Frans Ojala
6 Mar 2015

Distributed systems project

Assignment 3: overlay networks

The assignment was to construct an overlay network capable of routing messages between any two nodes that conformed to a set of restrictions: the nodes in the network were to use only their designated routing table, broadcasting was not allowed, there were to be 1024 nodes and the product of the largest routing table of any node (R) with the average number of hops (H) was to be less than 387 ($H * R < 387$). The overlay was to be implemented and tested on the Ukko HPC cluster.

The assignment was done in collaboration between I, the autor, and Jackob Lärfors. The reader should thus not be surprised of the similarity in our approaches as there was significant mutual contribution. All source code, on the other hand was developed and written separately and is not shared.

Kautz graphs

Based on previous knowledge and further exploration of options available, the static topology for the implementation was chosen to be based on Kautz graphs. There are several points of interest to consider in the choosing of a network topology:

1. The size of routing tables. The more connected each node is the shorter become paths from node to a randomly chosen node (in general). But higher degrees also increase the complexity and the H * R value.
2. Average number of hops to route a message within the network. There are two main components to consider here. The first is already described above: the routing table size of a node.
3. The second is the arrangement of nodes and their respective paths within the network. This is a far more complex problem and has had considerable research done in various forms of graphs. The nodes need to be connected in certain ways to achieve short paths which can be routed algorithmically.

It can be seen, therefore, that there needs to be a tradeoff in terms of routing table size and average number of hops in paths. There exists an optimal achievable value based on theories of certain graphs.

Kautz graphs are a specific graph that has been demonstrated [1, 2] to provide near optimal node distribution, path lengths and routing table sizes. It is closely related to the De Brujin graph, and thus follows similar semantics. Kautz graphs are defined as follows:

1. Describe the Kautz graph as a tuple: $K(d, k)$, where d represents the *degree* of the graph, and k represents the *Kautz string length* denoted henceforth as *k-length*.

2. The graph nodes are described by identifiers, which are formed from the *alphabet* and *k-length* of the Kautz graph.

3. The *alphabet* A is defined as symbols:
   $A = \{0, \dots, d-1, d\}$
   So, for instance the alphabet for a K(2, k) would be {0, 1, 2}

4. The *identifier* of a node is defined as a representation from the *alphabet* over the *k-length*:
   $\mathrm{id}(U) = u_0, \dots, u_{k-1}, u_k \qquad u_i \neq u_{i+1} \in A$
   for instance, an identifier for a node U_e in a K(2, 3) would be of form
   $\mathrm{id}(U_e) = 012$

5. The Kautz space is then, obviously, all the identifiers that can be formed with these constraints:
   Given a Kautz graph $K(d, k)$, the *K-space* is
   $K_s = \{U_1, \dots, U_z\} \qquad z = d^k + d^{k-1} \qquad U_i \neq U_j$
   and it is of size
   $|K_s| = \mathcal{N} = z$

6. Each node has d degree, i.e. d outgoing vertices, to nodes of defined identifiers. The out-neighbours are defined as:
   A node U has out-neighbour V iff.
   $\mathrm{id}(U) << \mathrm{id}(\mathcal{J})$ where << is the left shifting operator:
   given $\mathrm{id}(U) = u_0, \dots, u_{k-1}, u_k$
   and $\mathrm{id}(\mathcal{J}) = j_0, \dots, j_{k-1}, j_k$
   Then $\mathrm{id}(\mathcal{J}) = u_1, \dots, u_{k-1}, u_k, j_k \quad j_k \neq u_k$

Kautz graphs have near optimal properties for degree and average path length. The degree in a Kautz graph is, by definition, d, and the average path length is calculated in relation to the diameter D of the graph:

$$D = \log_d \mathcal{N} - \log_d (1 + 1/d)$$

$$Avg_{path} = D - 1/(d+1)$$

From [1]:

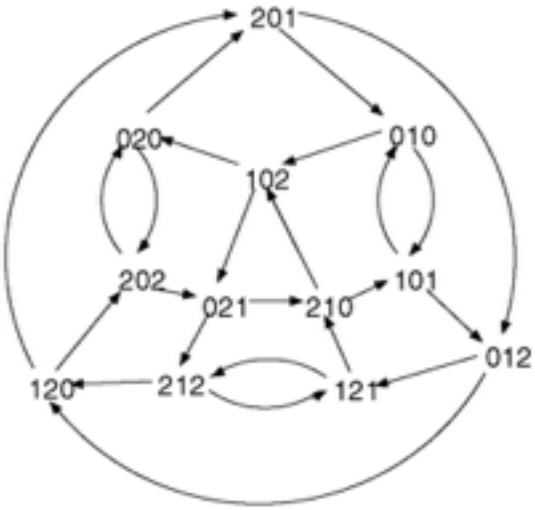

Fig. 1. The digraph $K(2,3)$.

As the static topology a graph of K(2, 10) was chosen because it provides a relatively good tradeoff between the number of required nodes (1536 vs 1024), degree and average path length. The number of nodes is substantially higher than specified in the assignment but as said it is a tradeoff that needed to be taken. The assignment constraints then take the form:

$$H = (\log_d N - \log_d (1 + 1/d)) - 1/(d+1)$$
$$= 9.6667$$
$$R = d = 2 \qquad \textit{(every node has the same degree)}$$
$$H * R = 2 * 9.6667 = 19{,}3334$$

Implementation and experimentation

Before work on a proper and fully functioning peer-to-peer system was started an experimental simulation system was built. The implementation is done in Node.js and is included in the package. The simulation consists of the following steps:

1. Generating the Kautz space identifiers
2. Building the graph vertices (routes)
3. Full routing simulation from all nodes to all nodes
4. Result logging

The purpose of the simulation was to experimentally verify that the FISSIONE system described in [2], which most of the work in this assignment is based on, is actually feasible and functional. According to the simulations the method for generating universal identifiers, while being feasible and (possibly) functional for larger k-lengths, is not uniform enough for smaller graphs. The main problem arises from the decreased uniformity when dealing with transformations from output of a general purpose cryptographic hashing algorithm to the Kautz space because of the special properties the identifiers have in Kautz graphs. To produce proper identifiers from input keys a (preferably perfect) hashing algorithm would

need to be devised specifically for the purpose. This is, however, far beyond the scope of this assignment.

The simulation also revealed that, in order to produce a fully functional overlay network that has a considerably smaller number of nodes than the graph would specify, the routing algorithm and neighbour selection need substantial tweaking. The FISSIONE system provides insight to the problem but the methods were not implemented at this stage. Rather it was decided that a peer-to-peer system would be built to accomodate the 1536 nodes requirement that a K(2, 10) graph presents.

The peer-to-peer network was also implemented in Node.js and is included in the package. There are several separate components to the program:

1. The master; its function is to generate the required identifiers in the K-space, assign them a host:port combination and finally start the operation of the nodes at the designated locations. The matser inculdes modules for routing, string generation and remote execution that may be reimplemented as one chooses.
2. The intermediate; its function is to start the node as a background process at a remote location. Being closely tied to the master and the style of remote execution the intermediate cannot be said to be a fully external component to the master. It can however be redesigned to be used over different network paradigms in different host environments.
3. The node; its purpose should be self-explanatory. Forms a single connected unit in the resulting peer-to-peer network that operates independently.

To use the network read the README included with the source. In practice there are not differences between the simulation and the actual built peer-to-peer network. Thus the results that are presented in the next section are just as valid even though they are mainly based on the outputs of the simulation rather than full fledged network activity.

Results

To gain insight into how the network would perform the simulation was run with several different parameters and configurations. As there were several implementation strategies only the most interesting and relevant are covered here. The interesting factors to take into consideration are:

- Postfix-prefix matching variance: since not all nodes are guaranteed to have exactly two out-neighbours with the exact properties of the Kautz out-neighbours the matching algorithm had to be tweaked. It matches two nodes with decreasing the size of the postfix to prefix match. In a perfect Kautz graph of K(*d, k*) the match is of exactly *k-1* length, but now at least of length $k - (k-1) = 1$.

- Collisions in identifiers: when the identifiers are generated from input keys the mechanism proposed in FISSIONE and implemented here produces collisions that are not resolved in the identifier generation algorithm.
- Routing information: number of routes, unreachable nodes, barely reachable nodes, dead ends, almost dead ends, looping routes (there are factors to play with here), too long routes, routes that die, successful routes, etc.. These and some more are presented below.

1. The simulation was run with the FISSIONE identifier generation mechanism with a static Kautz graph of K(2, 10) and a key count of 1024. Routing was done with longest postfix-prefix matching mechanism. Key: localhost+51000 as starting port which was incremented for each key.
2. The previous was repeated with the FISSIONE routing mechanism
3. Graph: K(2, 12), postfix-prefix routing, 1024 keys, key: localhost+51000 as starting port.
4. Graph: K(2, 12), FISSIONE routing, 1024 keys, key: localhost+51000 as starting port.
5. Graph: K(2, 10), FISSIONE routing, 1536 keys, keyspace generation
6. Graph: K(2, 10), FISSIONE routing, 1024 keys, key: 127.0.0.1 as starting ip, incremented each key and 51000 as starting port, also incremented each key
7. Graph: K(2, 10), FISSIONE routing, 1024 keys, keyspace generation and scaledown by removing excess keys
8. Graph: K(2, 100), FISSIONE routing, 1024 keys, 127.0.0.1 starting ip incremented, 51000 starting port incremented

Mode	Collisions	Total routes (vertices)	Inaccessible nodes (no in path)	Accessible from only 1 node	Dead ends (no path out)	1 out-neighbour
1	284	2041	3	1	3	4
2	284	2041	3	1	3	4
3	80	2039	4	1	4	5
4	80	2039	4	1	4	5
5	0	3072	0	0	0	0
6	280	2037	4	3	5	6
7	0	2047	0	1	0	1
8	0	2042	2	2	2	4

Table 1: Identifiers

It can be seen from the data in table 1, that the identifier generation by FISSIONE universal naming does not produce consistently good results. The FISSIONE identifier generation consistently produces collisions in smaller graphs, while in larger graphs collisions

are absent. However, using larger static graphs require that the neighbour selection and routing algorithms need to be redesigned, possibly to the FISSIONE methods. The amount of collisions in small graphs encouraged the project to move to the direction of entire keyspace generation as a fast solution. It should be noted that in mode 7, when the keyspace was purposefully scaled down by removing exactly 33.333% of the keys the resulting network does seem to have almost the desirable properties: all nodes are accessible and no dead ends.

Mode	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	16	36	57	156	361	592	744	0	0	x								
2	16	36	57	156	361	592	744	0	0	x								
3	25	21	20	38	60	104	242	415	496	389	229	x						
4	25	21	20	38	60	104	242	415	496	389	229	x						
5	0	0	0	0	0	0	0	0	3072	x								
6	31	17	18	48	81	137	328	640	737	x								
7	0	2	2	10	14	41	110	696	1145	x								
8	8	6	33	57	101	122	255	363	380	312	177	111	55	25	22	7	4	1

Table 2: Matching variance, identifier characters matched in route building

Looking at the neighbour selection patterns in table 2 that the longest postfix to prefix matching algorithm developed to match identifiers in an incomplete graph we can see that as expected the full graph (mode 5) is perfect in matching. There are only matches that are one less than the identifier length. In other cases the matching varies substantially and there are even several cases where there was only a single character that was matched. This is undesirable because routing becomes difficult and there are no implicit or explicit guarantees that all areas of the formed graph are actually connected.

Mode 7 produces a matching variance that pertains to a relatively well connected graph.

As expected the longest postfix-prefix routing does not perform well in this network topology (table 3). This can be seen in modes 1 and 3 where there are many looped routes and the average path length in dead routes is very high. The looping was cut off and marked when the algorithm arrived at a node that had been visited and would have routed the message to a successive out-neighbour that had also been visited from the current node, i.e. the path had already been taken. Also to note is that the average path length in the postfix-prefix mathing is long also for the successful routes. FISSIONE routing, however performs much better, as expected. There are no loops in the routes but in contrast there are 10 times more dead routes. The dying of the routes is due to there not being the proper out-neighbour to route a message to. This could be mitigated by having more connections (a higher degree of connectivity) between nodes like described in the FISSIONE system, but this also increasees the complexity of the graph and the H*R value.

Mode	Looped routes	Dead routes	Avg path lengh (dead)	Too long (>99 hops)	Successful routes	Avg path length (success)
1	952263	30778	11,76	22215	42296	29,64
2	0	1040400	2,30	0	7152	8,15
3	772301	178476	26,79	48140	48635	32,34
4	0	1044717	2,31	0	2835	7,40
5	0	0	0	0	2357760	9,64
6	0	1040876	2,29	0	6676	8,01
7	0	1028827	2,68	0	18725	8,80
8	0	1046139	2,31	0	1413	2,47

Table 3: Statistics on route information, routed from all nodes to all nodes

Noteworthy is that in FISSIONE routing the dead routes are very short in average, just over 2 hops. The connectivity of the nodes in a perfect Kautz graph is quite elegant: by arranging the identifiers in an alphabetical order and assigning them a number (as done in the simulation) one can see that there are three major parts to the graph. A node in part 1 (identifiers 1-512) routes to part 2 (identifiers 513-1024) and part 3 (identifiers 1025-1536). Respectively a node in part 2 routes to part 1 and 3, and a node in part 3 routes to part 1 and 2. It could be that the dead routes are on average length just over two because there are missing connections from some parts of the graph to others, this has not been verified. Also take note that each graph has different connectivity if the identifiers are formed using the FISSIONE generation and a different input key is used.

The mode 5 can be seen to produce an average path length that is a little lower than the expected theoretical value. This is due to an optimisation in the implementation of the FISSIONE routing algorithm. The optimisation stops routing the message onwards if it arrived at its destination. There are cases when the FISSIONE routing algorithm would needlessly route the message further and the optimisation removes this. This is also the mode that is used in the actual peer-to-peer network implementation because it infact is the only one that the author was able to construct in time.

The resulting H*R value is then 2*9.64 = 19,28.

Conclusions

As can be seen from the results the methods used in the assignment are not yet sufficient to produce a full-fledged peer-to-peer network to use in real life situations. It is however an interesting theoretical framework within which much work could still be done towards optimisations and robustness. As proven earlier there would be a need to devise a perfect hashing function that produces identifiers with certain properties specifically for Kautz

networks. There is also the need to revisit the neighbour selection algorithm to be able to form properly connected Kautz graph overlays. One would also need to rewrite the routing algorithm to accomodate the constant churn of real life peer-to-peer networks. The FISSIONE system has many of these and the mechanisms include, for example, dynamically increasing and decreasing Kautz subgraphs and keyspaces to counter churn.

REFERENCES:

1. Panchapakesan, Geetha, and Abhijit Sengupta. "On a lightwave network topology using kautz digraphs." *Computers, IEEE Transactions on* 48.10 (1999): 1131-1137. APA
2. Li, Dongsheng, Xicheng Lu, and Jie Wu. "FISSIONE: A scalable constant degree and low congestion DHT scheme based on Kautz graphs." *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. Vol. 3. IEEE, 2005.