Simulation and Implementation of Clock Synchronisation in Transputer networks

W.A. Vervoort

Department of Computer Science
University of Twente, The Netherlands.

Abstract

A Clock synchronisation algorithm based on the exchange of timestamped messages, with unpredictable delays, between neighbours in a transputer network is presented in [1]. This algorithm creates and maintains a global time reference, to be used in hard real-time applications, for all nodes of a transputer network of any topology. This global timebase is obtained by periodic filtered correction of the ***value*** *and* ***rate*** *of each local clock of a node, based on the timestamps received from its neighbours. The optimal values for the two parameters* α *and* β *of this filter were still unknown. Simulations in [2] have shown that although a* ***save*** *choice can be made in any case, an* ***optimal*** *choice depends on the topology of the network.*
The algorithm has been implemented and measurements have been performed on a transputer network ring of three nodes at the University Twente. It turned out that the optimal choice for α *and* β*, found by simulation, worked well in the implementation.*

Introduction

In the Clock synchronisation algorithm (described in the appendix) each node i of a distributed system maintains a logical clock $C_i(t)$ based on its physical clock. As physical clocks drift away from each other (typically for a transputer $|\rho| < 10^{-4}$ sec/sec) after R seconds the maximum difference between two clocks could be $2\rho R$ sec. The task of the algorithm is to correct, once in each resynchronisation period R, the value (using a factor α) and rate (using a factor β) of the logical clock, such that the clock *value* differences remain bounded by a known constant Δ (depending on the topology) and the logical clock *rates* grow to one. The algorithm should work for arbitrary network topologies and unpredictable message delays.

Estimation of α and β by simulation

Optimal values for α and β for the second order exponential smoother used in the synchronisation algorithm were found experimentally by A.D. Hoekstra in his M.Sc. thesis [2]. The following method was used. The simulation program [1] was run multiple times with the same randomly chosen set of starting values for the clocks and their drifts, while stepping through a rectangle of gridpoints in the α-β plane. To reduce simulation time this was done in a stepwise refinement fashion. At first the complete range for α and β from 0.05 to 1.0 was covered in relatively large steps (0.05), while in later runs the interval size as well as the stepsize was decreased. Earlier experiments had shown that the worst case performance is met with resynchronisation period R of 50 sec and a mean message delay of 0.5 msec; so these were used in all simulation runs.

Δ_{clock} is the mean value of the maximum difference between any two clocks of the network over the last 100 of 500 resynchronisations. It is used as a quality measure. As a rule of thumb it should not exceed 100d μsec, where d is the diameter of the topology. For each of the 20×20 gridpoints in the α,β - plane a simulation of 500 resynchronisations was carried out and the corresponding value $\Delta_{clock}(\alpha,\beta)$ was recorded. Its minimum, mean and maximum were used to find the most promising ranges of α and β for a further detailed search with smaller steps. For each (α,β) a quality factor was defined as $Q=100d/\Delta_{clock}(\alpha,\beta)$. Ideally Q is 1, which means that the requirement for that topology is met exactly for this choice of (α,β). In the α,β - plane regions can now be defined in which a least quality can be assured. The intersection -if any- of these regions for different topologies will then be a save choice, independent of the topology.

The experiments were carried out for the following topologies:

a ring of 20 nodes,
a torus of 10×10 nodes,
a double linked ring of 20 nodes,
a fully connected network of 20 nodes.

0-8186-2815-4/92 $3.00

Results of the simulation

In fig. 1a the results of the first scans of α and β from 0.05 to 1.0 by 0.05 are presented. For each topology sections of gridpoints are given where α and β assure the quality Q shown. The torus shows the largest area with Q≥1. The fully connected network (fcon) has a small area on the lower left side of the plane with Q≥1, and a large area with Q≥0.5. Both areas for the ring are much smaller and are located in the lower right corner. For the double ring (dring) no area was found with Q≥1. The best grid positions gave quality 0.3.

From fig. 1a we may conclude that as a rule of thumb, for any topology (α,β) could be chosen in the range (0.85-0.95 , 0.05-0.1). For all simulated topologies these values gave reasonable results:
ring: Q = 1.2
double ring: Q = 0.3
torus: Q = 1.0
fully connected: Q = 0.5

A quality of at least 1.0 is not possible with this choice of α and β. For resynchronisation intervals R smaller than 50 sec it is expected that this will be the case. For a 1kHz sampling real-time application an interval R=1 sec is possible while the algorithm will give still little overhead, because it will be active only once every 1000 samples.

After this overview the search for optimal values for α and β started with smaller ranges and gridpoints closer together. All were found below the line β = 0.05! In order to show a relationship with the topology, the three parameters: the diameter (d), the number of linked neighbours (l) and the number of nodes (n) are related to the optimal α and β found. The fully connected network (not possible with transputers) was added to study the relationship with l. The following figures were recorded:

ring (d=10, l=2, n=20)
Q≥1 for (α,β) = (0.9-1.0 , 0.0425-0.0475)
best (α,β): (0.952,0.0453), with Q=1.90.

double ring (d=5, l=4, n=20)
Q≥0.3 for (α,β) = (0.6-0.7 , 0.0175-0.1225)
best (α,β): (0.7,0.026), with Q=0.27.

torus (d=10, l=4, n=100)
Q≥1 for (α,β) = (0.425-0.52 , 0.0115-0.0400)
best (α,β): (0.44,0.013), with Q=4.03.

fully connected (d=1, l=20, n=20)
Q≥1 for (α,β) = (0.01-0.35 , 0.0-0.07)
best (α,β): (0.04,0.00045), with Q=3.07.

It is not yet understood why the double ring topology behaves worse than others in the simulations. It is the only topology that never meets the requirement of 100d.

For each topology fig. 1b shows the above optimal values of α and β. One should note that only gridpoints have been explored. Optimal values will for sure not be found exactly on these gridpoints.

From fig. 1b we may conclude:
- α and β dependent on the topology,
- β is always small,

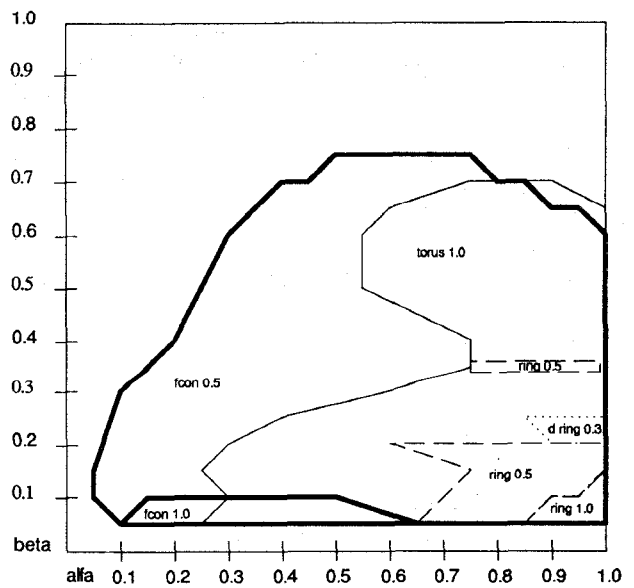


fig. 1a regions for alfa and beta for different topologies with indicated quality

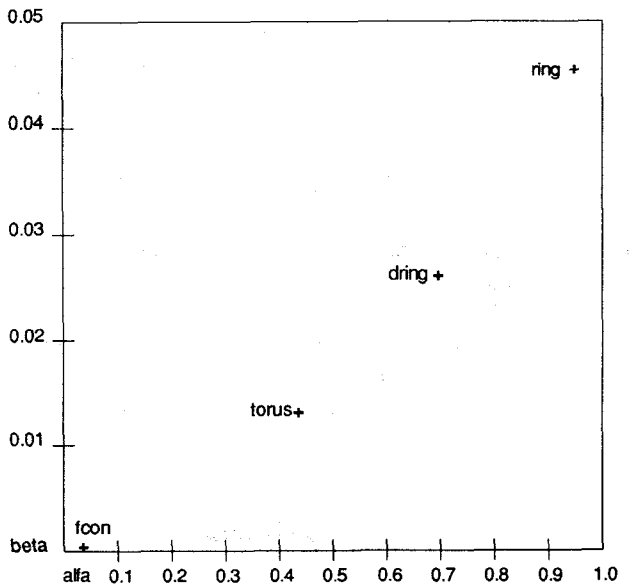


fig. 1b.. optimal values of alfa and beta for different topologies

- optimal values for topologies roughly are positioned on the parabolic line $\beta=0.055\alpha^2$ and are to be found closer to the origin the higher the *connectivity* (that is: small d and/or large l) is,
- the number of nodes does not seem to be of much importance.

Simulations have been done for smaller resynchronisation intervals R and other mean message delays too. The same optimal values for α and β were found, so these seem to depend on the topology only. Smaller values for R gave better quality factors Q.

It was found that calculating the mean value of Δ_{clock} in the last 100 of 500 resynchronisations could be a bit too early. The algorithm is still converging at that moment. Waiting a bit longer will increase the range of usable values for α and β too. Especially for small values of R waiting a bit longer will be no problem.

Implementation on a transputer network

The algorithm was implemented in Occam by A.D. Hoekstra [2] on a ring of three T800 transputers, see fig. 2. The three transputers in the ring were connected to a monitoring transputer used to measure the results of the clock synchronisation algorithm. Each transputer uses its high priority clock with a resolution of 1 μsec to maintain its logical clock. The algorithm is located in the node controller of the interface layer of the controller network, designed at the University Twente for real-time control by means of transputer networks [3]. The interface layer is located on top of the network layer, a high performance deadlock free point to point routing service, see fig 3.

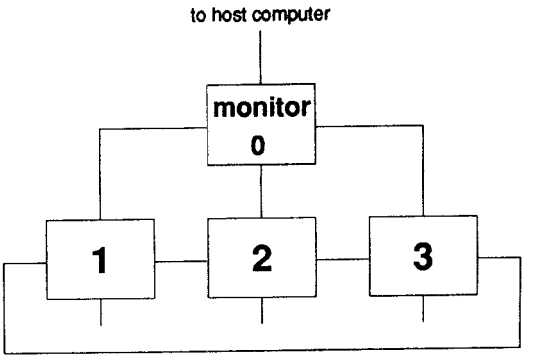


fig 2. test of clocksynchronisation on a ring of three transputers

The algorithm consists of four high priority processes. The local time is represented as a 32 bit integer, which gives the clock a turn around time of 71.6 minutes. A rate correction factor (rcf) is kept in a 32 bit real. It is used to calculate a logical clock interval lci from a hard clock interval hci by hci × rcf = lci. Its maximum difference from 1 is about 10^{-4}. If the hardware clock exhibits a drift ρ (runs with a physical rate $1+\rho$), rcf will finally grow to $(1+\rho)^{-1}$. As a result the rate of the logical clock (rcf×(1+ρ)) will grow to one. The process *split* picks a message from the network layer. If it is a time request from a neighbour node it is handed over to the *handle requests* process, which calculates and updates the logical clock and sends its new reading in a reply message to the requesting node. If *split* receives a reply from a neighbour node on a time request sent by this node it is handed over to the *get time* subprocess of the *clock sync* process that

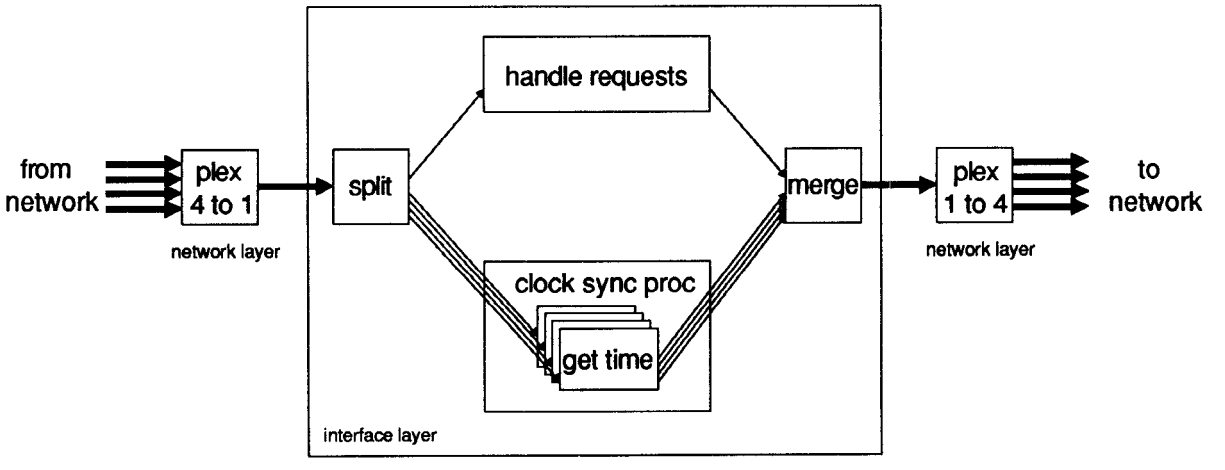


fig 3. Top level dataflow diagram of clock synchronisation program on a transputer

asked for a neighbour time. After each resynchronisation the hardware timer is set to wake up the *clock sync* process after a time interval R, when it is again time to resynchronize. It then executes the clock synchronisation algorithm by sending time requests to its neighbours by means of the subprocesses *get time*. The *merge* process collects all messages to be delivered to the network layer. Further details of the algorithm can be found in the appendix.

Special care had to be taken not to get arithmetic overflow while calculating the mean of the neighbour clock values. Another problem was encountered in calculating the mean value when some clocks had already passed their turn around point (from maximum positive to maximum negative integer value) while others had not.

On top of the interface layer application processes may run in the application layer. They may use services offered by the underlaying layers. One is a correct time service. An application process may ask the local time by means of the interface procedure *local clock*(now). The value is returned as a parameter. As a side effect the local clock is updated with a value rcf × number-of-ticks-since-last-update.

As the local clock is updated only when referenced, care should be taken not to mis an update while time has reached the turn around period of the clock representation (71.6 minutes). For safety an independent clock reading process is invoked every 10 minutes to read (and implicitly update) the local clock.

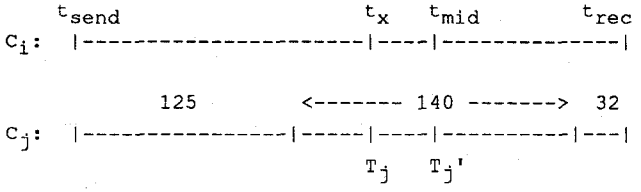
measurements on the network

Measuring clock differences in a distributed system is hard because the uncertainty of the delays of the messages containing the values to be measured are of the same order of magnitude as the clock differences. The results presented here should be interpreted with this in mind. Use of other methods like scaling the local clock with a factor 1000 in order to filter out the message delay noise [2], or the use of an oscilloscope to measure phase differences between the signals on the link [4] are possible but have not yet been applied.

The algorithm has been tested on the ring of three nodes. A simulation for such a ring showed that the same optimal α and β could be used as for the 20-node ring. As the diameter is 1 the requirement is that Δ_{clock} should not exceed 100 µsec.

The following observations were made:

- The algorithm can bring the three clocks quickly together which differ in initial value and rate. Rates are adjusted close to the mean of the initial values or follow a designated master clock.
- The rate is stabilized over time. Once it is compensated for crystal drifts it hardly changes any more. Variances were found in the order of 10^{-8} after one minute, decreasing to 10^{-12} after 30 minutes. Very slowly however the rates decrease. This may be an indication that the computed global time on every node systematically is too small compared to the value it actually is. This causes the difference between the global and local time always to be negative. This effect is probably caused by an observed asymmetry in the message delivery. It was found that sending a time request message to a neighbour node takes much more time (125 µsec) than receiving the answer (32 µsec). The uncertainty interval of 140 µsec is located between these two. So the assumption that the time T_j sent back by the neighbour generally lays in the middle of the message transport time interval (see appendix A) is wrong, its mean value T_j', corresponding with t_{mid} is 125+140/2=195 µsec after t_{send} and 32+140/2=102 µsec before t_{rec}. As a result t_{send}:t_{rec} = 2:1 is a better estimate. Tests have not yet been performed to verify this hypothesis.



- The total mean message delay appeared to be 300 µsec. This means that the assumption used in the simulations (500 µsec) is a good choice.
- Experiments were carried out with transputers on different boards on a Meiko In-Sun Computing Surface, with unrelated clock crystals. It was observed that when all transputers are on the same board the clock rate was almost constant and equal for all processors. If one of the processors is located on another board, two exhibit the same rate, while the other exhibits a relative difference which is very constant over time. This may mean that the clock synchronisation program correctly compensates the differences between clock crystals of different boards. The observed clock drift was $1.2.10^{-4}$, which conforms with the expected accuracy of the crystals.

Conclusions

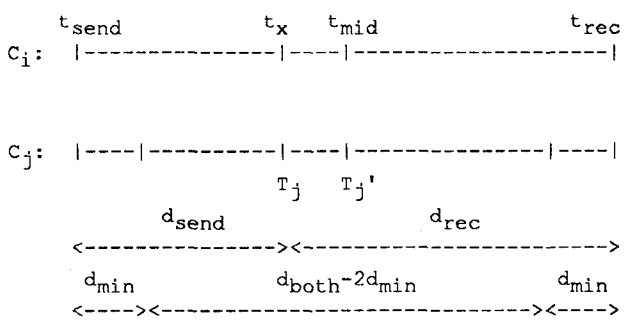
The clock synchronisation algorithm [1] seems to perform well for a real implementation on a ring of three transputers for the optimal values of α and β found by simulation of a 20-node ring. Simulations showed that α

and β depend on the configuration and on each other. Optimal values for topologies roughly are positioned on the parabolic line $\beta=0.055\alpha^2$ and are to be found closer to the origin the higher the *connectivity* (that is: small d and/or large l) is. In general α may be chosen in the range 0.85-0.95, and β should be small, in the range 0.05-0.1.

Appendix

The algorithm for node i to approximate its difference with the clock value of node j and the accuracy of that approximation is:

send(j,"give time"); $t_{send} := C_i(now)$;
receive(j,"time is T_j"); $t_{rec} := C_i(now)$;
$t_{mid} := (t_{send} + t_{rec})/2$; $d_{both} := t_{rec} - t_{send}$;
$\varepsilon_{ji}(t_{rec}) := T_j - t_{mid}$;
$\{C_{ji}(t_{rec}) = C_i(t_{rec}) + \varepsilon_{ji}(t_{rec})\}$
$weight_j$:= "value depending on d_{both}";



The following symbols are used:
i, j : nodes of the network
t, t_{send}, t_{rec}, t_{mid}, now : real-time moments
d_{min}, d_{send}, d_{rec}, d_{both} : message delays
$C_i(t)$: the clock value of node i at t
$\rho_i(t)$: drift of the clock of node i at t
$C_{ji}(t)$: i's estimation of $C_j(t)$ at t
$\varepsilon_{ji}(t) : C_{ji}(t) - C_i(t)$
$\varepsilon_i(t)$: (weighted) mean of $\varepsilon_{ji}(t)$ for $j \in$ neighbours of i

Explanation of the algorithm: Node i's message is received by j when j's clock reads T_j. As i does not know this moment t_x ($t_{send} + d_{send}$). it assumes reception at t_{mid} (which appeared to be wrong in the real case, as described above). Its estimation $C_{ji}(t_{mid})$ should have been T_j' instead of T_j. The error $T_j - T_j' = t_x - t_{mid} = (d_{send} - d_{rec})/2$, with a maximum of $(d_{both} - 2d_{min})/2$, where d_{min} is the minimum message delay. It increases with d_{both} and if confidence weight factors are to be used $\varepsilon_{ji}(t_{rec})$ should get a lower $weight_j$ the larger d_{both} is.

Adjustment of the local clock: Suppose that at t_0 the local clock of node i drifts away from the global time by $\rho_i(t_0)$ and that $\varepsilon_i(t_n)$ is the adjustment value for the next (n^{th}) resynchronisation interval ($t_n < t \leq t_{n+1}$) of length R. A part α of the adjustment $\varepsilon_i(t_n)$ is added gradually to the clock value, whereas another part β is added to the rate. Optimal values for α and β depend on the topology.
The clock value at real time t is defined as:
$C_i(t) = C_i(t_n) + (1 + \alpha.\varepsilon_i(t_n)/R + \rho_i(t_n)).(t - t_n)$
The rate $\rho_i(t_n)$ is recursively defined by:
$\rho_i(t_n) = \rho_i(t_{n-1}) + \beta.\varepsilon_i(t_n)/R$.

References

[1] Vervoort, W.A., et all, Distributed Time-Management in Transputer networks, proc Euromicro Workshop on Real-Time Systems, 1991, pp 224-230.
[2] Hoekstra, A.D., Clock Synchronisation in Transputer Networks, doctoral thesis, University of Twente, dec 1991.
[3], Wijbrans, K.C.J., et all, An Operating environment for control systems, Real-time en parallelle systemen, University of Twente, nov 1990.
[4] DeCarlini, U., et all, A simple Algorithm for Clock Synchronisation in Transputer Networks, software - Practice and Experience, apr 1988, pp. 331-347.