Gossip-Based Clock Synchronization for Large Decentralized Systems

Konrad Iwanicki, Maarten van Steen, and Spyros Voulgaris

Department of Computer Science
Vrije Universiteit Amsterdam
De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands
{iwanicki, steen, spyros}@few.vu.nl

Abstract. Numerous large-scale decentralized systems assume loosely synchronized clocks. Existing time protocols have not been designed for deployment in such systems, since they are complex and require manual configuration. We present the *Gossiping Time Protocol* (GTP), a completely self-managing epidemic time synchronization algorithm for peer-to-peer networks. In GTP, each node synchronizes its time by gossiping with other nodes. The decisions regarding sample evaluation and gossiping frequency are purely local, yet they result in consistent behavior of the whole system. Large-scale experimental evaluation of a 64,500-node network emulated on 65 machines indicates high scalability and reasonable accuracy of GTP.

1 Introduction

Many algorithms are designed for large-scale distributed systems in which nodes are assumed to have their clocks at least loosely synchronized. Examples include algorithms for replica consistency management in which consistency is formulated in terms of maximal allowed staleness [1], lease-based cache-invalidation algorithms [2], algorithms for large-scale systems monitoring and management [3], algorithms for ticket-based authentication [4], and real-time algorithms for wide-area overlay networks [5]. It is not difficult to extend this list.

In many cases, the developers of these algorithms simply assume that clocks are indeed to a certain extent synchronized and that some algorithm is deployed by the underlying system to meet this requirement. However, practice shows that such an assumption is often not realistic and that special measures need to be taken to ensure clock synchronization.

Unfortunately, simply deploying a clock synchronization algorithm is easier said than done. Good algorithms such as the Network Time Protocol (NTP) [6,7] are not only complex and require considerable manual configuration, but above all are not designed to be deployed for large decentralized systems in which nodes regularly join and leave. Virtually all peer-to-peer overlay networks fall into this latter category.

In this paper, we make a single contribution: we introduce a novel clock synchronization algorithm that is designed to operate in highly dynamic overlay networks. The algorithm requires no manual intervention, does not rely on special hardware or low-level systems software, can cope with a continuously changing network consisting of tens of thousands of nodes, and above all, is relatively simple while providing very reasonable accuracy.

The algorithm assumes only that there exists at least one, accurate and robust time server. The basic idea is that the time as produced by that server is rapidly and accurately disseminated throughout the network. Dissemination is a continuous process so that changes in the network can easily be accounted for. The dissemination speed is automatically adjusted to the rate of changes in the network in order to optimize resource consumption. We discuss our assumptions, algorithmic details, and large-scale experimental evaluations.

The rest of the paper is organized as follows. Section 2 introduces a terminology and a basic algorithm for clock synchronization in the network. We introduce our protocol in Sect. 3 and present experimental results in Sect. 4. Section 5 discusses the related work while Sect. 6 concludes.

2 Synchronization Principles

Throughout the paper, we adopt the following terminology [6]. The *epoch* of an event is an abstraction allowing to determine the ordering of events in some *time frame*. An *oscillator* is a generator capable of a precise frequency (relative to a given time frame) within a specified *tolerance*. A *clock* contains an oscillator and a counter recording the number of cycles since being initialized with a given value at a given epoch. The value of the counter at epoch t defines the *time* of that epoch $T(t)$.

In general, T is a monotonically non-decreasing function of t. For practical reasons, $T(t)$ – the time displayed by a clock at epoch t relative to the standard time frame – is approximated using the Taylor expansion:

$$T(t) = T(t_0) + F(t_0) \cdot (t - t_0) + D(t_0) \cdot \frac{(t - t_0)^2}{2} + \rho(t),$$

where $T(t_0)$ is the time at some previous epoch t_0, $F(t_0)$ is the frequency, $D(t_0)$ is the *drift* per time unit (the first derivative of the frequency), and $\rho(t)$ is the remainder. T and sometimes F are estimated in the synchronization process, while D and ρ, characterizing the random nature of a real clock, are ignored. Their influence on $T(t)$ is compensated for by periodically repeating the estimations.

The *stability* of the clock indicates how well it can maintain a constant frequency, the *accuracy* is how well its time compares with the reference time frame – the Universal Coordinated Time (UTC), and the *precision* is the granularity of the clock. The *offset* of clock A relative to clock B is their difference in time $T^{AB}(t) = T^{A}(t) - T^{B}(t)$ at a particular epoch t, while the *skew* is their difference in frequency $F^{AB}(t) = F^{A}(t) - F^{B}(t)$. For all t, we have: $T^{AB}(t) = -T^{BA}(t)$, $F^{AB}(t) = -F^{BA}(t)$, $T^{AA}(t) = 0$, and $F^{AA}(t) = 0$.

Synchronization of clocks in a network requires comparing them directly or indirectly with respect to differences in time and (possibly) frequency. The approach adopted by most popular protocols is based on a variant of Cristian's algorithm [8] (see Fig. 1).

The *time source* (B) is a node with authoritative time. Every other node (A) periodically records timestamp T_1^A according to its clock and sends a synchronization message. Immediately after reception of this message, B records timestamp T_2^B according to

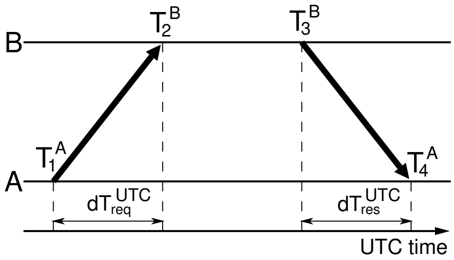


Fig. 1. Comparing time displayed by network clocks.

its clock and starts to prepare a response message containing the recorded timestamp. When the message is ready, B records timestamp T_3^B, stores it within a response (together with T_2^B) and sends the response back. As soon as the message is delivered, A records timestamp T_4^A according to its clock.

At this point, A has the following set of timestamps: T_1^A, T_2^B, T_3^B, and T_4^A (called a *synchronization sample* or just – a *sample*). Assume that time at both nodes is not running backwards and their clocks are stable and run with a similar frequency. If the difference between propagation delays from A to B and from B to A (the *differential delay*) is small ($dT_{req}^{UTC} \simeq dT_{res}^{UTC}$), the timestamps allow A to compute the *round-trip delay*:

$$\delta = T_4^A - T_1^A - (T_3^B - T_2^B) = T_4^A - T_1^A + T_2^B - T_3^B, \tag{1}$$

and the offset of B:

$$\theta = T_3^B + \frac{\delta}{2} - T_4^A = \frac{T_2^B - T_1^A + T_3^B - T_4^A}{2}. \tag{2}$$

Based on the value of θ, A can adjust its time to synchronize with B, by either changing the frequency of its clock for a certain period (*gradual* adjustment), or resetting the clock to the appropriate time (*immediate* adjustment).

Extensive error analysis for the presented algorithm can be found in [7]. To sum up, the following errors influence the time accuracy.

1. Errors in reading clocks depending on their precision and adjustment method.
2. Errors due to imperfect stability of the clocks since their time was last set.
3. Errors contributed by delay variations in the network and OS.
4. Errors caused by multiple nodes participating in the synchronization, which depend on the time differences among these nodes and a resolution method.

In practice, errors due to network delays dominate others [6]. Moreover, if synchronization of a node requires participation of other nodes, the errors contributed by these nodes accumulate.

3 Gossiping Time Protocol

Our **Gossiping Time Protocol** (**GTP**) assumes a network of nodes (equipped with clocks) with at least one accurate and robust time source. The basic idea is that the time, as provided by this source, is disseminated through the network by letting nodes regularly exchange clock settings by means of a gossiping protocol, that is, each node regularly selects another node from the network to exchange timing information with. Very simply put, if the time as registered by the contacted node is of higher quality, then the initiating node will adopt those clock settings. A crucial element of our synchronization protocol is evaluating the quality of the time as provided by a node, and taking local measures when clock settings are adopted. We will elaborate on these issues extensively later.

A second element important to our protocol is that if nodes are synchronized within acceptable bounds, the frequency by which they contact each other should be relatively low. In contrast, if nodes are not synchronized, we demand rapid dissemination of the correct time, that is, exchanges of clock settings must take place more often. The network as a whole should automatically adapt the gossiping frequency.

These are just a few self-managing aspects of our proposed system. Other issues concern the regular joining and leaving of nodes, as well as the robustness in the presence of (possibly massive) node and link failures. Our protocol is based on an instance of a *peer-sampling service* (PSS) which we have developed and extensively evaluated [9]. Originally, this service provides an interface to applications that allows them to contact a uniformly randomly selected *live* node. As we have demonstrated, PSS is fully decentralized, extremely robust, scales to hundreds of thousands of nodes, and can handle very high levels of churn. In particular, at least 69% of random nodes need to be removed simultaneously in order to partition the network. PSS considerably simplifies the design of our protocol and supports our claim about the robustness of GTP. It may be argued that if a number of nodes fails concurrently, the synchronization accuracy may decrease due to deteriorated overlay properties (e.g., a higher diameter). However, since PSS can repair the network much faster than the realistic clock skew can significantly degrade the quality of time, the clock accuracy is expected to remain almost unaffected.

In the extended description of our research [10], we present an improved PSS that can be tailored to application-specific requirements without loosing the ability of concurrent usage by multiple applications and other aforementioned properties. This new PSS, in particular, enables GTP to select not random nodes, but nodes that provide more accurate time. Due to space constraints, we do not present these results in this paper, and simply assume the classical PSS in all subsequent sections.

3.1 Principal Operation

The principal operation of GTP is quite simple and is sketched in the message-sequence chart shown in Fig. 2. When node *A* decides to initiate synchronization, it first requests PSS to uniformly select a peer (messages 1 and 2), say *B*. *A* then exchanges timestamps with *B* (messages 3 and 4), as we sketched in Fig. 1. At that point, *A* evaluates the returned sample and can either reject it (5a), use it to adjust its own local clock (5b), or decide to provide feedback to *B* (5c). The latter may happen, for example, because *A* can deliver a better time than B could, in which case B's clock may need to be adjusted. We will return to the evaluation of samples later. Returning feedback effectively means that our gossiping protocol follows a pull-push strategy, which has been shown to generally provide the fastest convergence behavior [9]. Finally, B may also decide to reject the returned feedback (6a) or to use it to adjust its clock (6b).

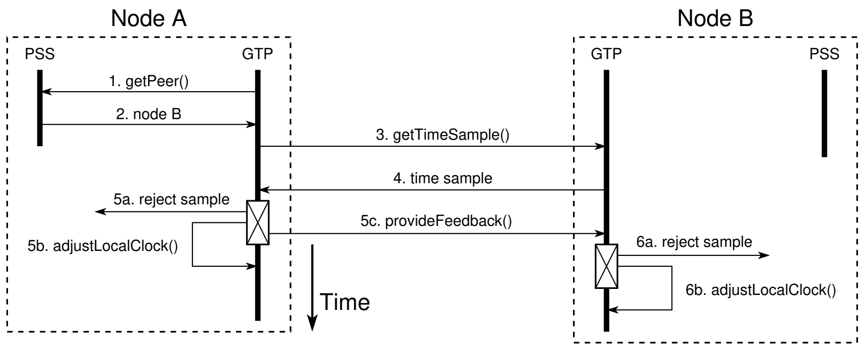


Fig. 2. The principal operation of GTP.

There are many design details of GTP, all of which are extensively discussed in [10]. However, for our discussion here, we concentrate only on the two key elements of the protocol, namely the evaluation of samples and the automatic adjustment of the gossiping frequency.

3.2 Sample Evaluation

In gossiping algorithms, peers are generally considered to be equal. However, when it comes to adopting clock settings, this is no longer the case. What it boils down to is that when node A contacts node B, we can generally assume that one of the two clocks is better synchronized with the time source. We need to figure out which one. As we show later, this issue is crucial to the macroscopic behavior of the algorithm. What we are seeking for are local-only solutions that lead to a self-organized dissemination scheme resulting in an accurate global synchronization. To this end, we explore two different evaluation metrics. First, we discuss a distance-based metric, followed by an error-based metric.

Hop-count metric. The *hop-count* metric is based on the observation that, since network delays are the major sources of errors, the accuracy of a node's time degrades with the number of nodes on the synchronization path from this node to the time source. According to this metric, the hop count is always equal to zero for the time source. Node A decides to use a sample from a timestamp exchange with node B if its hop count (H^A) is greater than B's hop count (H^B). In this case, A changes its hop-count value to $H^B + 1$.

Note that without any special measures all hop counts as maintained by the individual nodes would eventually converge to 1. This is because PSS guarantees a uniform random sampling from all nodes, so we can expect that every node will eventually have been able to read the time from the time source. However, it may take a considerable amount of time before a node can actually directly contact the time source again, while in the meantime the clock skew will constantly degrade the accuracy of its time. Therefore, periodical re-synchronization is simply a necessity. For this reason, a node may decide to use a sample from another node irrespective of current hop count values whenever it has not synchronized for a long time – a time we refer to as the *tolerance period*.

Furthermore, our experiments have shown that an additional mechanism is necessary for detecting samples suffering from fluctuations in network delays [10]. For this purpose, a filter containing a running window of round-trip delays of the last few samples can be employed. When a new sample is received, its round-trip delay (δ) is added to the filter and compared with a weighted average or a median of current values in the filter (Δ). The sample is rejected if $\delta > \Delta$. The filter addresses two important issues.

Firstly, it is capable of rejecting samples with round-trip delays higher than "normal," often an indication of a large error. A high round-trip delay is not necessarily equivalent to a large differential delay (i.e., the time to get from A to B is not the same as for going from B to A). However, without synchronized clocks it is not possible to determine a differential delay during a timestamp exchange. On the other hand, a large differential delay may be caused by a packet being stalled on its way in one direction, which implies a high round-trip delay.

Secondly, the filter is capable of adjusting to a changing network, that is, it exhibits self-adaptation properties. When round-trip delays increase due to escalated network load, Δ gradually grows and eventually some samples will justifiably pass the filter.

Dispersion metric. The *dispersion* metric provides a fine-grained estimation of sample errors leading to improved accuracy. Recall that in the previous metric, the hop count measures the expected error accumulation while the filter deals with exceptional cases. However, if a filter contains high round-trip delays, a clock sample with a small hop count but with a relatively large error may easily pass the filter. By using such a sample, a node's hop count may become small, in turn, leading to a situation in which its now erroneously adjusted clock value will suddenly be used by other nodes with which it gossips. We thus see a poor sample rapidly propagating to potentially many other nodes.

This situation is easily avoided by deploying the dispersion metric presented here. Intuitively, the dispersion value tells a node whether it is better off with an independently accumulated error since its last synchronization, than with the errors possibly inherited by using a clock sample from the selected peer. Assume that each node knows the precision and the frequency tolerance of its clock.[1] Node A calculates the dispersion of a sample obtained by gossiping with node B as follows:

$$\lambda = \varepsilon^B + \rho^B + |\theta^B| + \left(T^B_{NOW} - T^B_{LU}\right) \cdot \phi^B + \frac{\delta}{2}, \tag{3}$$

[1] Such a value is a common element of a clock specification.

where ε^B is the dispersion of B, ρ^B is the precision of B's clock, θ^B denotes the outstanding clock correction of B (in case of the gradual clock adjustment method), T^B_{NOW} is B's current time, T^B_{LU} is the time when B's clock was synchronized for the last time, ϕ^B is B's frequency tolerance, and δ is the round-trip delay. The dispersion of the sample incorporates all types of errors that can cause a clock to be inaccurate. Node A decides to use the sample for synchronization if and only if

$$\lambda \le \varepsilon^A + \rho^A + \left(T^A_{NOW} - T^A_{LU}\right) \cdot \phi^A \quad \textbf{and} \quad \left(\theta^A = 0 \textbf{ or } H^A > H^B\right), \tag{4}$$

where the meaning of the symbols is as before. The first inequality determines whether using the sample for synchronization will improve the constantly degrading time accuracy. The second heuristic allows to start the synchronization only if a possible previous clock adjustment is completed or node B has a lower hop count than node A. If the sample satisfies both conditions, node A sets its dispersion value to λ.

3.3 Gossiping Frequency

Synchronization in GTP is a continuous process in order to compensate for clock skew, leap seconds, and most of all, membership changes in the network. In the simplest scenario, a timestamp exchange could be initiated by a node at a fixed rate, that is, nodes would gossip at a constant frequency. Although a timestamp exchange is relatively inexpensive, it does consume resources. Obviously, the gossiping frequency can be small when the network is synchronized and stable. Ideally, however, it should automatically boost at certain events (e.g., when the time displayed by the time source is reset, or when many new nodes join the network such as after recovering from a major disruption). Again, the gossiping frequency should automatically return to lower values when the change is accounted for. This goal may be achieved in the following way.

Each node is allowed to change its gossiping frequency between ϑ_{min} and ϑ_{max}. To control the frequency, a node stores a sliding window containing absolute values of clock offsets derived from the last N samples. After m ($m \le N$) samples have been used for synchronization, the minimal (Θ_{min}) and maximal (Θ_{max}) values of the offsets in the window are determined. Additionally, a weighted average (denoted θ) of the last M offsets ($m \le M \le N$) is computed. If $\Theta_{max} - \Theta_{min} \ne 0$, the gossiping frequency is modified to:

$$(\vartheta_{max} - \vartheta_{min}) \cdot \frac{\theta - \Theta_{min}}{\Theta_{max} - \Theta_{min}} + \vartheta_{min}, \tag{5}$$

otherwise it is set to ϑ_{min}. The formula linearly maps a recent absolute offset θ from the interval $[\Theta_{min}, \Theta_{max}]$ to a value in $[\vartheta_{min}, \vartheta_{max}]$. The reaction speed and the change smoothness for a node are controlled by m, M and N parameters. In the next section, we will evaluate this adjustment for different parameter settings.

The presented algorithm allows each node to make its own decisions based on local information. Nevertheless, since the behavior of all nodes is consistent, the whole network adapts its gossiping frequency in a timely manner, as the change or the stability is discovered by subsequent nodes. Note that our solution is fully automatic in a sense that it does not require precalculating any critical parameters. The maximal and minimal gossiping frequencies can be derived by GTP from the clock specification and the expected time accuracy. As we show by experiments, the values for m, M and N, on the other hand, are not critical for the overall behavior.

4 Experimental Results

We have studied the properties of GTP on an emulated network of nodes, using a large server cluster. Each of 72 machines in this cluster contains two 1-GHz PIII CPUs, at least 1 GB of RAM and a 20-GB hard disk. The machines within the cluster are connected by 100-Mbps Ethernet. They run Red Hat Linux 7.2. Since the machines are not equipped with hardware capable of high-accuracy synchronization with UTC (e.g., WWV receivers), we had to rely on NTP to provide clock synchronization among them.

The GTP implementation [10] was written in Java and uses UDP for communication between nodes. Each machine was running two JVMs. One JVM in the whole testbed was hosting an RMI registry for utilities, while all others were hosting 500 GTP nodes each. We experimented with large overlay networks (64,500 nodes on 65 machines), medium ones (10,500 nodes on 11 machines) and small ones (1,500 nodes on 2 machines).

Emulation is superior to simulation as it involves a real implementation. This is especially true in the world of large-scale systems, where practice often diverges from theory. In the case of GTP, using real software implied real-world effects, for example, delays caused by the operating system on the path of a packet through the protocol stack, or an asymmetry in the time of execution between the code responsible for sending messages and the code responsible for receiving messages, which affect the quality of synchronization. Emulation is also more efficient than deployment, as it allows to study the behavior of incomparably bigger networks at a reasonable cost. Although our testbed was unable to model wide-area latencies, this drawback was partially compensated by the communication heterogeneity provided: intra-process (between threads within the same JVM), interprocess (between threads in two different JVMs on the same machine), and network communication (between threads on different machines).

It is important to note that although up to 1,000 nodes resided on a single machine, the random selection of a peer node, as provided by our PSS, ensures that, respectively, more than 98% (large system), 90% (medium system) and 33% (small system) of the messages were transmitted through physical network links. Consider, for example, a large system. For a node in such a configuration, at most 999 other nodes were running on the same machine while 63,500 were running on different machines. As observed during the tests and in similar experiments [11], peers subsequently selected by a node were uniformly distributed over all machines. Therefore, the expected communication between the nodes on the same machine for a large system constituted only 1.55% of total communication.

In the following sections we only highlight the most important aspects of GTP. More information on the full set of experiments that we conducted can be found in [10].

4.1 Dissemination Speed and Accuracy

The speed by which nodes synchronize is influenced by the overlay topology, the size of the network and the gossiping frequency. To simplify matters, we assume that each node initiates an exchange precisely once every ΔT units, that is, the gossiping frequency is fixed. When all nodes have performed such an exchange, we say that a *round* has completed. Note that during a round a node may be involved in more than one exchange as other nodes selected it for communication. We set $\Delta T = 25s$, leading to a gossiping frequency of $f = 0.04$ Hz.

We measured the dissemination speed by counting the number of nodes that received at least one sample that was used for synchronization. The results are shown in Fig. 3. Note that the actual synchronization speed may be smaller than the dissemination speed, for example, if the time is adjusted gradually (cf. Fig. 5).

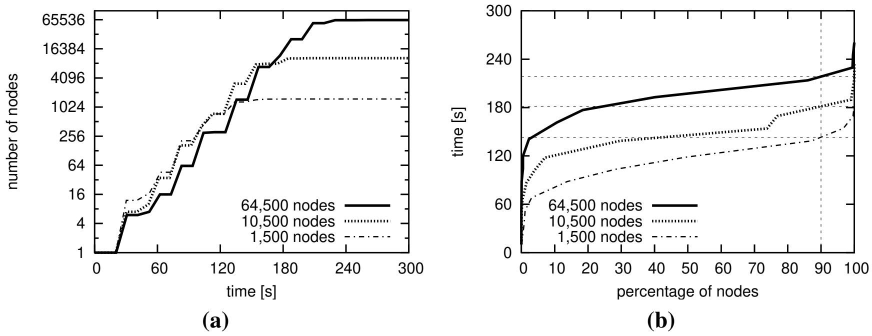
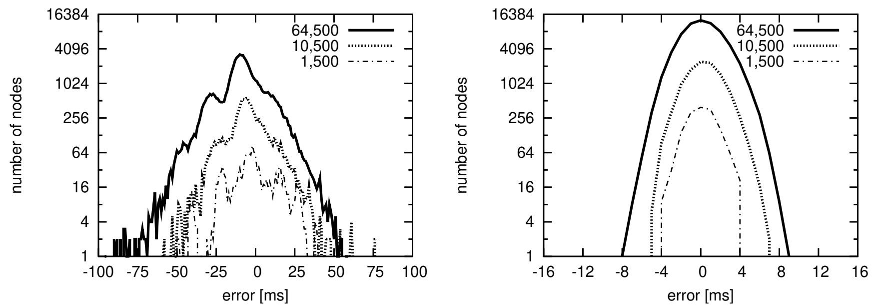


Fig. 3. Time dissemination speed of GTP: **(a)** shows that the number of nodes to which the time has been disseminated grows exponentially with time (note the log scale for the y-axis); the horizontal lines in **(b)**, which approximate 90% dissemination, show that the dissemination time grows logarithmically with the size of the network.

The results indicate that the dissemination speed depends logarithmically on the size of the network. This is consistent with the observation that in a network without differential delays if two GTP nodes, one of which is synchronized, exchange their timestamps, they are both synchronized afterwards. Assuming a topology with reasonable connectivity properties, the time should be propagated fast with a high probability.

We tested the accuracy by considering both hop counts (with sample filtering) and dispersion, on networks of different sizes. Initial clock errors were chosen randomly with uniform distribution from the [10 s, 60 s] interval (positive and negative). Figure 4 presents snapshots of error distribution taken in a *converged* state.

The hop-count metric with sample filtering performs significantly worse than the dispersion metric. For example, in a converged state in a 64,500-node network we have measured errors up to even 322 ms when using hop counts, whereas considering the dispersion criterion they do not exceed 12 ms. This difference is caused by the aforementioned scenario of error propagation which only becomes worse as the network grows. The dispersion metric, on the other hand, ensures high accuracy that scales gracefully with respect to the number of nodes.



(a) **(b)**

Fig. 4. Time accuracy of GTP: **(a)** using hop counts and sample filtering; **(b)** using the dispersion metric. Note the log scale for the y-axis and different scales for the x-axis.

4.2 Gossiping Frequency

The choice of the gossiping frequency is made individually by every node. To test the influence of such decisions on the macroscopic behavior of the network, we performed experiments in which, first, the network synchronized, and then, the clock setting of the time source was changed by 30 s. We used the earlier configurations and set the maximal and minimal gossiping frequency values of each node to 0.04 Hz and 0.01 Hz, respectively. Figure 5a illustrates the relationship between individual nodes and the whole system, consisting of 1,500 nodes.

While the system is synchronizing, all nodes gradually decrease their gossiping frequency as the offsets between nodes' clocks decrease. (Compare the curve representing the average absolute time error with the curve representing the average gossiping frequency during the initial 1700 s in Fig. 5a.) Later, when the time at the time source is changed, the frequency of individual nodes increases as they notice the change by observing high offset values reported by other nodes. (See the period from 2500 s to 4500 s in Fig. 5a.) The nodes that are closer, in terms of network hops, to the time source (e.g., node 1 in Fig. 5a), notice the time change, and consequently, adjust their frequency, earlier. Additionally, they keep this high gossiping frequency for a longer period of time as there are many other nodes that have not noticed the change, that is, they report high clock offsets. In contrast, the nodes that are farther from the time source (e.g., node 2 in Fig. 5a) and thus notice the change later, do not gossip long at a high frequency, because at that moment, a number of nodes should already be in the process of correcting their clocks, so they are reporting lower offsets. These nice properties ensure that if there is work to be done (i.e., there are many nodes unaware of the change), some nodes will gossip longer with a high frequency to accelerate completion of the work. On the other hand, when a significant amount of work has been already done, there is no need to operate long with high frequency.

Moreover, note that although some nodes gossip with the maximal frequency, the average value for the whole system is relatively small, because first, not all nodes notice the change at the same time, and second, the clock correction causes a gradual

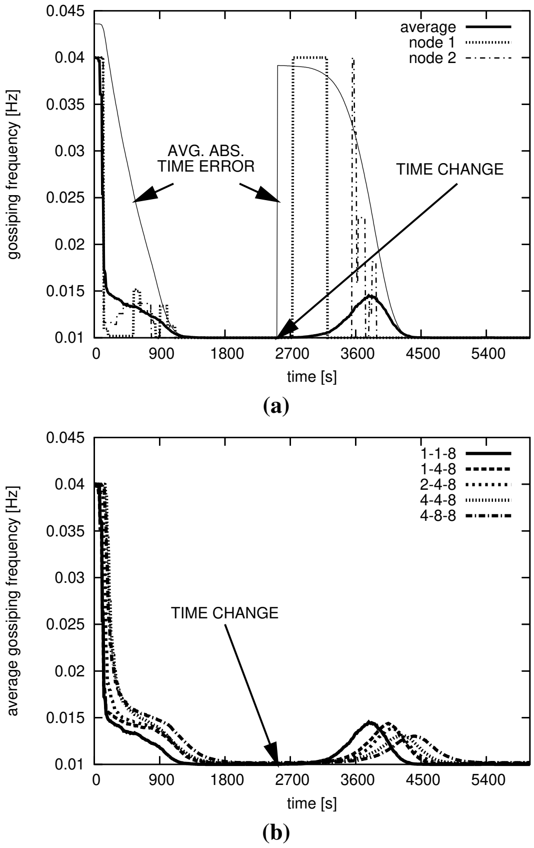


Fig. 5. Adjustment of gossiping frequency: **(a)** shows the changes for two randomly selected nodes and the average, with configuration [*m*-*M*-*N*] = [1-1-8]; **(b)** shows the average adjustment for different configurations.

decrease of reported offsets. This behavior is highly desirable as it prevents network congestion, which would result in high differential delays, and thus, slower and less accurate synchronization. When the system accounts for the change, the gossiping frequency is gradually decreased, such that the moment of synchronization, in which the time offsets between nodes are minimal, matches the moment of reaching the minimal gossiping frequency. (Compare the curve representing the average absolute time error with the curve representing the average gossiping frequency during the period from 2500 s to 4500 s in Fig. 5a.)

Figure 5b presents how different parameters influence frequency adjustment. Recall that a setting *m*-*M*-*N* indicates that the frequency is adjusted every *m* samples, by calculating the average absolute offset of the last *M* samples and comparing it to the maximal and minimal values derived from the last *N* samples. Varying *N* (the results not plotted) influences the amount of historical information taken into account when establishing the frequency bounds. The M parameter controls the smoothness and the range of the frequency changes, as it specifies how many historical offset values are taken into account when computing the average. Finally, m determines the reaction speed. What we see, however, is that actual parameter settings are not critical to the overall behavior, an issue we feel is important for self-organizing systems.

5 Related Work

Numerous time synchronization protocols focusing on clock accuracy have been proposed [12,13]. The most prevalent one, NTP ver. 3 [7,6], operates in a predefined hierarchical configuration where nodes are divided into primary time servers, secondary time servers and clients. The position in the hierarchy is determined by a stratum – a hop-count equivalent. A primary time server (stratum 1) is directly synchronized to an external reference time source (e.g., a radio clock or a high-quality GPS receiver), and is usually equipped with a precise and stable clock, such as an atomic clock. A secondary time server, intended to deliver the time to clients, synchronizes, possibly via other secondary servers, with one or more primary servers. The basic idea behind NTP is building a synchronization spanning tree in which a primary server is the root, secondary servers are the inner nodes, and clients are the leaves. To achieve the remarkable accuracy, NTP requires complex sample evaluation algorithms, rigorous support from hardware and an operating system, a stable network (well-defined by the administrators of the servers), and a fair amount of configuration.

In contrast, GTP focuses on very large, dynamic, decentralized systems. It sacrifices the nano-second accuracy for adaptability, scalability, simplicity, portability, and the ease of deployment or integration. The fine-grained error estimation of GTP has been inspired by NTP, but also many other algorithms employed by NTP can be successfully applied in GTP. Due to the assumptions about the dynamism of the network and only a basic support from an operating system, GTP, unlike NTP, does not synchronize clock frequencies, since this requires long measurement periods involving exactly same hosts, and special hardware [7]. Moreover, it does not prevent synchronization cycles if they do not degrade the clock accuracy. As the experiments show, despite these limitations, GTP is still able to deliver reasonably accurate time.

Another example, the UNIX timed [14], designed for LANs, uses an election algorithm to designate a single host as master. The master periodically polls every slave host to obtain its time. Based on the answers, it computes the average that determines the current global time. Afterwards, the clock corrections are propagated to all other hosts. This is an example of the internal synchronization, in which the network is not synchronized to the global reference time [13].

As opposed to the UNIX timed, GTP provides the external synchronization. Moreover, it has been designed for applications that usually work in the Internet, and not only LANs.

Recently a novel group of clock synchronization algorithms minimizing resource consumption and emphasizing robustness and self-organization, aimed at sensor networks, has been proposed [15]. Such networks require physical time, so data from different sensor nodes can be fused and delivered in a meaningful way to the sink. For example, the Reference-Broadcast Synchronization (RBS) [16] exploits the broadcast property of the wireless communication medium, which states that two receivers within listening distance of the same sender will receive the same message at approximately the same time (i.e., with very little variability in the delay). If every receiver records its time as soon as the message arrives, all receivers can synchronize with high accuracy by comparing their recorded time readings. This receiver-to-receiver synchronization removes two major sources of delays in wireless sensor networks, namely, waiting for the medium to be clear for transmission, and preparing the message. Since this type of synchronization requires all nodes to be in listening range of the sender, which may not be the case, RBS provides additional time forwarding. The topology for forwarding is determined by the senders of periodical synchronization messages.

The Timing-sync Protocol for Sensor Networks (TPSN) [17], Lightweight Time Synchronization (LTS) [18], and Tiny-Sync [19] use Cristian's algorithm and a spanning tree to synchronize time. However, due to the broadcast property of the wireless communication, a number of optimizations can be made. For example, a node deep in the tree can overhear messages exchanged between nodes closer to the root, and utilize some of the timestamps carried within these messages during its synchronization phase. Moreover, MAC-level packet timestamping, as employed by TPSN, significantly reduces possible errors.

The Flooding Time Synchronization Protocol (FTSP) [20] uses periodic floods from an elected root, rather than maintaining a spanning tree. In the case of root failure, the system elects a new root node. FTSP also refines the MAC-level timestamping to within microsecond accuracy.

Although time synchronization protocols for sensor networks bear some similarities with GTP, they all operate under completely different assumptions. They are very tightly coupled with the hardware they were devised for. This is particularly visible when considering how they exploit the broadcast property of the wireless communication medium or MAC-level timestamping, to reduce synchronization errors and to minimize the number of exchanged messages. Moreover, as opposed to ordinary PCs, sensor nodes are often equipped with high-precision timers. Due to such differences, a direct comparison of these protocols to GTP in terms of performance is difficult. For example, high-precision timers and various low-level optimizations influencing accuracy are in favor of the sensor network protocols. On the other hand, the time dissemination speed with respect to the number of nodes should be better for GTP, because of the properties of the physical network. Finally, the behavior of the sensor network protocols with respect to robustness and node population has not been widely studied, whereas in GTP we focus on very large, dynamic systems.

6 Conclusion and Future Work

Many modern large-scale systems require at least loosely synchronized clocks. Existing time protocols were not designed to operate in the large, often self-organizing environments we consider in this paper. GTP addresses this problem. It requires no hierarchy; instead, all nodes actively participate in epidemic-style time synchronization. Sample evaluation and decisions on adjusting the gossiping frequency are made individually by every node based only on locally-available information. Such local decisions not only inherently ensure high scalability, but above all lead to consistent behavior of the whole system, which is an essential self-management property. By using a peer-sampling service, such as described in [9], we obtain automatic organization of the overlay and robustness in the presence of various changes such as those in node membership.

The results of experiments, conducted on emulated networks using a real implementation, show that GTP is capable of attaining very reasonable accuracy and that it can automatically adjust its gossiping frequency when synchronization errors have become low or a sudden event occurs in the system.

We plan to improve GTP with mechanisms for dealing with time source failures and conduct experiments on larger physical networks that exhibit wide-area transmission delays. We are also interested in applying GTP to wireless sensor networks. Finally, we intend to investigate serverless synchronization, that is, heterogeneous nodes (in terms of the clock parameters) would establish and maintain consistent and accurate system-wide time without any designated time source. In this latter approach, because of the network dynamics, the stability of the algorithm may become an issue.

References

1. Yu, H., Vahdat, A.: Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Transactions on Computer Systems (TOCS) **20**(3) (2002) p.239–282
2. Duvvuri, V., Shenoy, P., Tewari, R.: Adaptive leases: A strong consistency mechanism for the World Wide Web. IEEE Transactions on Knowledge and Data Engineering **15**(5) (2003) p.1266–1276
3. van Renesse, R., Birman, K.P., Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Transactions on Computer Systems (TOCS) **21**(2) (2003) p.164–206
4. Garman, J.: Kerberos: The Definitive Guide. O'Reilly & Associates, Sebastopol, CA, USA (2003)
5. Rajendran, R.K., Ganguly, S., Izmailov, R., Rubenstein, D.: Performance optimization of VoIP using an overlay network. Technical report, NEC Laboratories America, Inc., Princeton, NJ, USA (2005)
6. Mills, D.L.: Improved algorithms for synchronizing computer network clocks. IEEE/ACM Transactions on Networking (TON) **3**(3) (1995) p.245–254
7. Mills, D.L.: Network Time Protocol (version 3) specification, implementation and analysis. Technical Report RFC 1305, The Internet Society (1992)
8. Cristian, F.: Probabilistic clock synchronization. Distributed Computing **3** (1989) p.146–158
9. Jelasity, M., Guerraoui, R., Kermarrec, A.M., van Steen, M.: The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In: Proc. of the Middleware 2004, Toronto, Canada (2004) p.79–98
10. Iwanicki, K.: Gossip-based dissemination of time. MSc thesis, Warsaw University and Vrije Universiteit Amsterdam (2005) Available at: `http://www.few.vu.nl/~iwanicki/`.
11. Voulgaris, S., Jelasity, M., van Steen, M.: A robust and scalable peer-to-peer gossiping protocol. In: Proc. of the 2nd Int. Workshop on Agents and Peer-to-Peer Computing (AP2PC), Melbourne, Australia (2003) p.47–58

12. Simons, B., Welch, J.L., Lynch, N.A.: An overview of clock synchronization. Lecture Notes in Computer Science **448** (1990) p.84–96
13. Patt, B.: A Theory of Clock Synchronization. PhD thesis, MIT (1994)
14. Gusella, R., Zatti, S.: The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD. IEEE Transactions on Software Engineering **15**(7) (1989) p.847–853
15. Sundararaman, B., Buy, U., Kshemkalyani, A.D.: Clock synchronization in wireless sensor networks: A survey. Ad-Hoc Networks **3**(3) (2005) p.281–323
16. Elson, J., Girod, L., Estrin, D.: Fine-grained network time synchronization using reference broadcasts. In: Proc. of the 5th USENIX Symposium on Operating System Design and Implementation (OSDI), Boston, MA, USA (2002)
17. Ganeriwal, S., Kumar, R., Srivastava, M.B.: Timing-sync protocol for sensor networks. In: Proc. of the 1st Int. Conf. On Embedded Networked Sensor Systems, Los Angeles, CA, USA (2003) p.138–149
18. van Greunen, J., Rabaey, J.: Lightweight time synchronization for sensor networks. In: Proc. of the 2nd ACM Int. Conf. on Wireless Sensor Networks and Applications, San Diego, CA, USA (2003) p.11–19
19. Sichitiu, M.L., Veerarittiphan, C.: Simple, accurate time synchronization for wireless sensor networks. In: Proc. of the IEEE Wireless Communications and Networking Conference (WCNC) 2003. (2003)
20. Maróti, M., Kusy, B., Simon, G., Lédeczi, A.: The Flooding Time Synchronization Protocol. In: Proc. of the 2nd Int. Conf. On Embedded Networked Sensor Systems, Baltimore, MD, USA (2004) p.39–49