An Experimental Framework for the Analysis and Validation of Software Clocks

Andrea Bondavalli[1], Francesco Brancati[1], Andrea Ceccarelli[1], and Lorenzo Falai[2]

[1] University of Florence, Viale Morgagni 65, I-50134, Firenze, Italy
{bondavalli,francesco.brancati,andrea.ceccarelli}@unifi.it
[2] Resiltech S.r.l., Via Filippo Turati 2, 56025, Pontedera (Pisa), Italy
lorenzo.falai@resiltech.it

Abstract. The experimental evaluation of software clocks requires the availability of a high quality clock to be used as reference time and a particular care for being able to immediately compare the value provided by the software clock with the reference time. This paper focuses i) on the definition of a proper evaluation process and consequent methodology, and ii) on the assessment of both the measuring system and of the results. These aspects of experimental evaluation activities are mandatory in order to obtain valid results and reproducible experiments, including comparison of possible different realizations or prototypes. As case study to demonstrate the framework we describe the experimental evaluation performed on a basic prototype of the Reliable and Self-Aware Clock (R&SAClock), a recently proposed software clock for resilient time information that provides both current time and current synchronization uncertainty (a conservative and self-adaptive estimation of the distance from an external reference time).

Keywords: experimental framework and methodology, assessment and measurements, software clocks, R&SAClock, NTP.

1 Introduction

Experimental evaluation (i.e., testing [2]) offers the possibility to observe a system in its actual execution environment and to perform fault forecasting and removal (e.g., through fault injection [1]). During experimental evaluation, measurement results (the data) are collected and allow to discover insight on the tested system. It is mandatory that these results are valid i.e., they are not altered due to an intrusive set up, a badly designed experiment or measurement errors. To achieve valid results it is necessary i) a proper evaluation process that is carefully designed, its objective pinpointed, and the relevant quantities carefully addressed, and ii) an assessment of the measuring system (the set of instruments used to perform the measurements) and of the results [13].

The focus on the paper is on the process and methodology for the experimental validation of software clocks. The main issues to address for properly cope with this problem are the provision of a high quality clock to be used as reference time and a particular care when designing the measuring system so to be able to immediately compare the value provided by the software clock with the reference time [15]. In this

S. Lee and P. Narasimhan (Eds.): SEUS 2009, LNCS 5860, pp. 69–81, 2009.

paper, the evaluation process is carefully planned, and the validity of the measuring system and of the results is investigated and assessed through principles of measurement theory. As further benefits, the experimental set-up and the whole evaluation process can be easily adapted and reused for the evaluation of different types of software clocks and for comparisons of possible different implementations or prototypes within the same category.

The paper illustrates the experimental process and set up by showing the evaluation of a prototype of the Reliable and Self-Aware Clock (R&SAClock [5]), a recently proposed software clock. R&SAClock exploits services and data provided by any chosen synchronization mechanism (for external clock synchronization) to provide both the current time and the current synchronization uncertainty (an adaptive and conservative estimation of the distance of local clock from the reference time).

The rest of this paper is organized as follows. In Section 2 we introduce our case study: the R&SAClock prototype that will be analyzed. Section 3 describes our experimental process and the measuring sub-system. Section 4 presents the results obtained by the planned experiments and their analysis. Conclusions are in Section 5.

2 The Reliable and Self-aware Clock

2.1 Basic Notions of Time and Clocks

Let us consider a distributed system composed of a set of nodes. We define *reference time* as the unique time view shared by the nodes of the system, *reference clock* as the clock that always holds the reference time, and *reference node* as the node that owns the reference clock. Given a local clock c and any *time instant* t, we define $c(t)$ as the *time value* read by local clock c at time t.

The behavior of the local clock is characterized by the quantities *offset*, *accuracy* and *drift*. The *offset* $\Theta_c(t) = t - c(t)$ is the actual distance of local clock c from reference time at time t [9]. This distance may vary through time. *Accuracy* A_c is an upper bound of the offset [10] and is often adopted in the definition of system requirements and therefore targeted by clock synchronization mechanisms. *Drift* $\rho_c(t)$ describes the rate of deviation of a local clock c at time instant t from the reference time [10].

Unfortunately, accuracy and offset are usually of practical little use for systems, since accuracy is usually a high value, and it is not a representative estimation of current distance from reference time, and offset is difficult to measure exactly at any time t. Synchronization mechanisms typically compute an *estimated offset* $\widetilde{\Theta}_c(t)$ (and an *estimated drift* $\tilde{\rho}_c(t)$), without offering guarantees and only at synchronization instants. Instead of the static notion of accuracy, a dynamic conservative estimation of the offset provides more useful information. For this purpose, the notion of uncertainty as used in metrology [4], [3] can provide such useful estimation: we define the *synchronization uncertainty* $U_c(t)$ as an adaptive and conservative evaluation of offset $\Theta_c(t)$ at any time t; that is $A_c \geq U_c(t) \geq |\Theta_c(t)| \geq 0$ [5].

Finally we define the *root delay* $RD_c(t)$ as the transmission delay (one-way or round trip, depending on the synchronization mechanism), including all system-related delays, from the node that holds local clock c to the reference node [9].

2.2 Basic Specifications of the R&SAClock

R&SAClock is a new software clock for external clock synchronization (a unique reference time is used as target of the synchronization) that provides to users (e.g., system processes) both the time value and the synchronization uncertainty associated to the time value [5]. R&SAClock is not a synchronization mechanism, but it acts as a new software clock that exploits services and data provided by any chosen synchronization mechanism (e.g., [9], [11]).

When a user asks the current time to R&SAClock (by invoking the function *getTime*), R&SAClock provides an enriched time value [*likelyTime, minTime, maxTime, FLAG*]. *LikelyTime* is the time value computed reading the local clock i.e., $likelyTime = c(t)$. *MinTime* and *maxTime* are computed using the synchronization uncertainty provided by the internal mechanisms of R&SAClock. More specifically, for a clock c at any time instant t, we extend the notion of synchronization uncertainty $U_c(t)$ distinguishing between a right synchronization uncertainty (positive) $U_{cr}(t)$ and a left synchronization uncertainty (negative) $U_{cl}(t)$, such that $U_c(t) = max[U_{cr}(t); -U_{cl}(t)]$. Values *minTime* and *maxTime* are respectively a left and a right bound of the reasonable values that can be attributed to the actual time: *minTime* is set to $c(t) + U_{cl}(t)$ and *maxTime* is set to $c(t) + U_{cr}(t)$.

The user that exploits the R&SAClock can impose an accuracy requirement, that is the largest synchronization uncertainty that the user can accept to work correctly. Correspondingly, R&SAClock can give value to its output *FLAG*, which is a Boolean value that indicates whether the current synchronization uncertainty is within the accuracy requirement or not. The main core of R&SAClock is the *Uncertainty Evaluation Algorithm* (*UEA*), that equips the R&SAClock with the ability to compute the synchronization uncertainty. Possible different implementations of the UEA may lead to different versions of R&SAClock (for example, in [5] and [12] two different versions are shown). Besides the R&SAClock specification shown, we identify the following two non-functional requirements:

REQ1. The service response time provided by R&SAClock is bounded: there exists a maximum reply time Δ_{RT} from a *getTime* request made by a user to the delivery of the enriched time value (the probability that the *getTime* is not provided within Δ_{RT} is negligible).

REQ2. For any *minTime* and *maxTime* in any enriched time value generated at time t, it must be $minTime \leq t \leq maxTime$ with a coverage Δ_{CV} (by coverage we mean the probability that this equation is true).

2.3 R&SAClock Prototype and Target System

Here we describe the prototype of the R&SAClock and the system in which it is executing as our target system used for the subsequent experimental evaluations. The R&SAClock prototype works with Linux and with the NTP synchronization mechanism. The UEA implemented in this prototype computes a symmetric left and right synchronization uncertainty with respect to *likelyTime* i.e., $-U_{cl}(t) = U_{cr}(t)$ and $U_c(t) = U_{cr}(t)$ [5]. Using functionalities of both NTP and Linux, the UEA gets i) $c(t)$ querying the local clock and ii) the root delay $RD_c(t)$ and the estimated offset $\widetilde{\Theta}_c(t)$ by

monitoring the NTP log file (NTP refreshes root delay and estimated offset when it performs a synchronization).

The behavior of the UEA is as follows. First, the UEA reads an upper bound δ_c, fixed to 50 part per million (ppm) in the experiments, on clock drift from a configuration file and listens on the NTP log file. When NTP updates the log file, the UEA reads the estimated offset and the root delay and starts a counter called *TSLU* which represents the *T*ime elapsed *S*ince *L*ast (more recent) *U*pdate of root delay and estimated offset.

Given t the most recent time instant in which root delay and estimated offset have been updated, at any time $t_1 \geq t$ the synchronization uncertainty $U_c(t_1)$ is computed as:

$$U_c(t_1) = |\widetilde{\Theta}_c(t)| + RD(t) + (\delta_c \cdot TSLU). \quad (1)$$

The basic idea of (1) is that, given $U_c(t) = |\widetilde{\Theta}_c(t)| + RD(t) \geq |\Theta_c(t)|$ at time t, we have $U_c(t_1) \geq \Theta_c(t_1)$ at $t_1 \geq t$ (a detailed discussion is in [5]).

The target system is depicted in Fig. 1. The hardware components are a Toshiba Satellite laptop, that we call PC_R&SAC, and the NTP servers connected to PC_R&SAC through high-speed Internet connection. The software components are the R&SAClock prototype, the NTP client (a process daemon) and the software local clock of PC_R&SAC. The NTP client synchronizes the local clock using information from the NTP servers.

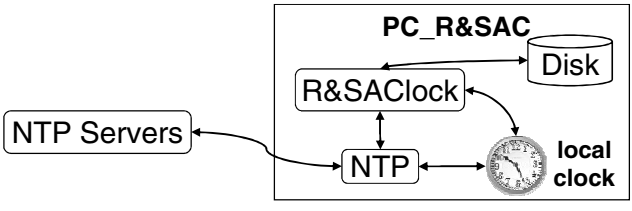

Fig. 1. The target system: R&SAClock for NTP and Linux

3 The Experimental Evaluation Process and Methodology

The process of our experimental evaluation starts by identifying the goals, and then designing the experimental set up (composed by injectors, probes and the experiment control subsystems), the planning of the experiments to conduct and finally defining the structure and organization of the data related to the experiments.

3.1 Objective

The objective of our analysis is in this case to validate a R&SAClock prototype, verifying if and how much it is able to fulfill its requirements in varying operating conditions, especially nominal and faulty. We aim to assign values to Δ_{RT} and Δ_{CV}.

3.2 Planning of the Experimental Set Up

The experimental set up is described by the grey components of Fig. 2. Its hardware components are a HP Pavilion desktop, that we call PC_GPS, and a high quality GPS (Global Positioning System [8]) receiver. Through such receiver, the PC_GPS is able to construct a clock tightly synchronized to the reference time that is used as the reference clock. Obviously such reference clock does not hold the exact reference time, but it is orders of magnitude closer to the reference time than the clock of the target system: it is sufficiently accurate to suit our experiments.

The PC_GPS contains a software component for the control of the experiment (Controller hereafter) that is composed of an actuator that commands the execution of workload and faultload to the Client (e.g., requests for the enriched time value, that the Client will forward to the R&SAClock), and of a monitor that receives the information from the Client of the completion of services, accesses the reference clock and writes data on the disk. The Client is a software component located on the target system: it performs injection functions, to inject the faultload and to generate the workload, and probe functions to collect the relevant quantities and write this data on the disk.

An experimental setup in which the target system and the Controller are placed on the same PC would require two software clocks, thus introducing perturbations that are hard to address.

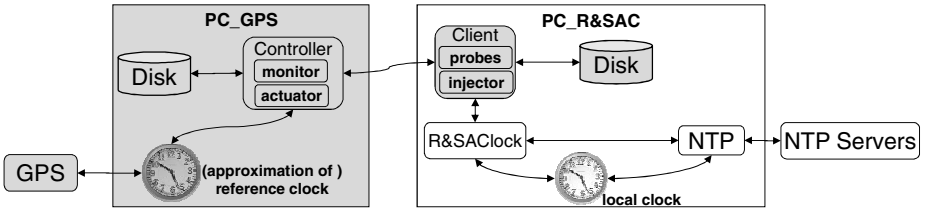

Fig. 2. Measuring system and target system

Given the description of the target system (an implementation of an R&SAClock) and of the experimental set up, we describe now how the relevant measures can be collected. To verify requirements REQ1 and REQ2, our measuring system implements solutions which are specific for R&SAClocks but general for any instance of it.

To evaluate REQ1, the Client logs, for each request for the enriched time value, the time *Client.start* in which the Client queries the R&SAClock and the time *Client.end* in which it receives the answer from R&SAClock (these two values are collected reading the local clock of PC_R&SAC).

To verify REQ2 (see Fig. 3), the measuring system computes a time interval [*Controller.start*, *Controller.end*] that contains the actual time in which the enriched time value is generated. This time interval is collected by the Controller's monitor that reads the reference clock. *Controller.start* is the time instant in which a request for the enriched time value is sent by the Controller to the Client, and *Controller.end* is the time instant in which the enriched time value is received by the Controller. REQ2 is satisfied if [*Controller.start*, *Controller.end*] is within [*minTime*, *maxTime*].

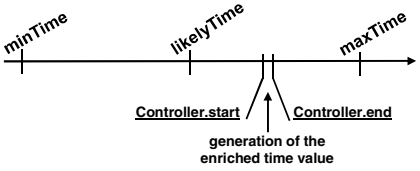

Fig. 3. The time interval [Controller.start, Controller.end] that allows to evaluate REQ2

3.3 Instrumentation of the Experimental Set Up

PC_R&SAC is a Linux PC connected to (one or more) NTP servers by means of an Internet connection and to the PC_GPS (another Linux-based PC) by means of an Ethernet crossover cable.

The Controller and the Client are two high-priority processes that communicate using a socket. Fig. 4 shows their interactions to execute the workload. The Client waits for Controller's commands. At periodic time intervals, the Controller sends a message containing a *getTime* request and an identifier ID to the Client, and logs ID and *Controller.start*. When the Client receives the message, it logs ID and *Client.start* and performs a *getTime* request to the R&SAClock. When the Client receives the enriched time value from R&SAClock, it logs the enriched time value, *Client.end* and ID, and sends a message containing an acknowledgment and ID to the Controller. The Controller receives the acknowledgment and logs ID and *Controller.end*. At the experiment termination, the log files created on the two machines are combined pairing entries with the same ID.

Controller and Client interact to execute the faultload as follows. The Controller sends to the Client the commands to inject the faults (e.g. to close the Internet connection or to kill the NTP client), and logs the related event; the Client executes the received command and logs the related event. Data logging is handled by NetLogger [6], a tool for logging data in distributed systems guaranteeing negligible system perturbation.

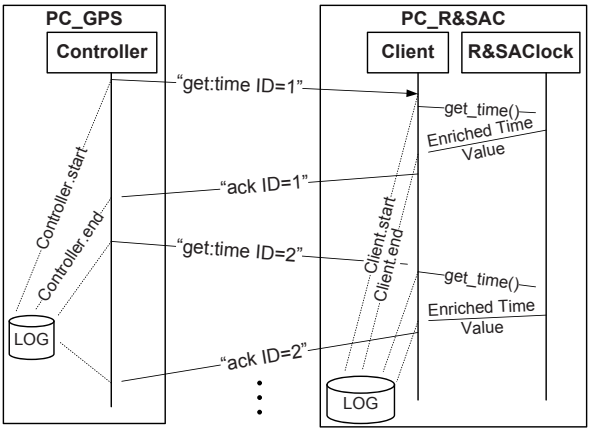

Fig. 4. Controller, Client and R&SAClock interactions to execute the workload

3.4 Planning of the Experiments

Here the execution scenarios, the faultload, the workload and the experiments need to be defined. Our framework allows to easily define and modify such aspects of the experimental planning as desired or needed by the objectives of the analysis. In this case of the basic R&SAClock prototype, with the objective of showing how our set up works, the selection has been quite simple and not particularly demanding (for the target system) or rich (to obtain a complete assessment).

Execution scenarios. Two execution scenarios are considered, that corresponds to the two most important operating conditions of the R&SAClock: i) *beginning of synchronization* (the NTP client has an initial transient phase and starts to synchronize to the NTP servers, and the PC_R&SAC is connected to the network), and ii) *nominal operating condition* of the NTP client (it represents a steady state phase: the NTP client is active and fully working, holding information on local clock).

Faultload. Beside the situation of *no faults*, the following situations are considered: i) *loose of connection* between the NTP client and servers (thus making the NTP servers unreachable), and ii) *failure of the NTP client* (the Controller commands to shut down the NTP client). These are the most common scenarios that the R&SAClock is expected to face during its execution.

Workload. The *workload* selected is simply composed of *getTime* requests to the R&SAClock to be sent once per second. This workload does not allow to observe the behavior of the target system under overload or stress conditions, and must be modified to a more demanding one if one wants to thoroughly evaluate the behavior of the target system.

Experiments. Combining the scenarios, the faultload and the workload, we identify four significant experiments: i) beginning of synchronization, and no faults injected; ii) nominal operating condition, and no faults injected; iii) nominal operating condition, and failure of NTP client; iv) nominal operating condition, and loose of connection. The duration of each experiment is 12 hours; the rationale (confirmed by the collected evidence) is that 12 hours are more than sufficient to observe all the relevant events in each experiment.

3.5 Structure of the Data

The structure and organization of the data related to the experiments is shown in Fig. 5, and are organized using a star-schema [7]. The organization of the data using a star schema is an intuitive model composed of facts and dimension tables. Facts tables (such as table R&SAClock_Results in Fig. 5) contain an entry for each experiment run. Each entry in its turn contains the values observed for the relevant metrics and values for the parameters of the experimental setup used in that specific experiment run. Each dimension table refer to a specific characteristic of the experimental set up and contain the possible values for that specific feature (in Fig. 5, tables Scenario, Workload, Faultload, Experiment, Target_System).

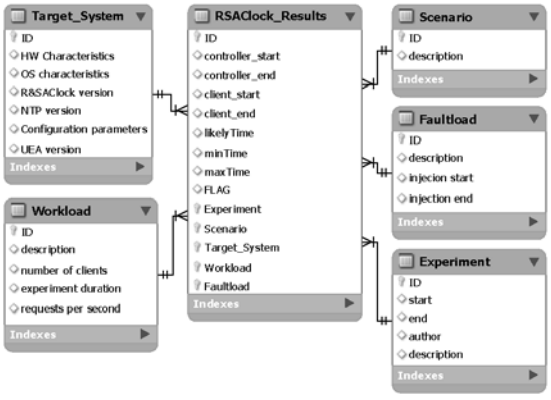

Fig. 5. Structure of the data related to the experiment organized following a star-schema

This model allows to structure and highlight the objectives, the results and the key elements of each evaluation; consequently, it helps to reason on and keep the purposes and contexts of the analysis clear.

4 Analysis of Results

We subdivide the offline process to investigate the results collected in three activities: i) data staging on the raw data collected, to populate a database where data can be easily analyzed, ii) investigation of the validity of the results, and iii) presentation and discussion of the results.

4.1 Data Staging

We organize the data staging in three steps: log collection, log parsing and database loading. In log collection the events are merged in a unique log file using NetLogger's API (Application Programming Interface). In log parsing we use an AWK script (AWK is a programming language for processing text-based data) to parse raw data and create CSV (Comma Separated Value: a standard data file format used for the storage of data structured in table form) files, that are easier to handle than the raw data. In database loading we create SQL (Structured Query Language) queries starting from the content of CSV files to populate the database.

4.2 Quality of the Measuring System and Results

We assess the quality of the measuring system along the principles of experimental validation and fault injection [1], [14] and the confidence of the results through principles of measurement theory [3], [4]; we focus on the uncertainty of the results and the intrusiveness of the measuring system [3], [4]. Furthermore, repeatability [4] is discussed to identify to what extend the results can be replicated in other experiments.

Intrusiveness (perturbation). Despite the Client is a high priority thread, its execution does not perturb the target system and does not affect results. In fact, the other relevant thread that if delayed could induce a change in the system behavior is the R&SAClock thread, responsible for the generation of the enriched time value. This thread is the one with the highest priority in the target system.

Uncertainty. The actual time instant in which the enriched time value is computed is within the interval [*Controller.start*, *Controller.end*] whose length is constituted by the length of the interval [*Client.start*, *Client.end*] plus the delay due to the communications between the Client and the Controller. The sampled duration of this interval is within 1.7 ms in the 99% of cases. Analyzing the other 1% of executions we discovered they were affected by large communications delay: we decided then to discard these runs. We set the time instant in which the enriched time value is computed as the middle value of the interval [*Controller.start*, *Controller.end*], that is (*Controller.end* + *Controller.start*) / 2. Such value is affected by an uncertainty of (*Controller.end* − *Controller.start*) / 2 = 0.85 ms (milliseconds) and confidence 1 [4].

Since the time instant in which the enriched time value is computed is the time instant in which the *likelyTime* is generated, we can attribute to the *likelyTime* the same uncertainty. As a consequence also the measured offset (difference between reference time and *likelyTime*) suffers from the same uncertainty.

In principle also the resolution of our measurement system should contribute to the final uncertainty. In our case, the Linux clock resolution is 1 μs (microsecond) and its contribution is irrelevant to the computation of uncertainty.

Repeatability. Re-executing the same experiment will almost certainly produce different data. Repeatability in deterministic terms as defined in [4] is not achievable. However, re-executing the same set of experiments will lead to statistically compatible results and to the same conclusions.

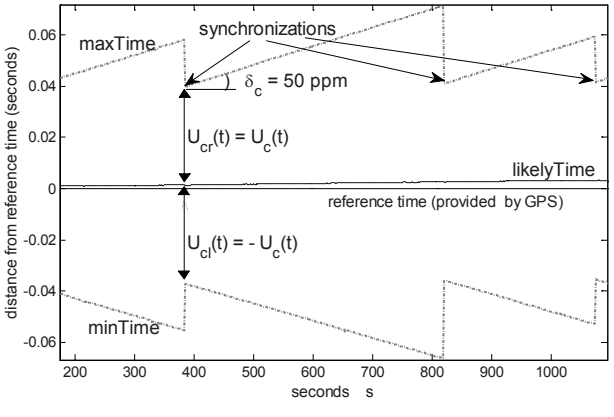

Fig. 6. A sample trace of execution of the R&SAClock

4.3 Results

In Fig. 6 we explain the results shown in Fig. 7-9. Reference time is on the *x*-axis. The central dashed line with label *likelyTime* is the distance between *likelyTime* and reference time: it is the offset of the local clock of PC_R&SAC that may vary during the experiments execution. The two external lines represent the distance of *minTime* and *maxTime* from reference time; this distance vary during experiments execution.

If the NTP client performs a synchronization to the NTP servers at time *t*, the synchronization uncertainty $U_c(t)$ is set to $\left|\widetilde{\Theta}_c(t)\right| + RD(t)$. After the synchronization, the synchronization uncertainty grows steadily at rate $\delta_c = 50$ ppm until the next synchronization. The time interval between two adjacent synchronizations varies depending on the behavior of the NTP client.

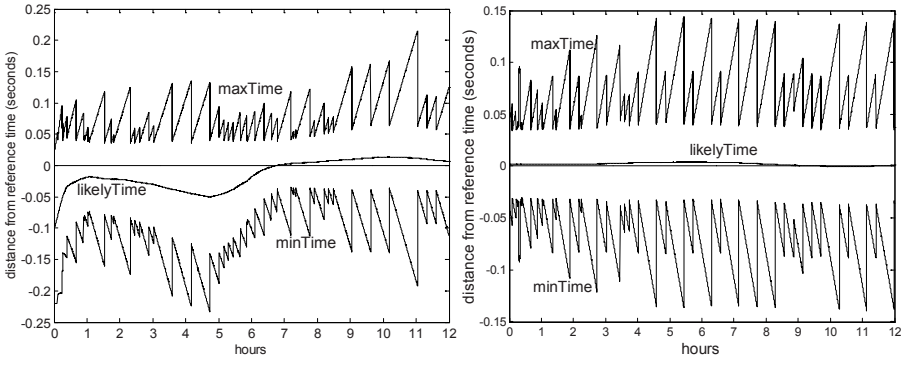

Fig. 7. a) Exp. 1: beginning of synchronization. b) Exp. 2: nominal operating condition.

Experiment 1. Fig. 7a shows the behavior of the R&SAClock prototype at the beginning of synchronization. The initial offset of the local clock of PC_R&SAC is 100.21 ms. At the beginning of the experiment, the NTP client performs frequent synchronizations to correct the local clock. After 8 hours, the offset is close to zero and consequently the NTP client performs less frequent synchronizations: this behavior affects $U_c(t)$, that increases. Reference time is always within [*minTime*, *maxTime*].

Experiment 2. Fig. 7b shows the behavior of the R&SAClock prototype when the target system is in nominal operating condition and no faults are injected. The offset is close to zero and the local clock of PC_R&SAC is stable: the NTP client performs rare synchronization attempts. Reference time is always within [*minTime*, *maxTime*]. $U_c(t)$ varies from 65.34 ms to 281.78 ms; the offset is at worst 4.22 ms.

Experiment 3. Fig. 8a shows the behavior of the R&SAClock prototype when the target system is in nominal operating condition and the NTP client is failed (the figure does not contain the beginning of synchronization, about 8 hours). *LikelyTime* drifts from reference time (the NTP client does not discipline the local clock): after 12 hours, the offset is close to 500 ms. Since the actual drift of local clock is smaller than δ_c, the reference time is always within [*minTime*, *maxTime*].

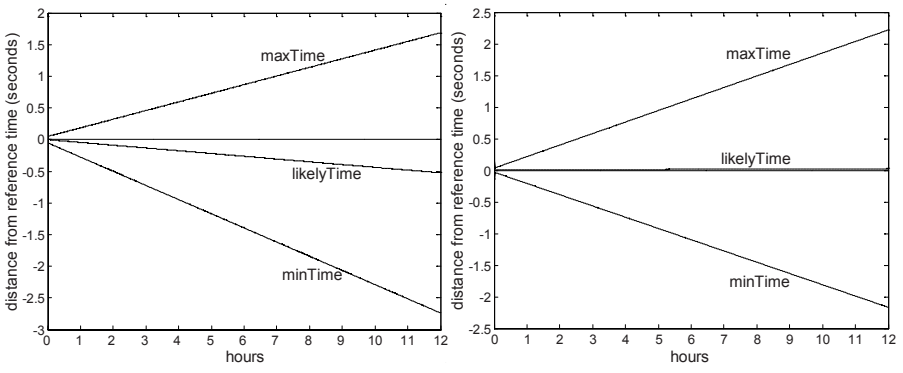

Fig. 8. a) Exp. 3: failure of the NTP client. b) Exp. 4: unavailability of the NTP servers.

Experiment 4. Fig. 8b shows the behavior of the R&SAClock prototype when the target system is in nominal operating condition and Internet connection is lost (the NTP client is unable to communicate with the NTP servers and consequently does not perform synchronizations). NTP client disciplines the local clock using information from the most recent synchronization. After 12 hours the offset is 26.09 ms: the NTP client, thanks to stable environmental conditions, succeeds in keeping *likelyTime* relatively close to the reference time. Reference time is within [*minTime*, *maxTime*].

Assessment of REQ1 and REQ2. The time intervals [*Client.start*, *Client.end*] from *all the samples* collected in the experiments are shown in Fig. 9. The highest value is 1.630 ms, thus REQ1 is satisfied simply by setting $\Delta_{RT} \geq 1.630$ ms. However such multimodal distribution shows that the response time of a *getTime* varies significantly depending on current system activity and possible overheads of system resources. This suggest the possibility to build a new improved prototype with a reduced Δ_{RT} and less variance in the interval [*Client.start*, *Client.end*] (e.g., implementing the R&SAClock within the OS layer and the *getTime* as an OS call).

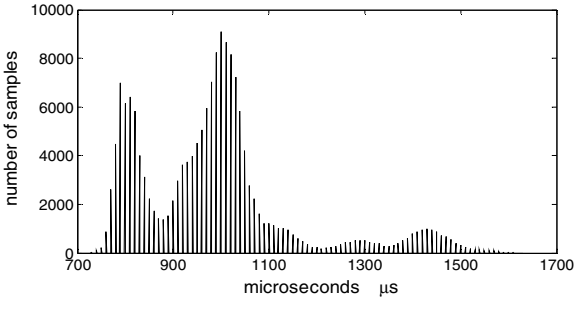

Fig. 9. Intervals [*Client.start*, *Client.end*]

In the experiments shown, REQ2 is always satisfied ($\Delta_{CV} = 1$). However, the interval [*minTime*, *maxTime*] is often a very large value, even when offset is *continuously* close to zero. Results of the experiments suggest that different (more efficient) UEAs that predict the oscillator drift behavior using statistical information on past values may be developed and used. A preliminary investigation is in [12].

6 Conclusions and Future Works

In this paper we described a process and set up for the experimental evaluation of software clocks. The main issues addressed have been the provision of a high quality clock (resorting to high quality GPS) to be used as reference time in the experimental set up, and a particular care in designing the measuring system, that has allowed to assess the validity of the measuring system and of the results. Besides the design and planning of the experimental activities, the paper illustrates the experimental process and set up by showing the evaluation of a prototype of the R&SAClock [5], a recently proposed software clock. Even the simple experiments described allowed to get insight on the major deficiencies of the considered prototype and to identify the directions for improvements.

Acknowledgments. This work has been partially supported by the European Community through the project IST-FP6-STREP-26979 (HIDENETS - HIghly DEpendable ip-based NETworks and Services).

References

1. Hsueh, M., Tsai, T.k., Iyer, R.K.: Fault Injection Techniques and Tools. Computer 30(4), 75–82 (1997)
2. Avizienis, A., Laprie, J., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. on Dependable and Secure Computing 1(1), 11–33 (2004)
3. BIPM, IEC, IFCC, ISO, IUPAC, OIML: Guide to the Expression of Uncertainty in Measurement (2008)
4. BIPM, IEC, IFCC, ISO, IUPAC, OIML: ISO International Vocabulary of Basic and General Terms in Metrology (VIM), Third Edition (2008)
5. Bondavalli, A., Ceccarelli, A., Falai, L.: Assuring Resilient Time Synchronization. In: Proceedings of the 27th IEEE Symposium on Reliable Distributed Systems (SRDS), pp. 3–12. IEEE Computer Society, Washington (2008)
6. Gunter, D., Tierney, B.: NetLogger: a Toolkit for Distributed System Performance Tuning and Debugging. In: IFIP/IEEE Eighth International Symposium on Integrated Network Management, pp. 97–100 (2003)
7. Kimball, R., Ross, M., Thornthwaite, W.: The Data Warehouse Lifecycle Toolkit. J. Wiley & Sons, Inc., Chichester (2008)
8. Dana, P.H.: Global Positioning System (GPS) Time Dissemination for Real-Time Applications. Real-Time Systems 12(1), 9–40 (1997)
9. Mills, D.: Internet Time Synchronization: the Network Time Protocol. IEEE Trans. on Communications 39, 1482–1493 (1991)

10. Verissimo, P., Rodriguez, L.: Distributed Systems for System Architects. Kluwer Academic Publishers, Dordrecht (2001)
11. Cristian, F.: Probabilistic Clock Synchronization. Distributed Computing 3, 146–158 (1989)
12. Bondavalli, A., Brancati, B., Ceccarelli, A.: Safe Estimation of Time Uncertainty of Local Clocks. In: IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS (to appear, 2009)
13. Bondavalli, A., Ceccarelli, A., Falai, L., Vadursi, M.: Foundations of Measurement Theory Applied to the Evaluation of Dependability Attributes. In: Proceedings of the 37th Annual IEEE/IFIP international Conference on Dependable Systems and Networks, pp. 522–533 (2007)
14. Arlat, J., Aguera, M., Amat, L., Crouzet, Y., Fabre, J.-C., Laprie, J.-C., Martins, E., Powell, D.: Fault Injection for Dependability Validation: a Methodology and Some Applications. IEEE Trans. on Software Engineering 16(2), 166–182 (1990)
15. Veitch, D., Babu, S., Pàsztor, A.: Robust Synchronization of Software Clocks across the Internet. In: Proceedings of the 4th ACM SIGCOMM Conference on internet Measurement, pp. 219–232 (2004)