Gambling Heuristic on a Chord Ring

Dario Rossi
Politecnico di Torino – Dipartimento di Elettronica
dario.rossi@polito.it

Ion Stoica
University of California, Berkeley – CS Division
istoica@cs.berkeley.edu

***Abstract*— Chord routing is greedy and *non-symmetric*, and is based on a skiplist-like data structure, whose entries are known as fingers. This work explores the benefits arising from a modified greedy lookup strategy that, without introducing any additional communication overhead, simply exploits the *implicit* symmetry knowledge intrinsic to the highly structured Chord ring. Through extensive simulation on a dynamic peer environment, we show a practical and feasible solution that actually boosts DHT lookup performance under a wide range of scenarios.**

I. Introduction

The aim of this work is to propose a general and viable approach for Distributed Hash Tables (DHTs) that is able to cope with both churn, i.e., the continuous change in system *membership*, as well as changes in the system *size*. Though our mechanism can be adapted to different DHT protocols, this paper presents its application to Chord [1]. The performance of Chord heavily depends, as for any other DHT system, on the parameters setting, which are usually *statically* tuned according to i) system size, ii) churn rate and iii) in reason of the tradeoff between the cost that we are willing to pay for the desired level of performance. However, although peer-to-peer system size is usually assumed to be roughly constant over time, this may be a rather simplistic assumption: therefore, it is important to asses the impact of the network size variation on the performance achieved for different static configurations of the system. Another critical point to stress it that the simple increase of the network connectivity, e.g., through extension of the peer routing tables, may not represent a valid solution at all: indeed, increasing the amount of structured data without increasing its update rate –and thus the associated maintenance overhead– may yield under certain circumstances to performance degradation. Therefore, to take into account the aforementioned problems, we propose an extremely simple mechanism that, without additional communication *overhead*, *dynamically* adds some useful soft-state entries to the peers routing tables. The intuition behind the approach is that the soft-state entries can naturally cope with varying-size systems, as well as significantly ameliorate the routing performance. The rest of the paper is organized as follows. Sec. II describes the proposed modification to the greedy Chord routing, while a preliminary analysis of the system configuration is performed in Sec. III; after evaluating in Sec. IV the impact of the proposed approach, Sec. V overviews the related works and Sec. VI concludes the paper.

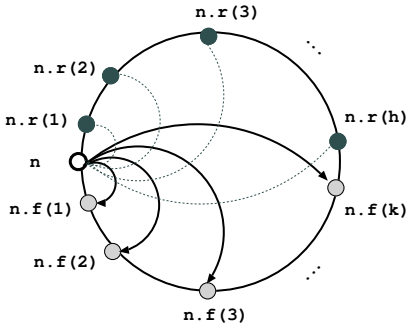

Fig. 1. Fingers and Regnifs in a Chord Ring

II. Reverse Fingers

This work proposes a mechanism that, without introducing any additional communication overhead, exploits the *implicit* symmetry knowledge intrinsic to the highly structured Chord ring. This is done by introducing into the routing table an additional "entity" beyond *successor* (consecutive nodes in the ID space) and *fingers* (entries at exponential distance in the ID space). Assuming reader's familiarity with Chord[1], we introduce the key idea of this work, i.e., the concept of *reverse finger*, with the help of Fig. 1. Adopting the Chord terminology, we may say that a peer f is a *finger* of n when f is an entry of n's finger table; symmetrically, in this case, we may say that n is a reverse finger –or, shortly, a *regnif*– of f, since f is being fingered by n. In Chord routing, each peer n uses its fingers f for routing purposes: therefore, each time f handles node n's request, then f implicitly knows that n is alive; our proposal is to transform this useful but *implicit* knowledge into an *explicit* and used information: by this, we mean simply to allow any peer to use its regnifs for lookup purposes only. However, in the case of failure of a regnif, we impose the lookup to conservatively fail back to Chord routing, i.e., the next-hop finger is chosen according to the original Chord scheme, as will be detailed later.

Since our primary concern is to explore a practical and feasible approach, an appealing solution is to consider a cache of *soft-state* regnif entries, which automatically expire after a timer T expiration. In order to select reasonable values for T, let us define the stabilization *period* S as the time elapsed among two consecutively scheduled execution of the stabilization process for two distinct entries in the finger table. We further define the stabilization *round* S_R as the time

elapsed among two consecutive stabilizations of the same entry in the finger table: during this interval, every other entry of the routing table is stabilized exactly once. Clearly, this time interval can be expressed as a function on both the stabilization period S and the number of finger table entries; indicating with N and F respectively the number of successors and fingers populating the routing table, the stabilization round can be written as S_R=(N+F)S. Let consider a finger f, whose stabilization process ended at time t_0, and a regnif r, added to the regnif cache at the same instant t_0; intuitively, the information carried by f and r has the same degree of reliability until the next stabilization of finger f, which will happen, on average, at time t_0+S_R; thus, selecting T on the order of S_R will yield a regnif cache which is as accurate as a finger cache without incurring in any additional stabilization cost. However, since there is no gain without gambling, we may decide to raise the regnif grace timer T well beyond S_R: indeed, on the worst case, the regnif-hop will incur in a timeout and the lookup will continue to the finger-hop selected by the original Chord routing.

Let now examine more in details how regnifs can be used in Chord; the more evident difference of the modified routing algorithm, which we will call in the following simply as Regnif, is constituted by the disposal of a larger number of *heterogeneous* routing entries, i.e., fingers and regnifs. However, irrespectively of the kind of next-hop entry chosen, routing is greedy and, as in the original Chord, subsequent steps are naturally sorted by decreasing ID-space length: thus, under this light, the extended cache does not affect the routing convergence properties and may rather mean a closer next-hop. A second and more subtle variation lies in the different *degree of trust* accorded to fingers and regnifs. Indeed, let consider for simplicity the iterative flavor of the greedy lookup, and assume that a node n has a request for a key k. At the first hop, the node n will: i) choose from its finger table, *à la* Chord, the finger $n.f$ nearest to the destination k; ii) choose from its regnif table the entry $n.r$ nearest to the destination k; iii) forward the request to the closest between the finger and the regnif, i.e. $p_1 = \arg\min_{n.f,n.r}(|k-n.f|_{ID}, |k-n.r|_{ID})$. Without loss of generality let assume that p_1, the first-hop node on the path along n and k, is alive and reachable. From now on, at the i-th hop, peer p_i performs a similar lookup, replying to n with a $(p_i.f, p_i.r)$ pair, and node n will prefer for its next hop the closest between the finger and the regnif. The aforementioned different degree of trust accorded to fingers and regnifs arise upon failure of the latters: indeed, in order to be conservative, if the contacted regnif entry $p_i.r$ fails through a timeout expiration, then the finger $p_i.f$ is used. This failback to Chord routing intuitively guarantees that Regnif's lookup success probability is not worse[1] than Chord's – which clearly introduces a tradeoff: the lookup probability is preserved to the detriment of the number of timeout experienced per lookup. Finally, a special attention is devoted to the first hop, since each peer may use a regnif of its *own* cache. Indeed, upon failure of the first-hop regnif r, this latter is vetoed[2] from being used for routing purposes, unless its status is refreshed before the soft-state timer expiration.

[1]Apart from the rare, though possible, case where: i) at time t_0, n forwards the request for k toward the regnif r, which is closer to k than finger f; ii) n acknowledges the failure of regnif r at time $t_0 + T_{timeout}$; iii) finger f leaves the system between t_0 and $t_0 + T_{timeout}$, but was active at time t_0 and could have possibly, depending on its load, handled n's request

III. Preliminary System Configuration

In order to explore the parameter space of Chord and Regnif into a dynamic environment we performed extensive simulations considering networks of different sizes: unless otherwise stated, we will refer to an average network size of 4096 nodes. Both the intra-node latency and the request processing time are exponentially distributed, with mean equal to 50 ms; timeout expiration is set to 500 ms, and request queues at nodes never overflow. We also take into account different churn rates: by default, nodes activity and inactivity times are exponentially distributed with mean equal to 1 hour, as reported in [2]. Nodes identifier do not persist across subsequent sessions, and are rather chosen uniformly at random. During its lifetime, each node performs several lookups for keys selected uniformly at random; lookups are generated according to a Poisson process and are spaced of 1 minute on average. Every simulation point corresponds to the average of five simulation runs with different initial seed; neglecting the initial transient, the number of lookup performed during each simulation run is proportional to the network size, specifically about $1.5 \cdot 10^6$ for a 4096-nodes networks; notice that this is done to ensure the simulation time to span over a significant number of stabilization rounds.

For reasons of space, we limit the analysis to only some of Chord's and Regnif's parameters, namely i) the stabilization period S, ii) the regnif timer T and iii) the regnif cache size R.

However, it is known that Chord successors and fingers have different impact on the system performance: successor pointers ensure lookup *correctness*, while finger provides low path length and latency and are thus tied to lookup *performance*. It should be said on this point that it may be possible to *decouple* the stabilization procedure, e.g., by lowering the successor's stabilization interval in order to enforce better lookup success rates; however, for the sake of simplicity, we do not consider the possibility of separately tuning the fingers and successors stabilization intervals. For brevity, we will report in the following results only the results achieved for N=F, indicating with C_x and R_x respectively a Chord and a Regnif rings having N=F=x; we will consider both over- provisioning and under-provisioning strategies, where the threshold for a well dimensioned system is considered to be the logarithm of the average system size. Besides, we experimentally verified that i) system performance benefits of a large number of fingers F, provided that the stabilization

[2]This is done to avoid to immediately evict a regnif from the cache, since the timeout may be due to congestion at the peer request queue rather than actual peer inactivity

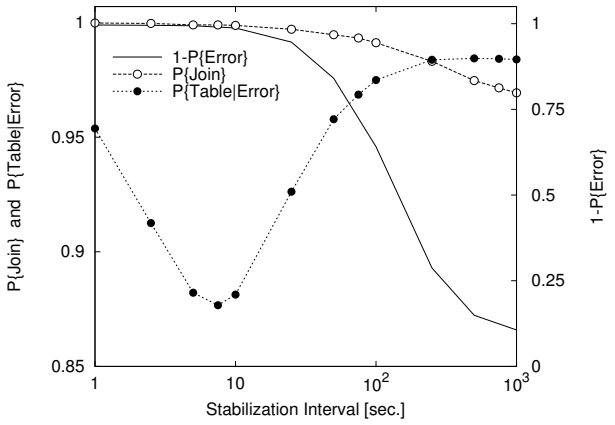

Fig. 2. Stabilization Interval: Lookup Success, Joining and Routing Table Inconsistency Probabilities

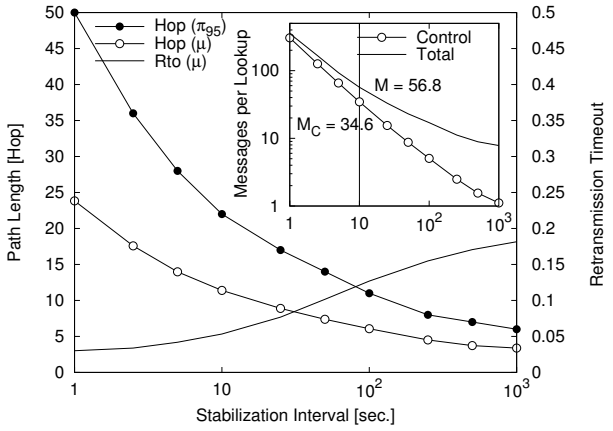

Fig. 3. Stabilization Interval: Path Length, Timeout and Communication Cost

epoch is much smaller than the average peer lifetime: otherwise, the time needed for the stabilization of the whole finger table can actually result in performance degradation; ii) as the number of fingers F increases, system performance are less sensitive to the successor number N and iii) indiscriminately raising the successor number N may result in a decrease of the lookup success rate. These considerations suggest that, under certain circumstances, over-provisioning can be as bad as under-provisioning.

A. Chord: Stabilization Interval

The stabilization interval clearly has a massive impact on Chord system performance: the aim of this section is to quantitatively investigate its importance. First of all, Fig. 2 depicts three curves as a function of the stabilization interval S, expressed in seconds, on the x-axis: namely, i) the lookup success probability, 1-$P\{Error\}$, ii) the probability of successfully joining the system, $P\{Join\}$ and iii) the probability that lookup errors are due to inconsistence of the routing table entries, $P\{Table|Error\}$. As expected, as the stabilization interval increase, the information contained in the routing tables is less correlated with the real system status: as a consequence, the probability of joining the system $P\{Join\}$, as well as the probability of performing a successful lookup 1-$P\{Error\}$ once joined the system, drop significantly: for instance, about 10% of the lookup succeed when S=10^3 seconds. In order to analyze the $P\{Table|Error\}$ results it must be said that inconsistent lookups are false-positive (i.e., in the sense that the lookup reaches a node which is not the true responsible for the key), while in the other cases the lookup terminates early (e.g., reaches a dead node with no further possibility to backtrack). This said, it is interesting to notice that a too small stabilization interval may result in *over-sampling* phenomena: this can be gathered observing the unexpected increase of routing table inconsistence $P\{Table|Error\}$ for values of S below 7.5 seconds. Besides, the probability of failure in joining the system keeps lower than the lookup failure probability.

A very interesting observation arise from the observation of Fig. 3, which depicts the average and 95^{th} percentile of the path length (on the left y-axis) as well as the average timeout number (on the right y-axis), as a function of the stabilization interval. It can be gathered that, as expected, the average timeout number increases as S increases, whereas, rather counter-intuitively, both the average and the 95^{th} percentile of the path length actually *decrease* as S increases. This, rather than being an index of better performance, reflect the actual limitation of the network horizon as seen by the peers: indeed, the longer the path, the higher the probability to incur in several timeouts and, eventually, the higher the probability for the search to fail, as evidenced in Fig. 2: in other words, by increasing S longer paths simply "vanish" from the system. Finally, let consider the Chord communication complexity, considering two distinct contributions: i) the communication overhead, which is expressed as the average number of control messages M_C per lookup and ii) the pure-lookup messages M_L, i.e., the average number of messages needed to route on the ring, purged from the Chord maintenance messages. The total cost of a lookup can be then expressed as the total number of messages M=M_C+M_L seen, on average, by each lookup. On the one hand, M_C represents the pure Chord overhead and, beside the dependence on the path length, is inversely proportional the stabilization interval S; on the other hand, M_L reflects the system performance and, in the case of iterative lookups, is equal to twice the average path length plus backtracking-hops contributions due timeout expirations. It should be said that these two contributions are tightly correlated: indeed, a higher maintenance cost M_C may result in better path length and timeout performance, eventually lowering the number of messages M_L needed per lookup. With these considerations in mind let observe the inset of Fig. 3, which depicts the communication overhead, expressed in terms of M and M_C, as a function of the stabilization interval. As expected, the overhead M_C decreases linearly as the stabilization interval increases; besides, the slope of total number of exchanged messages, M is also affected by two contrasting contributions (i.e., the decreasing path length and the increasing timeout number) and shows a slower decrease.

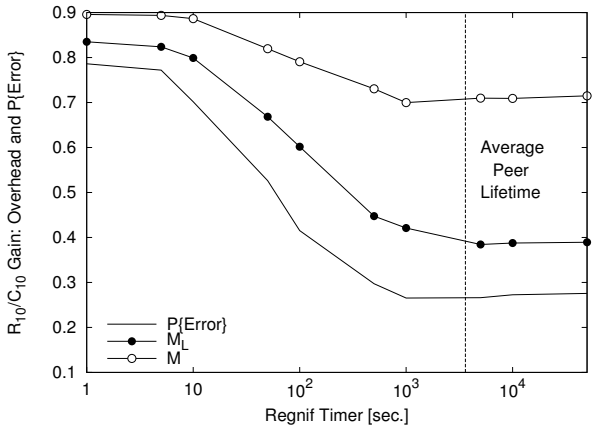

Fig. 4. Regnif Timer: Lookup Success and Communication Overhead Gain

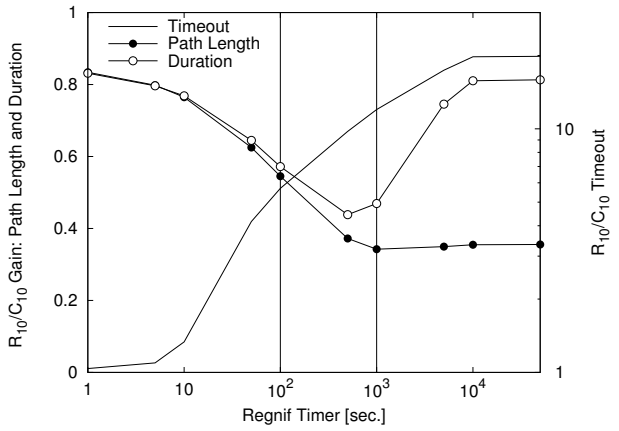

Fig. 5. Regnif Timer: Pathlength Gain and Timeout Increase Tradeoff

In the following, we set the default value of the stabilization interval to be S=10 seconds as a good tradeoff among the success probability, path length timeout and communication overhead (as evidenced in the picture, the control messages account for M_C/M=60%).

B. Regnif: Soft-State Timer

To investigate the impact of the Regnif soft-state timer T, we consider an under-provisioned R_{10} ring, using furthermore a finite Regnif cache, on a 4096-nodes network. Performance will be normalized over the results achieved on a Chord ring under the same configuration: it must be pointed out that C_{10} is considered to be well provisioned only for 1024-nodes networks. Fig. 4 depicts the error percentage decrease and the communication overhead gain, normalized over the results achieved by Chord under the same configuration, as a function of T. As expected, the $P\{Error\}$ gain is more evident as T grows, since Regnif performance is lower-bounded by the standard Chord-routing failback: in this scenario, up to 60% of the lookup errors may be avoided by the introduction of the Regnif cache. When, conversely, the timer T grows beyond the peer lifetime, the performance gain decreases almost beyond perception: in this case, some "useless" entries are wasting the finite regnif cache space. It should be noted that communication overhead decreases significantly: thus, not only the Regnif routing comes with no additional overhead, but can also actually *lower*, of about 30% in this experiment, the total communication cost M; this is essentially a consequence of the path length and network load reduction, as M_L testify.

Fig. 5 depicts the R_{10}/C_{10} ratio of the average timeout number (on the right y-axis), path length and lookup duration (on the left y-axis) as a function of the regnif timer T. As expected, the introduction of the regnif cache yields to a significant decrease of the path length at the price of a non-negligible increase of the timeout number. It is interesting to notice how the tradeoff between the path length and timeout drives the temporal duration of a lookup. When $T>10^3$ seconds, the lookup duration is negatively affected by the massive timeout increase; conversely, when the increase of the timeout number is modest, e.g., $T<10^3$ seconds, the lookup duration actually benefits of the path length reduction. Intuitively, the decrease of the path length translates into lighter load on the network, which in turns yields, on average, to shorter request queues for each peer: this compensate the increase of the timeouts number and eventually entails the reduction of the lookup duration. In the following we will set the soft-state timer to one of the references T=$\{10^2, 10^3\}$ seconds indicated in Fig. 4: the lower timer choice prefers system stability, while the higher timer follows from the belief that the higher bet, the higher can possibly be the gain.

C. Regnif: Cache Size

Let investigate the effects of the Regnif cache size when the Regnif timer is set to $T \in \{10^2, 10^3\}$ seconds; as before, network size is 4096-nodes and peers form a R_{10} ring.

To avoid cluttering the figures, we present in Fig. 6 only the timeout *degradation* and the path length *gain* as a function of the regnif cache size on the x-axis, for different values of the regnif timer T. The timeout degradation is expressed as the Regnif over Chord ratio R_{10}/C_{10} of the average timeout number; symmetrically, the path length *gain* is expressed as ratio C_{10}/R_{10} of the average path length. There are two main observation on this regard: first of all, the regnif cache may be extremely beneficial into under-provisioning scenarios, although as observed early the timeout degradation may tradeoff the path length gain. Secondly, the effect of the cache size saturates for both timer T values: i.e., adding more regnif yield no further gain nor degradation. Therefore, in the following we select the cache size to be 100 elements large.

IV. Regnif Performance Evaluation

This section compares Chord versus Regnif performance. Inspecting the impact of the network size and the peer activity model, we focus on on user-centric metrics such as the lookup success rate, path length and duration, timeout number. Conversely, we will no longer directly refer to system-wide metrics; indeed, it should be noted that both the $P\{Join\}$ probability and the the control messages overhead M_C are

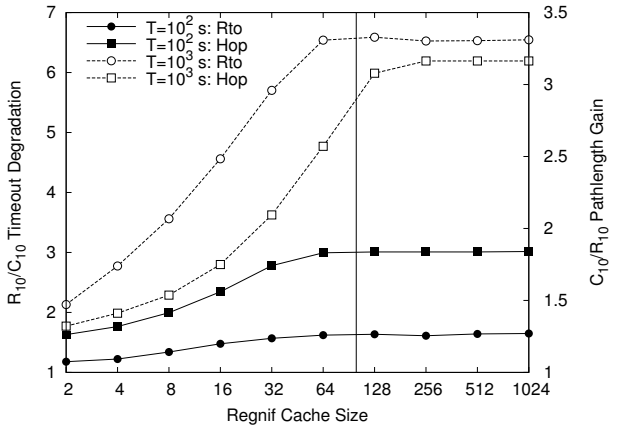

Fig. 6. Regnif Cache Size: Timeout Degradation and Pathlength Gain

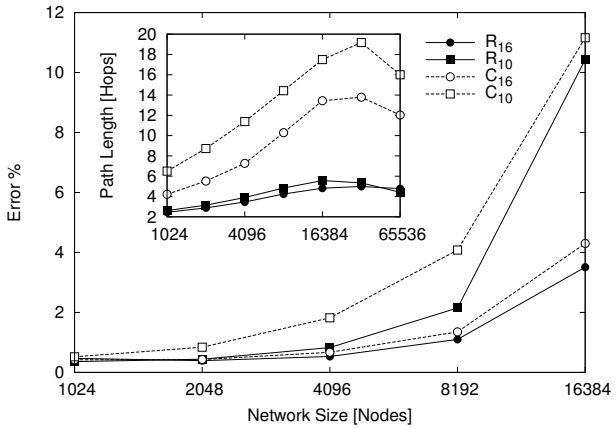

Fig. 7. Network Size: Regnif vs Chord Error Probability and Path Length

unaffected by the use of Regnif for solely *lookup* purposes, as in our case. However, as we previously observed, many of these metrics tradeoff and, being very complex to give a complete evaluation of the quality of the offered service, we need to distinguish at least a sort of ranking between them. We argue that, even though the duration of a lookup is extremely important to the user perception, nevertheless this should be weighted against both i) the user reaction times, which are in the order of unit to tens of seconds and ii) the subsequent download time, which is likely to be bigger than the lookup duration for a wide range of applications. Therefore, we are allowed to state that the success probability of the lookup itself is more important than its duration: in other words, guaranteeing the resource availability is more important than bounding the delay of its localization. Similarly, it could be argued that the number of hops in the path may be even more critical than the number of timeouts: indeed, incurring in a timeout does not necessary yield to lookup failure nor to longer lookups; moreover, as we previously noticed, shorter paths may translate into lower network load, which is beneficial to the whole system.

A. Network Size Effects

In this scenario we analyzed, for network size in the range from 1024 to 65536 peers, the performance of different provisioning policy; specifically, we considered i) *over-provisioned* C_{16}, R_{16} systems, where thus F=N=$\log_2(65536)$, ii) *well-provisioned* systems with F=N=$\log_2$(Peer) and iii) *under-provisioned* C_{10}, R_{10} systems. It should be noted that the well- and over-provisioning strategies overlap for the largest network size, whereas the well- and under-provisioned cases overlap for the smallest network size. Fig. 7 shows the Regnif and Chord lookup error percentage as a function of the network size, in the over- and under- provisioned cases only, thus providing a lower and upper bound to the average system performance. As a first consideration, the drastic degradation of the lookup probability for network sizes above 8192 peers is partly due to the choice of a small Chord finger base $b = 2$. This is especially true for under-provisioned systems: for example, the use of $b = 16$ on a 16384-nodes network more than *halves* the lookup errors in the C_{10} case; conversely, for network sizes below 4096 nodes, changing the base yielded no significant performance improvement: this suggest that, at least on large networks, in order to ensure good performance a careful tuning of the finger base is required. Secondly, and even more interestingly, it should be noted that the Regnif solution provides remarkable performance benefits for under-provisioning scenarios: e.g., the performance of R_{10} and C_{16} are similar for a 4096-nodes network. This means that, as expected, the Regnif cache is able to naturally cope with significant variation of the system size. Furthermore, it should be recalled that these results refer to a relatively small regnif cache $R = 100$: extending the cache size may yield further gain. The inset of Fig. 7 depicts the average path length as a function of the network size for different routing flavors and provisioning strategies. First of all, the path length increases from small to moderate network sizes; then, for network larger than 16834 nodes, the path length decreases: however, as we previously noticed, this *apparent* decrease must be coupled to the dramatic increase of the lookup failure. Secondly, the Regnif path length is evidently less affected, with respect to Chord, by the increase of the network size – which again confirms the effectiveness of the regnif cache. Finally, intra-system differences are less evident for different Regnif configurations (i.e., R_{10} and R_{16}) with respect to the Chord case, where under- and over-provisioning strategies exhibit a significant performance gap .

Finally, Fig. 8 depicts the average timeout number as a function of the network size. Interestingly, the relative order of the performance curves is the opposite in the Chord and Regnif cases for different provisioning strategies. This is easily explained considering that the use of Regnifs will likely higher the timeout probability: as a consequence, an under-provisioned Regnif R_{10} ring will be more likely to use its Regnifs than an over-provisioned R_{16} one, thus raising the average number of timeout per lookup. Secondly, in the Chord case, the over-provisioning strategy does not entails a reduction of the timeout number *per se* for all network sizes,

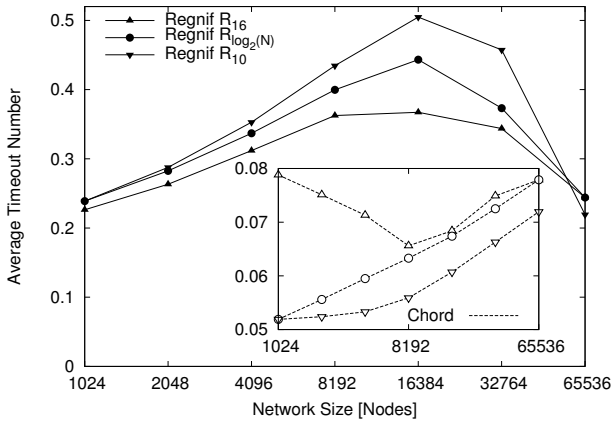

Fig. 8. Network Size: Regnif vs Chord Average Timeout Number

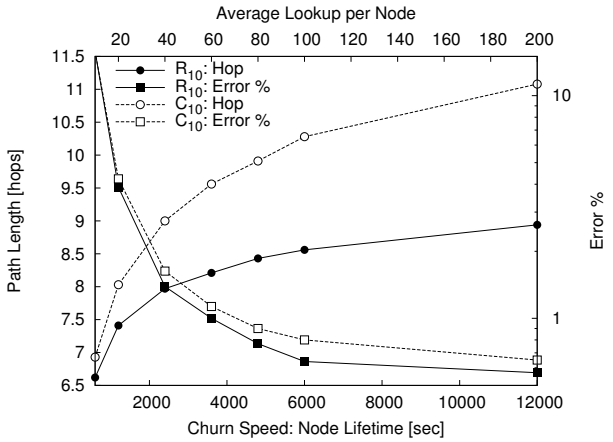

Fig. 9. Node Lifetime: Error Probability and Average Path Length

as testified by the C_{16} curve: the increase of the stabilization epoch in the C_{16} case entails a lower accuracy of the larger finger cache and is responsible for an higher number of timeouts experienced by 8192-peers or smaller networks.

B. Churn Model Effects

Eventually, let investigate the impact of the node lifetime on the system performance; let us anticipate that, considering an under-provisioned 4096-peers network, Regnif outperforms Chord under both fast and slow churns.

Fig. 9 depict the lookup failure percentage (on the right y-axis) and the average path length (on the left y-axis) as a function of the peer lifetime, expressed in seconds, on the bottom x-axis; besides, being the lookup rate constant and equal to 1/60 Hz, the top x-axis report the number of lookup effectuated per node. As the peer lifetime increases, both systems achieve more stable results: the lookup failure probability decreases and, under this light, the path length increase reflects a widening of the network horizon as seen by the average peer. As a consequence, the average lookup duration increases as well, whereas the decrease of the 95^{th} percentile of the lookup duration has instead to be coupled to the reduction of the timeout number.

V. Related Works

Since Chord's [1] first description many modifications to the protocol have been proposed and analyzed, some of which we will very briefly summarize here. For example, [3] address more sophisticated (and demanding) stabilization routines with stronger guarantees, while Koorde [4] proposes the combination of a ring structure with the low-diameter De Brujin graphs. In [5] a degree optimal version of Chord is obtained applying the small-word concept of neighbors of neighbors (NoN), first introduced in the DHT context by [6]. Also, [7] combines the benefits of the NoN approach with a different intra-finger distance distribution following the Fibonacci series, while bidirectional optimal routing scheme is presented in [8]. Finally, proximity matter have been considered, e.g., in [9] which investigated the benefits brought by topology-aware mechanism on the top of different DHTs, whereas load balancing issues have been addressed in [10].

VI. Conclusion

This paper proposed a zero-cost modification to Chord routing, based on eavesdropping of the request handled for other peers, which is viable and easily implementable. Through extensive simulation on a wide range of dynamic scenarios, we showed that the proposed technique is able to bring performance benefits, in terms of shorter path length and latency, at the price of the increase of the timeout number experienced by lookups.

References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, Vol. 11, No. 1, pp. 17-32, Feb. 2003.

[2] S. Saroiu, P. Gummadi and S.D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *In Proc. of Multimedia Computing and Networking* (MMCN), San Jose, CA, Jan. 2002

[3] D. Liben-Nowell, H. Balakrishnan and D. Karger, "Observations on the Dynamic Evolution of Peer-to-Peer Networks," *In Proc. of the 1st International Workshop on Peer-to-Peer Systems* (IPTPS), Cambridge, MA, Mar. 2002

[4] F. Kaashoek and D. R. Karger, "Koorde: A Simple Degree-Optimal Hash Table," *In Proc. of the 2nd International Workshop on Peer-to-Peer Systems* (IPTPS), Berkeley, CA, Feb. 2003

[5] G. S. Manku, M. Naor and U. Wieder, "Know thy Neighbor's Neighbor: The Power of Lookahead in Randomized P2P Networks," *In Proc. of the 36th Symposium on Theory of Computing* Chicago, IL, Jun. 2004

[6] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed Hashing in a Small World," *In Proc. of the 4th USENIX Symposium on Internet Technology and Systems* (USITS), Seattle, WA, Mar. 2003

[7] G. Cordasco, L. Gargano, M. Hammar, A. Negro and V. Scaran, "Non-uniform deterministic routing on F-Chord," *In Proc. of the 1st International Workshop on Hot Topics in Peer-to-Peer Systems* (HotP2P04), Volendam, The Netherland, Oct. 2004

[8] P. Ganesan and G.Manku, "Optimal routing in Chord," *In Proc. of the 15th ACM Symposium on Discrete Algorithms*, New Orleans, Jan. 2004

[9] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker and I. Stoica, "The Impact of DHT Routing Geometry on Resilience and Proximity," *In Proc. of SIGCOMM*, Karlsruhe, Germany, Aug. 2003

[10] B. Godfrey, B. Lakshminarayanan, S. Surana, R. Karp and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems", *In Proc. of IEEE INFOCOM*, Hong Kong, Mar. 2004.