Embedded Information Systems

4. Measuring time, clocks, clock synchronization

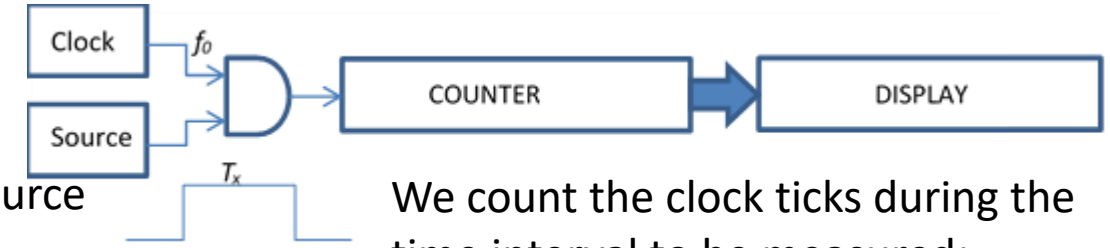
October 27, 2020

Reminder:

4. Measuring time, clocks, clock synchronization

Tools and methods:

(1) Measuring time using an electronic counter:



The gate time T_x generated by the source is the time duration to be measured.

We count the clock ticks during the time interval to be measured:

$$T_x \cong \frac{N}{f_0}$$

, where N is the content of the counter, and f_0 stands for the clock frequency. The approximate equality refers that N is always integer, while $T_x f_0$ is not necessarily. The (worst-case) relative error of the time measurement: This equation can be derived from the complete differential

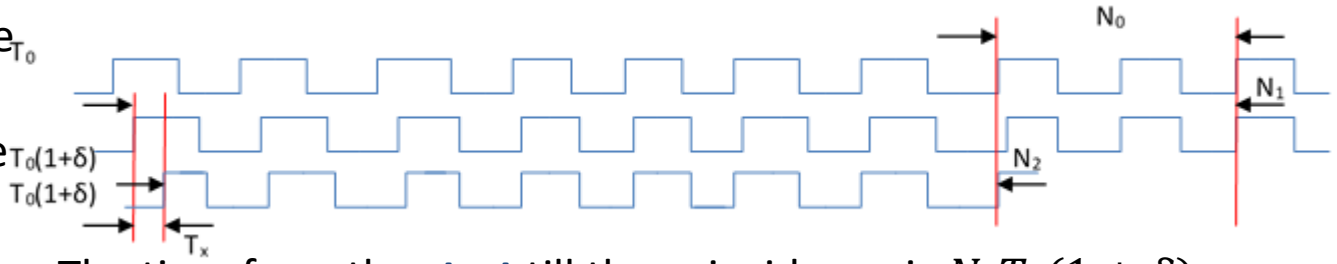
$$\left|\frac{\Delta T_x}{T_x}\right| \cong \left|\frac{1}{N}\right| + \left|\frac{\Delta f_0}{f_0}\right|$$

$dT_x = \frac{\partial T_x}{\partial N} dN + \frac{\partial T_x}{\partial f_0} df_0 = \frac{1}{f_0} dN - \frac{N}{f_0^2} df_0$, which divided by $T_x = \frac{N}{f_0}$ gives $\frac{dT_x}{T_x} = \frac{dN}{N} - \frac{df_0}{f_0}$.

Since the sign of the changes is not known, therefore we use the absolute value of the changes and express the worst case relative error.

(2) The dual vernier method:

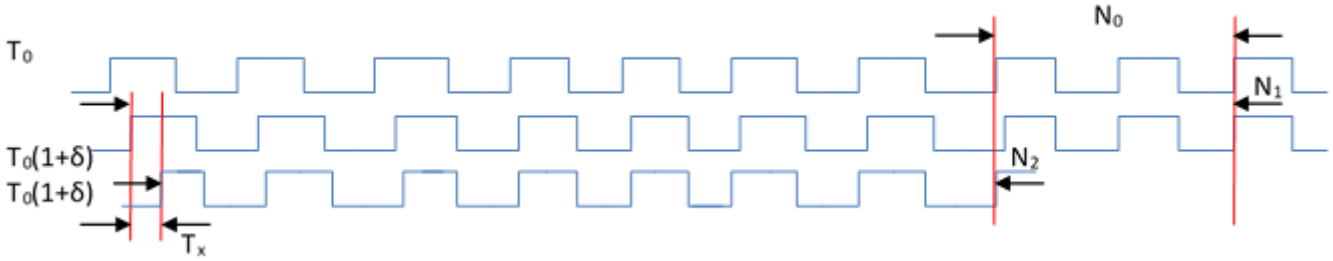
The sliding edge of the time interval to be measured and falling edge of the time interval start the **phase-startable phase-lockable oscillators** having periods of $T_0(1+\delta)$.



The time from the **start** till the coincidence is $N_1 T_0(1+\delta)$, while from the **end** till the coincidence it is $N_2 T_0(1+\delta)$.

The **time between** the two coincidences is $N_0 T_0$.

Reminder:

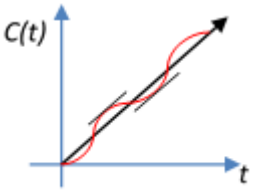


The time from the **start** till the coincidence is $N_1 T_0(1+\delta)$,
while from the **end** till the coincidence it is $N_2 T_0(1+\delta)$.

The time between the **two coincidences** is $N_0 T_0$. Thus $T_x = T_0[\pm N_0 + (N_1 - N_2)(1+\delta)]$

where the sign before N_0 is determined by the order of the two coincidences

If T_0**=5 nsec** and **δ=0.004**, then the shortest measurable duration is **20psec**.



Clocks are the sources of the knowledge of time with a given accuracy:

The source of the knowledge of time is called **clock**.

Reference clock or standard clock: if $C_k(t) = t;\ \forall t$.

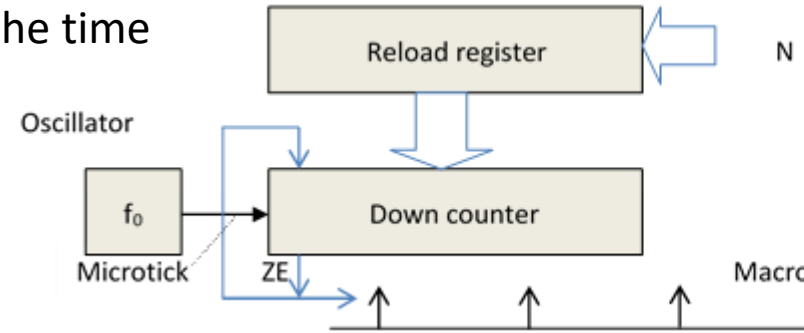
Correct clock: Clock k is correct in t_0, if $C_k(t_0) = t_0$.

Accurate clock: Clock k is accurate in t_0, if $\frac{\partial C_k(t)}{\partial t} = 1; t = t_0$.

Clock k is a function $C_k(t)$ of time, which maps real time to the time at clock *k*.

If a clock is inaccurate at some point of time, we say that the clock *drifts* at that point of time.

Physical clock: Oscillator + counter,
its granularity $g = \frac{1}{f}$ is the time between microticks.



The output of the high-frequency oscillator is divided by a down counter, which outputs an impulse at its ZE output as it reaches zero.

The physical reference clock:
denoted by $\boldsymbol{C}$, its granularity g^C. sec.
E.g.: 10^{15} macro tick/sec $\rightarrow g^C = 10^{-15}$

The frequency of this reference clock is in perfect agreement with the international standard of time.

Timestamp: $\boldsymbol{C}(e)$: is the absolute timestamp of the event e.

Clock drift: Let's measure the time between two arbitrary click of clock *k* with the reference clock and then relate the difference of the two timestamps to the true value of the time difference

$$drift_k(t_i, t_{i+1}) = \frac{C_k(t_{i+1}) - C_k(t_i)}{C(t_{i+1}) - C(t_i)}$$

Since the ideal value of the ***drift*** is 1, therefore we define the so-called: ***drift-rate:***

$\delta_k = \rho_k = |drift_k - 1|$. Typical maximum drift rates $\rho_{k,max}$ are in the range of 10^{-2} to 10^{-7} sec/sec, or better. Because every clock has a non-zero drift rate, free-running clocks, i.e., clocks that are never resynchronized, leave any bounded relative time interval after a finite time, even if they are fully synchronized at start-up.

This can result in serious consequences!

Example: During the **Gulf war** on February 25, 1991,a Patriot missile defence system failed to intercept an incoming scud rocket. The **clock drift over 100-hour** period (which resulted in **a delay of 0.3433 sec** causing a **tracking error of 678 meters**) was blamed for the Patriot missing the scud missile that hit an American military barrack in Dhahran, killing 29 and injuring 97.
The explanation of the error mentions that originally the Patriot systems were designed against muck slower devices, and they were further developed against scud systems only during the Gulf war.
The error causing this tragedy was identified already during the early days of February,

and the modified software was released on 16th of February, but unfortunately it was not forwarded to the system activated on February 25.

Offset: Consider two clocks having same granularity: $offset_{j,k}(t) = |C_j(t) - C_k(t)|$

Precision: Consider n clocks! The precision is the maximum offset between any two clocks:

$$\Pi(t) = \underbrace{max}_{\forall 1 \le j,k \le n} \left[ofszet_{j,k}(t) \right]$$

The process of mutual resynchronization of an ensemble of clocks to maintain a bounded precision is called **internal synchronization**.

Accuracy: The offset of clock k with respect to the reference clock:

$$offset_{k,ref}(t) = |C_k(t) - C_{ref}(t)|$$

The process of synchronization of a clock with the reference clock is called **external synchronization.**

Example: **Is the following statement true?**

It is!

If all clocks of an ensemble are **externally synchronized** with accuracy ***A***, then the ensemble is also **internally synchronized** with a precision of at most **2*A***.

The converse is not true!

Time measurement using more clocks. In a distributed system all nodes have their own clock.

Global time: is of the same accuracy as the reference clock, however its granularity is worse: it produces macroticks. These macroticks can be properly used, if $g > \Pi$, i.e. the synchronisation error is smaller than the resolution.

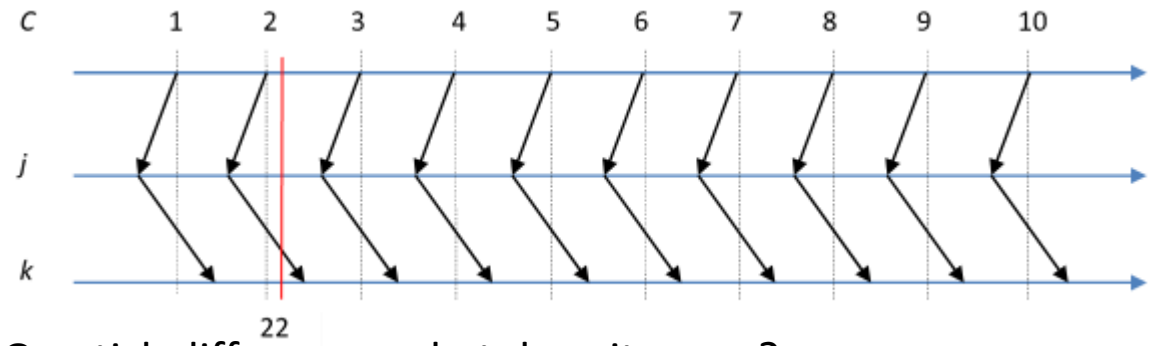
The difference of the timestamps of an event e, at nodes j and k, differ at most one tick.

$$|C_j(e) - C_k(e)| \le 1$$

This is the maximum we can achieve, since **it is always possible** that first clock *j* ticks, event *e* occurs, clock *k* ticks.

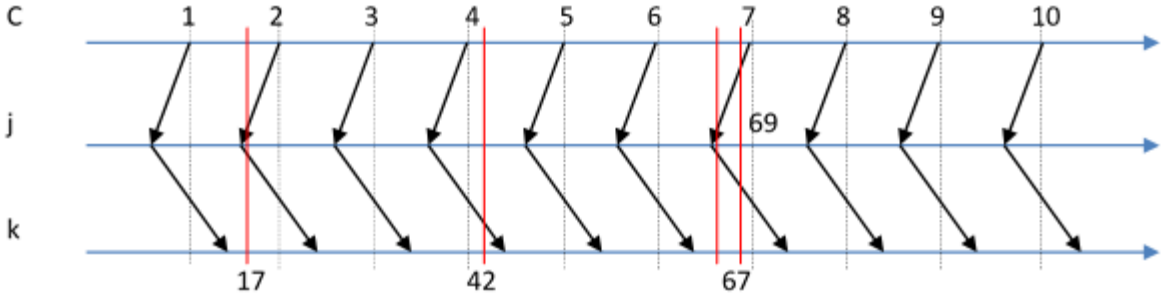
In this case the event will be timestamped with **one tick difference**.

Example: (every macrotick corresponds to 10 microticks):



At microtick e:22 clock j shows 2, while clock k shows 1.

One tick difference: what does it mean?



At microtick $e: 17$ $j: 2, k: 1$.

At microtick $e: 42$ $j: 4, k: 3$.

If the time difference of $e: 42$ and $e: 17$ is expressed by the difference of the C_k and $C_j$values, then the measurement will give **1** expressed in global time, while the real difference is **25** microticks. At microtick $e: 67$ $j: 7, k: 6$. At microtick $e: 69$ $j: 7, k: 6$.

If the time difference of events $e: 69$ and $e: 67$ are measured by the difference of $C_j$and C_k, then it gives 1, the actual difference is only 2 microticks.

Problem: The order of time is not decidable: at microtick $e: 67$ $j: 7$, at microtick $e: 69$ $k: 6$!

An interval is delimited by two events: the start event and the terminating event.

The measurement of these two events relative to each other can be affected by the synchronization error and the digitalization error.

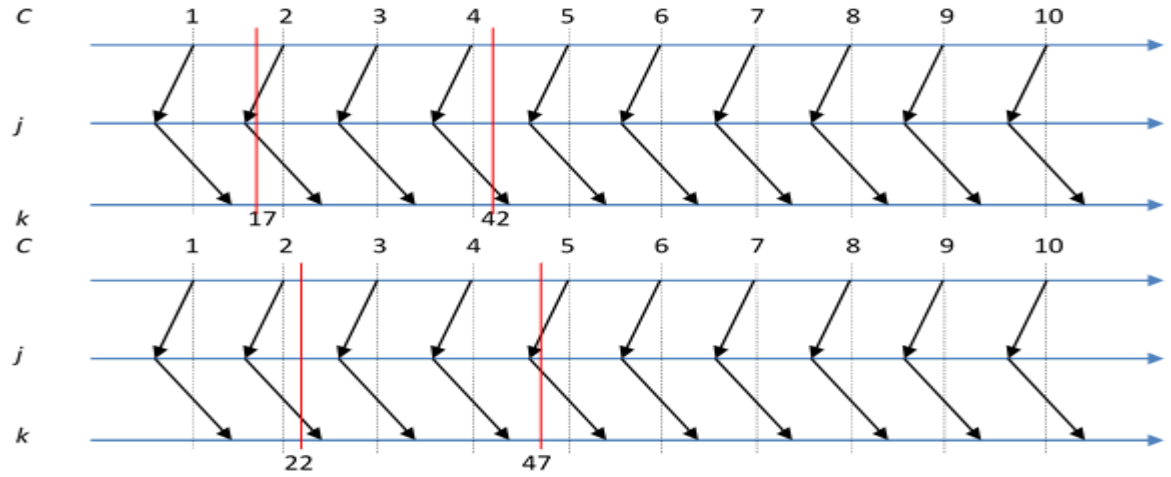
The sum of these two errors is less than $2g$ because the local implementation of the global time satisfies the condition $g > \Pi$.

Thus, if the difference is two macroticks then the order of time is decidable.

Measurement of time interval:

$$(d_{observed} - 2g) < d_{true} < (d_{observed} + 2g)$$

where d_{true} is the true duration of the interval $d_{observed}$ is the observed difference of the start event and the terminating event.



On the upper figure if we measure the difference of microticks $e: 42$ and $e: 17$ as the difference of clocks C_k and C_j then the result in macroticks is **1**, in contrast to the fact that the true difference in microticks is **25**.

On the lower figure the time difference of microticks $e: 47$ and $e: 22$ is measured by the difference of clocks $C_j$and C_k: the result in macroticks is **4** in contrast to the true difference in microticks is **25**.

Types of clock systems:

Central clock systems:

- **One accurate clock** provides the time knowledge to the whole system. The existence of other clocks in the system is „ignored” if there is no detectable failure of the central clock.
- For **fault tolerance** a standby redundancy for the central clock is used.
- The method is **accurate** (within ns to ms) and **expensive**.

- This method needs special purpose hardware integrated into the processor. The central clock sets this hardware to the proper value and any executing process can read it.
- The communication cost of this category is very low: Only one message is required for synchronizing a clock at any site. In a broadcasting environment, one message is required for synchronizing a group of sites.
- An example of this category is the **GPS** (Global Positioning System), which uses **4 broadcasting satellites** and achieves a clock synchronization with correctness within a few nanoseconds.

Centrally controlled clock systems:

In this category we distinguish between two types of nodes: **master** nodes and **slave** nodes.

- A nominated master clock polls slave clocks.
- Clock differences are measured, and the master dictates corrections to the slaves.
- If the master clock fails, an election of a new master (from the slaves) is initiated.
- Transmission times and delays are estimated, since they affect significantly the clock difference measured.
- Communication costs are higher than in the previous case.

Distributed clock systems:

- All the nodes are homogeneous, and each runs the same algorithm.
- Each node updates its own clock after receiving the time from other clocks and after estimating their correctness.
- The fault tolerance is protocol based. If a node fails, the other nodes are not affected; they only detect the failure and ignore the failed node thereafter.
- Generally, relatively heavy communication traffic is involved in this category, especially when robustness in the presence of malicious faults is required.

Time standards: Two are used in distributed real-time systems:

International Atomic Time (Temps Atomique Internationale, TAI):

TAI defines the second as the duration of 9 192 631 770 periods of the radiation of a specified transition of the **Cesium atom 133**. The intention was to define the duration of the **TAI** second so that it agrees with the second derived from astronomical observations.

Universal Time Coordinated (UTC):

UTC is a time standard that has been derived from astronomical observations of the rotation of the earth relative to the sun.

- The **UTC** time standard was introduced in 1972, replacing the Greenwich Mean Time (**GMT**) as an international time standard. The duration of the second conforms to the **TAI**.
- Because the rotation of the earth is not smooth, but slightly irregular, occasionally a leap second is inserted into the **UTC** to maintain synchrony between the **UTC** and astronomical phenomena, like day and night. Because of this leap second, the **UTC** is not a chronoscopic time scale, i.e., it is not free of discontinuities.
- It was agreed that on January 1, 1958 at midnight, both the **UTC** and the **TAI** had the same value. Since then the **UTC** has deviated from **TAI** about 40 seconds.
- The **US National Institute of Standards and Technology**: **NIST** provides a short-wave radio broadcast of continuous frequency and time signal at frequencies **2.5**, **5**., **10**, **15** and **20** MHz. The accuracy of these time signals is $\pm 1msec$, but due to random atmospheric disturbances at the receiver this accuracy will be about $\pm 10msec$. (In case of geostationary satellites: $\pm 0.5msec$.)

Time Format: Network Time Protocol (**NTP**).

This time format with a length of 8 bytes contains 2 fields: a 4-byte full second field, where the seconds are represented according to **UTC**, and a fraction of a second field, where the fraction of a second is represented as a binary fraction with a resolution of about 232 ps.
On January 1, 1972, at midnight the **NTP** clock was set to 2 272 060 800.0 seconds, i.e., the number of seconds since January 1, 1900 at 00:00h.

This ranges up to the year 2036, i.e., it has 136 years wrap around cycle.

Example: The significance of clock synchronisation:

UNIX **make** program

The sources of large programs are divided into smaller parts (e.g. 100 files).
Only those parts are to recompiled the source of which were modified.

If the timestamp of the source is later than that of the object file, e.g. **input.c** (timestamp **2151**), and **input.o** (timestamp **2150**), then the source file should be recompiled.

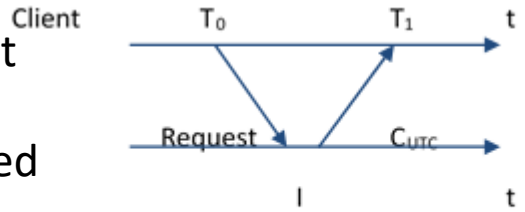
But if the editor and the compiler run on different computers, then if the clocks are not synchronised, the interpretation of the timestamps may fail!
If we realize that the timestamp of the source is earlier than that of the compiled one, i.e. **output.c** (timestamp **2143**), and **output.o** (timestamp **2144**), then we do not recompile the source.

But if this is due to the fact that the clock of the computer running the editor delays 2 time units, then this can cause serious problems!

Clock synchronisation:

Berkeley algorithm: Active time server: regularly requests the clock values of the nodes, computes their average and resends them.

Cristian algorithm: Synchronization is initiated by the client at time T_0 by requesting a server, which has a **UTC** receiver. After the arrival of the request an interrupt routine is executed and the **UTC** radio is requested. Finally, the value of the **UTC** clock is sent to the client. The message arrives at T_1.



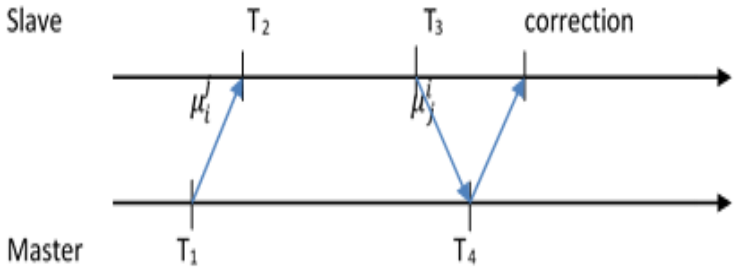
Server having UTC receiver

The received clock value must be corrected by the time needed for communication. If the communication requires nearly the same amount of time in both directions, then a good approximation of this correction is: $\sim \frac{T_1 - T_0 - I}{2}$.

Comment:

It might cause problems if $C_{UTC} + correction < T_1$, i.e., the clock of the client is to be reset to an earlier time. If the client clock is just timestamping subsequent events, it might happen that due to the reset a later event receives earlier time, i.e., seemingly changes the order. If such a situation is a real danger, then it is not allowed to reset the clock, only the slowing of the clock is permitted until its run will be synchronous with the **UTC** clock.

Master-slave algorithms:



1. The master clock i initiates synchronization at T_1.
   The error of the timestamp is e_1. $(T_1 = C_i(T_1) + e_1)$.
   The slave is node j.
   The message sent in T_1 will travel for μ_i^j time before it arrives to j at time T_2. The timestamp is $C_j(T_2)$, and $T_2 = C_j(T_2) + e_2$.

2. The slave computes the difference: $d_1 = C_j(T_2) - C_i(T_1)$

   Comparing the times of sending and receiving the message:

$$C_i(T_1) + e_1 + \mu_i^j = C_j(T_2) + e_2 \rightarrow d_1 = C_j(T_2) - C_i(T_1) = \mu_i^j + (e_1 - e_2)$$

   If ε_j denotes the mean of the difference of the slave clock and the master clock, then

$$\varepsilon_j = (e_1 - e_2) + E_j^1,$$

   where E_j^1 represents noise. ($(e_1 - e_2)$ is the difference of the actual values.)

   Replacing this difference by the mean value + noise:

$$d_1 = C_j(T_2) - C_i(T_1) = \mu_i^j + (e_1 - e_2) = \mu_i^j + \varepsilon_j - E_j^1$$

3. The slave sends its clock value at $T_3 = C_j(T_3) + e_3$ to the master. This message arrives at $T_4 = C_i(T_4) + e_4$ following a travel of μ_j^i duration. The master computes the

   $d_2 = C_i(T_4) - C_j(T_3)$ difference. Again, comparing the actual times of the events:

$$C_j(T_3) + e_3 + \mu_j^i = C_i(T_4) + e_4 \rightarrow d_2 = C_i(T_4) - C_j(T_3) = \mu_j^i + (e_3 - e_4).$$

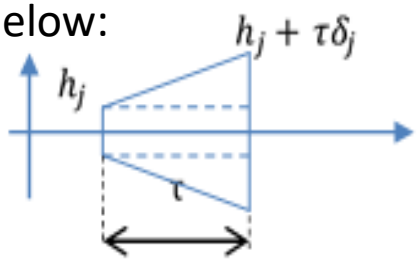
The mean difference of the slave and the master clocks ε_j can be expressed by $-\varepsilon_j = (e_3 - e_4) + E_j^2$, where E_j^2 represents noise. Thus $d_2 = \mu_j^i - \varepsilon_j - E_j^2$.

The half of the difference of $d_1 - d_2$ can be used to give the necessary correction value at the slave node:

$$(d_1 - d_2)/2 = \varepsilon_j + (\mu_i^j - \mu_j^i)/2 - (E_j^1 - E_j^2)/2 = \varepsilon_j + h_j,$$

where h_j stands for errors coming from the difference of the communication times and the quantization effects. The random components can be reduced by averaging.

After the synchronization the remaining error will increase due to the drift. See the figure below:



The figure illustrates that due to the remaining error and the drift, after a time duration of τ, the worst-case time distance of the clocks will be:

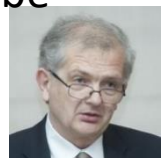
$$2\left(\tau \underbrace{max}_{j} \delta_j + \underbrace{max}_{j} |h_j|\right)$$

Comments:

(1) The accuracy of the correction has improved after **repeating** the measurement several times and **averaging** the results.

(2) If the details of the communication between master and slave are known (e.g. in LAN environment), the correction can be **further refined**.

(3) If the requirement is the synchronization of *n* processors, and every slave is requested *p* times, then the communication demand of the master-slave synchronization can be characterized by $(2p + 1)n$ in every τ period.



Example: Tempo algorithms: master-slave synchronization in the distributed Berkeley Unix.

Master side:

The basic algorithm is repeated N times:

for *k*=1 to *N*

do

Initialization:

do

$$T_A \leftarrow C_i(now)$$

$$\forall j \neq i \text{ Send } T_A \text{ to } j$$

endo

Processing data received from the slaves:

$\forall j \neq i$:

do

$$d_2^j \leftarrow C_i(now) - T_B$$

$$\Delta_j(k) \leftarrow \left(d_1^j - d_2^j\right)/2$$

endo

endo ; *N differences are available for* $\forall j$*:*

$\forall j \neq i$:

do

$$\Delta_j = (1/N)\sum_{k=1}^{N} \Delta_j\,(k)$$

$$\text{Send}\left(\Delta_j\right) \text{ to j}$$

endo

Slave side:

do

$$d_1^j \leftarrow C_j(now) - T_A$$

$$T_B \leftarrow C_j(now)$$

Send (T_B, d_1^j) to *i*

endo

do

$$C_j(t) \leftarrow C_j(t) - \Delta_j$$

endo

Distributed clock synchronization algorithms

The major advantage of the distributed approach is the higher degree of fault tolerance it achieves. This achievement increases the cost mainly in communication rather than in special hardware. The load imposed on the communication network by the distributed approach is therefore expected to be higher than that of the MS approach.

In the distributed clock systems, all the time servers use uniform approach with the following characteristics:

- Each node polls the rest of the clocks or a subset of them.
- Each applies a specific algorithm to the responses of the poll.
- Each node updates the local clock accordingly.

I. A Fundamental Ordering Approach

Let us consider an ordering approach based on message timestamping with the following properties:

- The accuracy of clock *i* is bounded by a drift rate δ: $\forall t$: $\left|1 - \frac{d}{dt} C_i(t)\right| < \delta_i \ll 1$.
- The communication graph of the algorithm is closely connected (every vertex/node can send synchronization messages to the rest of the vertices/nodes) with a diameter *d* (minimum number of hopes).
- The network imposes an unpredictable (yet bounded) message delay *D*. In other words, $\mu < D < \eta$ holds, where μ and η are the lower and upper bounds on *D*.

Each clock implements the following algorithm:

- On every local clock event occurrence, increment the local clock $C_i(t) \leftarrow C_i(t) + 1$.

- Each node with a clock sends messages to the others at least every τ seconds. Each message includes its timestamp T_m.
- Upon reception of an external T_m, the receiver sets its clock $C_i(t) \leftarrow max(C_i(t), T_m + \mu)$.

The communication cost of one update of the whole network is $n(n-1)$ messages.

The correctness of each clock due to this synchronization algorithm is $\forall i$: $\forall j$: $|C_i(t) - C_j(t)| < d(2\delta\tau + \eta)$ for all t.

This algorithm achieves only the ordering goal, bounding clock differences between sites. The algorithm results in updates according to the fastest clock in the system, and not necessarily the most accurate one.

II. Minimize Maximum Error

Every clock i "knows" it is correct within the interval: $[C_i(t) - E_i(t), C_i(t) + E_i(t)]$ where $E_i(t)$ is a bound on the error of clock i.

The error interval is constructed from the following contributors:

- The error that comes into effect right on the clock reset time (ρ_i), as discretization and other constant errors (ε_i).
- The delay from the time this clock i is read until another clock j uses the readout for its update (μ_i^j).
- The degradation of time-counting that develops between consecutive resets (δ_i).

The algorithm consists of two rules: a **response rule** and a **synchronizer rule**.

A request is transmitted by the **synchronizer rule** at node j activates i's response.

In this response, node i first updates its bound on the error, $E_i(t)$. It then replies its clock value $C_i(t)$ and the above bound on that clock's error. This message expresses a time interval within which the clock is correct.

The **synchronizer rule** is periodic, performed at least every τ time units. Its first step is a request for responses which it sends to the rest of the nodes. Then, for each of these nodes, it performs a response reception and a conditional clock reset.

Two conditions must hold for a reset:

1. The interval $[C_i(t) - E_i(t), C_i(t) + E_i(t)]$ that expresses the local knowledge of time must be **consistent** with the incoming interval $\left[C_j(t) - E_j(t), C_j(t) + E_j(t)\right]$.
   The consistency requires a **nonempty intersection** of these two intervals.
2. The error of the response, $E_j(t)$, plus the error of the response delay, $(1 + \delta_i)\mu_j^i$ generate an error smaller than the local one.

If these two conditions hold, the node can reset its clock and enhance its knowledge of time.

The reset involves **three parameters**.The local clock $C_i(t)$ is set to the value of the response clock. The error at local clock reset, ε_i, is set to the value of the response error and the delay combined. The time-of-reset record, ρ_i, is also set to the value of the response clock.

If on the other hand one of these conditions does not hold, the algorithm ignores the response.

The algorithm's two rules are given below:

Minimize Maximum Error:

Upon receiving a time Request from $j \neq i$:

do

$E_i(t) \leftarrow \varepsilon_i + (C_i(t) - \rho_i)\delta_i$

Send $(C_i(t), E_i(t))$ to j.

endo

Rule#1 (from the viewpoint of clock i).

At least once every τ time units:

$\forall j \neq i$: Request$\big(C_j(t), E_j(t)\big)$;

for $j \neq i$ do begin

Receive$\big(C_j(t), E_j(t)\big)$;

if $\big(C_j(t), E_j(t)\big)$ is consistent with $(C_i(t), E_i(t))$

then if $E_j(t) + (1 + \delta_i)\mu_j^i \leq E_i(t)$

then begin

$C_i(t) \leftarrow C_j(t)$

$\varepsilon_i \leftarrow E_j(t) + (1 + \delta_i)\mu_j^i$

$\rho_i \leftarrow C_j(t)$

end

else ignore it

end

endo

Rule#2

III. Intersection of time intervals

This algorithm also consists of two rules: a response rule and a synchronizer rule.
The response rule is identical to the response of the previous algorithm.
The synchronizer rule here is also periodic, performed at least every τ time units.
The first step of the synchronizer rule is a request for responses, which the algorithm sends to the rest of the nodes. The similarity to the previous algorithm ends here.
The second step of this algorithm is to receive all the responses.

Each response interval has a **left boundary** $L_j(t)$ and a right boundary $R_j(t)$ and the algorithm calculates both of them.

Then the algorithm selects the highest left boundary in the responses, α, and the lowest **right boundary**, β. If the responses are **consistent**, there must be a nonempty intersection of them all, and thus, $\alpha < \beta$. Otherwise the responses are considered inconsistent and therefore ignored.
If they are **consistent**, we can conclude that the real-time clock is within the interval $[\alpha, \beta]$.
Therefore, the algorithm sets its error to equal half this interval and the local clock to equal the interval's midpoint.

The first rule is exactly the same as previously.
The second rule:

At least once every τ time units:

$\forall j \neq i$: Request$\big(C_j(t), E_j(t)\big)$;
$\forall j \neq i$: Receive$\big(C_j(t), E_j(t)\big)$;
$\forall j \neq i\ L_j(t) \leftarrow \big(C_j(t) - E_j(t)\big)$; the left boundary
$\forall j \neq i$: $R_j(t) \leftarrow \big(C_j(t) + E_j(t)\big) + (1 + \delta_i)\mu_j^i$
the right boundary
$\alpha \leftarrow max\big(L_j\big); \beta \leftarrow min\big(R_j\big)$
if $\alpha < \beta$
then
$\varepsilon_i \leftarrow \frac{1}{2}(\beta - \alpha)$;
$C_i(t) \leftarrow \frac{1}{2}(\alpha + \beta)$;
$\rho_i \leftarrow \frac{1}{2}(\alpha + \beta)$
end
else ignore them all
endo

Comments:

(1) The intersection algorithm is superior in its accuracy, however, less robust. It may ignore the responses of all the participants because of an erroneous response from one participant.

(2) The communication demand of the distributed algorithm in case of *n* clocks is $2n(n-1)$ in every τ time units.

(3) The distributed algorithms require good knowledge of time.

Jitter of the synchronization message: *Jitter*: $e = d_{max} - d_{min}$

- at the application software level: $500\mu s$ …5ms
- at the kernel of the operating system: $10\mu s$ …$100\mu s$
- at hardware of the communication controller: <$10\mu s$.

The important role of the latency jitter *e* for internal synchronization is emphasized by an impossibility result: It is not possible to internally synchronize the clocks of an ensemble consisting of *N* nodes to a better precision than $\Pi = e\left(1 - \frac{1}{N}\right)$.

IV. Fault-Tolerant-Average (FTA) algorithm:

In a system with *N* nodes *k* Byzantine faults should be tolerated.

The FTA algorithm is a one-round algorithm that works with inconsistent information, and bounds the error introduced by inconsistency. At every node, the *N* measured time differences between the node's clock and the clocks of all other nodes are collected (the node considers itself a member of the ensemble with time difference zero).

These time differences are sorted by size.

Then the k largest and the k smallest differences are removed (if the erroneous time value is either larger or smaller than the rest).
The remaining $N - 2k$ time differences are, by definition, within the precision window.

The average of these remaining time differences is the correction term for the node's clock.