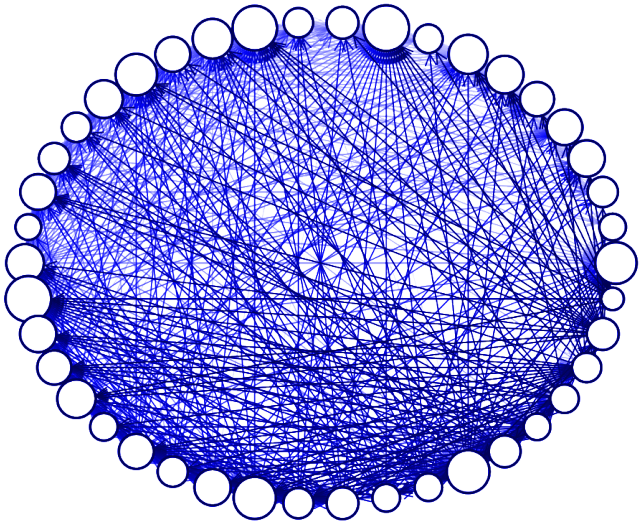
Kademlia DHT
An analysis of the network



Peer To Peer Systems
And Blockchains

a.y. 2018-2019

Contents

1 Introduction

Kademlia is a *redundant* , *iterative* , *prefix based* DHT that can run in parallel and is based on UDP. Such an algorithm is exploited in P2P systems to provide an efficient strategy to map files to peers and allow a fast retrieval of such files by any peer. As any other DHT, Kademlia is self organizing and offers a good scalability. In particular, each file is hashed into a key, made of m digits, just like any peer in the network. For the sake of clarity we refer to hashes of file as *key* and to the hashes of peer as *ids* . Each key in the network is assigned to the id it shares the longest prefix with. The intuition behind the algorithm used to retrieve keys is to discover iteratively ids which share more and more "prefix-bits" (from left to right) with the target key.

The logical distance between m-bits identifiers is computed by using bitwise `XOR` to figure out the longest common prefix of the two identifiers. The distance is then the number of bits that don't belong to this shared prefix. Among other properties, this computation is *symmetric* .

The goal of our work is to provide some insights about the structure of the networks that are generated by the Kademlia algorithm, assuming that we are looking at the network in a time frame when there is no churn.

2 Kademlia: a theoretical analysis

Kademlia is widely used by many P2P systems, such as eMule, BitTorrent and Ethereum, the reason lies in its simplicity and effectiveness.

Kademlia is initialized with three parameters:

`alpha:` represents the parallelism degree in forwarding callsbacks to other peers in the network;

`m:` indicates the amount of bits used to represent an identifier;

`k:` quantifies the maximum number of "neighbours" per each distance level in the routing table of any peer.

Since any identifier is represented using m bits, if we denote with p the number of prefix bits shared by two *different* identifiers (hence $p \neq m$) we can assert that the $p + 1$-th bit from left is necessarily different among such keys. Thanks to this observation, we can note that the possible number of keys that are at a distance p from a given string are 2^{m-p-1}.

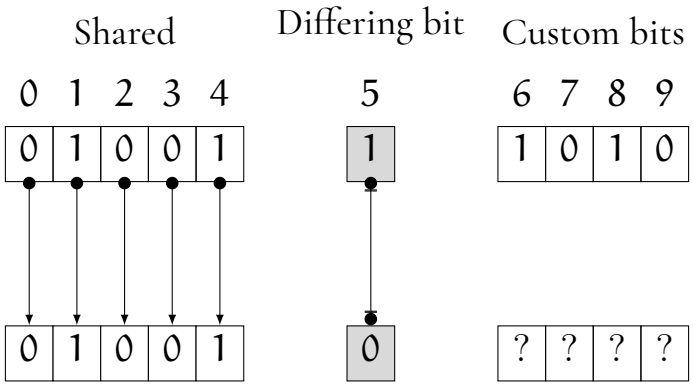


Figure 1: A pictorial representation of two binary strings that are at a distance of 5 *one from the other. In this example* $m = 10$.

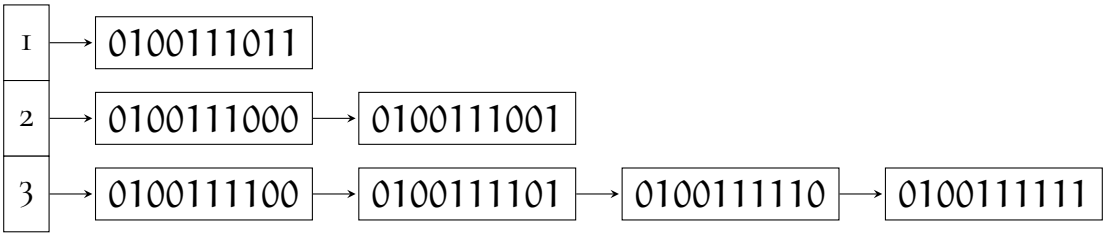


Figure 2: A possible configuration of the first three rows of the routing table of the node $n1$ *which id is* 0100111010*, for* $m = 10$.

Any peer in the network has a partial view on the network, in particular, for each distance value d in the range $[1, m]$ he stores the ids of at most k peers at a distance d from itself. To make the picture clearer, see Figure 2.

The functions implemented by Kademlia are the following:

`ping:` which is used to test if a peer is still online;

`store:` requests to a certain peer to store a file;

`find_node:` it is sent to find out the nearest peers in the network to a certain key. When a peer receives such a call returns the peers contained in its routing table that are nearet with th requested key;

`find_value:` it is sent to find the location of a key. When a node receives such a request returns te value if it is in charge of it, returns the closest peers otherwise.

The reader is now ready to understand our implementation.

3 Kademlia: our case study

We implemented a portion of a Kademlia P2P system, in particular we were interested in simulating the construction of the Kademlia routing tables.

The programming language we chose for the implementation was Java, in which we developed three classes:

`Coordinator:` which performs a first initialization of the network choosing a random id, corresponding to the first peer to join the network. Subsequently, the whole network is populated, until it contains n peers;

`Node:` represents a node and contains exaclty two fields: the node id and its routing table;

`RoutingTable:` this class represents a routing table, hence stores all the rows of such a table and implements some utility functions, such as print and store operations;

`Key:` is how we implemented the identifiers of both nodes and keys in the DHT. This class make use of Java BitSet, but extends it with some useful operations such as "ad hoc" functions for computing the distance between identifiers.

A detailed documentation is available in `doxygen` format in the source code.

This code has been used to build many instances of a Kadmlia network, using random choices, varying the following three parameters: n (total number of peers in the network), m (lenght of the identifiers in terms of bits) and k (the maximum number of keys contained in a row of a routing table).

We chose the following sets of values for the parameters presented above:

$$\begin{cases} k \in \{2, 5, 10, 20\} \\ m \in \{8, 16, 32, 160\} \\ n \in \{100, 1000, 10000\} \end{cases}$$

We also ran the simulation for $n = 100000$, with $m = 32$ and $k \in \{2, 5, 10\}$

The protocol specifications * set $k = 20$, so we decided to move around that value.

On the other hand, the hash function that is used by Kademlia is SHA1, which has a key length of 160 bits, which is the value that guarantees a unform distribution among those keys.

The highest reasonable value for n is 100000, in order to allow fast computations.

* For more details visit:
http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

4 Experimental results

We are now ready to discuss what we found out in this analysis.

In Table 1 there are some measures of the many networks we generated.

Most of such measures are intuitive, less intuitive is *eccentricity* (the average among the maximum distances of each node from any node in the graph) and *clustering coefficient* .

The clustering coefficient of an oriented graph (such Kademlia network) is defined as follows:

$$C_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)}$$

where N_i for a vertex v_i is defined as its immediately connected neighbours: $N_i = \{v_j : e_{ij} \in E \vee e_{ji} \in E\}$.

In order to show the in and out degree distribution we pick the highest n we used (10000) and we show the possible values for k and m, in Figure 8, Figure 6, Figure 7, Figure 5, Figure 3, Figure 4.

As expected,

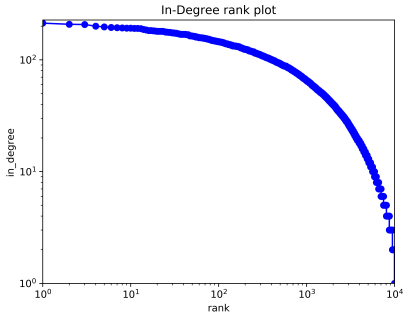
As expected, the in-degree distribution is very different from the out-degree distribution. Indeed, our expectation is that the out-degrees of every node is very similar, as they only depend on the parameters k and m and the IDs of the other nodes in the network. Thus, the expected Gaussian distribution for them, is clearly shown in the plots listed above. On the other hand, in-degrees follow a preferential attachment model (as nodes with higher input degree are more likely to be discovered earlier by a new connecting node, and thus more likely to end up in their routing tables). Thus, the expected Zipf distribution for them is shown in the in degree plots when the number of nodes is high enough.

For what concerns eccentricity, Figure 9 the resulting networks have a low diameter. Moreover, we can see that the distribution of eccentricities is basically constant, showing that the worst-case time to reach another node does not vary much across the network.

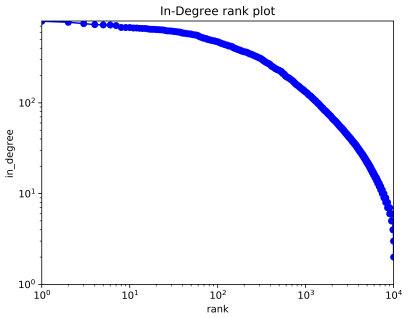
5 Conclusions

We conclude that a Kademlia network satisfies the small-world network definition, since it has a small average degree but at the same time a small diameter, approximately logarithmic.

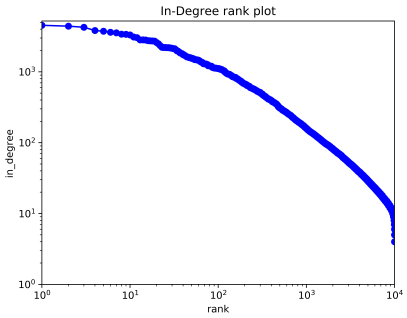
k	m	n	Edges	Density	Diam	Ecc	CC	deg
2	8	100	1,148	0.12	4	3.77	0.29	11.48
2	8	1,000	14,023	$1.4 \cdot 10^{-2}$	6	5.33	0.2	14.02
2	8	10,000	$1.45 \cdot 10^{5}$	$1.45 \cdot 10^{-3}$	6	5.31	0.23	14.49
2	16	100	1,215	0.12	5	3.99	0.29	12.15
2	16	1,000	18,583	$1.86 \cdot 10^{-2}$	6	5.12	0.15	18.58
2	16	10,000	$2.46 \cdot 10^{5}$	$2.47 \cdot 10^{-3}$	7	6.05	0.12	24.65
2	32	100	1,218	0.12	4	3.9	0.28	12.18
2	32	1,000	18,608	$1.86 \cdot 10^{-2}$	6	5.14	0.15	18.61
2	32	10,000	$2.52 \cdot 10^{5}$	$2.52 \cdot 10^{-3}$	7	6.02	$9.6 \cdot 10^{-2}$	25.2
2	160	100	1,200	0.12	4	3.95	0.28	12
2	160	1,000	18,574	$1.86 \cdot 10^{-2}$	6	5.01	0.15	18.57
2	160	10,000	$2.52 \cdot 10^{5}$	$2.52 \cdot 10^{-3}$	7	6.05	$9.63 \cdot 10^{-2}$	25.2
5	8	100	1,996	0.2	4	3.17	0.43	19.96
5	8	1,000	28,352	$2.84 \cdot 10^{-2}$	5	4.85	0.26	28.35
5	8	10,000	$2.92 \cdot 10^{5}$	$2.92 \cdot 10^{-3}$	6	5.63	0.29	29.21
5	16	100	2,208	0.22	4	3.13	0.4	22.08
5	16	1,000	37,737	$3.78 \cdot 10^{-2}$	5	4.36	0.21	37.74
5	16	10,000	$5.28 \cdot 10^{5}$	$5.28 \cdot 10^{-3}$	6	5.18	0.14	52.82
5	32	100	2,247	0.23	3	3	0.41	22.47
5	32	1,000	37,614	$3.77 \cdot 10^{-2}$	5	4.01	0.21	37.61
5	32	10,000	$5.38 \cdot 10^{5}$	$5.38 \cdot 10^{-3}$	5	5	0.14	53.81
5	160	100	2,204	0.22	3	3	0.41	22.04
5	160	1,000	37,703	$3.77 \cdot 10^{-2}$	5	4.03	0.21	37.7
5	160	10,000	$5.39 \cdot 10^{5}$	$5.39 \cdot 10^{-3}$	6	5	0.13	53.9
10	8	100	2,643	0.27	3	3	0.46	26.43
10	8	1,000	39,713	$3.98 \cdot 10^{-2}$	6	5.36	0.33	39.71
10	8	10,000	$3.88 \cdot 10^{5}$	$3.88 \cdot 10^{-3}$	9	8.29	0.34	38.77
10	16	100	3,060	0.31	3	2.66	0.46	30.6
10	16	1,000	57,676	$5.77 \cdot 10^{-2}$	4	3.97	0.28	57.68
10	32	100	3,027	0.31	3	2.66	0.48	30.27
10	32	1,000	58,024	$5.81 \cdot 10^{-2}$	4	3.85	0.27	58.02
10	160	100	3,074	0.31	3	2.88	0.46	30.74
10	160	1,000	57,978	$5.8 \cdot 10^{-2}$	4	3.87	0.27	57.98
20	8	100	3,104	0.31	3	2.89	0.52	31.04
20	8	1,000	36,877	$3.69 \cdot 10^{-2}$	5	5	0.37	36.88
20	8	10,000	$3.66 \cdot 10^{5}$	$3.66 \cdot 10^{-3}$	9	8.74	0.4	36.6
20	16	100	3,656	0.37	3	2.02	0.54	36.56



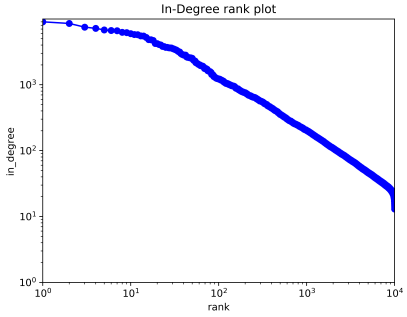
(a) k=2



(b) k=5

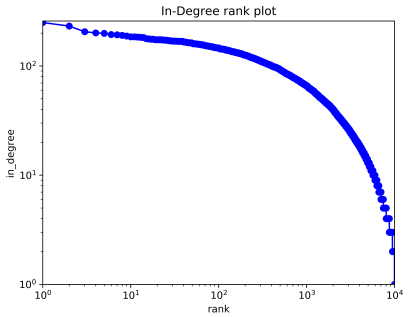


(c) k=10

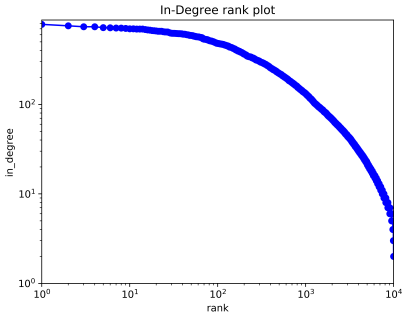


(d) k=20

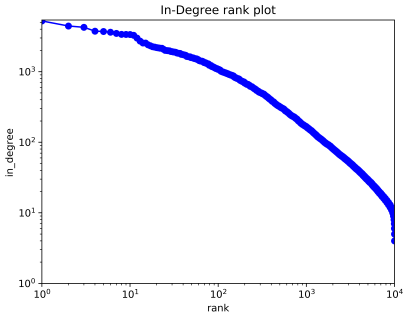
Figure 3: In degree for m=160



(a) k=2

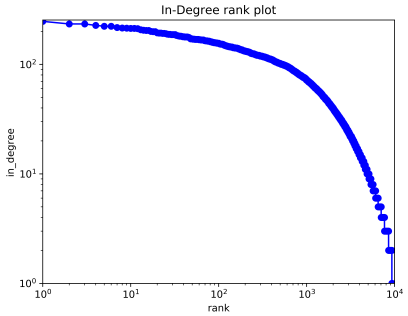


(b) k=5

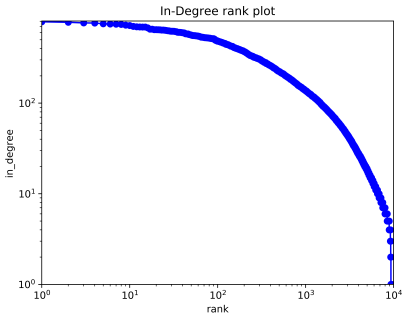


(c) k=10

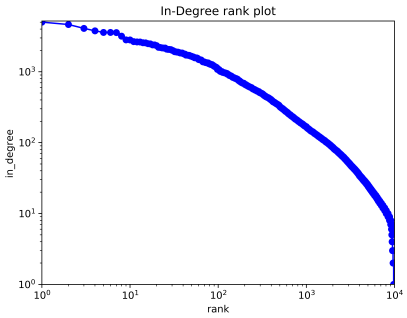
Figure 4: In degree for m=32



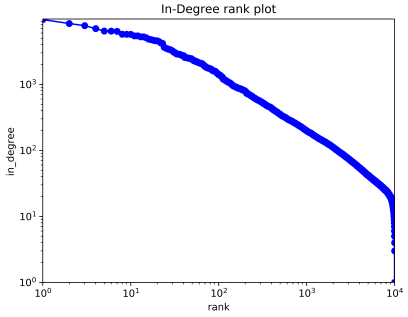
(a) $k=2$



(b) $k=5$

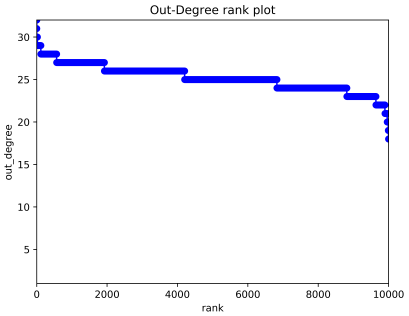


(c) $k=10$

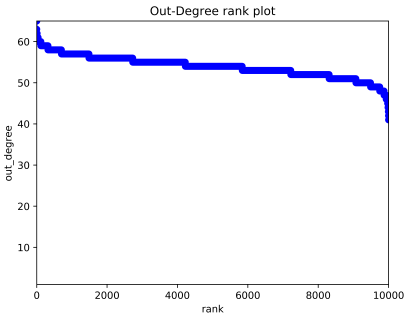


(d) $k=20$

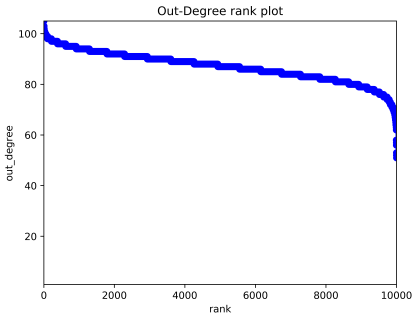
Figure 5: In degree for m=16



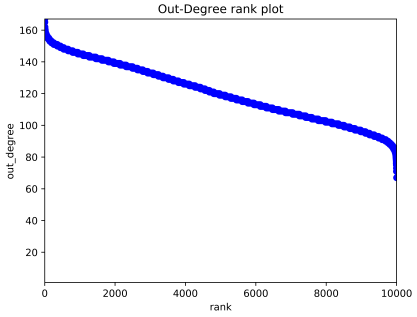
(a) $k=2$



(b) $k=5$

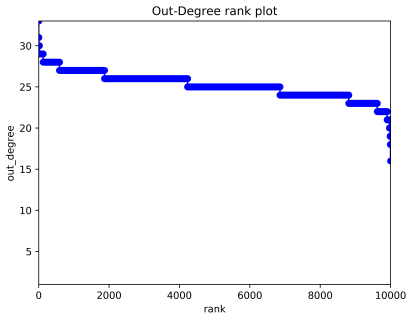


(c) $k=10$

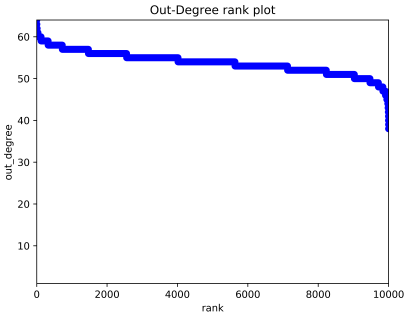


(d) $k=20$

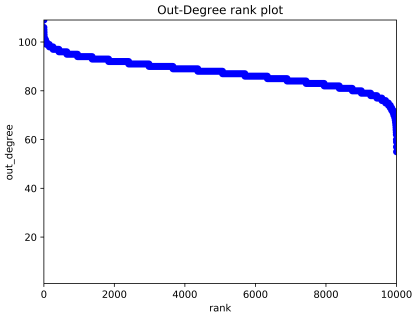
Figure 6: Out degree for m=160



(a) k=2

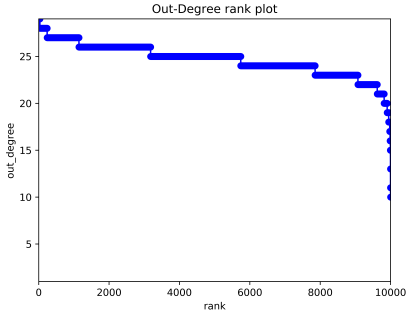


(b) k=5

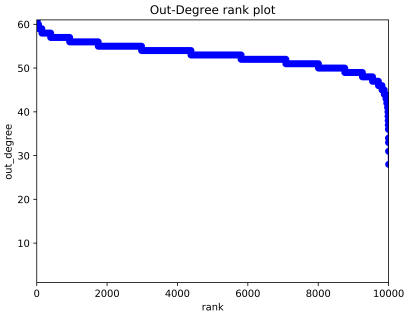


(c) k=10

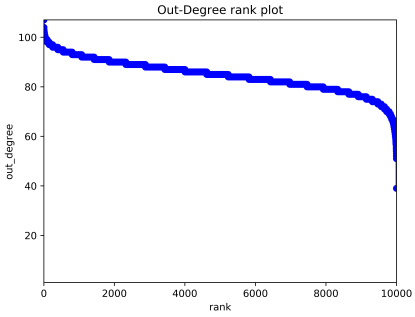
Figure 7: Out degree for m=32



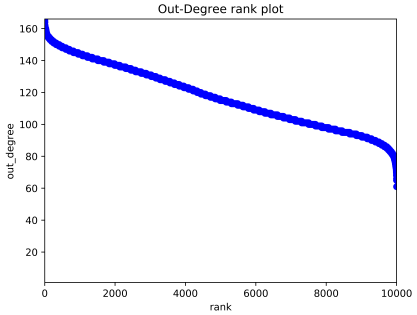
(a) k=2



(b) k=5



(c) k=10



(d) k=20

Figure 8: Out degree for m=16

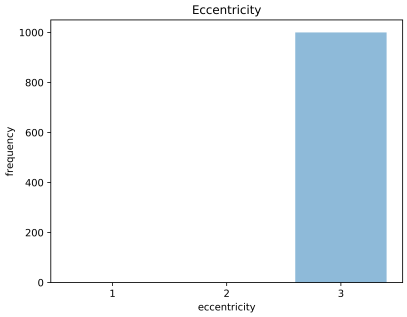


Figure 9: Eccentricity distribution fo m=160, k=20, n=1000