

PariSync: Clock Synchronization in P2P Networks

P. Bertasi, M. Bonazza, N. Moretti, E. Peserico

Department of Information Engineering

University of Padova

Email: {bertasi, bonazzam, morettin, enoch}@dei.unipd.it

Abstract—This paper presents PariSync, a distributed system for clock synchronization in DHT-based peer to peer networks. PariSync is formed by two modules: a topology module, that chooses for each node a small subset of neighbors with which to exchange timing information (piggybacking on the DHT link structure) and an estimation module, that assembles the information into an estimate of the node’s offset and drift from a global virtual clock emerging from the consensus of all peers. PariSync works on extremely large peer-to-peer networks (millions of nodes) exhibiting good performance even in the presence of churn and malicious nodes. We provide a version of PariSync in pure Java and in JXTA.

I. INTRODUCTION

This paper presents PariSync, a system for efficient clock synchronization in large Peer to Peer networks implemented in Java and compatible with the JXTA framework [1]. This introduction briefly reviews the problem of clock synchronization (Subsection I-A) and motivates and summarizes our work (Subsection I-B).

A. Clock Synchronization

Clock Synchronization is a problem that has been extensively studied over many years, both in the distributed systems community, and (for its great practical importance) by the networking community. In a nutshell, it consists in having all the nodes in a distributed system agree on a common virtual clock, ideally “sufficiently close” to real time (perhaps with a few “hints” from accurate, external sources) despite variable communication delays and misbehaving nodes.

The distributed systems community typically casts the problem of clock synchronization as follows. Assume a network of n nodes v_1, \dots, v_n , all with communication delays, all with clocks, initially perfectly synchronized but with potentially different speeds. Each node v_i witnesses a sequence of events e_i^1, e_i^2, \dots . The goal is to develop a communication protocol that allows nodes to reach a consensus on a virtual time $t(e_i^j)$ to be assigned to each event e_i^j so that no event witnessed by a node is timestamped “out of order”. Ideally $t(e_i^j)$ should also lie within a multiplicative factor (bounded away from both 0 and ∞) of the real time at which e_i^j occurs, and the protocol should tolerate the largest possible number of “Byzantine” nodes - nodes that maliciously conspire, with perfect coordination and information, to make the protocol fail (a conservative model both of nodes controlled by actively malicious actors and of accidentally misconfigured nodes). Roughly speaking, protocols exist [2], [3] that tolerate a very large fraction of byzantine nodes (from $\approx \frac{1}{3}$, to $\approx \frac{1}{2}$ if a

shared digital signature system is available to all nodes). On the other hand, these protocols tend to require from each node considerable space and considerable bandwidth (both linear or polynomial in the size of the network), and so they are not really capable of scaling to networks with millions of nodes. The guarantees they provide on the timestamp assigned to each event are also relatively lax: many P2P applications require virtual time to remain within at most a few seconds from real time, and often less than a second.

The networking community has instead focused on less pessimistic models in terms of malfunctioning nodes and communication errors/delays ([4], [5], [6], [7]), and assumed the presence of a few trusted “correct” clocks (e.g., the tightly synchronized atomic clock servers of NTP) to provide lower communication overheads and tighter clock synchronization — indeed, some work aims at sub-millisecond and even, for LANs, sub-microsecond synchronization [8]. Unfortunately, these algorithms tend to rely on tree-like structures that are less robust, since a single failure close to the root of the tree (particularly a “malicious” one) can cause disastrous effects on a large fraction of the network.

B. Why PariSync?

We believe that neither of the basic mechanisms adopted in the two communities suits large P2P networks. In particular, a clock synchronization system for such networks should implement a virtual, global clock (allowing each node to accurately estimate its relative offset and drift) in a fashion:

- 1) Scalable in terms of node bandwidth — requiring the bandwidth used by each node to scale at most logarithmically in the size of the network.
- 2) With synchronization errors comparable to the communication delay between a pair of nodes (in practical terms, a few hundred ms over the Internet).
- 3) Tolerant of a small, but non-vanishing fraction of actively malicious nodes and link failures.
- 4) With maximum availability (subject to the connectivity of the P2P network) despite firewalls, lack of superuser permissions etc.
- 5) Ideally piggybacking as much as possible on the existing DHT (Distributed Hash Table) infrastructure, to minimize communication overhead.

PariSync is a system for DHT based P2P networks that achieves the aforementioned goals. PariSync is implemented in Java and compatible with the JXTA [1] framework; it runs in userspace without requiring administrator privileges. In a

nutshell, PariSync is formed by two modules: a *topology* module that chooses, for each node, a small subset of neighbors with which to exchange timing information, and a *estimation* module, that chooses how to process that information to estimate the current time and speed of the global virtual clock and/or upper and lower bounds on them.

Two points are worth noting, in particular in terms of comparison with the very successful NTP (and similar protocols such as SNTP). First, although NTP access is often less restricted by network administrator policies than P2P traffic, by definition the only P2P systems on which one needs clock synchronization are those that do allow P2P traffic (it's pointless synching with a P2P network if one cannot communicate with it). Thus, *a system that piggybacks on existing P2P protocols will always be available to synchronize those protocols, whereas NTP may not be* - in fact, every large P2P network is virtually guaranteed to have a sizable fraction of all its nodes unable to employ NTP, whether because of misconfiguration, traffic restrictions, or lack of administrator privileges to run or configure the NTP daemon. Second, NTP and other similar protocols relying on a tree-like structure do allow peering between nodes at the same level of the tree to achieve greater robustness. However, none of these protocols defines the peer link structure, and showing how to set-up such a link structure in a way that allows scalable, effective synchronization is, indeed, one of the main contributions of this paper.

The rest of the paper is organized as follows. Section II, after briefly reviewing DHTs, describes and motivates the structure and operation of PariSync. Section III provides some preliminary experimental evaluation. Finally, Section IV summarizes our results, comments on their significance, and looks at some directions of future (and current) work.

II. SYSTEM DESCRIPTION

After briefly reviewing the structure of a "typical" Distributed Hash Table like PariDHT, Kademlia, Chord or Pastry (in Subsection II-A), this section describes the algorithm behind PariSync (in Subsection II-B).

A. Distributed Hash Tables

As stated above one of the strength of PariSync is to exploit the DHT on which is run. We give here a brief description of a typical DHT. After a surge of interest in the academic community (e.g. Chord [9], CAN [10], Pastry [11], Kademlia [12]) recent years have seen DHTs adopted in several applications with a vast public (e.g. the popular eMule [13] and Azureus [14] filesharing clients and the JXTA system [1]). While DHTs can be implemented by many different data structures, virtually all the mainstream ones (including all the ones we cited above) function according to a basic scheme that we summarize below.

Roughly speaking, each node in a DHT is assigned a random address in a b -bit ID space (b is chosen sufficiently large, typically 160 or more, to avoid collisions). Some form of distance (pseudo-)metric is defined on this address space (e.g.

the XOR metric for Kademlia [12]), so that one can partition the space, for any given node, in the 2^{b-1} addresses in the "other half" of the network, the 2^{b-2} addresses in the same half but in the other quarter, the 2^{b-3} addresses in the same quarter and the other eighth, and so on. Each node then keeps contacts with a small number k of nodes in the other half of the network, k nodes in the other quarter, k in the other eighth and so on (see Figure 1). Theoretically $k = 1$ would suffice, but in practice some redundancy is introduced to provide robustness and typically $5 \leq k \leq 20$ is used. Of course less than k nodes might be present in some of the smallest regions, in which case all the nodes in any such region are kept as contacts.

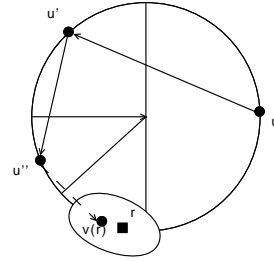


Fig. 1. The search structure in a typical DHT.

Each resource r (e.g. a file) is also mapped into the same address space using a pseudorandom hash of the keyword(s) that will be used to locate it; information about how to reach it (e.g. the IP of the machine from which it can be accessed) is stored in the node $v(r)$ closest to it in the address space. To locate $v(r)$ — whether to retrieve the information on how to access r , or to store it in the first place — a node u will forward the query to the node u' , among its contacts, closest to r in the address space. In the worst case, u' will be in the other half of the network — but it will certainly be in the same half as r . It can then forward the query to another node u'' that will certainly be in the same quarter as r — and so on, until $v(r)$ is reached in a number of steps with high probability logarithmic in the size of the network.

B. PariSync

This subsection provides a description of the PariSync, motivating its design choices. The basic goal of PariSync is to allow a system of peers in a DHT based P2P network to agree on a common virtual clock that should run as closely aligned to real time as possible. Each node maintains, at all times, its own clock, but also attempts to maintain (with good approximation) its offset and drift w.r.t. the global virtual clock, and/or upper bounds on this offset and drift.

The two fundamental difficulties in estimating offset and drift lie in "noisy" communication delays between nodes (as in classic clock synchronization) and in the fact that, since we want the algorithm to scale to networks of millions of nodes, each node should be allowed to communicate directly with only a small subset of the whole network, ideally piggybacking on the pre-existing DHT link structure. Thus, the main problem can be decomposed into two, almost orthogonal

subproblems, that are, in fact, addressed by two independent software modules in PariSync:

- 1) A *topology* subproblem: have each node choose the appropriate neighbors with which to communicate.
- 2) An *estimation* subproblem: have each node estimate the current state of the global, virtual clock based on its own clock and the information received from its neighbors.

The latter aspect has received considerable attention in the literature; on the contrary, almost nothing is known about the former, and all solutions in the literature essentially either consider (non-scalable) fully connected graphs, or (fragile) trees or tree-like graphs. This paper focuses on the topology subproblem, adopting only a very basic solution to the estimation one — though more sophisticated strategies may be easily “swapped in” into the appropriate estimation module.

Two important estimation issues should be considered even when focusing on topology, however. First, ideally a sizable fraction of all nodes should have access to NTP or other means of accurately estimating their own time. It would be desirable for the global clock to be “anchored” to these accurate nodes. Second, a small but non-trivial fraction of all nodes should be expected to provide completely inaccurate results (e.g. due to misconfiguration or to an active attempt to disrupt the global clock). In the absence of some means to certify which nodes have access to accurate, external sources (and note that even if there is access to an accurate source, the intervening communication process may introduce large, undetectable errors) the two goals of anchoring the global clock to accurate external sources and of making it resilient to coordinated faults are conflicting: a minority of nodes whose clocks are perfectly consistent with each other, but widely divergent from those of the rest of the network, could either be the group of peers with access to accurate, external information, or simply a group of malicious peers trying to disrupt the network.

Since robustness to (coordinated) faults is probably a higher design priority than external clock anchoring for P2P networks — at least as long as the global virtual clock behaves “reasonably” — any robust system will have each node give little or no weight to the information provided by a small but non-vanishing fraction of “outliers”. It is not difficult to see that this makes the natural scheme of having each node estimate the global clock on the basis of its DHT neighbors alone ineffective. Assume that the address space of a network of n nodes is partitioned into k segments, each holding approximately n/k nodes. The (sets of nodes in the) different segments will then never synchronize if they start with different offsets and drifts and if the estimation rejects a fraction at least $\frac{\log(k)}{\log(n)}$ of all outliers, since only a fraction $\frac{\log(k)}{\log(n)}$ of all links of a node falls outside its segment.

PariSync instead has each node increase its pool of long distance links by simply asking each neighbor (according to the DHT structure) to reply not only with its own time information, but also with the time information about a long distance neighbor of that node (i.e. a neighbor residing in the

other half of the network). While this does introduce some extra noise due to the extra hop, it provides (as we shall see in the next subsection) a particularly well-behaved topology that offers rapid convergence of the consensus on the global clock; and it does so *essentially for free*, since each node in a DHT based peer to peer network will periodically contact its DHT neighbors anyway (to check liveness, to forward queries etc.), and adding time information about two nodes in such a communication exchange has a negligible impact on the performance of the system.

Once a node has retrieved from each of its neighbors the local time (and the local time of their long distance contact), the estimation module of PariSync evaluates its offset and drift from the virtual clock. Note that there exist some sophisticated schemes to do this, and to bound the resulting error (e.g. [15], [7]). The focus of this paper, however, is on the topology module, and the estimation scheme provided below, while extremely basic, seems to work reasonably well — though certainly more sophisticated schemes could be introduced without having to modify the topology module.

Each node v at time t maintains a vector of T assessments of the global clock speed $s_v^t(t), s_v^{t-1}(t), \dots, s_v^{t-(T-1)}(t)$ where T is ideally at least logarithmic in the size of the network. Informally, $s_v^\tau(t)$ measures the estimate, at time t , of the global clock speed at time τ . $s_v^t(t)$ is always set equal to v ’s own speed. $s_v^\tau(t+1)$, with $\tau \leq t$, is set equal to the average of all $s_u^\tau(t)$ such that u is a node providing time information to v that v had not rejected at time τ (note that v can easily estimate the speed of a neighbor relative to its own by comparing that neighbor’s local elapsed time over an interval, and its own elapsed time during that interval, making the estimate slightly more accurate by taking into account communication delays estimated as half of the roundtrip time). Thus, the larger the gap between t and τ , the less up-to-date the information, but also the larger the set of nodes that contribute to it (note that, according to our experiments, clock speeds remain fairly stable over periods of at least a few hundred minutes). Instead of estimating the global clock speed, one could also — with the same mechanism — compute a lower and upper bound on it by simply taking the minimum and maximum instead of the average at each step.

Exactly in the same fashion, a node can obtain an assessment of the global clock *time* at (its own) timepoints $t, t-1, \dots, t-T+1$, $g_v^t(t), g_v^{t-1}(t), \dots, g_v^{t-(T-1)}(t)$ by averaging the times assessed by its neighbors. Again, an estimate relative to a recent timepoint may be not very accurate, because only few nodes have contributed to it. However, if (as it appears to be the case, see the next Section) the clock speeds of individual machines appear to be relatively stable over a period equal to the time between consecutive measurements (a few minutes) times the base 2 logarithm T of the size of the network (at most 20 to 30), a better evaluation of the current time of the global clock can be achieved by taking the estimate of that time at (local) time $t-T$, and correcting it by the elapsed local time adjusted by the (estimated) speed ratio over the intervening interval.

III. EXPERIMENTAL ANALYSIS

This section provides some preliminary experimental evaluation of PariSync. Subsection III-A shows how the standard procedure of estimating communication delay as half of the roundtrip time between nodes still holds even over multihop paths with large delays, such as those typically between peers in large P2P networks. Subsection III-B evaluates PariSync on a smallish (100 – 200 node) network of nodes dispersed throughout Europe (part of the AEOLUS testbed [16]). The resulting latencies, combined with clock drift data from several PCs, allow us to run with realistic parameters an extensive simulation of PariSync over a network of a million (simulated) nodes, detailed in Subsection III-C.

A. Network Latency

NTP clock adjustments are based on the assumption that UDP packets travel from host A to host B in exactly one half of the Round Trip Time between A and B . Defining t_{AB} the time it takes for a UDP packet to travel from host A to host B , this means $t_{AB} = t_{BA}$. In a large P2P network the geographic distance between communicating hosts can be very changeable. Neighborhood in a DHT like Kademlia’s is defined on an ID-oriented metric, and has nothing to do with geographic proximity. For this reason our system must take into consideration a large variety of links, ranging from LAN to intercontinental. IP’s connectionless design makes it seem very likely to have packets following one route when going from A to B and a different one when going back to A . The longer the link, the higher the probability of having different hops in the two paths. We developed a plug-in for PariPari

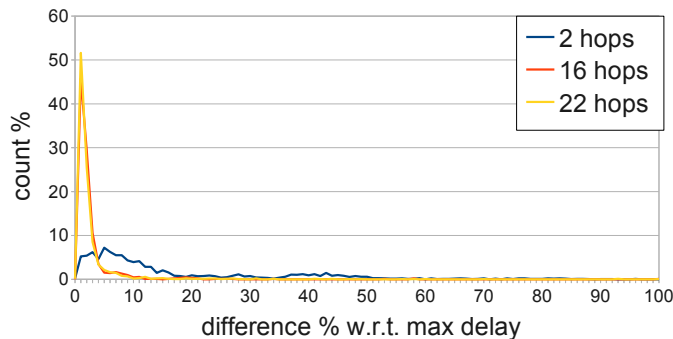


Fig. 2. Differences in clockwise and counter-clockwise pings. Peak values for the 3 groups are: 7% count at 5% difference, 46% at 1%, 52% at 1%.

that performs several *UDP pings* between hosts at regular intervals and logs results to perform statistical analysis. A *UDP ping* is performed by sending a small (a 64-bit signature) packet to a host using UDP and registering how much time passes before receiving a reply UDP packet from that host. In order to collect a significant amount of information, we ran these tests using AEOLUS [16] testbed. This testbed [17] is a network connecting 16 European Universities, each providing a number of heterogeneous hosts, all of which run 1 to 10 (virtual) instances of JXTA. We split hosts into 3 groups of 3 hosts each, so that each host could compute $|t_{ABCA} - t_{ACBA}|$.

Group 1 simulates A and B being in the same LAN, Group 2 simulates A and B being 14 to 16 hops apart (a continental link), Group 3 simulates A and B being 20 to 24 hops apart (an intercontinental link). The number of hops between hosts has been computed by running ICMP *traceroute* several times for each link. Hosts are placed in an overlay network in a token-ring fashion. A map of the ring instructing hosts about their neighbors is shared among hosts of the same group. A *UDP ping* begins with the first node appearing in the map starting a timer and sending a packet containing its signature to one of its neighbors. Every host that receives a packet from one of its neighbors forwards the message to the other neighbor, unless the packet has its own signature in it. If it does, the first ping is complete and the result is logged. A second ping is then performed by the first node, this time sending the packet to its other neighbor. The operation is repeated ten times before passing a token to the node’s clockwise-next neighbor. A *round* is then complete. When a node receives the token, it waits for 10 minutes before starting its *UDP pings* in order to statistically separate rounds. After 500 rounds we analyzed the values of $|t_{ABCA} - t_{ACBA}|$ logged by every host. Figure 2 shows the distribution of these values for each group. In a pathological case, when hosts are in the same LAN, a certain amount of noise appears. This, though, seems more addressable to Java operations on packets than on changes on delivery time on the underlying network. Delays are in the order of half a millisecond, so a difference of 50% in the two pings means a difference of a quarter of millisecond, very close to our timer precision limit. We’ll further investigate on this aspect relying on more precise Java timers, but from the figure it is evident that, for common links, the assumption made by NTP is valid and can be used also in very large networks.

B. PariSync for Real

Initially we ran tests on physical hosts using the AEOLUS testbed. We started each of the 103 hosts with different offsets and drifts choosing them according to a Gaussian distribution. We let our algorithm run with: 10 minutes as the interval between distributed synchronizations, 8 minutes as minimum interval between NTP synchronizations, α taken as 6 for distributed and as 40 for NTP synchronization. Each node polled 7 ($\log(103)$) hosts per distributed synchronization. As we can see from figure 3 and 4 the consensus is achieved after just 11 synchronizations (110 minutes). After 450 minutes we turned off the NTP server and the stability of the system remained good, with an average offset of nearly 35 ms after 24 hours.

C. Simulations

Since PariSync is primarily designed to provide large P2P networks with synchronization we tested it on a simulated environment. We wrote a multi-threaded simulator and modeled it using data from real drifts.

We retrieved data from 6 hosts. `ntpd` was disabled on these hosts and hosts were rebooted to drop any `ntpd` drift correction. The drifts appeared different between hosts but

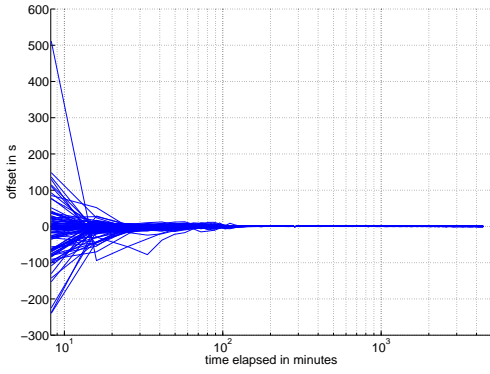


Fig. 3. Offset converging on AEOLUS testbed.

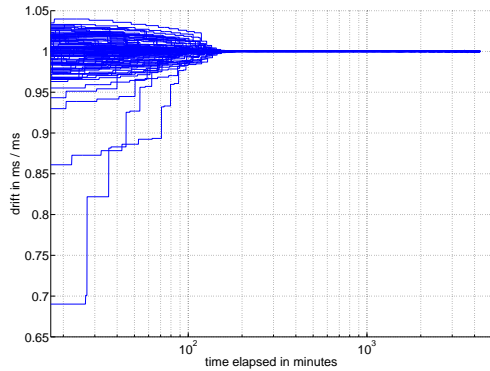


Fig. 4. Drift converging on AEOLUS testbed.

reasonably constant (Fig. 5). We then modeled drift in the simulator assuming a gaussian distribution with the measured average and variance.

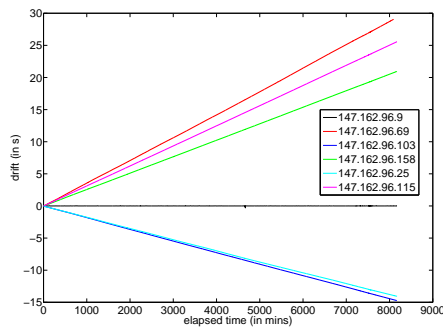


Fig. 5. Drifts on 6 hosts: they range from $-14\mu\text{s/s}$ to $52\mu\text{s/s}$.

As already stated, communication delay can be safely estimated from the measured Round Trip Time (possibly filtering and averaging to get more precise measures). As said above PariSync performs such measurements and corrects data received by other nodes with these estimated delays.

First of all we tried to discover, using a relatively small amount of nodes, the best parameters to let the network

converge steadily and quickly. We run several trials changing the percentage of nodes to be accepted after the filtering process (i.e. accepted nodes) and the weight to be given to their average (i.e. ppsweight). In order to compare simulation performances, we consider the network to be *stable* as the distribution of the time values on all of its nodes reaches a standard deviation equal to or minor than 10^{-5} . Let t_{conv} be the time (in seconds) it takes to the network to become *stable*. Every simulation has been run for a total of 10000 seconds. The height of each bar in Fig. 6 represents $10000 - t_{conv}$, that is the amount of seconds left before the end of the simulation at the time the network became *stable* (hence higher is better). The figure clearly shows that the choice of parameters strongly affects network behavior. Filtering nodes with “strange” values proves to be very effective and increasing the average weight speeds up the convergence.

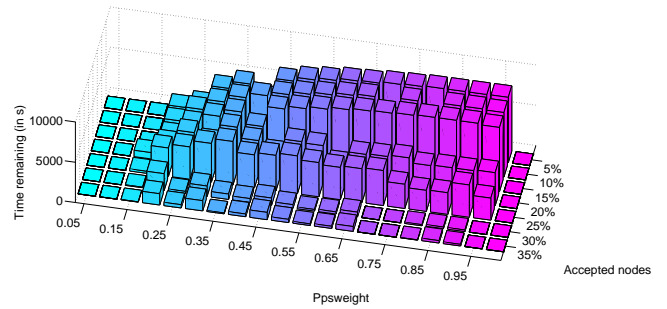


Fig. 6. Seconds remaining to the end of the simulation at the time the network became stable for different values of accepted nodes - ppsweight.

Moreover, PariSync exhibits a good behavior even in the presence of nodes with very different drifts. We ran several simulations setting the same incorrect clock value to some groups of nodes in order to stress the system. The higher the number of “bad” nodes the slower the system achieved the consensus.

We then tested PariSync’s resilience to situations where nodes in the same segment of the ID space sport the same clock behavior, but this behavior varies between segments. We tested the simplest case, with all nodes with ID prefix 0 in one segment and all nodes with ID prefix 1 in another. Recall that, in this situation, simply averaging the information from DHT neighbors without including that from their long range contacts would result in slow or no convergence (see previous SectionII). Fig. 8 shows the difference between standard PariSync, and PariSync without long range contact polling.

Finally, we ran two simulations to test PariSync with and without NTP connection enabled. PariSync is slightly slower with NTP enabled but stabilizes on “real” NTP time. Fig. 9 shows mean and standard deviation evolution.

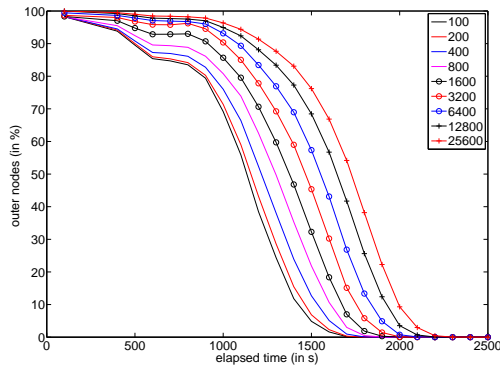


Fig. 7. Outer nodes of 100K nodes in function of nodes with 0 ms/ms drift (accepted nodes: 5%, ppsweight: 0.95).

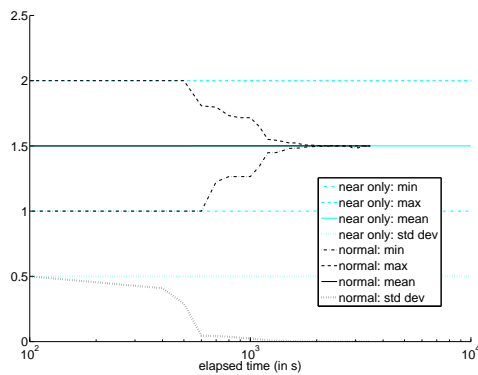


Fig. 8. 1M nodes in a “split” network (accepted nodes: 5%, ppsweight: 0.95) with standard polling (“normal”) vs. polling without long range contacts (“near”).

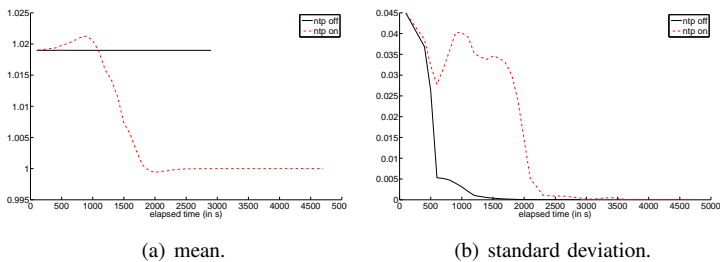


Fig. 9. 1M nodes with and without NTP.

IV. CONCLUSIONS AND FUTURE WORK

PariSync is an efficient clock synchronization system for large, DHT-based P2P networks (e.g. eMule’s Kad [13] or Azureus/ μ Torrent DHTs [14], [18]). PariSync has each peer communicate with only a small subset of other peers, carefully chosen to guarantee a maximal flow of timing information (thus leading to high synchronization performance) while piggybacking on the existing DHT link structure (and thus guaranteeing minimal communication overhead and high robustness). Indeed, the link structure adopted by PariSync could be easily adopted by NTP for efficient intra-stratum

peering.

PariSync then aggregates timing information at each node to produce an estimate of (and/or upper and lower bounds on) the local drift and offset from to a virtual global clock that represents the consensus of all nodes. The aggregation mechanism is completely orthogonal to that of link structure formation (and, in fact is implemented in a different module) and could thus be replaced transparently with more sophisticated mechanism (the quest for which is one direction of future research). Still, preliminary experimental results, conducted both on a testbed of real machines dispersed throughout Europe, and through simulations of very large networks (1M nodes), seem to show that PariSync achieves excellent results “out of the box”.

ACKNOWLEDGMENTS

This work was supported in part by EU IP AEOLUS (FP6-015964) and by Univ. of Padova Strategic Proj. “Algorithms and Architectures for Computational Science and Engineering”. The authors would like to thank Joachim Gehweiler for access to, and assistance with, the AEOLUS testbed.

REFERENCES

- [1] Jxta. <https://jxta.dev.java.net/>.
- [2] L. Lamport and P. M. M. Smith, “Byzantine clock synchronization,” *SIGOPS Oper. Syst. Rev.*, vol. 20, no. 3, pp. 10–16, 1986.
- [3] D. Dolev, J. Halpern, and H. R. Strong, “On the possibility and impossibility of achieving clock synchronization,” in *STOC ’84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1984, pp. 504–511.
- [4] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Probabilistic clock synchronization,” in *Distributed Computing*, September 1989, pp. 146–158.
- [5] A. Sobeih, M. Hack, Z. Liu, and null Li Zhang, “Almost peer-to-peer clock synchronization,” *Parallel and Distributed Processing Symposium, International*, vol. 0, p. 21, 2007.
- [6] J. F. B. Ogden and B. White, “Ibm system z9 109 technical introduction,” 2005.
- [7] C. Lenzen, T. Locher, and R. Wattenhofer, “Clock synchronization with bounded global and local skew,” in *FOCS ’08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 509–518.
- [8] T. T. Machizawa, A. Iwawma, “Software-only implementations of slave clocks with sub-microsecond accuracy,” in *Precision Clock Synchronization for Measurement, Control and Communication, 2008. ISPCS 2008*, 2008, pp. 17–22.
- [9] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” in *ACM SIGCOMM 2001*, San Diego, CA, September 2001.
- [10] S. Ratnasamy, P. Francis, S. Shenker, R. Karp, and M. Handley, “A scalable content-addressable network,” in *In Proceedings of ACM SIGCOMM*, 2001, pp. 161–172.
- [11] A. Rowstron and P. Druschel, “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems,” 2001.
- [12] P. Maymounkov and D. Mazires, “Kademlia: A peer-to-peer information system based on the xor metric,” 2002.
- [13] emule. <http://www.emule-project.net>.
- [14] Azureus. <http://azureus.sourceforge.net/>.
- [15] J.-M. Berthaud, “Time synchronization over networks using convex closures,” *IEEE/ACM Trans. Netw.*, vol. 8, no. 2, pp. 265–277, 2000.
- [16] AEOLUS PROJECT. <http://aeolus.ceid.upatras.gr/>.
- [17] AEOLUS PROJECT testbed. <http://aeolus.cs.upb.de/>.
- [18] μ torrent. <http://www.utorrent.com/>.