

A Symphony implementation for Peersim

Project of the Peer-to-Peer course

Andrea Esposito
September 16, 2013

1 THE MODULE

The module for the simulator Peersim [1] implements, in the most faithful as possible to the paper [2], the DHT Symphony in particular:

- join to the network
- search of a key
- voluntary leave
- involuntary leave
- a custom algorithm to estimate the network size

The overlay is extended to allow to each node to has more than 1 neighbour per side and so be able in this way to contrast the presence of churn.

The estimation of the network size is done by monitoring only the local information of each node without exchanging any data. The algorithm anyway is based on the length segments to each neighbours.

1.1 CODE

The code is written to be, hopefully, reusable such a way to allow future integration in other works or to be simulated in a more complex and heterogeneous overlays.

The protocols are essentially 3:

1. SymphonyEstimationProtocol, implements the custom algorithm to estimate the network size;
2. SymphonyNetworkManager, implements the join operation, the voluntary leave operation and all the operations such a way the overlay can be created (e.g. instauration of the long range links) and maintained also in presence of churn (e.g. keep-alive messages). This protocol is both event-driven and cycle based.
3. SymphonyProtocol, represents actually the symphony overlay and implements the routing operations. This protocol is event-driven.

Using two different protocols to implement the overlay management and its routing is possible to use different transport channels. For example is possible to put the routing on a reliable channel and the overlay management on an unreliable but faster one.

The controls are 7:

1. NetworkEstimationTest, calculates the real network size and the estimated one such a way to compare them;
2. SymphonyNetworkBuilder, builds the initial network;
3. SymphonyNetworkChecker, verifies that the overlay is actually okay (e.g. the number of the short range links is greater than 0) and how many nodes are disconnected;
4. RandomRouteTest, simple control that sends some random requests just to simulate a workload on the overlay;
5. RingRouteTest, it sends a series of requests such a way to move through all the ring (move from the current node to the nearest one in a clockwise direction);
6. LeaveTest, allows to decrease the network size by voluntary leave of nodes;
7. SymphonyStatistics, calculates statistics mainly to verify the average hops of routing and total number of messages exchanged.

1.2 CONFIGURATION

It's possible to specify the following parameters:

- routing algorithm (unidirectional/bidirectional);
- number of the short range links;
- number of the long range links (fixed or $\log(n)$);
- use of 1-lookAhead;

- use of the Relinking criteria;
- the range that the Re-Linking criteria is based on to decide if to do or not a “fresh refresh” of the long range links;
- the number of the attempts that are requested before is declared failed the procedure of the long range links creation;
- the number of the timeouts that are received after had sent the keep-alive messages to declare a node offline;
- independently choice of the algorithm to estimate the network size.

Actually is not provided but the module is able to manage an heterogeneous overlay with nodes that have different settings.

In the file *example.cfg* is possible to find a configuration file with the ALL possible parameters that can be specified.

2 MEASURES PERFORMED

The measures are done such a way to validate:

1. the custom algorithm to estimate the network size;
2. the values that the paper says about the average number of hops required to search and find a key inside the overlay;
3. the speed up about using or not the 1-lookAhead.

2.1 MEASURES ABOUT THE ESTIMATION OF THE NETWORK SIZE

The overlay used is initially composed by 2^5 nodes and after it will expand to 2^{10} nodes and finally will reduce to 2^3 nodes.

In Figure 2.1 is possible to see that the algorithm is quite good to estimates in the average but it's not really reactive, mainly it needs some time before it starts to decrease the estimation. The motivation of this behaviour is because the algorithm use only local information and so it is intrinsically more unstable.

Moreover has been measured that, again for the usage of only local information, there is an high variance between the nodes (for nodes just joined the estimate of 2 and for nodes that are really close each other an overestimated value of 20% of the real one).

As noticed in the paper the number of segments s is not critic and increasing that value doesn't improve a lot the estimation.

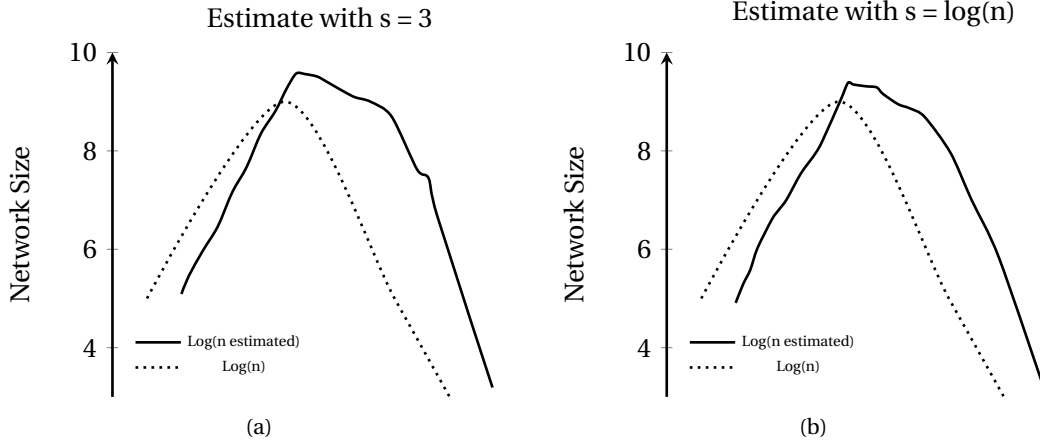


Figure 2.1: Average estimation of the network size

2.2 MEASURES OF THE NUMBER OF HOPS REQUIRED TO FIND A KEY

In order to validate the values presented in the paper has been performed various scenarios where each one change for:

- the routing algorithm;
- the usage of the 1-lookAhead;
- number k of Long Range Links;
- type of the overlay: static, expanding namely increase the size along the time and expanding with the use of the relinking.

First of all we remember that the paper says (actually thanks to the Kleinberg Model [3]) that with $k = O(1)$ Long Range Links, the Latency is $O(\frac{1}{k} \log^2(n))$ where n means the network size. Applying that to the special cases $k = 1$ and $k = \log(n)$ we have that the average hops are: $O(\log^2(n))$ and $O(\log(n))$.

In Figure 2.2 and in Figure 2.3 are reported the graphs varying the network type and the number k (constant) of Long Range Links. As we can see from the graphs for $k = 1$ the average has a quadratic trend and increasing the number of Long Range Links that trend becomes more flat as a linear one.

Notice that the bidirectional routing has (as the paper says) a speed-up of more or less 30% compared to the unidirectional one.

In Figure 2.4 there is the graph with the all types of network and routing algorithms with $k = \log(n)$ and, as the paper says, the trend is linear (or even sub-linear) on the network size.

2.3 MEASURES ABOUT THE SPEEDUP OBTAINED USING THE 1-LOOKAHEAD

The above simulations have been repeated but with the use of the 1-lookAhead and have measured how in percentage the paths are more shorter.

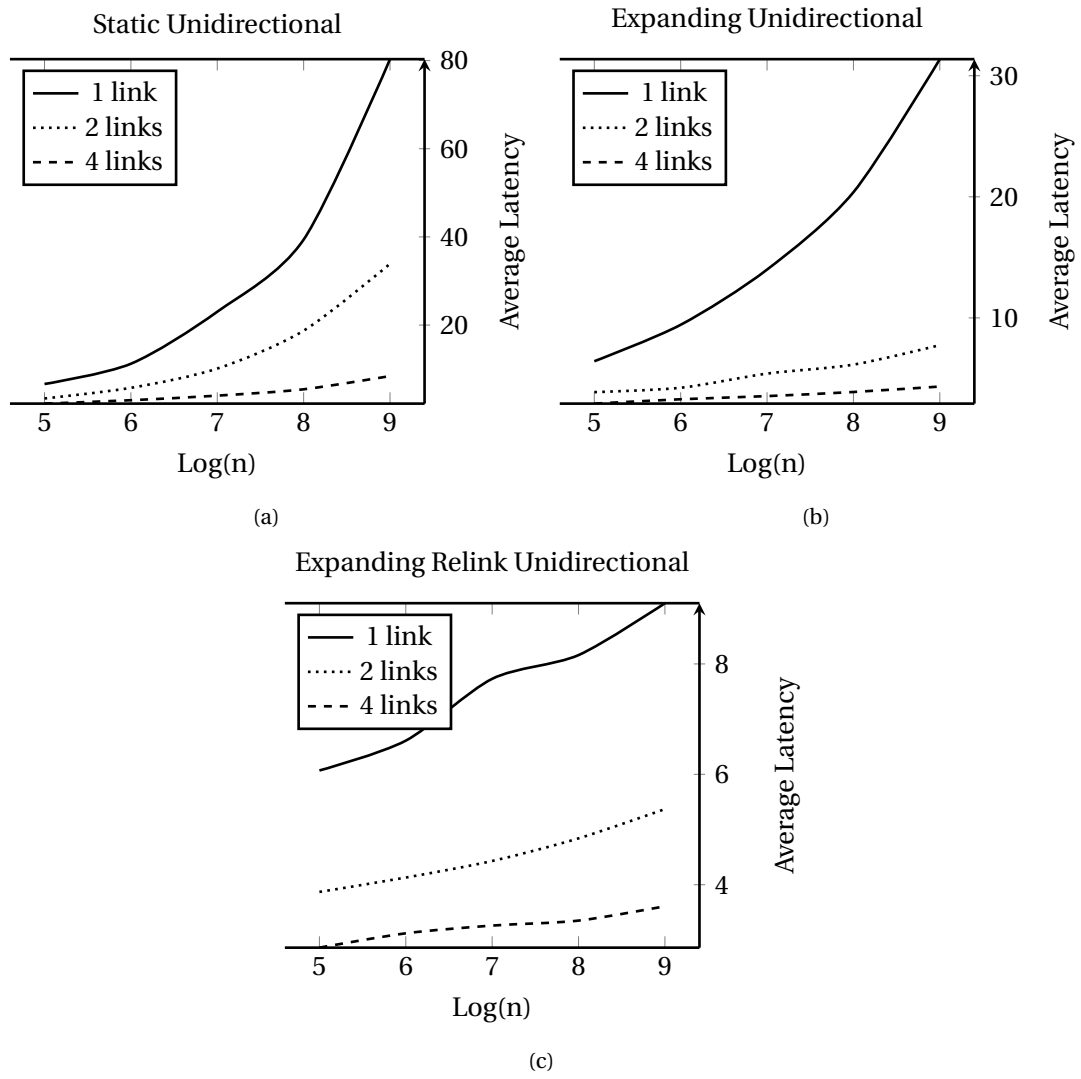


Figure 2.2: Latency Unidirectional Cases

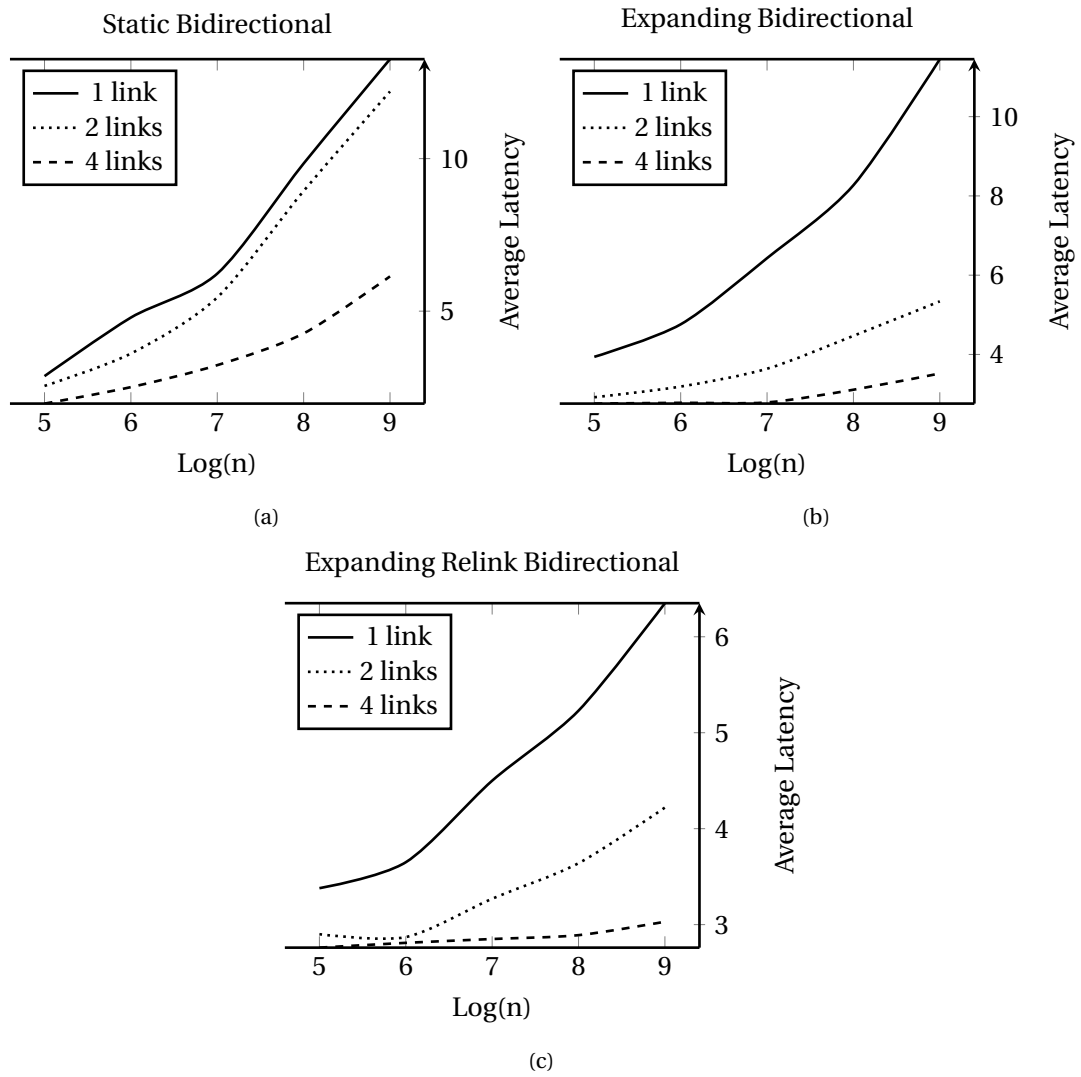


Figure 2.3: Latency Bidirectional Cases

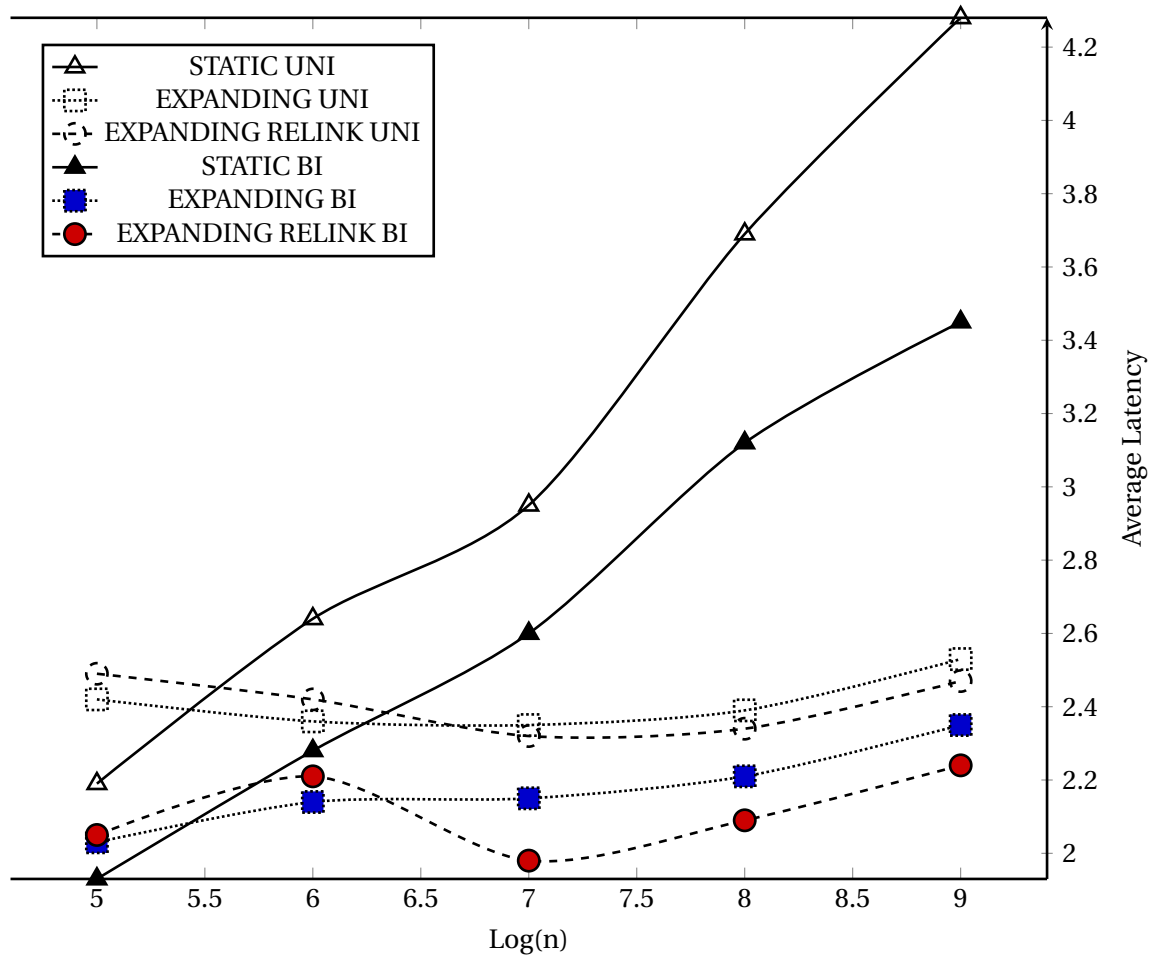


Figure 2.4: Latency with $k = \log(n)$

In Figure 2.5 and Figure 2.6 is possible to see that there is a big speedup like 50% for a low number of links instead, increasing the number of them the percentage decrease to be less than 20%.

Notice that in a more dynamic network the speedup is lower. This happen because the exchanging information between the nodes are less reliable and so could be some inconsistencies.

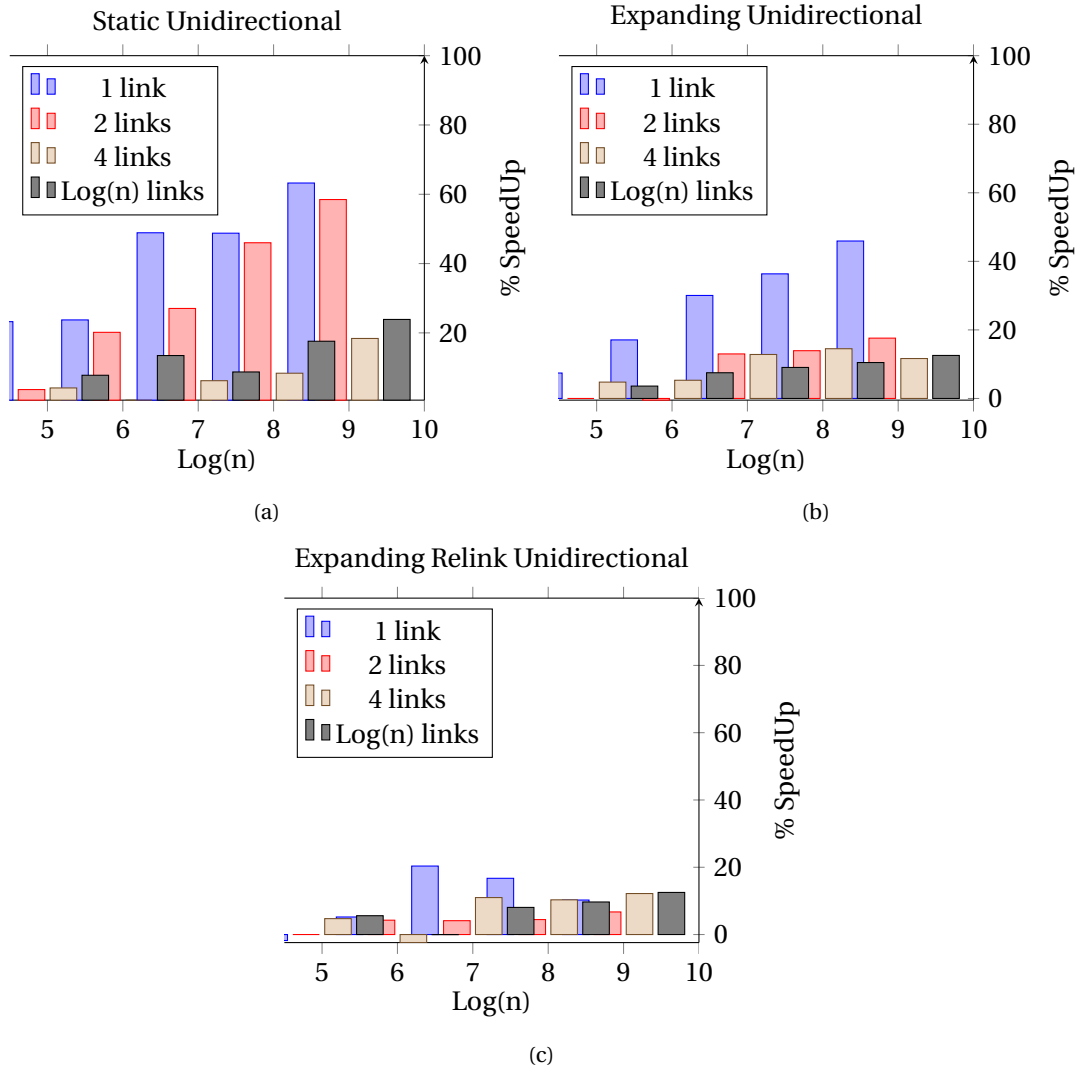


Figure 2.5: Latency Unidirectional Case with the usage of the 1-lookAhead

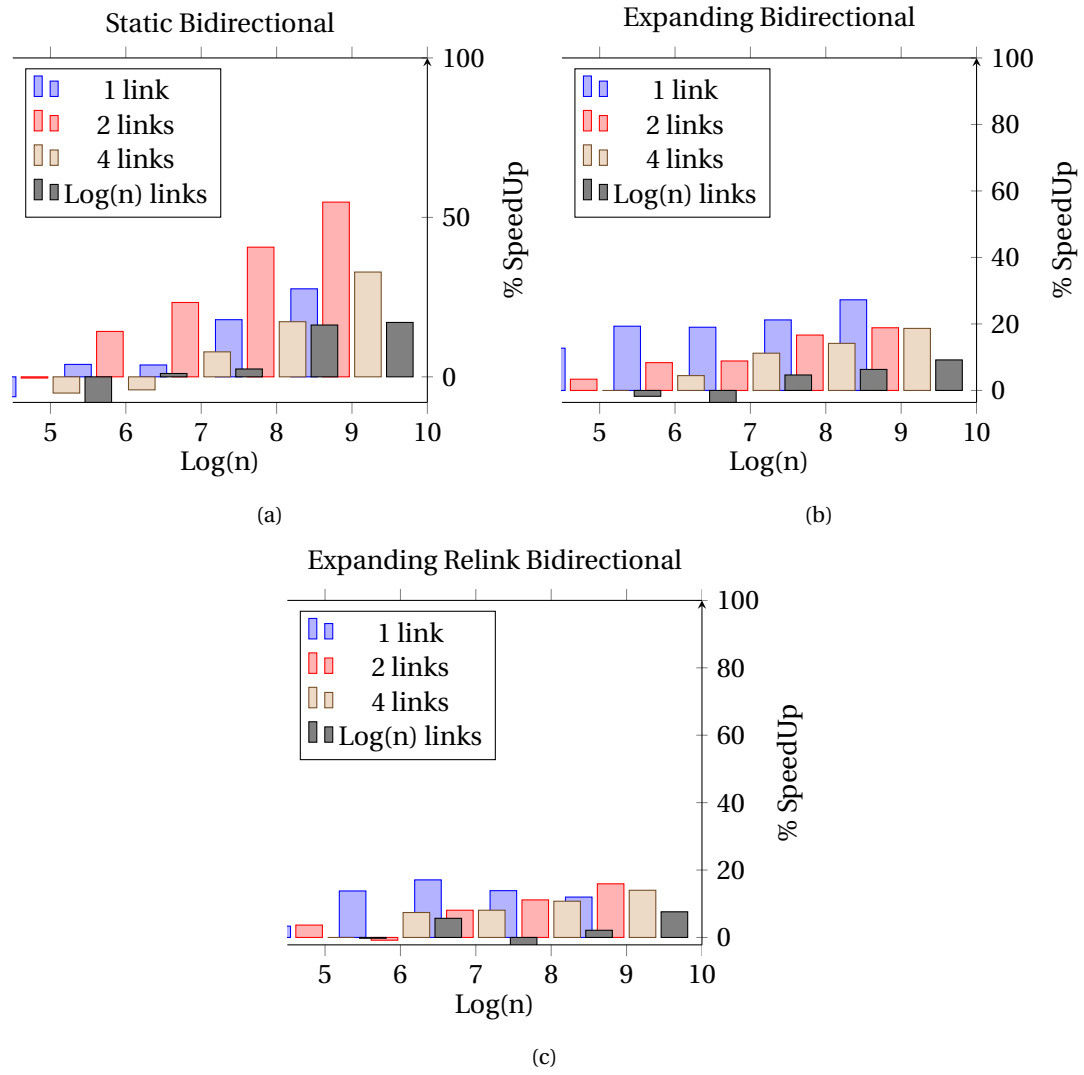


Figure 2.6: Latency Bidirectional Case with the usage of the 1-lookAhead

REFERENCES

- [1] *The Peersim Simulator*, <http://peersim.sourceforge.net/>
- [2] G. S. Manku, M. Bawa and P. Raghavan, *Symphony: Distributed Hashing in a Small World* USITS 2003, 4th USENIX Symposium on Internet Technologies and Systems, pp. 127- -140, March 2003.
- [3] J. Kleinberg, *The Small-World Phenomenon: An Algorithmic Perspective*, 32nd ACM Symposium on Theory of Computing (STOC 2000), 2000, pp. 163-170.