SYNCHRONIZED UNIVERSAL TIME COORDINATED FOR DISTRIBUTED REAL-TIME SYSTEMS

U. Schmid*

Technical University of Vienna, Department of Automation, Treitlstraße 3, A-1040 Vienna, Austria. Email: s@auto.tuwien.ac.at

This paper presents a novel technique for establishing a highly accurate global time in fault-tolerant, large-scale distributed real-time systems. Unlike the usual clock synchronization approaches, the proposed *clock validation technique* provides a precise system time that also relates to an external time standard like UTC with high accuracy. The underlying idea is to validate time information of external time sources like GPS-receivers against a global time maintained by the local clocks in the system. As an example, a promising *interval-based clock validation algorithm ICV* that exhibits excellent fault-tolerance properties is outlined and analyzed. It requires only a few highly-accurate external time sources and provides each node with the actual accuracy of its clock.

Clock validation; universal time coordinated (UTC); clock synchronization; fault-tolerant distributed real-time systems

1. INTRODUCTION

During the past few years, much research has been conducted towards a common view of time in fault-tolerant distributed systems. There are more than 60 papers listed in a 1993 bibliography (Yang and Marsland, 1993) on clock synchronization in distributed systems. As a result, the problem of *internal synchronization*, i.e., keeping local clocks within well-defined bounds of each other, is relatively well understood.

The nature of the problem, however, radically changes when the requirement of a mutually consistent global time is extended to a global time that relates to some external time standard like *universal time coordinated* (UTC). The reason is that there always exists a certain tradeoff between *precision* (the maximum deviation of two different clocks in the system) and *accuracy* (the maximum deviation of any clock from real time), since internal synchronization is achieved by (slightly) modifying the progress of global time. Thus, as observed in (Christian, 1989), p. 151, the problem of fault-tolerant *external synchronization* constitutes a research topic in its own right.

Taking into account that real-time systems are becoming more and more prevalent in daily life, where UTC is the only common (and official!) notion of time, it is obvious that systems employing their own idea of time might be of questionable use for future applications. Promising sources of UTC are readily available now, most notably the NAVSTAR *global position system* (GPS). Consequently, it is high time to focus on the problem of how to provide a global time synchronized to UTC for large-scale, fault-tolerant, distributed real-time systems.

Intended for the *new ideas/promising approaches* section of WRTP'94, this paper presents the directions and some results of related research performed in the project *SynUTC*[1] at TU Vienna. It is organized as follows: Section 2 presents the system model underlying the investigations. Section 3 is devoted to the discussion of a novel *clock validation technique*, including its relation to previous work. The description and (partial) analysis of a particular *interval-based clock validation algorithm ICV* is given in Section 4, along with some practical considerations in Section 5. Finally, the directions of current and further research are presented in the concluding Section 6.

2. SYSTEM MODEL

Large-scale distributed real-time systems are usually built upon a more or less hierarchical internetwork comprising several heterogeneous subnetworks. In today's automation systems, for example, there might be an Ethernet-based interconnection of *top-level* nodes, some of which act as gateways to *cell-level* networks relying on fieldbuses like *Profibus*, and possibly further *machine-level* subnetworks connecting *programmable logic*

[1] SynUTC is supported by the Austrian Science Foundation (FWF) under grant no. P10244-ÖPH.

controllers (PLCs) and sensor/actor devices via a *contoller area network* (CAN) bus.

Commonly, there is a large number of nodes, in particular at low levels in the hierarchy, often with considerable geographical distances between them. The number of suitable communication architectures is limited by often extremely poor environmental conditions (i.e., noise) and lack of space for "voluminous" (e.g., multiple) interconnections.

A system model suitable for that type of framework is made up of

- an arbitrary topology of *synchronization subnets* (SSNs), which are interconnected by one or more *gateways*, i.e., nodes participating in two or more SSNs,
- an *improper*, external synchronization subnet virtually "connecting" external time sources like GPS-receivers.

The most common topology is of course tree-like, as exemplified in the automation system above.

Let SSN_i denote the i-th SSN according to some suitable enumeration, with SSN_0 representing the external one. A single (proper) SSN may be modelled as follows:

- An "ordinary" *broadcast data network* connecting all n *nodes* of the SSN. Let g_i be the number of nodes which act as a gateway to the i-th interconnected SSN_i. Note that a single node may be a gateway to multiple SSNs.
- Synchronous network behaviour, with *(broadcast) transmission delay* δ_{ij} from node i to j satisfying $\delta_{ij} \in [d_{ij} - \varepsilon_{ij}, d_{ij} + \varepsilon_{ij}]$; d_{ij} represents the deterministic part and ε_{ij} a (usually reasonably small) bound on the random part of δ_{ij}. Data integrity may be reliably checked via certain checksumming methods.
- Nodes (and network) may suffer from *crash*, (send and receive) *omission* and *performance* (*timing*) faults, but not from arbitrary *malicious* —*byzantine*, but see below— ones. (Generally, however, there is a low probability of failure.)
- Each node i is endowed with a (continuously) adjustable *clock* $C_i(t)$ with *rate deviation* bounded by ρ_i, ensuring that $(t-t_0)(1-\rho_i) \leq |C_i(t) - C_i(t_0)| \leq (t-t_0)(1+\rho_i)$ for all real-times $t \geq t_0$, providing there is no adjustment in $[t_0, t]$ and the clock is not faulty.
  A faulty clock (counted as a node fault) may exhibit an arbitrary behaviour, including a *two-faced* one, thus accounting for (restricted) *byzantine* node faults.
- A node i may be connected to an external time source disciplining its clock $C_i(t)$. If there is no fault, it provides UTC with some *accuracy* α_0, i.e., $|C_i(t) - t| < \alpha_0$ for all real-times t. A faulty clock (counted as a node fault) may behave arbitrarily as well.

The appropriateness of that system model for large-scale distributed real-time systems, as mentioned at the beginning of this section, needs some additional remarks.

First, it may reasonably be assumed that all networks employed in today's real-time systems (802.3DCR, token ring, Profibus, CAN, ...) support some kind of broadcast and provide reasonably synchronous behaviour (even CSMA/CD provides enough synchrony, at least with some hardware support); see (Veríssimo and Rodrigues, 1992) for a similar reasoning. Note that dedicated time channels (e.g., clock signals) of any kind are not considered (except for the external time sources like GPS, of course). Apart from being of questionable success in noisy environments, such an approach would need additional interconnections when applied in conjunction with existing network technology.

Another point of discussion concerns the fault model stated above: Ruling out arbitrary node faults may seem to be too unrealistic an assumption in the fault-tolerant real-time systems' context. However, it turns out that the underlying broadcast network restricts the number of classes of overall system behaviour considerably. In the most general case of totally arbitrary behaviour of a faulty ("adversary") node, nothing may be done about possible total blocking of overall system operation — just consider a node jamming the channel completely or maliciously impersonating several/all other nodes in the system. Thus, there is no other way but to exclude such arbitrary malicious faults (or the broadcast network assumption, of course) from the system model.

Note that although it is difficult to *protect* the system against a faulty node that impersonates other nodes in the general case, it is nevertheless possible to *detect* such malicious behaviour more or less "transparently" at the network level. More specifically, one just has to ensure that any node broadcasts its time messages at most once. This is trivial to achieve in the absence of crashes, but requires some additional measures if such faults are to be considered; see (Schmid and Pusterhofer, 1995) for similar issues. If a node receives an impersonating message (carrying its own ID), it immediately transmits its original message as well, causing other nodes to discard both. Of course, the question arises whether such detection-only measures are actually worth while.

On the other hand, for a more restrictive fault model, where any node has to adhere at least to the (low-level) network protocol (the faulty ones behaving arbitrarily only on top of it, ruling out malicious impersonation a priori), there are protocols like *Deterministic Ethernet* (DOD/CSMA-CD), see (Le Lann and Rivierre, 1993) that guarantee bounded message delivery times. In that case, byzantine behaviour of a faulty node is restricted to broadcasting duplicated or replicated messages (produced by nodes not conforming to the protocol or caused by faults) to (*all*) the other nodes.

This is exactly the (byzantine) faulty behaviour covered by the two-faced clock assumption in the system model above. It is important to realize, however, that this type of node faults produces a byzantine fault in the *overall* system only if it leads to an inconsistent view among the nodes of the distributed system, that is, if there is a receive omission of the first message at a node m and another one of a duplicate at a node $l \neq m$.

Assuming that receive omissions of the same broadcast message as experienced by different receivers are independent and have probability $p << 1$, the probability $b_{n,l,m}$ that two successive broadcasts of two (different) messages M_1 and M_2 (by the same sender) partition the total number n of nodes in three subsets $\mathcal{L}$, $\mathcal{M}$ and $\mathcal{O}$ of cardinalities l, m and $n-l-m$, respectively, (where all nodes in $\mathcal{L}$ receive M_1 only, all in $\mathcal{M}$ $M2$ only, and the remaining ones in $\mathcal{O}$ either both or none of them) is required. This is of course a multinomial probability, namely

$$\begin{aligned} b_{n,l,m} &= \binom{n}{l, m, n-l-m}(1-p)^l p^l \\ &\quad \cdot p^m (1-p)^m \big(1-2p(1-p)\big)^{n-l-m}. \end{aligned}$$

Hence, the probability β_n that there is at least one node in $\mathcal{L}$ and at least another one in $\mathcal{M}$ reads

$$\begin{aligned} \beta_n &= \mathbf{P}\{l \geq 1 \text{ and } m \geq 1\} = \sum_{i=2}^{n}\sum_{l=1}^{i-1} b_{n,l,i-l} \\ &= \sum_{i=2}^{n} p^i (1-p)^i \big(1-2p(1-p)\big)^{n-i} \\ &\quad \cdot \sum_{l=1}^{i-1} \binom{n}{l, i-l, n-i}. \end{aligned}$$

An elementary manipulation involving the definition of the binomials shows that

$$\sum_{j=0}^{i} \binom{n}{j, i-j, n-i} = \sum_{j=0}^{i} \binom{n}{i}\binom{i}{j} = \binom{n}{i} 2^i,$$

so it follows that

$$\begin{aligned} \beta_n &= \sum_{i=2}^{n} \Big[\binom{n}{i} 2^i - 2\binom{n}{i}\Big] \big(p(1-p)\big)^i \\ &\quad \cdot \big(1-2p(1-p)\big)^{n-i} \\ &= 1 - \big(1-2p(1-p)\big)^n \\ &\quad -2np(1-p)\big(1-2p(1-p)\big)^{n-1} \\ &\quad -2\big(1-p(1-p)\big)^n \\ &\quad +2\big(1-2p(1-p)\big)^n \\ &\quad +2np(1-p)\big(1-2p(1-p)\big)^{n-1} \\ &= 1 - 2\big(1-p(1-p)\big)^n + \big(1-2p(1-p)\big)^n \end{aligned}$$

Employing $a^b = e^{b \log a}$ and the well-known series expansions $\log(1-x) = -\sum_{k \geq 1} x^k/k$ and $e^x = \sum_{k \geq 0} x^k/(k!)$ for $x \to 0$, an asymptotic[2] expression for $p \to 0$ and large $n < 1/p$ is obtained, namely

$$\begin{aligned} \beta_n &= 1 - 2e^{-np(1-p)+\mathcal{O}(np^2)} \\ &\quad + e^{-2np(1-p)+\mathcal{O}(np^2)} \\ &= 1 - 2\big(1-np(1-p)\big) + \mathcal{O}(n^2p^2) \\ &\quad +1 - 2np(1-p) + \mathcal{O}(n^2p^2) \\ &= \mathcal{O}(n^2p^2) \qquad \text{for } p \to 0. \end{aligned}$$

This simple analysis can be easily extended to the case of k transmissions of two messages M_1 and M_2 (giving a total of $2k$ transmissions); essentially, one has to replace $1-p$ by $k(1-p)$ and p by p^{2k-1}. In any case, the worst situation (largest value of β_n) occurs for $k = 1$, i.e., in the case above. Given a probability of omission in the range of $p \approx 10^{-3} \ldots 10^{-4}$, meaning that one message in 1000...10000 messages is lost on the average, and a total of $n = 100$ nodes, it follows that $\beta_{100} \approx 10^{-2} \ldots 10^{-4}$, which is reasonably small. Note that $p \approx 10^{-3} \ldots 10^{-4}$ is a pessimistic assumption for real networks, since *system-wide* omissions (experienced by all nodes due to electromagnetic noise on the transmission media, for example) do *not* contribute to p.

Clearly, β_n is the probability of only *one* byzantine fault, arising from two different messages transmitted by a *single* sender node. The probability of $f \geq 2$ byzantine faults (involving f senders) is obviously less than $\beta_n^f = \mathcal{O}(n^{2f}p^{2f})$; in the example above, one obtains a probability in the range $10^{-2f} \ldots 10^{-4f}$. Hence it follows that a large number f of (restricted) byzantine faults

[2] As usual, $f(x) = \mathcal{O}\big(g(x)\big)$ for $x \to 0$ if there is a positive constant M such that $|f(x)| \leq M|g(x)|$ for $x \to 0$. Note that the truncation of a convergent power series $f(x) = \sum_{k \geq 0} f_k x^k$ satisfies $f(x) = \sum_{k=0}^{K-1} f_k x^k + \mathcal{O}(x^K)$ for all $x \to 0$.

—although covered by the system model above— is very unlikely in practice.

3. CLOCK VALIDATION TECHNIQUES

Unlike in the context of (internal) clock synchronization, where much work has been done (see (Yang and Marsland, 1993) for a comprehensive bibliography and (Simons *et al.*, 1990; Ramanathan *et al.*, 1990) for overviews), there are only a few papers devoted to the problem of external synchronization. The latter research may be categorized as follows:

In (Kopetz and Ochsenreiter, 1987) some helpful general considerations concerning external synchronization and an outline of a rather straightforward solution based on a periodic modification of the rate of *all* clocks of the distributed system are given. The rate adjustment is based upon the comparison of a single node's clock against an external time standard. Hence, this algorithm is not fault-tolerant and provides only modest accuracy.

Another non-fault-tolerant algorithm for providing (slave) nodes of a distributed system with time information from a time server (a master node that has access to an external time standard) is based upon the probabilistic clock reading technique introduced in (Christian, 1989). A similar probabilistic approach underlies the carefully engineered *network time protocol* NTP, designed for providing a reliable time service in the *Internet*, see (Mills, 1991). Such probabilistic algorithms, however, are used for a very different application domain, usually involving asynchronous system models. The statistical methods required for that type of systems are not suitable for the system model in Section 2.

A promising approach to external synchronization is the *interval based paradigm* underlying the non-probabilistic, fault-tolerant time service developed in (Marzullo, 1984; Marzullo and Owicki, 1983). Interval-based algorithms represent (local) time information relating to an external standard like UTC by intervals that are known (i.e., supposed) to contain UTC. Given a set of such intervals from different time servers, a (small) interval that actually contains UTC may be determined — even if some of the intervals are faulty.

A different synchronization technique is used in the fault-tolerant, interval-based time service described in (Lamport, 1987). It employs a technique similar to (byzantine) agreement for disseminating time information provided by dedicated nodes with access to an external time standard. Clock synchronization techniques are used to synchronize the other node's local clocks to a global time derived from the disseminated time intervals.

The major deficiency of all the existing approaches lies in the fact that their idea of accurate time depends only on the reliability of some —or even a single— dedicated time server(s). In other words, they do not exploit all the timing information available in the distributed system. Hence, in order to tolerate a reasonable number of faults, a relatively large number of time servers is required.

This paper proposes an alternative approach that explicitly takes into account the various nodes keeping track of the time by means of their (low-accuracy but reliable) local clocks as well. More specifically, instead of relying on time information of some high-accuracy time servers only, one may compute a *validity interval* from all local clocks in the system to judge whether the information provided by high-accuracy time servers is valid. This *clock validation technique* allows the design of a fault-tolerant time service that relates to an external time standard — with only a few nodes actually having access to it.

Clock validation combines time information from external time sources like GPS-receivers and a global time maintained by techniques related to clock synchronization. The appropriateness of this "hybrid" approach results from the following facts:

- An external time source like a GPS-receiver may be characterized by
  - + very good accurracy (100 ns range),
  - + no overhead,
  - − low (and not sufficiently understood!) reliability,
  - − high cost.
- Clock synchronization techniques provide
  - − moderately good precision (μs range with hardware-support),
  - − some overhead,
  - + high (fully understood!) reliability,
  - + cheapness.

4. THE CLOCK VALIDATION ALGORITHM ICV

Although the basic principle of clock validation is rather simple, there are obviously several ways to design a particular clock validation algorithm. This section provides a brief description and analysis[3] of a promising *interval-based clock validation algorithm ICV*. It is based on ideas introduced in (Marzullo, 1984), which are strikingly

[3] The purpose of this exposition is to make clear the principles of ICV. To that end, abstracting away from several details is necessary in order to make the analysis easier to follow. A forthcoming Technical Report will contain all the omitted details.

well-suited for clock validation purposes.

In the course of the description, the following notation will be used: Real times (i.e., UTC) are represented by lower-case variables, local times as displayed by a node's clock are usually upper-case; for example, the current local time T_i at i's clock is $T_i = C_i(t)$. Intervals I are written in either of the two forms $I = [x, y]$, $x \leq y$, or $I = [c \pm w] = [c - w, c + w]$, $w \geq 0$ as appropriate; a single variable z may be used to denote the interval $[z, z]$. For an interval $I = [x, y]$, $|I| = y - x$ denotes its length, and $\text{center}(I) = (x + y)/2$ its midpoint. The sum of two intervals is defined as $[x, y] + [u, v] = [x + u, y + v]$, the intersection of two intervals is equal to $[x, y] \cap [u, v] = [\max(x, u), \min(y, v)]$ if $u \leq y$ and $v \geq x$, or $\emptyset$ otherwise.

Now, given a set $\mathcal{I}$ of intervals (of time) $\mathcal{I} = \{I_1, \ldots, I_m | m \geq 1\}$ with the property that at least $m - f$, $0 \leq f < m/2$ of the intervals are (mutually) *consistent*, meaning that they all contain a single point (of time) t (i.e., UTC), the smallest interval that is assured to contain t is provided by *Marzullo's function*: $\mathcal{M}^{m-f}(\mathcal{I})$ is defined to be the largest interval whose endpoints belong to at least $m - f$ of the I_j's. It can be computed in $\mathcal{O}(m \log m)$ time on the average by sorting the intervals' endpoints.

$\mathcal{M}$ has a number of interesting properties; in this paper, however, only the following one is needed: If $g \geq m - f$ of the intervals in $\mathcal{I}$ are consistent and d_f denotes the length of the $(g - f)$-largest of them, then

$$|\mathcal{M}^{m-f}(\mathcal{I})| \leq \begin{cases} 2d_f & \text{if } f < m/2, \\ \min(d_{2f}, 2d_f) & \text{if } f < m/3. \end{cases} \quad (1)$$

The proof is inspired by the line of reasoning in (Marzullo, 1984): The two endpoints of $\mathcal{M}^{m-f}(\mathcal{I})$ must intersect with at least $m - f$ of the I_j's and hence with at least $m - f - (m - g) = g - f$ of the consistent intervals (but not necessarily the same ones); note that $g - f \geq m - 2f > 0$ for $f < m/2$. Hence, the endpoints are not further apart than twice the length of the $(g - f)$-largest consistent interval. For $f < m/3$, the two endpoints even intersect with at least $g - 2f \geq m - 3f > 0$ *single* intervals: Assuming the contrary would imply at least $2(g - f)$ *different* consistent intervals; there are, however, only g ones available so that $2(g - f) - g = g - 2f > 0$ must be the same.

Now assume an SSN_l with n nodes, h of which act as gateways to a high-accurate SSN_h, and $l = n-h$ ordinary ones. In this paper, only the *deterministic* fault model which underlies almost all research on clock synchronization is considered: At most $f < l/2$ of the l low-accuracy nodes and $e < h/2$ of the h high-accuracy ones may exhibit a failure (caused by a node/clock fault or a transmission fault) at the same time. It should be pointed out, however, that it is the more realistic, that is, *probabilistic*, fault model that ultimately needs to be considered.

Supplied with those prerequisites, the principles of the clock validation algorithm ICV may be sketched as follows: First, each node i in the SSN maintains a state variable *accuracy* $\alpha_i(t)$ holding the current accuracy of i's clock with respect to UTC, i.e., $t \in \big[C_i(t) \pm \alpha_i(t)\big]$. Without resynchronizations, $\alpha_i(t)$ *deteriorates* according to the rate deviation ρ_i.

Every node i performs the following functions:

- *Transmission*: Periodically at times $C_i(t_i^I) = T_i^I = kT$, $k \geq 1$ where T denotes the *resynchronization period*, node i initiates the broadcast of a *time message* M_i that is actually performed at time t_i^S; the broadcast latency is assumed to be bounded by $t_i^S - t_i^I \leq \lambda$. The time message M_i contains (1) the local time $T_i^S = C_i(t_i^S)$ and (2) the current accuracy $\alpha_i = \alpha_i(t_i^S)$.
- *Message Reception*: When a time message M_j from node j arrives at node i at time t_j^R, a tuple consisting of the message and i's local time $T_j^R = C_i(t_j^R)$ of reception is stored in a set $\mathcal{S}$.
- *Synchronization*: At some time $C_i(t_i^\Lambda) = T_i^\Lambda = T_\ell^R + \Lambda$, where T_ℓ^R denotes the time of reception of the first valid time message (from node ℓ) and Λ is chosen appropriately to cover the maximum broadcast latency λ, the information stored in $\mathcal{S}$ is used to compute the intervals

$$\begin{aligned} I_j(t_i^\Lambda) \;=\; & [T_j^S \pm \alpha_j] \\ & +[d_{ji} \pm \varepsilon_{ji}] \\ & +\big[(T_i^\Lambda - T_j^R) \pm (T_i^\Lambda - T_j^R)\rho_i\big] \end{aligned} \quad (2)$$

  for all nodes j that provided time messages; $\mathcal{S} := \emptyset$ afterwards. The resulting intervals are then partitioned (according to their lengths) into a set $\mathcal{L}$ of *low*- and a set $\mathcal{H}$ of *high*-accuracy ones. From the $l - f \leq |\mathcal{L}| \leq l$ low-accuracy intervals, a *validity interval* $I_\mathcal{L} = \mathcal{M}^{l-f}(\mathcal{L})$ is computed. Similarly, an interval $I_\mathcal{H} = \mathcal{M}^{h-e}(\mathcal{H})$ is derived from the $h - e \leq |\mathcal{H}| \leq h$ high-accuracy intervals. Finally, i's clock is set to $\text{center}(I_\mathcal{H} \cap I_\mathcal{L})$ and its accuracy to $|I_\mathcal{H} \cap I_\mathcal{L}|/2$, providing that this resynchronization would constitute an improvement.

Lack of space prohibits a full analysis of that algorithm, even under the simple deterministic fault model. However, the following considerations should be convincing that this algorithm works and should also give some insight to the general line of reasoning.

Assume that at least $h - e$ of the high-accuracy nodes have been consistent during the (current) resynchronization period, satisfying an accuracy bound $\alpha_i(t) \le \alpha_h$, and the same for at least $l - f$ low-accuracy nodes with the accuracy bound α_l. Note that α_h is determined by the high-accuracy SSN_h to which the h gateway nodes are connected; it is (usually) not affected by the algorithm for SSN_l. If SSN_h is the external SSN_0, i.e., if the h gateway nodes are equipped with external time sources, then $\alpha_h = \alpha_0$ according to the system model in Section 2. Furthermore, let $\varepsilon_{ij} \le \varepsilon$ and $\rho_i \le \rho$ denote bounds on the maximum transmission delay uncertainities and the rate deviations, respectively, and α be either α_h or α_l depending on the context.

Then it follows from the construction of the intervals in Eq. (2) that the $I_j(t_i^\Lambda)$ as perceived at a non-faulty node i are also consistent: The first term of $I_j(t_i^\Lambda)$ is the "original" interval at node j, the second one accounts for the uncertainity arising from the transmission of the time message, and the third term covers the deterioration between reception and synchronization due to the rate deviation of i's clock; note that terms of order $\mathcal{O}(\rho_i^2)$, $\mathcal{O}(\alpha\rho)$, ... are ignored, so that $(1+\rho_i)^{-1} \approx 1 - \rho_i$, for example. Moreover, it is obvious that the intervals constructed at any two different (correct) nodes are also consistent, although consistency is not a transitive relation in general.

Taking into account that all consistent nodes are initiating their broadcasts almost simultaneously since $|t_j^I - t_l^I| \le 2\alpha$ for any two such nodes l, j, all messages from consistent nodes must be transmitted and hence be received within $\lambda + 2\alpha < \Lambda$. Ignoring small terms, it turns out that $(T_i^\Lambda - T_j^R)\rho_i \le (T_i^\Lambda - T_\ell^R)\rho_i = \Lambda\rho_i$ so that

$$|I_j(t_i^\Lambda)| \le 2\bigl(\alpha + \varepsilon + \Lambda\rho_i\bigr)$$

Hence it follows by Eq. (1) that $\mathcal{M}$ applied to the lower-accuracy intervals provides an interval with length at most $2c_l\bigl(\alpha_l + \varepsilon + \Lambda\rho_i\bigr)$ ($c_l = 1$ if $f < l/3$ or 2 otherwise) that contains UTC. Similarily, $\mathcal{M}$ of the higher-accuracy ones provides such an interval with length at most $2c_h\bigl(\alpha_h + \varepsilon + \Lambda\rho_i\bigr)$ ($c_h = 1$ if $e < h/3$ and 2 otherwise). Obviously, the latter one dominates the intersection of both (which must be non-empty due to consistency), so that *all* low-accuracy clocks are set to a value that differs by no more than $2c_h\bigl(\alpha_h + \varepsilon + \Lambda\rho_i\bigr)$; all accuracies are initialized to a value of at most $c_h\bigl(\alpha_h + \varepsilon + \Lambda\rho_i\bigr)$. Since within a resynchronization period the accuracy of a correct clock $C_i(t)$ detoriates at most by $\pm(T - \Lambda)\rho_i$, one eventually finds

$$\alpha_l = c_h(\alpha_h + \varepsilon + T\rho) \text{ with } c_h = \begin{cases} 2 & \text{if } e < h/2, \\ 1 & \text{if } e < h/3. \end{cases}$$

Therefore, the algorithm keeps the clocks closely synchronized with both each other and UTC. Note carefully that assuming high accuracy (which automatically implies high precision) is assumed here; hence, there is no need to consider the problem of achieving high precision in cases of low accuracy, cf. (Lamport, 1987).

ICV has the following desireable properties:

- The accuracy of each node is available locally at the node for failure recognition/recovery and system diagnosis purposes.
- If a node's clock has been in error, it usually recovers after the next synchronization.
- Gateway nodes are handled automatically in that a node's clock and accuracy are only altered if this constitutes an improvement.
- Since α_l is kept relatively small by ICV, h may be small (even $h = 1$ is not completely unreasonable) since validation intervals are usually reasonably discriminating.

The actual ICV currently under development will provide several additional features:

- Measurement of parameters like the deterministic part of transmission delays d_{ij} or maximum clock rate deviations ρ_i, since usually nobody is willing to carry out the necessary measurements beforehand. Note carefully that the correctness of ICV depends on a correct estimation of the rate deviation bounds ρ_i and the transmission uncertainities ε_{ij}. On the other hand, too coarse an estimation has an adverse effect on the achieveable performance (in particular, on accuracy).
- Flywheeling in the case of total (external) UTC loss and quick (initial) resynchronization.
- Providing improved average case accuracy and reliability, including graceful degradation.

5. PRACTICAL CONSIDERATIONS

For the successful application of a clock validation algorithm like ICV in practice, it has to provide several "engineering" features as well. Some of the most important ones are:

- Achieving a system-wide accuracy in the range of μs by taking every reasonable measure —including some hardware support— applicable to existing system architectures.

Steadily increasing processor and transmission speeds provide a rather tempting basis for increasing the "resolution" (granularity) of a distributed real-time system with respect to the environmental events of interest. In view of the 100 *ns* accuracy of GPS, it is not too unreasonable to aim at real-time applications with granularities about 1 μs.

Clearly, hardware is the only means to attain such high accuracies. Any hardware support, however, should be designed to be applicable to a reasonably broad class of existing system (in particular, network) architectures — users are seldom willing to sacrifice well-established standard technology for an "exotic" one, even if the latter promises increased functionality!

- Providing both full time (UTC) and easy-to-handle timestamps.

  UTC is the only official notion of time, and hence important for real-time systems interacting with the "official" world. Still, full UTC time is a memory-consuming entity and not very suitable for computations. Moreover, it has already been observed in (Kopetz and Ochsenreiter, 1987) that UTC is not *chronoscopic* in the sense that it does not (always) allow one to measure small intervals of time. In fact, UTC is generated from the *atomic time TAI* by occasionally inserting *leap seconds* in order to cope with the small disagreements between TAI and *astronomic time*.

  Therefore, it does not make much sense to choose UTC as the actual notion of system time. One should rather use a system time that is
  - chronoscopic,
  - easily convertible to UTC, TAI and related time notions,
  - compatible with (short) timestamps.

  A promising candidate is the 64-bit NTP time format (Mills, 1991) that consists of a 32-bit high-order part counting the number of real (TAI) seconds since January 1, 1900 (wrapping around every 136 years) and a 32-bit low-order part containing the fractional part of the current second (236 *ps* resolution). Short timestamps are easily obtained by using only a portion of the whole 64-bit NTP time information.

- Connection of various external time sources, in particular GPS-receivers.

  There are a couple of different external time sources available, most notably GPS and some less accurate ones like the German DCF-77. Most of them provide a certain "on-time signal" like a 1 *pps* (pulse-per-second) signal accurately synchronized to UTC; the higher-order part of UTC, however, is often available only some time after the on-time signal.

Those above-listed features of course require provisions in the clock validation algorithm itself. However, most of them will be supported by means of a dedicated ASIC (*UTCSU*). Inspired by the pioneering *clock synchronization unit* CSU of (Kopetz and Ochsenreiter, 1987), the *universal time coordinated synchronization unit* UTCSU will provide the following major functions:

- Rate and state adjustable clock (in NTP-format) with continuous amortization, extended by current clock accuracy and reliability information.
- Highly accurate (1 μs-range or less) message-timestamp-based clock offset estimation between different nodes of a SSN, even in conjunction with existing network technologies.
- Connection of external time sources providing standard signals like a 1 *pps* (pulse-per-second) signal synchronized to UTC, and support of gateway nodes, i.e., multiple network controllers.

Various timing features as well as interrupt capabilities and provisions for BST (*boundary scan test*) complete the intended functionality of the UTCSU.

6. DIRECTIONS OF CURRENT RESEARCH

As already mentioned Section 1, the major goal of this paper was to introduce basic issues of clock validation. The mainstreams of current research —performed in the context of the project *SynUTC*[4]— are:

- Development of an implementation of the ICV algorithm outlined in Section 4, that provides the engineering features mentioned in Section 5.
- Providing a full (average case) analysis of the algorithm's behaviour, relying on a probabilistic rather than a deterministic fault model.
- Development of the UTCSU ASIC.
- Design of an evaluation testbed consisting of several M68030 VME-CPUs equipped with a (deterministic) Ethernet-Coprocessor (802.3DCR) and the UTCSU.
- Systematic experimental evaluation, including dependability of GPS-receivers with respect to our fault model.

Apart from those issues dealing with the particular clock validation algorithm ICV of Section 4 (and possibly alternative ones), it is also necessary

[4] *SynUTC* is a joint project of the Department of Automation (U. Schmid, K. M. Schossmaier) and the Department of Computer Technology (D. Loy, M. Horauer), TU Vienna.

to investigate the relations to the external synchronization problem in general, see (Patt-Shamir and Rajsbaum, 1994) for related issues. In fact, much work remains to be done.

7. ACKNOWLEDGEMENTS

Several comments of W. A. Halang and an anonymous referee on a project proposal submitted to the Austrian Science Foundation, which all pointed out GPS as a promising UTC source are gratefully acknowledged. K. M. Schossmaier (UMass, now TU Vienna) suggested several improvements of an earlier version of the manuscript.

8. REFERENCES

Christian, F. (1989). Probabilistic Clock Synchronization. *Distributed Computing* **3**, 146–158.

Kopetz, H., and Ochsenreiter, W. (1987). Clock Synchronization in Distributed Real-Time Systems. *IEEE Trans. Comput.* **C-36(8)**, 933–939.

Lamport, L. (1987). Synchronizing Time Servers. *Technical Report Digital System Research Center* **18**, 1–33.

Le Lann, G., and Rivierre, N. (1993). *Real-Time Communications over Broadcast Networks: the CSMA-DCR and the DOD-CSMA-CD Protocols*. INRIA Rapport de recherche **1863**.

Marzullo, K. (1984). *Maintaining the Time in a Distributed System: An Example of a Loosely-Coupled Distributed Service*. Ph.D. thesis, Dept. of Electrical Engineering, Stanford University.

Marzullo, K., and Owicki, S. (1983). Maintaining the Time in a Distributed System. *ACM Operating System Review* **19(3)**, 44–54.

Mills, D.L. (1991). Internet Time Synchronization: The Network Time Protocol. *IEEE Trans. Commun.* **39(1)**, 1482–1493.

Patt-Shamir, B., and Rajsbaum, S. (1994). A Theory of Clock Synchronization. To appear in *Proc. 26th Symposium on the Theory of Computing (STOC)*.

Ramanathan, P., Shin, K.G., Butler, R.W. (1990). Fault Tolerant Clock Synchronization in Distributed Systems. *IEEE Computer* **23(10)**, 33–42.

Schmid, U., and Pusterhofer, A. (1995). SSCMP: The Sequenced Synchronized Clock Message Prototol. To appear in *Computer Networks and ISDN Systems*.

Simons, B., Lundelius-Welch, N., and Lynch, N. (1990). An Overview of Clock Synchronization. *Proc. Fault-Tolerant Distributed Computing*, 1990, Simons, B., Spector, A., ed., 84–96, Springer LNCS 448, Berlin - Heidelberg.

Verissimo, P., and Rodrigues, L. (1992). *A posteriori Agreement for Fault-tolerant Clock Synchronization on Broadcast Networks*, Boston, July 1992, 527–535, IEEE Computer Society Press.

Yang, Z., and Marsland, T.A. (1993). Annotated Bibliography on Global States and Times in Distributed Systems. *ACM Operating System Review* **27(3)**, 55–72.