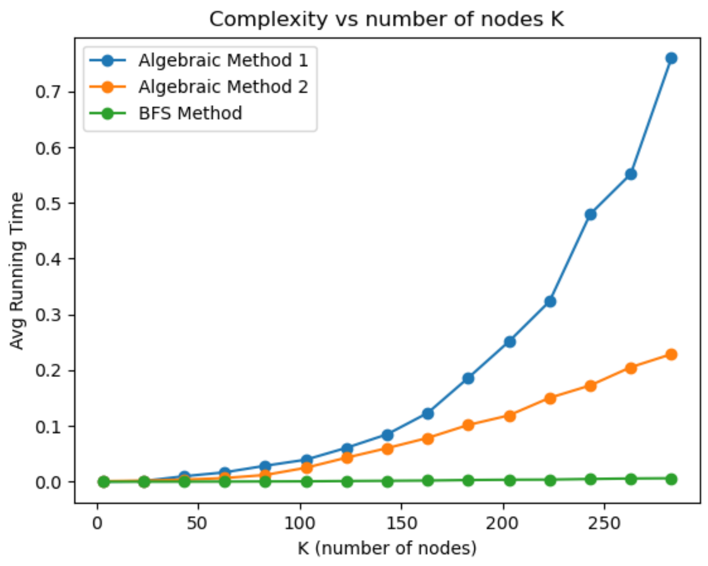
Networking for Big Data and Laboratory
Report Homework 1
Team "Nyquist"

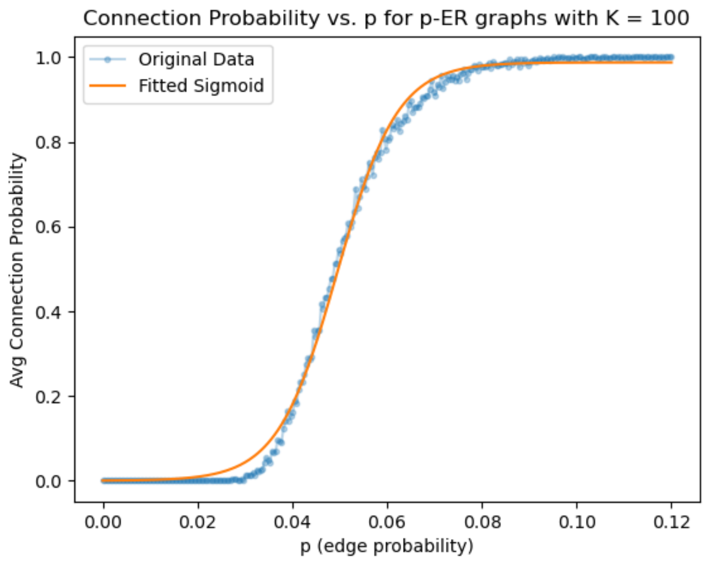
Francesco Mari - 1919565
Lorenzo Pannacci - 1948926
Cristiano Perrone - 2128080
Riccardo Violano - 2148833

May 2024

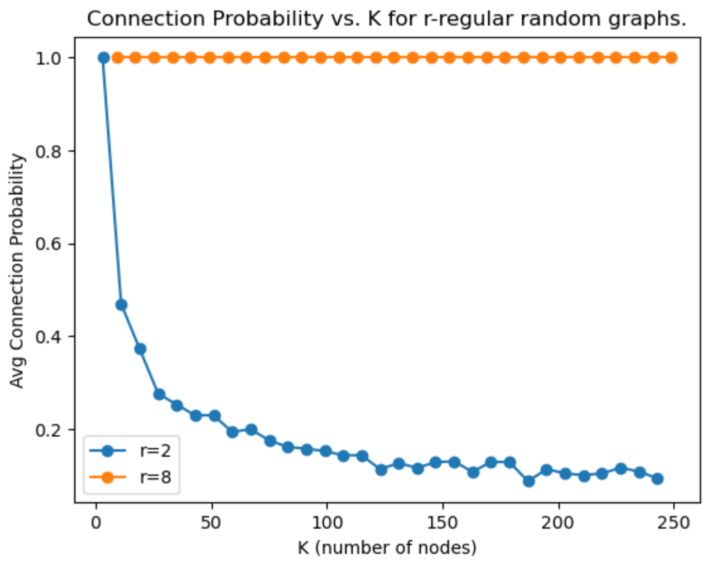
Part 1 - Connectivity algorithms and prob. of connected random graphs



(a) First figure



(b) Second figure



(c) Third figure

The **first figure** shows the complexity estimation of the 3 algorithms that check the connectivity of a given graph via a Monte Carlo approximation for the **average running time**. We run the tests using a complete graph for all the algorithms. In particular for the algebraic method 1, that studies if the adjacency matrix A is irreducible by checking $\mathbf{I} + A + A^2 + \cdots + A^{n-1} > 0$, we used the *Horner's method* in order to avoid the computationally expensive matrix multiplications. For the other algorithms we used well-known built-in functions both to retrieve the eigenvalues of the adjacency matrix and for the BFS. For the algebraic method 2 we considered eigenvalues as zero if lower than a fixed tolerance, to account for machine precision (tolerance=10^{-12}). We can see that algorithms are sorted by increasing performance from the algebraic method 1 to the BFS one.

In the **second figure** we can see the connection probability for a Erdős–Rényi graph as p (probability of existence of an edge) varies. We estimated the connection probability with a Monte Carlo approximation with 500 iterations using the previous BFS method. As we can see connection probability is consistently near 1.0 for $p > 0.1$ and from the orange fitted line we understand the shape is reasonably a **sigmoid**.

In the **third figure** it is shown the connection probability for r-regular random graphs for $r = 2$ and $r = 8$ (number of neighbors of each node) as the number of nodes K varies. As we can see for $r = 8$ the connection probability stays constant at 1, whereas for the 2-regular random graphs it seems to go asymptotically to zero. In truth, this is a special case of a more **general theorem** that says that for $r \geq 3$ a graph with large size K is asymptotically almost surely connected (Bollobás, section 7.6: Random Regular Graphs).

Part 2 - Computational Job: local versus distributed run

This second part tasks us with the creation of a distributed computation scenario with two different topologies to understand when distributing the work has the most positive effects both in computations times and computations cost, measured as the total work time dedicated to the task. The calculations for the two metrics are straightforward for the baseline, since there is no data transfer to consider:

$$E[R] = T_0 + E[X] = 30 + 8\dot{3}.600 = 28.830 \text{ seconds}$$

$$S = E[R] + \xi E[\Theta] = (1+\xi)\dot{(}T_0 + E[X]) = 31.713 \text{ seconds}$$

Since the **Fat-Tree topology is deterministic** we can obtain the distances between the origin server A and any other server as a function of N: the nearest 31 servers are in the same rack of A with a number of hops of 2, the next 992 servers will be in the same pod with 4 hops and any other server will require an inter-pod path with length 6. Using the provided formulas from the array of the distances we can obtain the Round-Trip Time (RTT) of each connection and therefore their throughputs and times to send the input data, which are still completely deterministic, but not identical for all servers.

Since the mean response time is given by the slowest server response we need to calculate the expected value of the **maximum of N random variables**, which are the sum of the input data transfer time, the computation time which is partially fixed (T_0) and partially distributed as a negative exponential and the output data transfer time, which follows an uniform distribution. Due to the many factors that play out in the definition of the random variable we consider the most suitable approach to solve this problem the use of **simulations**. As we managed to boil up the non-deterministic part to just two random distributions we will be able to avoid the construction of real graphs which would considerably slow down the execution of the simulation.

Most of the reasonings for the Jellyfish topology are the same as above, the only difference is the topology itself and its random nature. Even if the **Jellyfish topology is random**, we can find some deterministic behaviors inside it. We know that the 31 server in the same rack as the origin server A will be the closest ones with 2 hops of distance and this is true for every possible configuration of the topology. We also know that the switch of this rack is uses 32 ports to connect to other switches and since there are no parallel edges in the topology we have a guarantee for other 1024 servers at a distance of 3 hops.

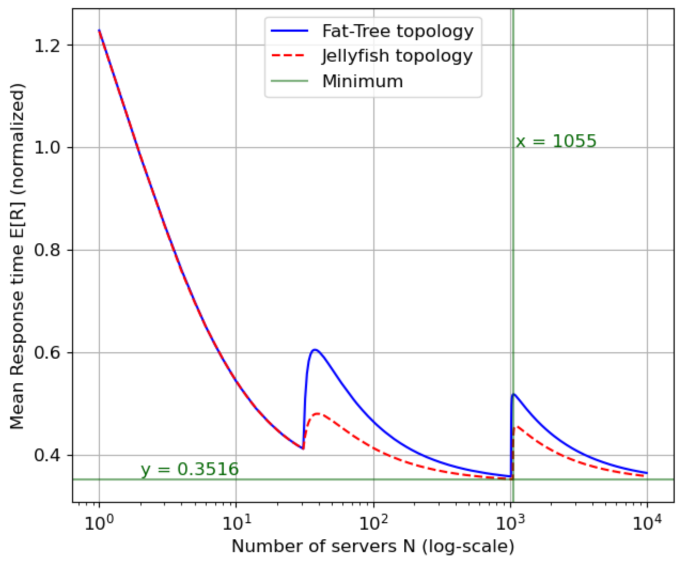
The fact that **getting farther we increase the distance of only** 1 **hop** is an important difference with the Fat-Tree topology and it is caused by the fact that Jellyfish does not use switches dedicated solely to connecting only other switches.

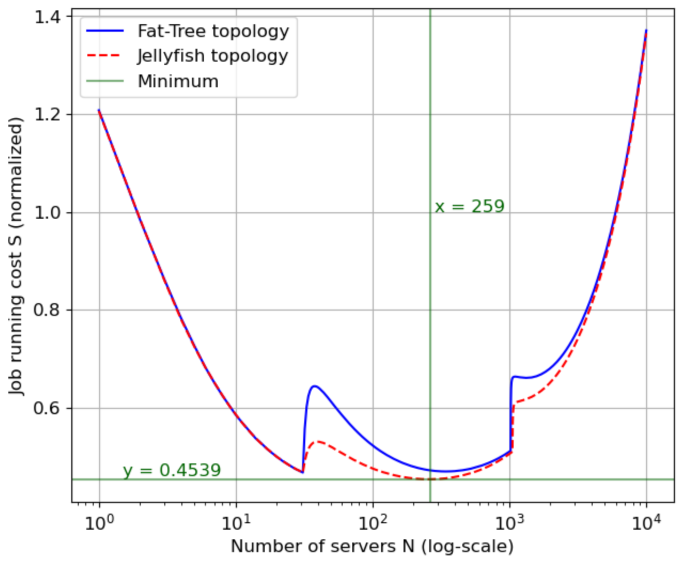
From here on out the behavior of the topology is no longer deterministic: we have 32 switches each with 32 ports for other switches, but there is the possibility to connect to switches that we already counted. While we need only a small portions of those links to be connected to 'unexplored' switches to satisfy our need of at most 10.000 servers there are no guarantees and in unfortunate configurations the number of hops can increase to more than 4 hops.

From the article that proposed Jellyfish (available here and included in the archive of the homework) we understand that in the case where every switch has the same number of ports and servers the network (considering only switches) can be seen as a **Random Regular Graph (RRG)** sampled uniformly from the space of all *r-regular graphs*. From this insight we can design a function that creates the array of distances starting from the *random_regular_graph* method of the Python library *NetworkX*.

While this considerably reduces the computation times of the simulation compared to the application of the Jellyfish construction algorithm we still consider it too high to get a reasonably smooth curve. We therefore decided to implement another simulation to find out the probability distribution of the maximum distance in the Jellyfish topologies of the kind we are interested in (2048 switches with 32 ports for switches connections) and found out that **the probability of the max distance being above the minimum of** 4 **hops is virtually zero**, allowing us to use an algorithm analogous to the one used for Fat-Tree.

What we present is the result of simulations of 100.000 iterations for every N considered. We decided to use a **logarithmic scale for the x-axis**, as it manages to capture more clearly the phenomenon. This also allowed us to exponentially increase the sparsity of the N used in simulations. We however decided to **keep a high density near interest points** to better capture their behavior with high precision.





We can observe the following behaviors:

- **The topologies behave the same for** $N \leq 31$. This is because here they only use same-rack servers, meaning the distances arrays are exactly the same.
- There are **two spikes** on both plots for both topologies, corresponding to the increase of the maximum length of the paths between the origin server and the others.
- **Jellyfish results are always better or par with Fat-Tree for both plots**. Values are noticeably better right after an increase of maximum length and converge the closer we are to a new spike. Right before the second spike the Jellyfish curve is slightly below the one of Fat-Tree: this difference is the effect given by the lower path lengths of the Jellyfish topology.
- **The second spike for the Jellyfish topology happens after** the one of Fat-Tree since the number of servers before reaching the new maximum length is slightly higher.
- **Jellyfish gets the minimum Mean Response Time** with a value of 0.3516 at $N = 1055$, the maximum amount of servers we can possibly get before increasing the maximum path length to 4. Fat-Tree's minimum is a little higher (0.3570) but follows the same reasoning as we found it at $N = 1023$, the maximum amount before path length increases to 6.
- **Jellyfish gets the minimum Job Running Cost** with a value of 0.4539 at $N = 259$, an intermediate value between the first and the second spike. This time the result of Fat-Tree is very different, with the minimum at $N = 31$ and with a noticeable higher value (0.4675).

Overall in our scenario Jellyfish presents itself as the best choice for both the evaluation metrics in every case, and this is to be added to other factors Jellyfish excels in, as the easiness inserting new nodes, cheaper expandability and resistance to human error during the physical wiring of the topology.