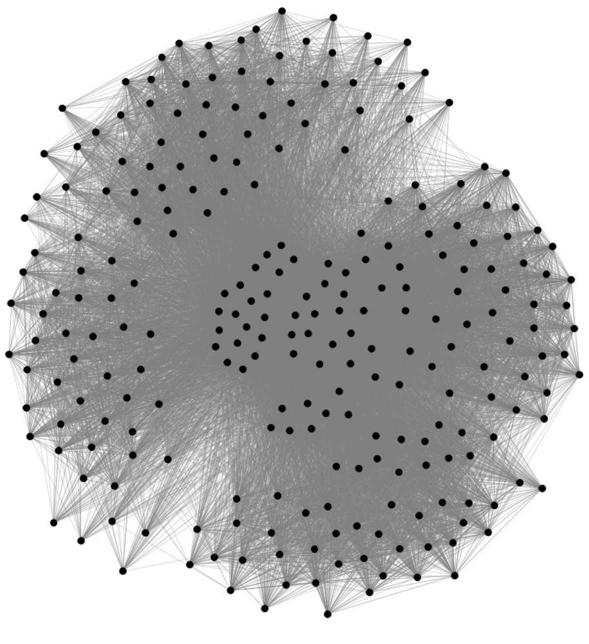
The value of the alpha parameter in the LookUp procedure has been fixed to 3. Since wasn't explicitly required to implement a LookUp, we decided not to make the things more complicated adding another variable.

The file 'allstats.csv' with all the results obtained in the 151 different runs will be attached to this report.

Graph Topology

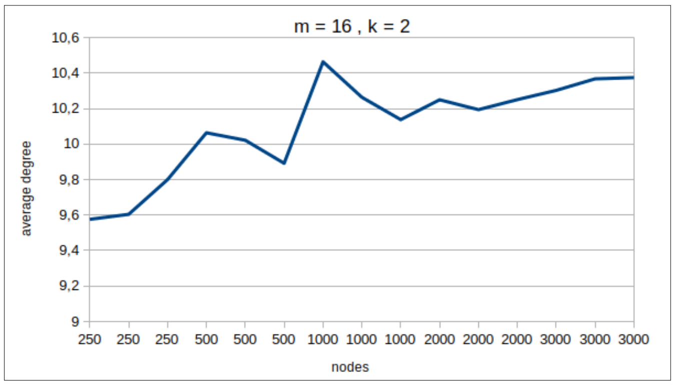

k = 2

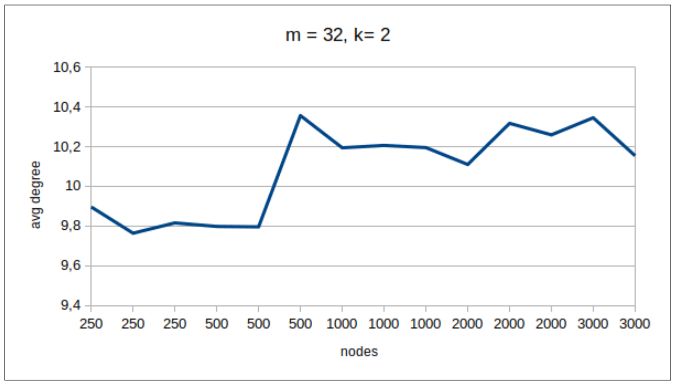

k = 8

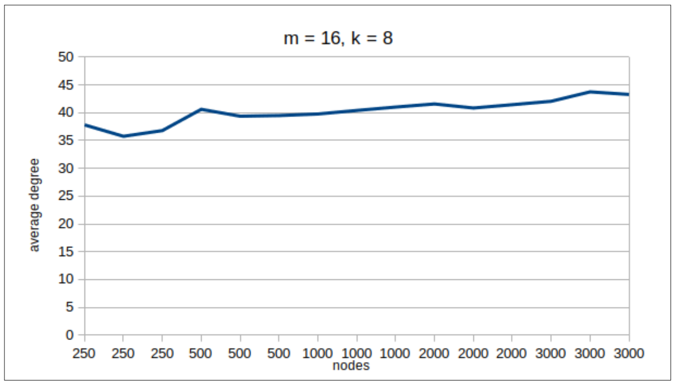

k = 20

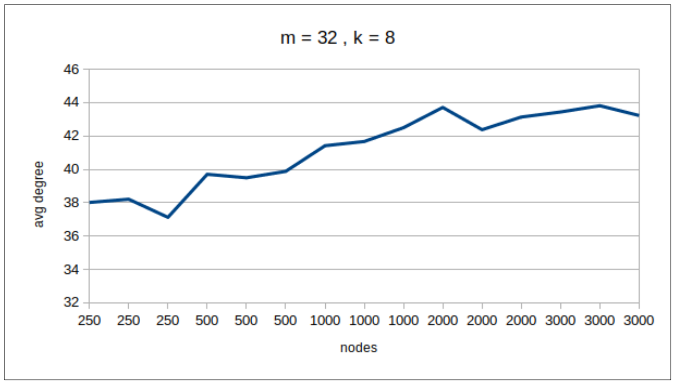
Here we display some of the obtained graphs. All the three images represent a netwotk with n=250 and m = 16. The edeges density is given by the different values of k.

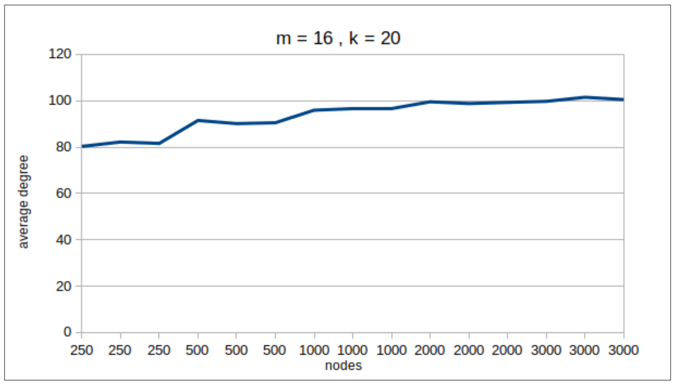
Average Degree

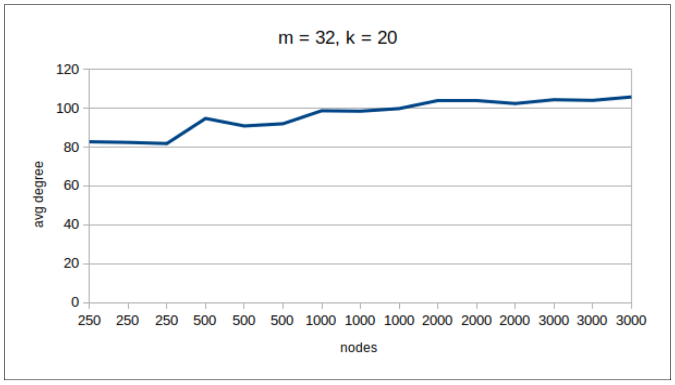






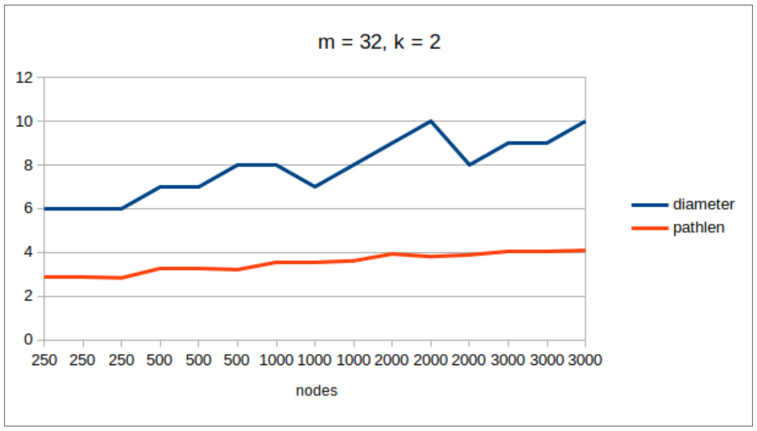


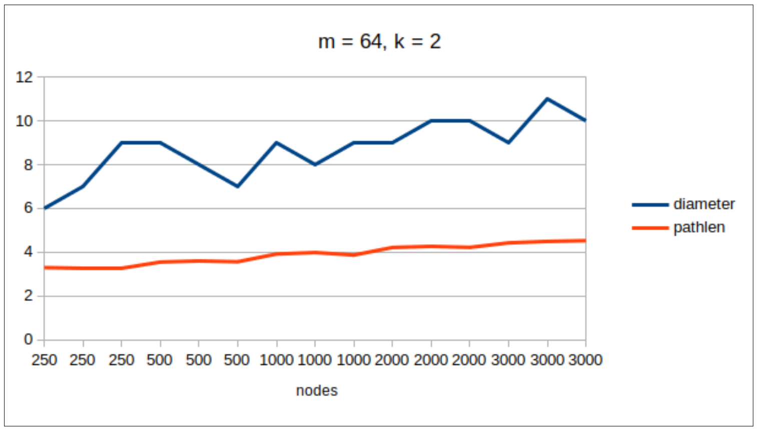


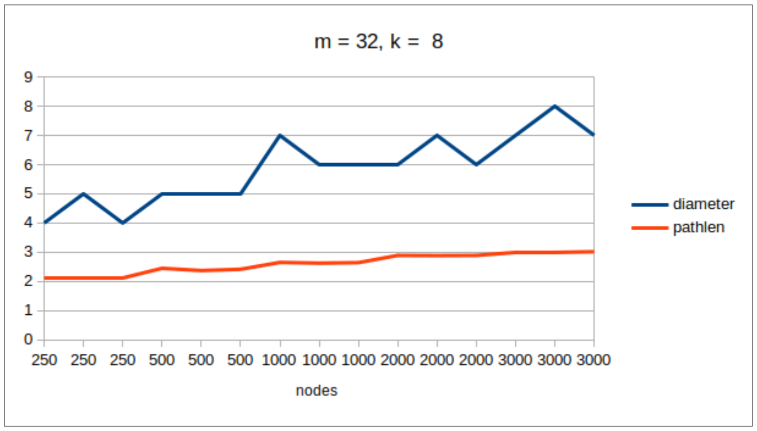


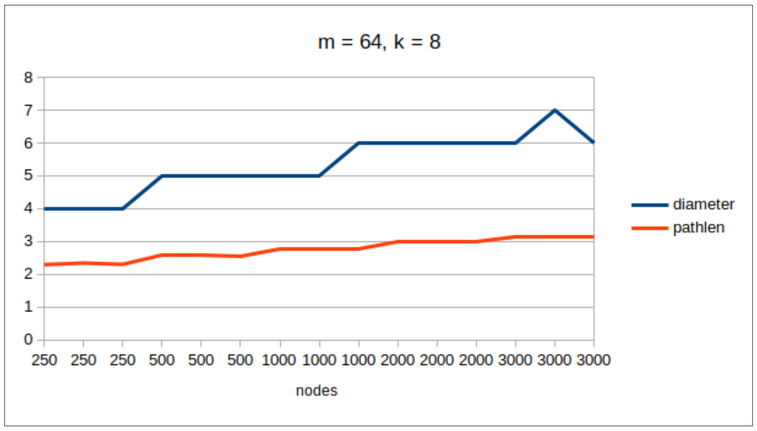
We can see from the charts that increasing n and k also the average degree increases. This trend makes sense because we expected that giving more capacity to the buckets also the nodes inserted in the routing table will increase.
We can see this fact better when we switch from k = 8 to k = 20 : doubling the bucket dimension leads to a doubling of the node degree (from 40-45 to 85-95).

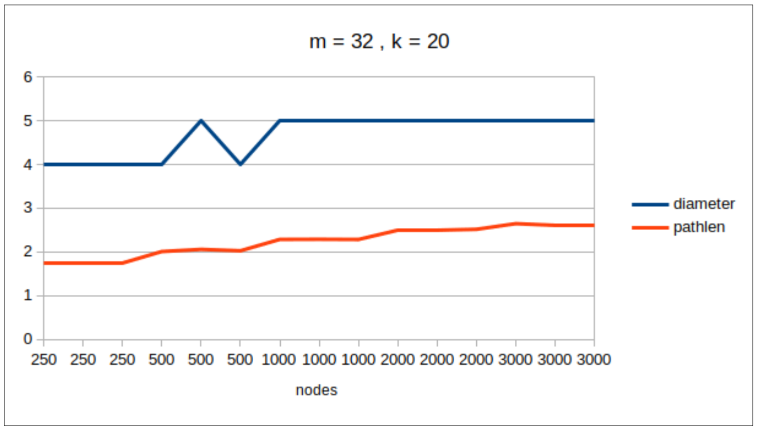
Diameter and Average Path length

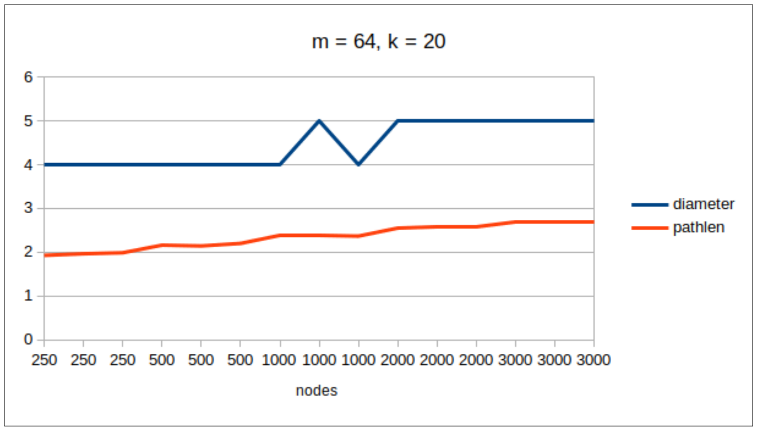








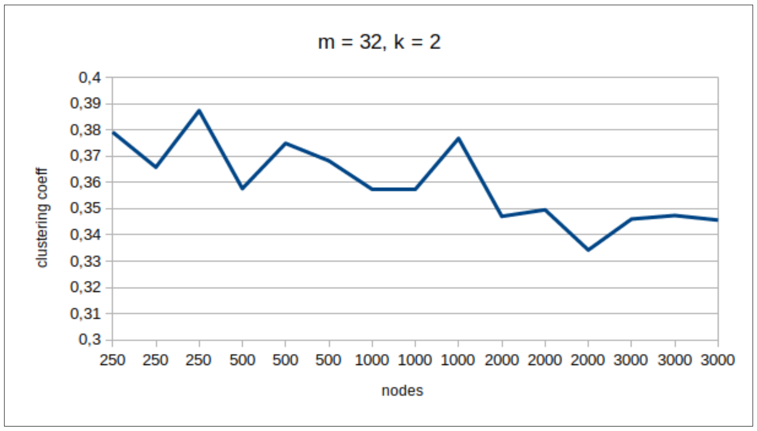


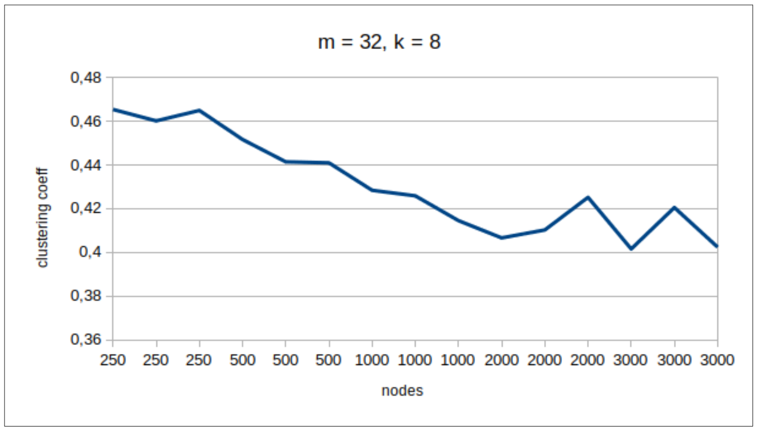


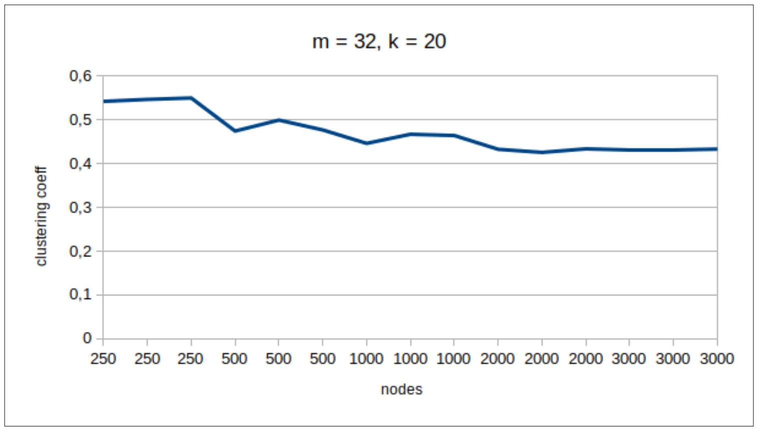
Diameter and Avergae Path legth have been displeyed together because they are strictly related. Looking at the charts we can say two things:

1. If n increases also the diameter and the path length increase. This fact is reasonable because if we have more nodes in the network problably it will take more hops to reach one node from another one. It's good to notice that the increase is linear.
2. Indreasing k brings to diameter and path length decreasing. This fenomenon is due to the previous results about the degree. Higher values of k allow a node to know more nodes in the network reducing the distance between them.

Clustering Coefficient





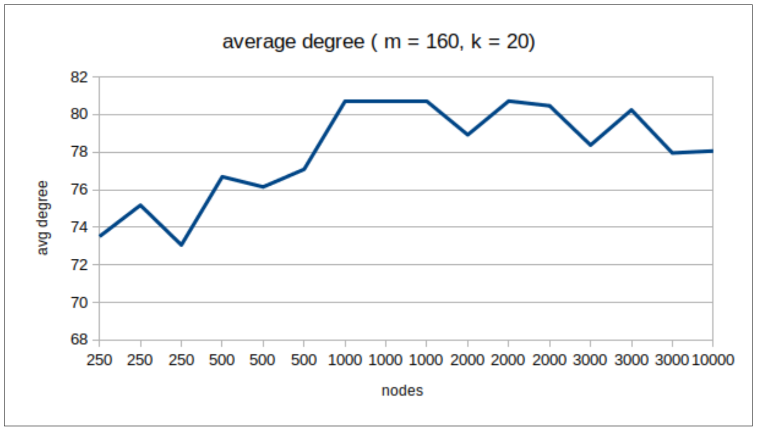


Also for the clustering coefficient is possible to make considerations like the ones done for the diameter.
The greater k , the greater the clustering coefficient because of the bucket size and the number of nodes in the routing tables.
Considering n, we can see that if n increases the clustering coefficient seems to slightly decrease. This shouldn't be strange because we could expect a smaller clustering coefficient with a larger number of nodes.

It's necessary also to compare this chart with that ones concerning the diameter and the path length. Again we can notice that k has a strong influence: if we choose k in the right way it's possible to have high values of clustering coefficient and low values of diameter and path length.

More experiments

Now we will analize the experimets done with m = 160 and k = 20. Also the experiment with n=10000 is considered.



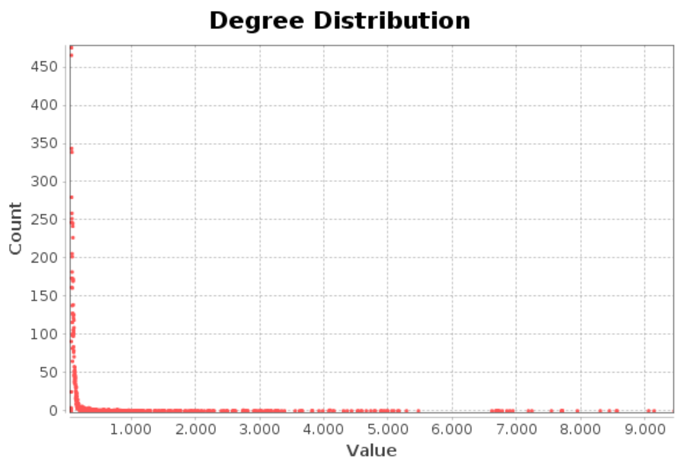
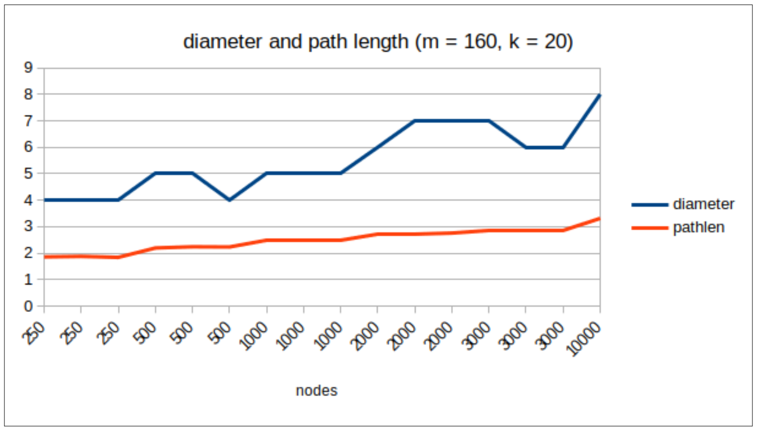
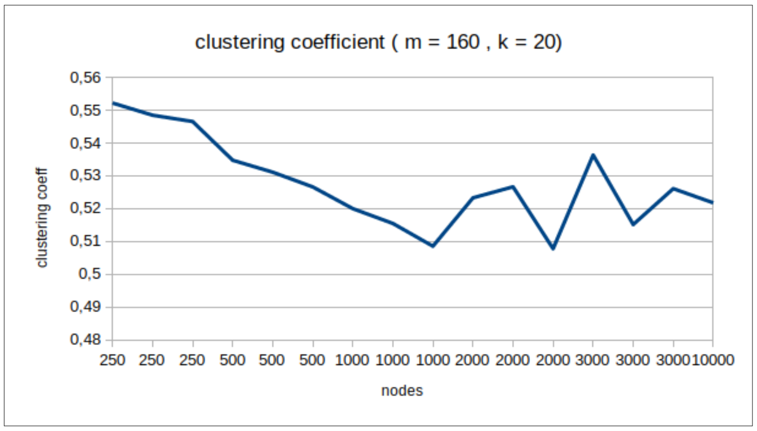


Figure 1 : degree distribution n = 10000





Also with more realistic values of m and k we obtain the same results previously described.

Let’s discuss the *Figure 1* which represents the degree distribution obtained with n = 10000. Looking at the figure we can say that the network is made of few nodes with a really high degree and more nodes with a smaller number of edges (around 78). This trend can’t be taken as a general rule because is the result of a single experiment and it may not reflect the average case.

Conclusions

We can clonclude saying that the assignment has been implemented keeping in mind two main objectives:

- make the code clean, easy to understand and easily changeable reducing strong dependecies.
- try to automate as much as possible the computation of the statistics so that we can try more experiments to get better results.

The analisys of the topology has shown that the choice of k seems to have the most important role in the shaping of the network. Chossing values of k around 20 can bring to good values of degree, diameter, path length and clustering coefficient.
From the performed experiments we didn’t notice a strong impact of the value of m which seems not affecting too much the construction of the graph.

Future Developments

One possible future development can be to perform experimetns with higher values of n to see if the trends described here are confirmed also in more realistic situations. To achieve this goal it’s necessary to use a computer with more computational power or to parallelize the code.