## 3.1 Clock Synchronisation

This section first presents some commonly-used definitions and notations in the field of the clock synchronisation problem, and then gives an overview of the known solutions that fits to the GUARDS constraints. At last, the GUARDS synchronisation algorithm is detailed, including the initial synchronisation.

### 3.1.1 Definitions and Notations

We consider a set of fully connected nodes. Each node has a physical clock, and computes a logical clock time.

***Definition 1: Clock time***

Each node maintains a logical clock time T = C(t), meaning that at real-time t the clock time of the local node is T.

By convention, variables associated with clock time (local to a given node) are in uppercase, whereas variables associated with real time (measured in an assumed global Newtonian frame) are in lowercase.

***Definition 2 : drift rate***

A nonfaulty physical clock C has a maximum drift rate ρ if and only if for any $t_2 > t_1$:

$$1 - \rho < \frac{C(t_2) - C(t_1)}{t_2 - t_1} < 1 + \rho$$

***Definition 3 : agreement***

The agreement condition is satisfied if and only if the skew δ between any nonfaulty logical clocks $C_i$ and $C_j$ is bounded:

$$\left| C_i(t) - C_j(t) \right| \leq \delta$$

***Definition 4 : accuracy***

The accuracy condition is satisfied if and only if any nonfaulty logical clock $C_i$ stay within a linear envelop of real time. That is, there exists a constant $\gamma > 0$ and two constants a and b (that depend on initial conditions) such that:

$$(1 - \gamma)t + a \leq C_i(t) \leq (1 + \gamma)t + b$$

***Definition 5 : synchronisation algorithm***

A synchronisation algorithm is an algorithm which satisfies the agreement and accuracy conditions.

Some typical values of the parameters introduced so far or used in the sequel of the section are given in $$$Table 1.

| Variable | Content | Typical value |
|---|---|---|
| ρ | Maximum drift rate of all physical clocks | $10^{-4}$ |
| f | Quartz frequency of physical clocks | 10 MHz |
| δ | Maximum skew between any two nonfaulty logical clocks | less than 100 μs |

| Variable | Content | Typical value |
|---|---|---|
| d | Upper bound on the transmission delay | 50 μs |
| ε | Upper bound on the read error[2] | 1 μs |
| n | Number of clocks | at most 4 |
| m | Maximum number of (arbitrarily) faulty clocks | 1 |
| R | Resynchronisation interval | 500 ms |

**Table 1 : Typical values**

Note that the maximum skew between any two logical clocks is necessarily greater than the skew between their physical clocks, and that this latter skew can easily be obtained through the simple formula $\delta_{physical} = 2\rho R$, where R is the interval between two resynchronisations. With the values given in $$$Table 1, this gives $\delta_{physical} = 2*10^{-4}*500$ ms = 100 μs. Hence the fact that the typical value of δ in $$$Table 1 is greater than 100 μs (see section $$$3.1.1.3 for a more precise estimation of δ).

### 3.1.2 Existing Algorithms

A lot of clock synchronisation algorithms are described in the literature. [Ramanathan et al. 1990] provides a good survey of these algorithms. Restricting only to software implemented algorithms[3] yields three main kinds of algorithms (see $$$Figure 1):

---

[2] The read error is mainly due to the transmission delay incertitude. Thus, d-2ε is the minimum message transit delay between any two nodes in the system.

3 In order to minimize the amount of specific hardware, in conformance to the GUARDS requirement to use as much COTS components as possible.

- Convergence algorithms, where nodes periodically resynchronise by exchanging synchronisation messages.
- Consistency algorithms, where nodes use an interactive consistency algorithm to achieve agreement on clock values. The little amount of literature on this kind of algorithm [Lamport & Melliar-Smith 1985] does not allow us to use them within GUARDS without spending significant efforts on carefully studying them.
- Probabilistic algorithms [Cristian 1989], where nodes can make the worst-case skew as small as desired, but with an increasing probability of loss of synchronisation. This is clearly unacceptable in the field of safety-critical applications.

Focusing on convergence algorithms, we see that there exists two main kinds of such algorithms:

- Convergence-averaging algorithms, where each node resynchronises according to clock values obtained through periodic one-round clock exchanges. On each node, the other clocks can be taken into account through a mean-like function [Lamport & Melliar-Smith 1985], or a median-like function [Lundelius-Welch & Lynch 1988].
- Convergence-nonaveraging algorithms, where each node periodically seeks to be the system synchroniser. To deal with possible byzantine behaviors, the exchanged messages can be authenticated or broadcast [Srikanth & Toueg 1987; Dolev et al. 1995].

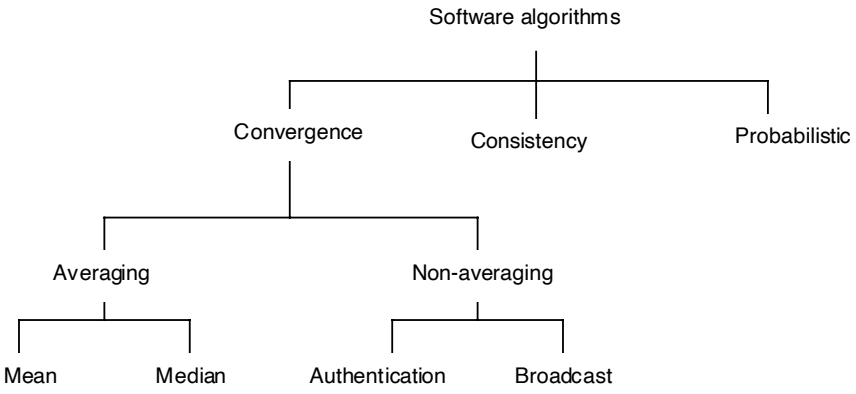


Figure 1 : Synchronisation algorithms

### 3.1.2.1 Convergence-averaging Algorithms

These synchronisation algorithms can tolerate m arbitrary faulty nodes in a (fully connected) network of n nodes, under the sufficient condition[4] that $n \geq 3m+1$. This implies in particular that they *cannot* tolerate a byzantine faulty clock in a three node system (see $$$Figure 2).

The basic idea of these algorithms is as follows. The resynchronisation is performed periodically: each node broadcasts a resynchronisation message when its local clock time has counted R seconds since the last resynchronisation period. At the same time, the node collects during a given waiting period (which is far shorter than R) the resynchronisation messages broadcast by other nodes: it records the arrival time (according to its local clock time) of each received resynchronisation message. After the waiting period, it uses a fault-tolerant averaging function on the arrival times to compute an averaged clock time. It then

4 This condition is also necessary if authentication is not used.

corrects its own clock time by setting it to the averaged clock time before the next resynchronisation period.

In [Lamport & Melliar-Smith 1985], the fault-tolerant averaging function F is the mean of all values (with values differing from the local value by more than δ replaced by the local value).

In [Lundelius-Welch & Lynch 1988], the function F is a median-like function. It consists in rejecting the m highest and m lowest values, and then performing the average of all remaining values.

With three clocks, the above fault synchronisation function F consists simply in taking the median value. So a byzantine faulty clock can send synchronisation messages to two other nodes in such a way that they find themselves as being always the median: the byzantine faulty clocks has only to send synchronisation messages so that the earliest (respectively latest) nonfaulty clock receive the Byzantine synchronisation message before (respectively after) its own synchronisation emission. Thus each nonfaulty node keeps its own value and the two nonfaulty clocks drifts apart progressively.

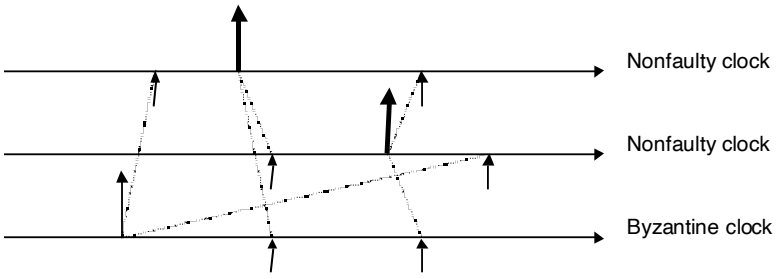

Figure 2 : Desynchronisation of two nonfaulty clocks in presence of a byzantine clock

#### 3.1.2.2 Convergence-nonaveraging Algorithms

When authentication is used for inter-node message exchanges, these synchronisation algorithms can tolerate m arbitrary faulty node with only $n \geq 2m+1$ nodes. This implies in particular that they *can* tolerate a byzantine faulty clock in a three node system.

In these algorithms [Srikanth & Toueg 1987; Dolev et al. 1995], each node seeks to be the system synchroniser, and succeeds if it is the fastest nonfaulty node. More precisely, each node broadcasts its resynchronisation message and collects during a given waiting period the resynchronisation messages broadcast by other nodes (like in convergence-averaging algorithms). As soon as it has received m+1 resynchronisation messages (including possibly its own), it restarts its local clock for the next period and relays the m+1 resynchronisation messages to all other nodes.

It should be noted that the worst case skew of all such algorithms proposed so far is greater than the transmission delay: $\delta \geq d$.

To better understand the mechanisms involved here, we will detail the algorithm proposed in [Srikanth & Toueg 1987]. In this algorithm, each node starts a new local clock $C^k(t)$ at each successive resynchronisation round. The core of the algorithm can be described by the following pseudo-code (the constant a being chosen so that clock-time does not go backward when resynchronisation occurs, i.e. $C^k(t) > C^{k-1}(t)$):

```
If C^(k-1)(t) = kR
  Then    sign and broadcast (round k)
//
If (m+1) signed (round k) messages received
  Then    C^k(t) = kR + a
          Relay all (m+1) messages to all
```

When a node broadcasts its synchronisation message, it is assumed that it receives it immediately (because for the local node, there is no transmission delay for the reception of its own synchronisation message). So, the test of reception of (m+1) signed messages includes the possible locally emitted synchronisation message.

$$$Figure 3 gives an example of tolerance of a byzantine faulty clock in a three node system (m is equal to 1). The faulty clock (clock number 3 on the figure) broadcasts its (signed) resynchronisation message[5] so that the fastest nonfaulty clock (clock number 1 on the figure) receives this message before the end of its own resynchronisation interval and the slowest nonfaulty clock (clock number 2 on the figure) receives it after the end of its own resynchronisation interval. We have seen that such an inconsistent broadcast can desynchronise the two nonfaulty clocks under a convergence-averaging synchronisation algorithm. Let us analyze what happens here:

- The fastest nonfaulty clock receives the resynchronisation message of clock 3. A short time after, this non-faulty clock reaches the end of its own resynchronisation interval. According to the algorithm, it has received at this time m+1 resynchronisation messages (the one coming from clock 3 and its own). So it starts its new local clock (horizontal bold arrow on the figure) and relays to the two other nodes two (signed) resynchronisation messages.

- The slowest nonfaulty clock receives the two signed resynchronisation messages (just emitted by the clock 1), and thus according to the algorithm restarts its new local clock (horizontal bold arrow on the figure) and relays to the two other nodes these two (signed) resynchronisation messages. A short time after, this clock 2 reaches the end of its own resynchronisation interval and then broadcast its own resynchronisation message. A short time after, this clock 2 receives the

5 The signature of a message m by a node i is denoted by m:i on the figure.

resynchronisation message from clock 3 which it may identify as being faulty.

We can see that the clock 2 starts its new local clock at most d seconds (maximum transmission time) after clock 1 has started its own.

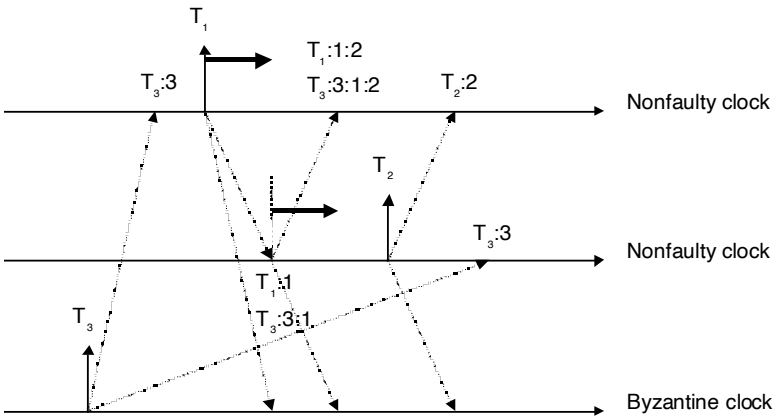


Figure 3 : Tolerance of a byzantine clock in a 3 node system

### 3.1.2.3 Maximum Skew Estimation

One of the most important characteristics of a synchronisation algorithm is its ability to achieve a small maximum skew between non-faulty clocks. Therefore, it is important that we estimate and compare the maximum skew of [Lundelius-Welch & Lynch 1988] (hereafter denoted by LL) and [Srikanth & Toueg 1987] (hereafter denoted by ST).

Theoretical results provide us with the two following formulas:

- for the LL convergence-averaging algorithm (theorem 16, page 21)[6]: $\delta = \beta + \varepsilon + \rho(7\beta+3d+4\varepsilon) + 4\rho^2(2+\rho)(\beta+d)$, with $\beta = 4(\varepsilon+\rho R)$
- for the ST convergence-nonaveraging algorithm (lemmas 6 and 8, page 630-631): $\delta = [R(1+\rho) + d][\rho(2+\rho)/(1+\rho)] + d(1+\rho)$

Using the typical values given in $$$Table 1, we have $\varepsilon = 10^{-6}$ s, $\rho R = 5.10^{-5}$ s, $d = 5.10^{-5}$ s, and we can see that terms including factors like $\rho^2$, $\rho\varepsilon$, $\rho d$ are negligible. By dropping these higher order terms, we obtain:

- for the LL algorithm: $\delta \sim 5\varepsilon + 4\rho R$
- for the ST algorithm: $\delta \sim d + 2\rho R$

We can see that the maximum skew $\delta$ is of the same order of magnitude in the two algorithms (a few hundreds of microseconds). However, several remarks must be made:

- We have taken a rather pessimistic maximum drift rate of $10^{-4}$ for physical clocks. For example, $\rho$ is estimated at $10^{-6}$ by [Ramanathan et al. 1990], leading to $\rho R = 5.10^{-7}$ s. If we take such a smaller value for $\rho$, then LL becomes better than ST.
- We have taken a rather optimistic maximum transmission delay d, taking into account that GUARDS channels are expected to be near each other (typically a few decimeters). If channels are distributed over longer distances, then this could lead to an increase of d, thus making again LL better than ST.

---

6 The factorisation of this fomula follows actually those given in [Ramanathan et al. 1990]. But note that the exact reformulation of [Ramanathan et al. 1990] is incorrect (omission of the scalar 3 in front of d in the third factor).

- From a practical viewpoint, it is not clear whether the transmission delay d is the same for LL and for ST. Indeed, LL is a single-round algorithm where synchronisation messages are deterministically and synchronously broadcast. Thus d corresponds to the maximum time required for a message to be prepared by a node, broadcast to all other nodes and processed by these other nodes. For ST the issue is that a node may have to broadcast its synchronisation message (first round), and then to relay immediately after the synchronisation messages it received (second round), precisely because it received the $(m+1)^{th}$ signed synchronisation message just after having sent its own synchronisation message. The maximum transmission delay has to take into account this unfavorable case (which is likely to happen precisely when clocks are well synchronised), thus leading to a larger value.