Chapter 3
Global Time

Overview This chapter starts with a general discussion on time and order. The notions of causal order, temporal order, and delivery order and their interrelationships are elaborated. The parameters that characterize the behavior and the quality of a digital clock are investigated. Section 3.2 proceeds along the positivist tradition by introducing an omniscient external observer with an absolute reference clock that can generate precise time-stamps for all relevant events. These absolute time-stamps are used to reason about the precision and accuracy of a global time base, and to expose the fundamental limits of time measurement in a distributed real-time system.

In Sect. 3.3, the model of a *sparse time base* is introduced to establish a consistent view of the order of computer-generated events in a distributed real-time system without having to execute an agreement protocol. The *cyclic model of time* presented in this section is well suited to deal with the progression of time in cyclic systems, such as in many control and multimedia systems.

The topic of internal clock synchronization is covered in Sect. 3.4. First, the notions of convergence function and drift offset are introduced to express the synchronization condition that must be satisfied by any synchronization algorithm. Then, the simple central-master algorithm for clock synchronization is presented, and the precision of this algorithm is analyzed. Section 3.4.3 deals with the more complex issue of fault-tolerant distributed clock synchronization. The jitter of the communication system is a major limiting factor that determines the precision of the global time base.

The topic of external synchronization is studied in Sect. 3.5. The role of a time gateway and the problem of faults in external synchronization are discussed. Finally, the network time protocol (NTP) of the Internet, the time format of the IEEE 1588 clock synchronization protocol, and the time format of the TTA are presented.

H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Real-Time Systems Series, DOI 10.1007/978-1-4419-8237-7_3,

3.1 Time and Order

Applying the principles of *utility* and *parsimony* (Sect. 2.2.1), we base our model of time on Newtonian physics, because the models of *Newtonian physics* are simpler than the models of *relativistic physics* and sufficient to deal with most temporal phenomena in embedded systems. In many engineering disciplines (e.g., Newtonian mechanics), time is introduced as an independent variable that determines the sequence of states of a system. The basic constants of physics are defined in relation to the standard of time, the physical second. This is why the global time base in a cyber-physical real-time system should be based on the metric of the physical second.

In a typical real-time application, the distributed computer system performs a multitude of different functions concurrently, e.g., the monitoring of real-time (RT) entities (both their value and rate of change), the detection of alarm conditions, the display of the observations to the operator, and the execution of control algorithms to find new set-points for many distinct control loops. These diverse functions are normally executed at different nodes. In addition, replicated nodes are introduced to provide fault tolerance by active redundancy. To guarantee a consistent behavior of the entire distributed system, it must be ensured that all nodes process all events in the same consistent order, preferably in the same temporal order in which the events occurred (see also the example in Sect. 5.5) in the controlled object. A proper global time base helps to establish such a consistent temporal order on the basis of the time-stamps of the events.

3.1.1 Different Orders

Temporal Order. The continuum of Newtonian real time can be modeled by a *directed timeline* consisting of an infinite set {T} of *instants* (or *points in time*) with the following properties [Wit90, p. 208]:

1. {T} is an ordered set, that is, if p and q are any two instants, then either p is simultaneous with q, or p precedes q, or q precedes p, where these relations are mutually exclusive. We call the order of instants on the timeline the *temporal order*.
2. {T} is a dense set. This means that there is at least one q between p and r *iff p is not the same instant as r*, where p, q, and r are instants.

A section of the time line between two different instants is called a *duration*. In our model, an *event* takes place at an instant of time and does not have a duration. If two events occur at the *same* instant, then the two events are said to occur *simultaneously*. Instants are totally ordered; however, events are only partially ordered, since simultaneous events are not in the order relation. Events can be totally ordered if another criterion is introduced to order events that occur simultaneously, e.g., in a distributed computer system, the number of the node at which the event occurred can be used to order events that occur simultaneously [Lam78].

Causal Order. In many real-time applications, the causal dependencies among events are of interest. The computer system must assist the operator in identifying the *primary event* of an *alarm shower* (see Sect. 1.2.1). Knowledge of the exact temporal order of the events is helpful in identifying this primary event. If an event $e1$ occurs after an event $e2$, then $e1$ cannot be the cause of $e2$. If, however, $e1$ occurs before $e2$, then it is possible, but not certain, that $e1$ is the cause of $e2$. The *temporal* order of two events is necessary, but not sufficient, for their *causal* order. Causal order is *more* than temporal order.

Reichenbach [Rei57, p. 145] defined causality by a mark method without reference to time: If event $e1$ is a cause of event $e2$, then a small variation (a mark) in e1 is associated with small variation in $e2$, whereas small variations in $e2$ are not necessarily associated with small variations in $e1$.

Example: Suppose there are two events $e1$ and $e2$:
$e1$ Somebody enters a room.
$e2$ The telephone starts to ring.

Consider the following two cases
1. $e2$ occurs after $e1$.
2. $e1$ occurs after $e2$.

In both cases the two events are temporally ordered. However, while it is unlikely that there is a causal order between the two events of case (1), it is likely that such a causal order exists between the two events of case (2), since the person might enter the room to answer the telephone.

If the (partial) temporal order between alarm events has been established, it is possible to exclude an event from being the *primary event* if it *definitely occurred later* than another alarm event. Subsequently, we will show that a precise global time base helps to determine the event set that is in this *definitely-occurred-later-than* relation (see also the example in Sect. 1.2.1).

Delivery Order. A weaker order relation that is often provided by distributed communication systems is a consistent *delivery order*. The communication system guarantees that all nodes see a defined *set of related events* in the same delivery order. This delivery order is not necessarily related to the temporal order of event occurrences or the causal relationship between events. Some distributed algorithms, e.g., *atomic broadcast algorithms* require a consistent delivery order.

3.1.2 Clocks

In ancient history, the measurement of durations between events was mainly based on subjective judgment. With the advent of modern science, objective methods for measuring the progression of time by using *physical clocks* have been devised.

Digital Physical Clock. A (*digital physical*) *clock* is a device for measuring time. It contains a *counter* and a *physical oscillation mechanism* that periodically generates

an event to increase the counter. The periodic event is called the *microtick* of the clock. (The term *tick* is introduced in Sect. 3.2.1 to denote the events generated by the global time).

Granularity. The duration between two consecutive microticks of a digital physical clock is called a *granule* of the clock. The granularity of a given clock can be measured only if there is a clock with a finer granularity available. The granularity of any digital clock leads to a digitalization error in time measurement.

There also exist analog physical clocks, e.g., *sundials* that do not have granularity. In the following, we only consider digital physical clocks.

In subsequent definitions, we use the following notation: clocks are identified by natural numbers 1, 2, ..., n. If we express properties of clocks, the property is identified by the clock number as a superscript with the microtick or tick number as a subscript. For example, microtick i of clock k is denoted by $microtick_i^k$.

Reference Clock. Assume an *omniscient external observer* who can observe all events that are of interest in a given context (remember that relativistic effects are disregarded). This observer possesses a *unique reference clock* z with frequency f^z, *which* is in perfect agreement with the international standard of time. The counter of the reference clock is always the same as that of the international time standard. We call $1/f^z$ the *granularity* g^z of clock z. Let us assume that f^z is very large, say 10^{15} microticks/s, so that the granularity g^z is 1 fs (10^{-15} s). Since the granularity of the reference clock is so small, the digitalization error of the reference clock is considered a second order effect and disregarded in the following analysis.

Absolute Time-Stamp. Whenever the omniscient observer perceives the occurrence of an event e, she/he will instantaneously record the current state of the reference clock as the time of occurrence of this event e, and, will generate a *time-stamp* for e. $Clock(e)$ denotes the time-stamp generated by the use of a given *clock* to time-stamp an event e. Because z is the single reference clock in the system, $z(e)$ is called the *absolute time-stamp* of the event e.

The *duration* between two events is measured by counting the microticks of the reference clock that occur in the interval between these two events. The *granularity* g^k of a given clock k can now be measured and is given by the nominal number n^k of microticks of the reference clock z between 2 microticks of this clock k.

The temporal order of events that occur between any two consecutive microticks of the reference clock, i.e., within the granularity g^z, cannot be reestablished from their absolute time-stamps. This is a fundamental limit in time measurement.

Clock Drift. The *drift* of a physical clock k between microtick i and microtick $i + 1$ is the frequency ratio between this clock k and the reference clock, at the instant of microtick i. The drift is determined by measuring the duration of a granule of clock k with the reference clock z and dividing it by the nominal number n^k of reference clock microticks in a granule:

$$drift_i^k = \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k}$$

Because a good clock has a drift that is very close to 1, for notational convenience the notion of a *drift rate* ρ_i^k is introduced as

$$\rho_i^k = \left| \frac{z(microtick_{i+1}^k) - z(microtick_i^k)}{n^k} - 1 \right|.$$

A perfect clock will have a drift rate of 0. Real clocks have a varying drift rate that is influenced by environmental conditions, e.g., a change in the ambient temperature, a change in the voltage level that is applied to a crystal oscillator, or aging of the crystal. Within specified environmental parameters, the drift rate of an oscillator is bounded by the *maximum drift rate* $\rho_{\max}^k$, which is documented in the data sheet of the resonator. Typical maximum drift rates $\rho_{\max}^k$ are in the range of 10^{-2} to 10^{-7} s/s, or better, depending on the quality (and price) of the oscillator. 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 startup.

> **Example:** During the Gulf war on February 25, 1991, a Patriot missile defense system failed to intercept an incoming Scud rocket. The clock drift over a 100-h period (which resulted in a tracking error of 678 m) was blamed for the Patriot missing the Scud missile that hit an American military barracks in Dhahran, killing 29 and injuring 97. The original requirement was a 14-h mission. The clock drift during a 14-h mission could be handled [Neu95, p. 34].

Failure Modes of a Clock. A physical digital clock can exhibit two types of failures. The counter could be mutilated by a fault so that the counter value becomes erroneous, or the drift rate of the clock could depart from the specified drift rate (the shaded area of Fig. 3.1) because the clock starts ticking faster (or slower) than specified.

3.1.3 Precision and Accuracy

Offset. The *offset* at microtick i between two clocks j and k with the same granularity is defined as

$$offset_i^{jk} = \left| z(microtick_i^j) - z(microtick_i^k) \right|$$

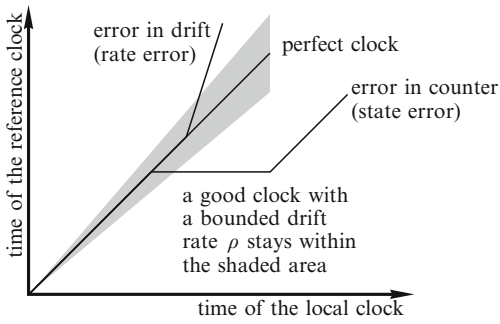

Fig. 3.1 Failure modes of a physical clock

The offset denotes the time difference between the respective microticks of the two clocks, measured in the number of microticks of the reference clock.

Precision. Given an ensemble of n clocks $\{1, 2, \ldots, n\}$, the maximum offset between any two clocks of the ensemble

$$\Pi_i = \max_{\forall 1 \leqslant j,k \leqslant n} \left\{ offset_i^{jk} \right\}$$

is called the *precision* Πof the ensemble at microtick i. The maximum of Π_i over an *interval of interest* is called the *precision* Π of the ensemble. The precision denotes the *maximum offset* of respective microticks of any two clocks of the ensemble during a duration of interest. The precision is expressed in the number of microticks of the reference clock.

Because of the drift rate of any physical clock, the clocks of an ensemble will drift apart if they are not resynchronized periodically (i.e., brought closer together). 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 z at microtick i is called the $accuracy_i^k$. The maximum offset over all microticks i that is *of interest is* called the $accuracy^k$ of clock k. The accuracy denotes the maximum offset of a given clock from the external time reference during a duration of interest.

To keep a clock within a bounded interval of the reference clock, it must be periodically resynchronized with an external time reference. This process of resynchronization of a clock with an external time reference is called *external synchronization*.

If all clocks of an ensemble are externally synchronized with an accuracy A, then the ensemble is also internally synchronized with a precision of at most $2A$. The converse is not true. An ensemble of internally synchronized clocks will drift from the external time if the clocks are never resynchronized with the external time base.

3.1.4 Time Standards

In the last decades, a number of different time standards have been proposed to measure the time difference between any two events and to establish the position of an event relative to some commonly agreed origin of a time base, the *epoch*. Two of these time bases are relevant for the designer of a distributed real-time computer system, the *International Atomic Time* (TAI) and the *Universal Time Coordinated* (UTC).

International Atomic Time (TAI – Temps Atomique Internationale). The need for a time standard that can be generated in a laboratory gave birth to the International Atomic Time (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. TAI is a *chronoscopic* timescale, i.e., a timescale without any discontinuities (e.g., leap seconds). The epoch of TAI starts on January 1, 1958 at 00:00 h Greenwich Mean Time (GMT). The time base of the global positioning system GPS is based on TAI with the epoch starting on January 6, 1980 at 00:00 h.

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. It is the basis for the time on the *wall-clock*. However, there is a known offset between the local wall-clock time and UTC determined by the time zone and by the political decisions about when daylight savings time must be used. The UTC time standard was introduced in 1972, replacing the Greenwich Mean Time (GMT) as an international time standard. Because the rotation of the earth is not smooth, but slightly irregular, the duration of the GMT second changes slightly over time. In 1972, it was internationally agreed that the duration of the second should conform to the TAI standard, and that the number of seconds in an hour would have to be modified occasionally by inserting a *leap second* into the UTC to maintain synchrony between the UTC (wall-clock time) and astronomical phenomena, like day and night. Because of this leap second, the UTC is not a chronoscopic timescale, 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 by about 30 s. The point in time when a leap second is inserted into the UTC is determined by the Bureau International de l'Heure and publicly announced, so that the current offset between the UTC and the TAI is always known.

Example: In *Software Engineering Notes* of March 1996 [Pet96, p. 16] was the following story:

> Ivan Peterson reported on a problem that occurred when a leap second was added at midnight on New Year's Eve 1995. The leap second was added, but the date inadvertently advanced to Jan. 2. Ivars heard from a source at AP radio that the synchronization of their broadcast networks depends on the official time signal, and this glitch affected their operation for several hours until the problem was corrected. You can't even count on the national timekeepers to get it right all the time.
> Bob Huey responded that making corrections at midnight is obviously risky: (1) The day increments to January 1, 1996, 00:00:00. (2) You reset the clock to 23:59:59, back one second. (3) The clock continues running. (4) The day changes again, and it is suddenly January 2, 1996, 00:00:00. No wonder they had problems.

3.2 Time Measurement

If the real-time clocks of all nodes of a distributed system were perfectly synchronized with the reference clock z, and all events were time-stamped with this reference time, then it would be easy to measure the interval between any two events or to reconstruct the temporal order of events, even if variable

communication delays generated differing delivery orders. In a loosely coupled distributed system where every node has its own local oscillator, such a tight synchronization of clocks is not possible. A weaker notion of a universal time reference, the concept of *global time*, is therefore introduced into a distributed system.

3.2.1 Global Time

Suppose a set of nodes exists, each one with its own local physical clock c^k that ticks with granularity g^k. Assume that all of the clocks are internally synchronized with a precision Π, i.e., for any two clocks j, k, and all microticks i

$$\left|z(microtick_i^j) - z(microtick_i^k)\right| < \Pi.$$

(In Sect. 3.4, methods for the internal synchronization of the clocks are presented). It is then possible to select a *subset of the microticks* of each local clock k for the generation of the local implementation of a global notion of time. We call such a selected local microtick i a *macrotick* (or a *tick*) of the global time. For example, every tenth microtick of a local clock k may be interpreted as the global tick, the *macrotick* t_i^k, of this clock (see Fig. 3.2). If it does not matter at which clock k the (macro)tick occurs, we denote the tick t_i without a superscript. A global time is thus an *abstract notion* that is *approximated* by properly selected microticks from the synchronized local physical clocks of an ensemble.

Reasonableness Condition. The global time t is called *reasonable*, if all local implementations of the global time satisfy the condition

$$g > \Pi$$

the *reasonableness condition* for the global granularity g. This reasonableness condition ensures that the synchronization error is *bounded* to less than one *macrogranule*, i.e., the duration between two (macro) ticks. If this reasonableness condition is satisfied, then for a single event e, that is observed by any two different clocks of the ensemble,

$$\left|t^j(e) - t^k(e)\right| \leqslant 1,$$

i.e., the global time-stamps for a single event can differ by at most one tick. *This is the best we can achieve*. Because of the impossibility of synchronizing the clocks

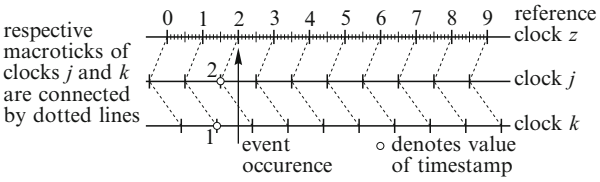

Fig. 3.2 Time-stamps of a single event

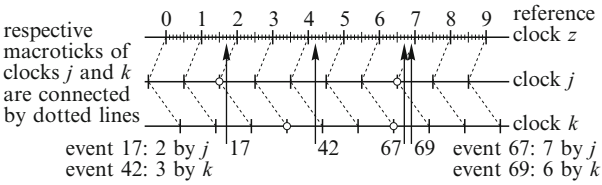

Fig. 3.3 Temporal order of two events with a difference of one tick

perfectly, and the granularity of any digital time there is always the possibility of the following sequence of events: clock j ticks, event e occurs, clock k ticks. In such a situation, the single event e is time-stamped by the two clocks j and k with a difference of one tick (Fig. 3.2).

One Tick Difference – What Does It Mean? What can we learn about the temporal order of two events, observed by different nodes of a distributed system with a reasonable global time, given that the global time-stamps of these two events differ by one tick?

In Fig. 3.3, four events are depicted, *event 17*, *event 42*, *event 67* and *event 69* (time-stamps from the reference clock). Although the duration between *event 17* and *event 42* is 25 microticks, and the duration between *event 67* and *event 69* is only 2 microticks, both durations lead to the *same* measured difference of one macrogranule. The global time-stamp for *event 69* is *smaller* than the global time-stamp for *event 67*, although e*vent 69* occurred *after event 67*. Because of the accumulation of the synchronization error and the digitalization error, it is not possible to reconstruct the temporal order of two events from the knowledge that the global time-stamps differ by one tick. However, if the time-stamps of two events differ by two ticks, then the temporal order can be reconstructed because the sum of the synchronization and digitalization error is always less than two granules in a clocking system with a reasonable global time-base.

This fundamental limitation in time measurement limits the *faithfulness* of the digital computer model of a controlled physical subsystem. The time-base in the physical part of a cyber-physical system is dense, while the time base in the computer system is discrete. Whenever two events in the physical subsystem occur close together, compared to the granularity of the global time, it is not possible to reconstruct the physical temporal order of the events in the computer system faithfully. The only way out of this dilemma is the provision of a global time base with a smaller granularity, such that temporal errors are reduced [Kop09].

3.2.2 Interval Measurement

An interval is delimited by two events, the start event of the interval and the terminating event of the interval. 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 of the reasonableness

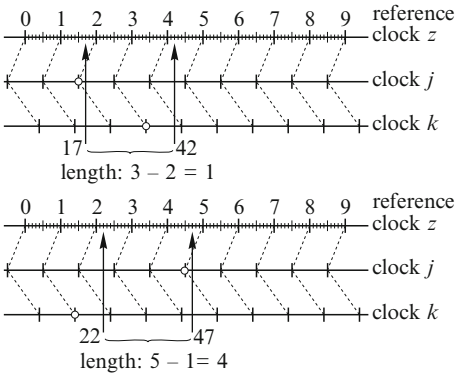

Fig. 3.4 Errors in interval measurement

condition, where g is the granularity of the global time. It follows that the true duration d_{true} of an interval is bounded by

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

where d_{obs} is the observed difference between the start event and the terminating event of the interval. Figure 3.4 depicts how the observed duration of an interval of length 25 microticks can differ, depending on which node observes the start event and the terminating event. The global tick, assigned by an observing node to an event delimiting the interval is marked by a small circle in Fig. 3.4.

3.2.3 π/Δ-Precedence

Consider a distributed computer system that consists of three nodes j, k, and m that support a global time. Every node is to generate an event at its view of the global instants 1, 5, and 9. An omniscient outside observer will see the scenario depicted in Fig. 3.5.

All events that are generated locally at the same global clock tick will occur within a small interval π, where $\pi \leq \Pi$, the precision of the ensemble (because of the reasonableness condition). Events that occur at different ticks will be at least $\boldsymbol{\Delta}$ apart (Fig. 3.5). The outside observer should not order the events that occur within π, because these events are *supposed* to occur at the same instant. Events that occur at different ticks should be ordered. How many granules of silence must exist between the event subsets such that an outside observer or another cluster will always recover the temporal order intended by the sending cluster? Before we can answer this question (in Sect. 3.3.2) we must introduce the notion of $\pi/\boldsymbol{\Delta}$ precedence.

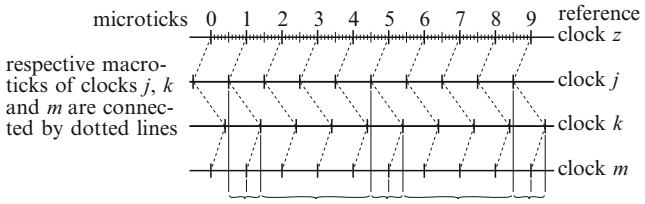

Fig. 3.5 π/Δ precedence

Given a set of events {E} and two durations π and Δ where $\pi << \Delta$, such that for any two elements e_i and e_j of this set, the following condition holds:

$$[|z(e_i) - z(e_j)| \leqslant \pi] \text{ v } [|z(e_i) - z(e_j)| > \Delta]$$

where z is the reference clock. Such an event set is called π/Δ-*precedent*. π/Δ-*precedence* means that a subset of the events that happen at about the same time (and that are therefore close together within π) is separated by a substantial interval (at least Δ) from the elements in another subset. If π is zero, then any two events of the $0/\Delta$-*precedent* event set occur either at the same instant or are at least a duration Δ apart.

Assume a distributed system with a reasonable global time base with granularity g and two events, $e1$ and $e2$, that are produced at the same locally generated global tick of two different nodes. Due to the synchronization error, these events can differ by up to but less than one granule. These events are observed by some of the other nodes.

Because of the synchronization and digitalization error, the two (simultaneous by intention) events can be time-stamped by the observers with two ticks difference. In order to be able to establish the intended temporal order of events from their time-stamps a sufficient duration of silence is needed before the next event may occur in order to ensure that the intended simultaneity of the events can always be recovered by all observers [Ver94].

3.2.4 Fundamental Limits of Time Measurement

The above analysis leads to the following four fundamental limits of time measurement in distributed real-time systems with a reasonable global time base with granularity g.

1. If a single event is observed by two different nodes, there is always the possibility that the time-stamps differ by one tick. A one-tick difference in the time-stamps of two events is not sufficient to reestablish the temporal order of the events from their time-stamps.

2. If the observed duration of an interval is d_{obs}, then the true duration d_{true} is bounded by

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

3. The temporal order of events can be recovered from their time-stamps if the difference between their time-stamps is equal to or greater than 2 ticks.
4. The temporal order of events can *always* be recovered from their time-stamps, if the event set is at least $0/3g$ precedent.

These fundamental limits of time measurement are also the *fundamental limits to the faithfulness* of a digital model of a physical system.

3.3 Dense Time Versus Sparse Time

Example: It is known a priori that a particular train will arrive at a train station every hour. If the train is always on time and all clocks are synchronized, it is possible to uniquely identify each train by its time of arrival. Even if the train is slightly off, say, by 5 min, and the clocks are slightly out of synchronization, say, by 1 min, there will be no problem in uniquely identifying a train by its time of arrival. What are the limits within which a train can still be uniquely identified by its time of arrival?

Assume a set {E} of events that are of interest in a particular context. This set {E} could be the ticks of all clocks, or the events of sending and receiving messages. If these events are allowed to occur at any instant of the timeline, then, we call the time base *dense*. If the occurrence of these events is restricted to some *active intervals* of duration ε, with an interval of silence of duration $\mathbf{\Delta}$ between any two active intervals, then we call the time base $\varepsilon/\mathbf{\Delta}$-*sparse*, or simply *sparse* for short (Fig. 3.6). If a system is based on a sparse time base, there are time intervals during which no significant event is allowed to occur. Events that occur only in the active intervals are called *sparse events*.

It is evident that the occurrences of events can only be restricted if the given system has the authority to control these events, i.e., these events are in the sphere of control of the computer system [Dav79]. The occurrence of events outside the sphere of control of the computer system cannot be restricted. These external events are based on a dense time base and cannot be forced to be *sparse events*.

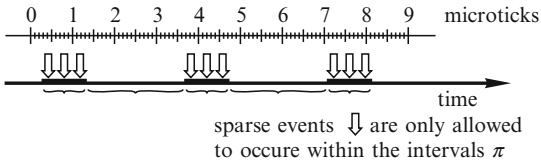

Fig. 3.6 Sparse time-base

Example: Within a distributed computing system, the sending of messages can be restricted to some intervals of the timeline and can be forbidden at some other intervals – they can be designed to be *sparse events*.

3.3.1 Dense Time-Base

Suppose that we are given two events $e1$ and $e2$ that occur on a dense time base. If these two events are closer together than $3g$, where g is the granularity of the global time, then, it is not always possible to establish the temporal order, or even a consistent order of these two events on the basis of the time-stamps generated by the different nodes if no *agreement protocol* (see below) is applied.

Example: Consider the scenario of Fig. 3.7 with two events, $e1$ and $e2$ which are 2.5 granules apart. Event $e1$ is observed by node j at time 2 and by node m at time 1, while $e2$ is only observed by node k that reports its observation "$e2$ occurred at 3" to node j and node m. Node j calculates a time-stamp difference of one tick and concludes that the events occurred at about the same time and *cannot* be ordered. Node m calculates a time-stamp difference of 2 ticks and concludes that $e1$ *has definitely occurred* before $e2$. The two nodes j and m have an *inconsistent* view about the order of event occurrence.

Agreement Protocol. To arrive at a *consistent view* of the order of *non-sparse events* within a distributed computer system (which does not necessarily reflect the temporal order of event occurrence), the nodes must execute an *agreement protocol*. The first phase of an agreement protocol requires an information interchange among the nodes of the distributed system with the goal that every node acquires the differing local views about the state of the world from every other node. At the end of this first phase, every correct node possesses exactly the same information as every other node. In the second phase of the agreement protocol, each node applies a deterministic algorithm to this consistent information to reach the same conclusion about the assignment of the event to an active interval of the sparse time base–the commonly agreed value. In the fault-free case, an agreement algorithm requires an additional round of information exchange as well as the resources for executing the agreement algorithm.

Agreement algorithms are costly, both in terms of communication requirements, processing requirements, and – worst of all – in terms of the additional delay they introduce into a control loop. It is therefore expedient to look for solutions to the consistent temporal ordering problem in distributed computer systems that do not

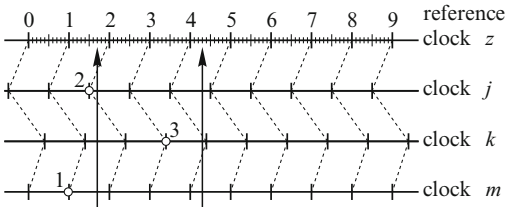

Fig. 3.7 Different observed order of two events $e1$ and $e2$

require these additional overheads. The sparse time model, introduced below, provides for such a solution.

3.3.2 Sparse Time-Base

Consider a distributed system that consists of two clusters: cluster A generates events, and cluster B observes these generated events. Each one of the clusters has its own cluster-wide synchronized time with a granularity g, but these two cluster-wide time bases are not synchronized with each other. Under what circumstances is it possible for the nodes in the observing cluster to reestablish consistently the *intended temporal order* of the generated events without the need to execute an agreement protocol?

If two nodes, nodes j and k of cluster A, generate two events at the same cluster-wide tick t_i, i.e., at tick t_i^j and at tick t_i^k, then these two events can be, at most, a distance Π apart from each other, where $g > \Pi$, the granularity of the cluster-wide time. Because there is no intended temporal order among the events that are generated at the same cluster-wide tick of cluster A, the observing cluster B should *never* establish a temporal order among the events that have occurred at about the same time. On the other hand, the observing cluster B should *always* reestablish the temporal order of the events that have been occurred at different cluster-wide ticks. Is it sufficient if cluster A generates a $1g/3g$ precedent event set, i.e., after every cluster-wide tick at which events are allowed to be generated there will be silence for at least three granules?

If cluster A generates a $1/3g$ precedent event set, then it is possible that two events that are generated at the same cluster-wide granule at cluster A will be time-stamped by cluster B with time-stamps that differ by 2 ticks. The observing cluster B should not order these events (although it could), because they have been generated at the same cluster-wide granule. Events that are generated by cluster A at different cluster-wide granules ($3g$ apart) and therefore should be ordered by cluster B, could also obtain time-stamps that differ by 2 ticks. Cluster B cannot decide whether or not to order events with a time-stamp difference of 2 ticks. To resolve this situation, cluster A must generate a $1/4g$ precedent event set. Cluster B will not order two events if their time-stamps differ by ≤ 2 ticks, but will order two events if their time-stamps differ by ≥ 3 ticks, thus reestablishing the temporal order that has been intended by the sender.

3.3.3 Space-Time Lattice

The ticks of the global clock can be seen as generating a space-time lattice, as depicted in Fig. 3.8. A node is allowed to generate an event (e.g., send a message) at the filled dots and must be silent at the empty dots. This rule makes it possible for

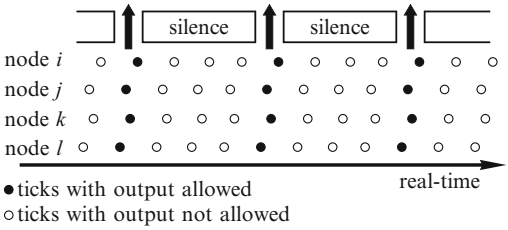

Fig. 3.8 Sparse time base

the receiver to establish a consistent temporal order of events without executing an agreement protocol. Although a sender is allowed to generate an event only at the filled dots, this is still much faster than executing an agreement protocol, provided a global time base of sufficient precision is available. Events that are generated at the filled dots of the sparse time lattice are called *sparse events*.

Events that occur outside the sphere of control of the computer system cannot be confined to a sparse time base: they happen on a dense time base and are therefore not *sparse events*. To generate a consistent view of events that occur in the controlled object, and that are observed by more than one node of the distributed computer system, the execution of an agreement protocol is unavoidable at the interface between the computer system and the controlled object or other systems that do not participate in the global time. Such an agreement protocol transforms a *non-sparse event* into a *sparse event*.

3.3.4 Cyclic Representation of Time

Many processes in the technical and biological world are cyclic [Win01]. A cyclic process is characterized by a regular behavior, where a similar set of action patterns is repeated in every cycle.

> **Example:** In a typical control system, real-time is partitioned into a sequence of control cycles (Fig. 3.9). Every control cycle starts with reading the state variables of the controlled object, proceeds with the execution of the control algorithm, and finishes with the output of new set-points to the actuators at the interface between the computer system and the controlled object.

In the cyclic representation of time, the linear time is partitioned into cycles of equal duration. Every cycle is represented by a circle, where an instant within a cycle is denoted by the *phase*, i.e., the angular deviation of the instant from the beginning of the cycle. *Cycle* and *phase* thus denote an instant in a cyclic representation. In the cyclic representation of sparse time, the circumference of the circle is not a *dense* line, but a *dotted* line, where the size and the distance between dots is determined by the precision of the clock synchronization.

A sequence of consecutive processing and communication actions, such as the actions in Fig. 3.9, are *phase-aligned*, if the termination of one action is immediately followed by the start of the next consecutive action. If the actions

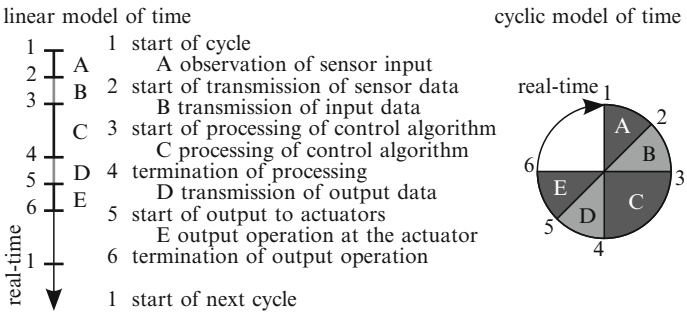

Fig. 3.9 Linear versus cyclic representation of time in a control system

within a *RT-transaction* (see Sect. 1.7.3) are phase-aligned, then the overall duration of the RT transactions is minimized.

If we look at Fig. 3.9, we see that communication services in a typical control loop are periodically required only in the intervals *B* and *D* of a cycle. The *shorter* these intervals B and D, the *better*, since the dead time of the control loop is reduced. This requirement leads to the model of *pulsed data streams*, where, in a time-triggered system, the highest possible bandwidth is allocated periodically in the intervals B and D, while, during the rest of the cycle, the communication bandwidth can be allocated to other requests [Kop06].

An extension of the cyclic representation is the *spiral representation of time*, where a third axis is introduced to depict the linear progression of the cycles.

3.4 Internal Clock Synchronization

The purpose of internal clock synchronization is to ensure that the global ticks of all correct nodes occur within the specified *precision* Π, despite the varying drift rate of the local real-time clock of each node. Because the availability of a proper global time base is crucial for the operation of a distributed real-time system, the clock synchronization should not depend on the correctness of a single clock, i.e., it should be fault-tolerant.

Every node of a distributed system has a local oscillator that (micro)ticks with a frequency determined by the physical parameters of the oscillator. A subset of the local oscillator's microticks called the ticks (or macroticks – see Sect. 3.2.1) is interpreted as the global time ticks at the node. These global time ticks increment the local node's global time counter.

3.4.1 The Synchronization Condition

The global time ticks of each node must be periodically resynchronized within the ensemble of nodes to establish a global time base with specified precision.

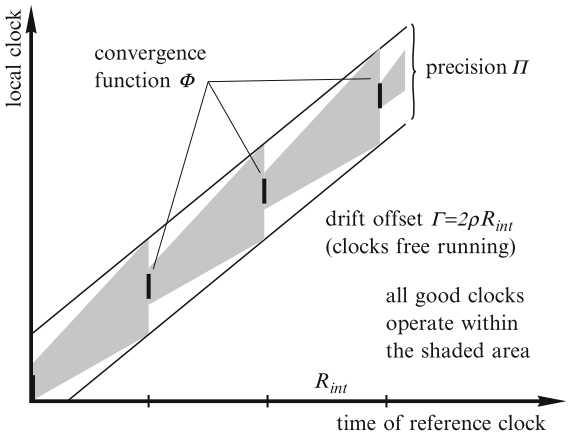

Fig. 3.10 Synchronization condition

The period of resynchronization is called the *resynchronization interval* R_{int}. At the end of each resynchronization interval, the clocks are adjusted to bring them into better agreement with each other. The *convergence function* Φ denotes the offset of the time values immediately after the resynchronization. Then, the clocks drift again apart until they are resynchronized at the end of the next resynchronization interval R_{int} (Fig. 3.10). The *drift offset* Γ indicates the maximum accumulated divergence of any two good clocks from each other during the resynchronization interval R_{int}, where the clocks are free running. The drift offset Γ depends on the length of the resynchronization interval R_{int} and the maximum specified drift rate ρ of the clock:

$$\Gamma = 2\rho R_{\text{int}}$$

An ensemble of clocks can only be synchronized if the following *synchronization condition* holds between the *convergence function* Φ, the *drift offset* Γ and the *precision* Π:

$$\Phi + \Gamma \leqslant \Pi$$

Assume that at the end of the resynchronization interval, the clocks have diverged so that they are at the edge of the precision interval Π (Fig. 3.10). The synchronization condition states that the synchronization algorithm must bring the clocks so close together that the amount of divergence during the next free-running resynchronization interval will not cause a clock to leave the precision interval.

Byzantine Error. The following example explains how, in an ensemble of three nodes, a malicious node can prevent the other two nodes from synchronizing their clocks since they cannot satisfy the synchronization condition [Lam82]. Assume an ensemble of three nodes, and a convergence function where each of the three nodes sets its clock to the average value of the ensemble. Clocks A and B are good, while clock C is a malicious *two-faced* clock that disturbs the other two good clocks in

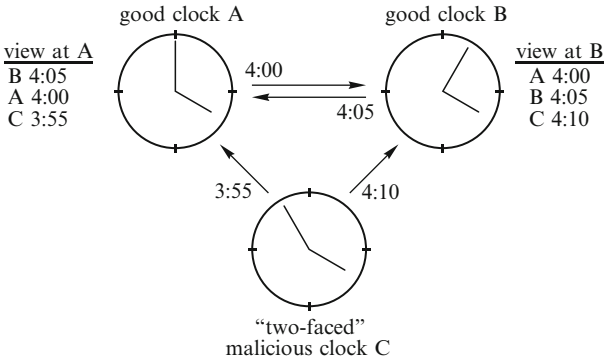

Fig. 3.11 Behavior of a malicious clock

such a manner that neither of them will ever correct their time value (Fig. 3.11), and will thus eventually violate the synchronization condition.

Such a malicious, *two-faced* manifestation of behavior is sometimes called a *malicious error* or a *Byzantine error* (see also Sect. 6.1.3). During the exchange of the synchronization messages, a Byzantine error can lead to inconsistent views of the state of the clocks among the ensemble of nodes. A special class of algorithms, the *interactive-consistency algorithms* [Pea80], inserts additional rounds of information exchanges to agree on a consistent view of the time values at all nodes. These additional rounds of information exchanges increase the quality of the precision at the expense of additional communication overhead. Other algorithms work with inconsistent information, and establish bounds for the maximum error introduced by the inconsistency. An example of such an algorithm is the Fault-Tolerant-Average algorithm, described later in this section. It has been shown [Lam85] that clock synchronization can only be guaranteed in the presence of Byzantine errors if the total number of clocks $N \geq (3k + 1)$, where k is the number of Byzantine faulty clocks.

3.4.2 Central Master Synchronization

This is a simple non-fault tolerant synchronization algorithm. A unique node, the central master, periodically sends the value of its time counter in synchronization messages to all other nodes, the slave nodes. As soon as a slave node receives a new synchronization message from the master, the slave records the time-stamp of message arrival. The difference between the master's time, contained in the synchronization message, and the recorded slave's time-stamp of message arrival, corrected by the known latency of the message transport, is a measure of the deviation of the clock of the master from the clock of the slave. The slave then corrects its clock by this deviation to bring it into agreement with the master's clock.

The convergence function Φ of the central master algorithm is determined by the difference between the fastest and slowest message transmission to the slave nodes of the ensemble, i.e., the *latency jitter* ε between the event of writing the synchronization time value by the master and the events of message arrival time-stamping at all slaves.

Applying the *synchronization condition*, the precision of the central master algorithm is given by:

$$\Pi_{central} = \varepsilon + \Gamma$$

The central master synchronization is often used in the startup phase of a distributed system. It is simple, but not fault tolerant, since a failure of the master ends the resynchronization, causing the free-running clocks of the slaves to leave the precision interval soon thereafter. In a variant of this algorithm, a multi-master strategy is followed: if the active master fails silently and the failure is detected by a local time-out at a *shadow master*, one of the *shadow masters* assumes the role of the master and continues the resynchronization.

3.4.3 Fault-Tolerant Synchronization Algorithms

Typically, distributed fault-tolerant clock resynchronization proceeds in three distinct phases. In the first phase, every node acquires knowledge about the state of the global time counters in all the other nodes by the exchange of messages among the nodes. In the second phase, every node analyzes the collected information to detect errors, and executes the convergence function to calculate a correction value for the local global time counter. A node must deactivate itself if the correction term calculated by the convergence function is larger than the specified precision of the ensemble. Finally, in the third phase, the local time counter of the node is adjusted by the calculated correction value. Existing algorithms differ in the way in which the time values are collected from the other nodes, in the type of convergence function used, and in the way in which the correction value is applied to the local time counter.

Reading the Global Time. In a local-area network, the most important term affecting the precision of the synchronization is the jitter of the time messages that carry the current time values from one node to all the other nodes. The known minimal delay for the transport of a time message between two nodes can be compensated by an a priori known delay-compensation term [Kop87] that compensates for the delay of the message in the transmission channel and in the interface circuitry. The delay jitter depends more than anything else on the system level, at which the synchronization message is assembled and interpreted. If this is done at a high level of the architecture, e.g., in the application software, all random delays caused by the scheduler, the operating system, the queues in the protocol software, the message

Table 3.1 Approximate jitter of the synchronization message

Synchronization message assembled and interpreted	Approximate range of jitter
At the application software level	500 μs–5 ms
In the kernel of the operating system	10–100 μs
In the hardware of the communication controller	Less than 1 μs

retransmission strategy, the media-access delay, the interrupt delay at the receiver, and the scheduling delay at the receiver, accumulate and degrade the quality of the time values, thus deteriorating the precision of the clock synchronization. Table 3.1 gives approximate value ranges for the jitter that can be expected at the different levels [Kop87]:

Since a small jitter is important to achieve high precision in the global time, a number of special methods for jitter reduction have been proposed. Christian [Cri89] proposed the reduction of the jitter at the application software level using a probabilistic technique: a node queries the state of the clock at another node by a query-reply transaction, the duration of which is measured by the sender. The received time value is corrected by the synchronization message delay that is assumed to be half the round-trip delay of the query-reply transaction (assuming that the delay distribution is the same in both directions). A different approach is taken in the Time-Triggered Architecture. A special clock synchronization unit has been implemented to support the segmentation and assembly of synchronization messages at the hardware level, thereby reducing the jitter to a few microseconds. The new IEEE 1588 standard for clock synchronization limits the jitter by hardware assisted time stamping [Eid06].

Impossibility Result. The important role of the latency jitter ε for internal synchronization is emphasized by an impossibility result by Lundelius and Lynch [Lun84]. According to this result, it is not possible to internally synchronize the clocks of an ensemble consisting of N nodes to a better precision than

$$\Pi = \varepsilon\left(1 - \frac{1}{N}\right)$$

(measured in the same units as ε) even if it is assumed that all clocks have perfect oscillators, i.e., the drift rates of all the local clocks are zero.

The Convergence Function. The construction of a convergence function is demonstrated by the example of the distributed Fault-Tolerant-Average (FTA) algorithm in a system with N nodes where 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 the 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 time differences are removed (assuming that an erroneous time value is

either larger or smaller than the rest). The remaining $N - 2k$ time differences are, by definition within the precision window definition (since only k values are assumed to be erroneous and an erroneous value is larger or smaller than a good value). The average of these remaining $N - 2k$ time differences is the correction term for the node's clock.

Example: Figure 3.12 shows an ensemble of seven nodes and one tolerated Byzantine fault. The FTA takes the average of the five accepted time values shown.

The worst-case scenario occurs if all good clocks are at opposite ends of the precision window Π, and the Byzantine clock is seen at different corners by two nodes. In the example of Fig. 3.13, node j will calculate an average value of $4\Pi/5$ and node k will calculate an average value of $3\Pi/5$; the difference between these two terms, caused by the Byzantine fault, is thus $\Pi/5$.

Precision of the FTA. Assume a distributed system with N nodes, each one with its own clock (all time values are measured in seconds). At most k out of the N clocks behave in a Byzantine manner.

A single Byzantine clock will cause the following difference in the calculated averages at two different nodes in an ensemble of N clocks:

$$E_{byz} = \Pi/(N - 2k).$$

In the worst case a total of k Byzantine errors will thus cause an error term of

$$E_{k-byz} = k\Pi/(N - 2k).$$

Considering the jitter of the synchronization messages, the convergence function of the FTA algorithm is given by

$$\Phi(N, k, \varepsilon) = (k\Pi/(N - 2k)) + \varepsilon.$$

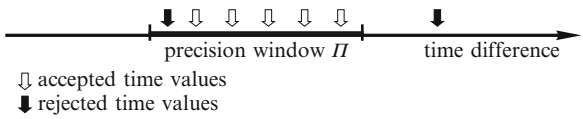

Fig. 3.12 Accepted and rejected time values

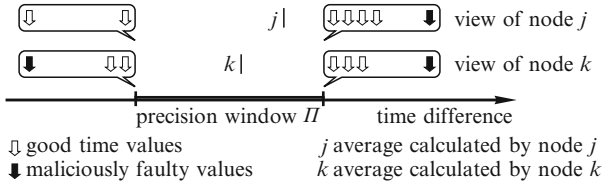

Fig. 3.13 Worst possible behavior of a malicious (Byzantine) clock

Table 3.2 Byzantine error term $\mu(N, k)$

	Number of nodes in the ensemble							
Faults	4	5	6	7	10	15	20	30
1	2	1.5	1.33	1.25	1.14	1.08	1.06	1.03
2				3	1.5	1.22	1.14	1.08
3					4	1.5	1.27	1.22

Combining the above equation with the synchronization condition (Sect. 3.4.1) and performing a simple algebraic transformation, we get the precision of the FTA algorithm:

$$\Pi(N, k, \varepsilon, \Gamma) = (\varepsilon + \Gamma)\frac{N - 2k}{N - 3k} = (\varepsilon + \Gamma)\mu(N, k).$$

where $\mu(N, k)$ is called the *Byzantine error term* and is tabulated in Table 3.2.

The Byzantine error term $\mu(N, k)$ indicates the loss of quality in the precision due to the inconsistency arising from the Byzantine errors. In a real environment, at most one Byzantine error is expected to occur in a synchronization round (and even this will happen very, very infrequently), and thus, the consequence of a Byzantine error in a properly designed synchronization system is not serious.

The drift-offset Γ is determined by the quality of the selected oscillator and the length of the resynchronization interval. If a standard quartz oscillator with a nominal drift rate of 10^{-4} s/s is used, and the clocks are resynchronized every second, then Γ is about 100 µs. Because the stochastic drift rate of a crystal is normally two orders of magnitude smaller than the nominal drift rate that is determined by the systematic error of the quartz oscillator, it is possible to reduce the drift offset Γ by up to two orders of magnitude by performing systematic error compensation.

Many other convergence functions for the internal synchronization of the clocks have been proposed and analyzed in the literature [Sch88].

3.4.4 State Correction Versus Rate Correction

The correction term calculated by the convergence function can be applied to the local-time value immediately (*state correction*), or the rate of the clock can be modified so that the clock speeds up or slows down during the next resynchronization interval to bring the clock into better agreement with the rest of the ensemble (*rate correction*).

State correction is simple to apply, but it has the disadvantage of generating a discontinuity in the time base. If clocks are set backwards and the same nominal-time value is reached twice, then, pernicious failures can occur within the real-time software (see the example in Sect. 3.1.4). It is therefore advisable to implement rate

correction with a bound on the maximum value of the clock drift so that the error in interval measurements is limited. The resulting global time base then maintains the chronoscopy property despite the resynchronization. Rate correction can be implemented either in the digital domain by changing the number of microticks in some of the (macro)ticks, or in the analog domain by adjusting the voltage of the crystal oscillator. To avoid a common-mode drift of the complete ensemble of clocks, the average of the rate correction terms among all clocks in the ensemble should be close to zero.

3.5 External Clock Synchronization

External synchronization links the global time of a cluster to an external standard of time. For this purpose it is necessary to access a *timeserver*, i.e., an external time source that periodically broadcasts the current reference time in the form of a *time message*. This time message must raise a synchronization event (such as the beep of a wrist watch) in a designated node of the cluster and must identify this synchronization event on the agreed time scale. Such a time scale must be based on a widely accepted measure of time, e.g., the physical second, and must relate the synchronization event to a defined origin of time, the *epoch*. The interface node to a timeserver is called a *time gateway*. In a fault-tolerant system, the time-gateway should be a fault-tolerant unit (FTU – see Sect. 6.4.2).

3.5.1 External Time Sources

Assume that the time gateway is connected to a GPS (Global Positioning System). The accuracy of a GPS receiver is better than 100 ns and it has an authoritative long-term stability – in some sense, GPS is the worldwide measurement standard for measuring the progression of time. Alternatively, the external time source can be a temperature compensated crystal oscillator (TCXO) with a drift rate of better than 1 ppm, causing a *drift offset* of better 1 μs/s or an atomic clock, e.g., a Rubidum clock with a drift rate in the order of 10^{-12} , causing a *drift offset* of about 1 μs in 10 days (more expensive atomic clocks are even better). The time gateway periodically broadcasts time messages containing a synchronization event, as well as the information to place this synchronization event on the TAI scale. The time gateway must synchronize the global time of its cluster with the time received from the external time source. This synchronization is unidirectional, and therefore asymmetric, as shown in Fig. 3.14. It can be used to adjust the rate of the clocks without any concern for the occurrence of emergent instability effects.

If another cluster is connected to this *primary* cluster by a secondary time gateway, then, the unidirectional synchronization functions in the same manner. The secondary time gateway considers the synchronized time of the primary cluster as its time reference, and synchronizes the global time of the secondary cluster.

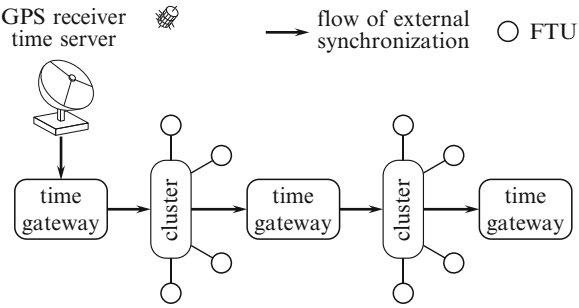

Fig. 3.14 Flow of external synchronization

While internal synchronization is a cooperative activity among all the members of a cluster, external synchronization is an authoritarian process: the time gateway forces its view of external time on all its subordinates. From the point of view of fault tolerance, such an authoritarian regime introduces a problem: if the authority sends an incorrect message, then all its *obedient* subordinates will behave incorrectly. However, for external clock synchronization, the situation is under control because of the *inertia* of time. Once a cluster has been synchronized, the fault-tolerant global time base within a cluster acts as a *monitor* of the time gateway. An external synchronization message will only be accepted if its content is sufficiently close to the cluster's view of the external time. The time gateway has only a limited authority to correct the drift rate of a cluster. The enforcement of a *maximum common-mode correction rate* is required to keep the error in relative time-measurements small. The software in each node of the cluster checks the maximum correction rate.

The implementation must guarantee that it is impossible for a faulty external synchronization to interfere with the proper operation of the internal synchronization, i.e., with the generation of global time within a cluster. The worst possible failure scenario occurs if the external timeserver fails maliciously – a very low probability failure mode if the external timeserver is GPS. This leads to a common-mode deviation of the global time from the external time base with the *maximum permitted deviation rate*. In a properly designed synchronization system, this drift from the external time base will not affect the internal synchronization within a cluster.

3.5.2 Time Gateway

The time gateway must control the timing system of its cluster in the following ways:

1. It must initialize the cluster with the current external time.
2. It must periodically adjust the rate of the global time in the cluster to bring it into agreement with the external time and the standard of time measurement, the second.

3. It must periodically send the current external time in a time message to the nodes in the cluster so that a reintegrating node can reinitialize its external time value.

The time gateway achieves this task by periodically sending a time message with a rate-correction byte. This rate-correction byte is calculated in the time gateway's software. First, the difference between the occurrence of a significant event, e.g., the exact start of the full second in the timeserver, and the occurrence of the related significant event in the global time of the cluster, is measured by using the local time base (microticks) of the gateway node. Then, the necessary rate adjustment is calculated, bearing in mind the fact that the rate adjustment is bounded by the agreed maximum rate correction. This bound on the rate correction is necessary to keep the maximum deviation of relative time measurements in the cluster below an agreed threshold, and to protect the cluster from faults of the server.

3.5.3 Time Formats

Over the last few years, a number of external-time formats have been proposed for external clock synchronization. The most important one is the standard for the time format proposed in the Network Time Protocol (NTP) of the Internet [Mil91]. This time format (Fig. 3.15) with a length of 8 B contains two fields: a 4 B full seconds 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 s, i.e., the number of seconds since January 1, 1900 at 00:00 h.

The NTP time is not chronoscopic because it is based on UTC which has to accommodate for the switching second. The occasional insertion of a leap second into UTC can disrupt the continuous operation of a time-triggered real-time system.

Another time format is the IEEE 1588 standard time format [Eid06]. In this time format the epoch starts on January 1, 1970 at 00:00 h or is user defined. The full seconds are counted according to TAI, while the unit of the fraction of a second is the nanosecond. This leads to abrupt change in the representation whenever a full second is reached.

The Time-Triggered Architecture (TTA) uses a time format that is a combination of IEEE 1588 and NTP. The full seconds are counted as in TAI (such as IEEE 1588), but parts of a second are represented in a binary fraction of the full second (such as NTP). It is thus chronoscopic and conforms fully to the dual system.

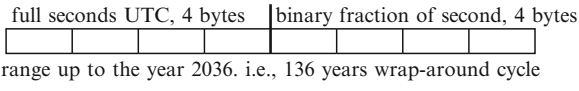

Fig. 3.15 Time format in the Network Time Protocol (NTP)

Points to Remember

- An event happens at an *instant*, i.e., at a point of the timeline. A *duration* is a section of the timeline delimited by two instants.
- A consistent delivery order of a set of events in a distributed system does not necessarily reflect the temporal or causal order of the events.
- A *physical clock* is a device for time measurement that contains a counter and a physical oscillation mechanism that periodically generates an event to increase the counter.
- Typical maximum drift rates ρ of physical clocks are in the range from 10^{-2} to 10^{-7} s/s, or lower, depending on the quality (and price) of the resonator.
- The *precision* denotes the maximum offset of respective ticks of any two clocks of an ensemble during the time interval of interest.
- The *accuracy* of a clock denotes the maximum offset of a given clock from the external time reference during the time interval of interest.
- TAI is a *chronoscopic* timescale, i.e., a timescale without any discontinuities, that is derived from the frequency of the radiation of a specified transition of the cesium atom 133.
- UTC is a non-chronoscopic timescale that is derived from astronomical observations of the rotation of the earth in relation to the sun.
- A *global time* is an abstract notion that is approximated by properly selected microticks from the synchronized local physical clocks of an ensemble.
- The *reasonableness condition* ensures that the synchronization error is always less than one granule of the global time.
- If the difference between the time-stamps of two events is *equal to or larger than* 2 ticks, then that temporal order of events can be recovered, provided the global time is reasonable.
- The temporal order of events can *always* be recovered from their time-stamps if the event set is at least $0/3g$ precedent.
- If events happen only at properly selected points of a *sparse time base*, then it is possible to recover the temporal order of the events without the execution of an agreement protocol.
- The *convergence function* Φ denotes the offset of the time values immediately after the resynchronization.
- The *drift offset* Γindicates the maximum divergence of any two good clocks from each other during the resynchronization interval R_{int}, in which the clocks are free running.
- The synchronization condition states that the synchronization algorithm must bring the clocks so close together that the amount of divergence during the next free-running resynchronization interval will not cause a clock to leave the precision interval.
- Clock synchronization is only possible if the total number of clocks N is larger or equal to $(3k + 1)$, if k is the number of clocks behaving *maliciously* faulty.

- The most important term affecting the precision of the synchronization is the *latency jitter* of the synchronization messages that carry the current time values from one node to all other nodes of an ensemble.
- When applying the fault-tolerant average algorithm, the *Byzantine error factor* $\mu(N, k)$ denotes the loss of quality in the precision caused by the Byzantine errors.
- *State correction* of a clock has the disadvantage of generating a discontinuity in the time base.
- While internal synchronization is a cooperative activity among all members of a cluster, external synchronization is an authoritarian process: the timeserver forces its view of external time on all its subordinates.
- The *NTP time*, based on UTC, is not chronoscopic. The occasional insertion of a leap second can disrupt the continuous operation of a time-triggered real-time system.
- The *time gateway* maintains the external synchronization by periodically sending a time message with a rate correction byte to all the nodes of a cluster.

Bibliographic Notes

The problem of generating a global time base in a distributed system has first been analyzed in the context of the SIFT [Wen78] and FTMP [Hop78] projects. A VLSI chipfor clock synchronization in distributed systems was developed by Kopetz and Ochsenreiter [Kop87]. The Network Time Protocol of the Internet was published in 1991 by Mills [Mil91]. Kopetz presented the concept of a sparse time model first in [Kop92]. The excellent book by Eidson [Eid06] covers the IEEE 1588 protocol for clock synchronization in detail. The integration of internal and external clock synchronization is discussed in Kopetz et al. [Kop04]. For a more philosophical treatment of the problem of time, the reader is advised to study the excellent book by Withrow in [Wit90] entitled *The Natural Philosophy of Time*.

Review Questions and Problems

3.1 What is the difference between an *instant* and an *event*?

3.2 What is the difference between *temporal* order, *causal* order, and a consistent *delivery* order of messages? Which of the orders implies another?

3.3 How can clock synchronization assist in finding the *primary* event of an alarm shower?

3.4 What is the difference between UTC and TAI? Why is TAI better suited as a time base for distributed real-time systems than UTC?

3.5 Define the notions of *offset*, *drift*, *drift rate*, *precision*, and *accuracy*.

3.6 What is the difference between *internal* synchronization and *external* synchronization?

3.7 What are the fundamental limits of time measurement?

3.8 When is an event set $\varepsilon/\mathbf{\Delta}$-*precedent*?

3.9 What is an *agreement protocol*? Why should we try to avoid agreement protocols in real-time systems? When is it impossible to avoid agreement protocols?
3.10 What is a *sparse time base*? How can a sparse time base help to avoid agreement protocols?
3.11 Give an example that shows that, in an ensemble of three clocks, a Byzantine clock can disturb the two good clocks such that the synchronization condition is violated.
3.12 Given a clock synchronization system that achieves a precision of 90 μs, what is a reasonable granularity for the global time? What are the limits for the observed values for a time interval of 1.1 ms?
3.13 What is the role of the *convergence function* in internal clock synchronization?
3.14 Given a latency jitter of 20 μs, a clock drift rate of 10^{-5} s/s, and a resynchronization period of 1 s, what precision can be achieved by the central master algorithm?
3.15 What is the effect of a Byzantine error on the quality of synchronization by the FTA algorithm?
3.16 Given a latency jitter of 20 μs, a clock drift rate of 10^{-5} s/s and a resynchronization period of 1 s, what precision can be achieved by the FTA algorithm in a system with ten clocks where one clock could be malicious?
3.17 Discuss the consequences of an error in the external clock synchronization. What effect can such an error have on the internal clock synchronization in the worst possible scenario?