# A time synchronization protocol for A664-P7

Frédéric Boulanger, Dominique Marcadet, Martin Rayrole, Benoît Valiron, Safouan Taha

# A time synchronization protocol for A664-P7

Frédéric Boulanger*, Dominique Marcadet*, Martin Rayrole†, Safouan Taha* and Benoît Valiron*

*LRI, CentraleSupélec, Université Paris Saclay, Gif-Sur-Yvette, France. Email: firstname.lastname@lri.fr
†Thales Avionics, Vélizy, France. Email: Martin.Rayrole@fr.thalesgroup.com

***Abstract*—The norm for A664-P7 networks used in avionic does not offer any mean of defining a common time reference to the various applications running on end-systems. In this paper, we propose an algorithm to provide such a time reference. To support the proposal, we implemented the algorithm and ran experiments with OMNeT++ to explore the robustness of the approach.**

***Index Terms*—A664-P7, Time-synchronization, Asynchronous**

## I. Introduction

The AME platform (*Avionique Modulaire Étendue*, i.e. "Extended Modular Avionics") is an embedded computer architecture made of computers and various heterogeneous communication media and services. The objective of the platform is to provide support for certified avionic functionalities. The network part of this architecture conforms to the ARINC-664 Part 7 standard, also known as AFDX (*Avionics Full Duplex Switched Ethernet*).

An A664-P7 network consists of switches, end-systems and ethernet links. Virtual links are statically configured over this network: they are one-to-many multicast communications links giving guaranties on available bandwidth and maximum delays. However, the standard does not define any way of providing the various applications running on end-systems with a common time reference.

This paper proposes and experiments such a time reference and describes its architecture, protocol and algorithms. In the literature, a realistic A664-P7 network consists of about 100 end-systems, 10 switches and 900 virtual links. This is our baseline for this work.

Several constraints should be imposed on a distributed time reference function for A664-P7. Firstly, it should be using the existing network and hardware: this means for example that packets cannot be timestamped by switches, which requires specialized hardware. Then, the protocol has to be deadlock-free, even in case of failure of one element. The synchronization algorithm has moreover to guarantee the convergence of the time references of the end-systems. Finally, for safety and robustness reasons, the protocol will be insulated from any external time reference. The initial expectations of the function are: the function should be operational within 200ms after starting, it should maintain a reasonable time precision, and the function should be able to run for 48 hours without resetting the clock.

Research partially funded by CORAIL project of CORAC (Conseil pour la Recherche Aéronautique Civile).

### A. Existing, implemented proposals

Time synchronization protocols usually consider two classes of nodes in the network: servers and clients. Servers are expected to keep among them a uniform notion of time that clients are able to access through specific protocol primitives. One of the main difficulty in these protocols is to ensure the synchronization between the servers. There are several ways of ensuring this synchronization. Some protocols such as NTP [1] make use of external time references such as GPS or atomic clocks. Other protocols such as PTP [2], SynUTC [3] and the SAE AS6802 [4] standard assume the use of switches and bridges able to timestamp packets. Finally, an alternative approach consists in integrating the time protocol directly into the protocol, as with TTP [5] or FlexRay [6].

Each of these implemented solutions fails short with respect to at least one of our constraints. The NTP protocol is ruled out because it requires precise external clocks, and all the other approaches asks either for specialized hardware or for dedicated networks.

### B. Our solution

At the origin of all existing synchronization protocols is an algorithm that does not come with a concrete protocol implementation: the Welch-Lynch algorithm [7]. As a purely theoretical algorithm, it offers a solution to the synchronization of distributed clocks and it abstracts most of the problems. The core task of the algorithm is to maintain the convergence of the clocks and to ensure some precision.

Our proposal for time synchronization in an A664-P7 network is based on the Welch-Lynch algorithm for the core time synchronization among the servers. We propose a setting where servers are in charge of setting up the time reference, keeping it up to date (using the Welch-Lynch algorithm), and communicating it to clients. The role of a client is to deliver the time reference obtained from the servers to applications running on the same end-system. Our study shows that the expected precision of $100\,\mu$s cannot be obtained in worst-case scenarios on a 100 Mb/s network, but the results are however useful in an avionics context. In order to validate our proposal, we modeled A664-P7 and the proposed time reference function within OMNeT++. We ran simulations on a realistic network instance. Our experiments show that our proposal satisfies the constraints, offering a solution to the problem of time synchronization in A664-P7 networks with standard hardware.

## II. Context

This section presents the problem we address.

### A. The A664-P7 Network

*Aeronautical Radio, Incorporated* (ARINC), held by Rockwell Collins since 2013, is a company providing communication systems for several industrial applications, including avionics.

ARINC standards are developed by the *Airlines Electronic Engineering Committee* (AEEC). The ARINC 664 standard defines an avionic bus with a deterministic Ethernet protocol. It consists in 8 sections [8]:

1) Systems Concepts and Overview
2) Ethernet Physical and Data Link Layer Specification
3) Internet-Based Protocols and Services
4) Internet-Based Address Structure & Assigned Numbers
5) Network Domain Characteristics and Interconnection
6) *Reserved*
7) Avionics Full-Duplex Switched Ethernet Network
8) Interoperation with Non-IP Protocols and Services

An A664-P7 network is built from end-systems, switches and Ethernet links. It is also called AFDX, for *Avionic Full Duplex Switch Ethernet*.

Ethernet links are full-duplex. Links and switches are redundant, as shown in Figure 1, width red and blue switches and links. Communications are unidirectional, from a source end-system to one or several destination end-systems. These logical communication paths are abstracted away using the notion of *virtual link* (VL), specifying the characteristics of the communication occurring on the path. Such characteristics include for instance the maximal size of a transmitted frame, and the minimal time between each frame. The configuration of the network is completely static, in order to guarantee the absence of collisions, bringing determinacy to the system.

### B. Dimensions of the Problem

The algorithm we present in this paper is meant to be used in the A664-P7 network of an airplane. The literature provides the following dimensioning parameters for such a network:

- in [9] the authors propose parameters for a realistic network: 104 end-systems, 8 switches, 974 virtual links, 6501 latency constraints. The characteristics of the network are summarized in Table I.
- in [10], Grieu describes a network proposed by Airbus, considered to be representative of the complexity of the network of the A380 airliner. The described network consists in about 50 end-systems and 8 switches. There are 500 communication links, each with between 1 and 5 destination end-systems. Most of them transport small packets of about 20 bytes. The period of emission of these packets is about 30 milliseconds. A handful of links are different and consists in larger packets (several hundreds of bytes), emitted with a periodicity of about one millisecond.
- in [11], [12], the authors provide a graphical representation of the A664 network of the A380 that we used for making Figure 1.

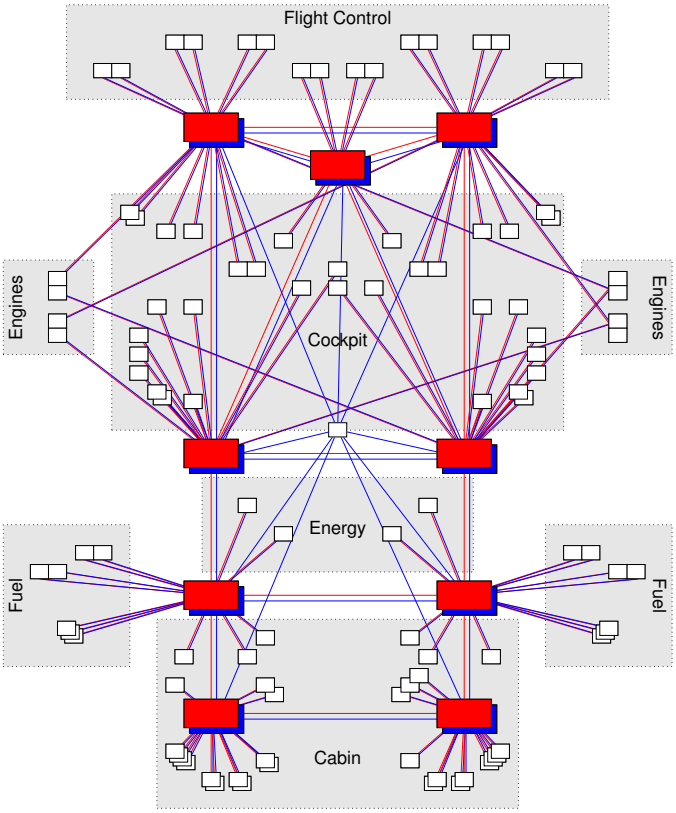


Fig. 1. Structure of the A664-P7 network of the A380

### C. Requirements for a clock synchronization algorithm

The objective of this paper is to propose a clock synchronization algorithm for the A664-P7 network. The postulate is that a plane in operation should not have to uniquely rely on an external clock to synchronize the internal end-systems of the network: the algorithm should be able to work in isolation.

In a purely distributed system, every node needs to communicate with every other node. For the number of end-systems we are considering, the number of broadcasts sent on the network will incur a high load on the equipments. In order to prevent an increase in the number of packets as the number of end-system grows, we aim at a protocol with servers delivering time to clients. In this setting, only the servers are going to synchronize in a distributed manner. To ensure robustness, there should be more than one server: these servers should therefore be able to synchronize their internal time with each others.

The clocks of two end-systems might differ in two ways: there can be a difference in the origin of the time of these two clocks, and the two clocks might not have the exact same frequency. The corrections to apply to the clocks of the end-systems to derive a common reference time depend on how this time reference is used. There are two typical use-cases:

TABLE I
DIMENSIONS OF A REALISTIC A664-P7 NETWORK

| | VL destinations | BAG | Max. Packet Size | Traversed Switches | Latency |
|---|---|---|---|---|---|
| **Minimum** | 1 | 2 ms | 100 bytes | 1 | 1 ms |
| **Average** | 6.6 | 60 ms | 380 bytes | 1.3 | 10.04 ms |
| **Maximum** | 84 | 128 ms | 1500 bytes | 4 | 30 ms |

(1) to timestamp data or messages, and (2) to measure time durations between events. If a discrepancy is found between the ideal time and the local time of the server, this discrepancy must be corrected progressively so that the order of message timestamps match the order of the messages: time cannot decrease. Similarly, a discrepancy between ideal time and physical time of end-systems needs to be adjusted slowly to keep a good precision in the measure of time durations.

It is therefore necessary to separate the time synchronization algorithm into two distinct phases:

1) an initialization phase, during which the servers can make aggressive corrections to their local clocks in order to identify the initial parameters: drift (difference in frequency) and origin of time, in order to establish the temporal reference. If an external reference time is available (a GPS for example), one can also possibly choose as servers the end-systems with the most stable physical clock.
2) an operational phase, during which corrections are applied in a smooth way in order to preserve the absolute order of dates and the precision of time durations. In this phase, a server requiring a too large correction could be considered as faulty and eliminated from the set of time servers.

In general, the requirements for the function are as follows:

**R1** The function must establish a time reference.
**R2** The function must deliver the time reference to any avionic program that needs it.
**R3** The function must function within the environment specified by ARINC 664. The redundancy of the network must be transparent to the function.
**R4** The function must recover from the dysfunction or a failure of one of the equipments it relies on.
The failure of a switch or of an ethernet link is already covered by the redundancy of the network. The function must however recover from the failure of an end-system.
**R5** The function must not require the modification of network equipment. It must be deployable on existing backends.
In particular, solutions requiring time-stamping at the level of switches are not an option.
**R6** The function must be resilient to a reboot at any time.
**R7** On the other hand, the function must be able to function without reboot for enough time to cover any kind of mission.
**R8** In operational mode, the time reference established by a server must be strictly increasing.
**R9** Each server is on its own end-system, and each server has its own VL to communicate with the other servers and with the clients.
**R10** There must be at least 3 servers, and the function must be able to run with one server failure.
**R11** the required precision must be attained no later than the end of the initialization mode.
**R12** If a server detects that another server sends faulty data, it must definitively discard the data from this server when building its temporal reference.
**R13** There must be at most one client per end-system.
**R14** Each client must receive time from at least 2 servers. It establishes its client-time from the received server-times.
**R15** The client-time must be strictly increasing.
**R16** The server precision is smaller or equal to 100 $\mu$s.
**R17** The client precision is smaller or equal to 100 $\mu$s.
**R18** The duration of the initialization mode is at most 200 ms.

The synchronization protocol that we chose for the servers is inspired from existing protocols, and in particular it is based on the Welch-Lynch algorithm, presented in subsection II-D. Our protocol is adapted to the A664-P7 architecture, in particular to the existence of two links of communication. If the bandwidth is guaranteed by A664-P7, this does not necessarily gives guarantees on the symmetry of the latency, an important factor in the estimation of the difference between local clocks.

### *D. The Welch-Lynch Algorithm*

The academic reference on distributed clocks synchronization was published in 1988 by Welch and Lynch [7]. The algorithm proposes a fault-tolerant algorithm allowing Byzantine failures: faulty devices can have arbitrary behavior. The algorithm maintains the clocks of $n$ systems in approximate synchronization provided that no more than one third of them are faulty.

The algorithm is periodic with period $P$. The behavior is quasi-synchronous, and all nodes behave the same (there is no distinction between slave and master, or client and server).

- For each time $T = k \times P$: broadcast a SYNC packet to each of the other nodes.
- Wait a time $\Delta$. Compute the correction to apply using the mean of the reception time of the SYNC packets from the other nodes.
- Continuously, in parallel: note the emission and reception time of each received SYNC packet.

The SYNC packets are not timestamped: the date is implicitly taken as $T = k \times P$. The parameters $P$ and $\Delta$ are fixed and chosen depending on the desired properties of the system: jitter, drift, *etc*.

The algorithm ensures the following properties:

- Agreement. The difference between the clocks of two non-faulty nodes in the network is bounded.
- Validity. The clocks of non-faulty nodes are within a linear envelope of the real time.

## III. Our Proposal

The function is composed of clients, servers and controllers. Servers and clients are partitions in the sense of ARINC-653. Servers are in charge of establishing and keeping a common time reference, and to deliver it to clients. The role of a client is to give access to the time reference to programs running on the same end-system. For this, the clients receive information about the time reference from the servers. Controllers are in charge of the command of the function and the management of failures. There is always a client and a controller on an end-system. Servers are only present on a limited number of end-systems. The typical number of servers is 4, and 3 is a minimum. A typical network contains about a hundred of end-systems. Since the number of servers is small, each server will broadcast its information to all the clients.

Communication between clients and servers is done using VLs. Each server communicates with every other server to establish and maintain the time reference, using one particular VL. The same VL is used to broadcast the time reference to the clients. Finally, the communication between a client and its controller is done using the functionalities of the operating system running on the end-system.

To minimize the dispersion of transmission delays, the packets used to construct, maintain and communicate the time reference through VLs have a fixed length. They consists of:

- A versioning number and the type of the packet: INIT or TIME. Each information is coded on 4 bits.
- A notification for the controller, coded on one byte.
- A date: the time reference of the server sending the packet, coded on 8 bytes.

The rationale for the date is as follows. We do not use floating-point numbers because of the accumulation of rounding errors. 32 bit integers can store a bit more than 71 minutes counted in micro-seconds, which is not enough. 64 bit integers can store more than 290 years in nano-seconds, which is enough for our needs, hence the 8 bytes for the time reference.

Because the network is very deterministic, transmission delays between servers, and between servers and clients can be characterized with some precision. This is statically estimated from the network architecture and it is mainly based on the number of switches the transmission goes across.

Note that for a client (and for a server), there are several times to be kept in parallel:

- The time reference, inferred from the data received from the servers (`currentTime`).
- The time given by the hardware clock of the end-system (`localTime`).
- The "real" time, to which no one has access.

### A. Operating Modes

*1) Initial Mode:* In this mode, each server sends INIT packets to signal that it is in this mode. The sending period of these messages is noted `serverActivationPeriod`.

A server leaves the initial mode when it has received either `numberOfServers` $-$ 1 INIT packets (in this case the whole function was in initial mode), or `numberOfServersQuorum` TIME packets (in which case the whole function is in operational mode). In the case where both conditions are true, the second one prevails. Each server estimates the reference time on the other servers from the time reference it receives from them, the local time at which it received their packet, and the estimated transmission time.

In the case where the function is in the initial mode, each server in initial mode sets its reference time when it switches to the operational mode to the maximum value of its current time and the estimated times of all other servers.

In the case where the function is in the operational mode, a server in initial mode sets its reference time when it switches to the operational mode to the average of the estimated times of all the other servers.

In initial mode, each client waits to receive `numberOfServersQuorum` TIME packets before entering operational mode, where the client will be activated with a `clientActivationPeriod` period.

*2) Operational Mode:* A server in operational mode sends TIME packets to maintain and broadcast the time reference. As in initial mode, these messages are send periodically every `serverActivationPeriod`.

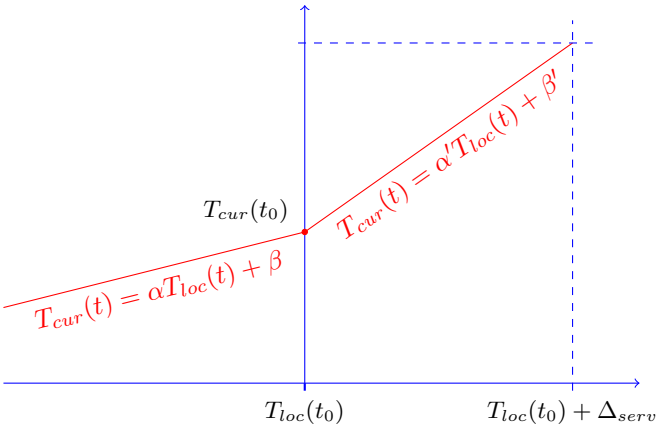


Fig. 2. Computation of `currentTime` ($T_{cur}$)

To meet the R8 requirement (in operational mode, the `currentTime` of a server is strictly increasing) and R15 (the client `currentTime` must be strictly increasing), the `currentTime()` function, which computes the current reference time for a server or client `x` from its local time, is piecewise linear with an always positive slope. There are two parameters, `coefficient` (the slope) and `offset` (the intercept) that are updated at each activation and then kept constant during one period as illustrated in Figure 2, on which, for the sake of brevity, `x.currentTime` is $T_{cur}$, `x.localTime` is

$T_{loc}$, and the parameters `x.coefficient` and `x.offset` are $\alpha$ and $\beta$. $\Delta_{serv}$ is the `serverActivationPeriod`.

### B. Algorithms

Each server and client have some attributes and methods:

- `localTime()`: the time provided by the hardware clock
- `offset`: the current offset used by `currentTime()`
- `coefficient`: the current coefficient used by `currentTime()`
- `currentTime()`: the reference time computed as shown in Figure 2.

Some constants are needed:

- `numberOfServers`: the total number of servers.
- `numberOfServersQuorum`: the number of operational servers needed for the function to be operational.
- `serverActivationPeriod`: the period of activation of the servers.
- `clientActivationPeriod`: the period of activation of the clients.
- `maximumTimeDifference`: the maximum accepted difference between the current time and the time estimated from packet timestamps.
- `minimumDelay(s,x)`: the minimum transmission delay between a sender `s` and a destination `x`.

Each packet received by a server or client end-system is stored with its time of arrival.

*1) Algorithm for servers in initial mode:* When a server is booting, or re-booting, two cases are possible:

- The whole function might be starting. In this case, the function is in initial state and the servers must establish the time reference.
- The whole function has already (re)booted. In this case, the server must acquire the current reference time from the other servers.

The algorithm for servers in initial mode is as follows.

*At (re)boot*:

- Initialization of the current time at 0:
  $\texttt{coefficient} = 1.0$
  $\texttt{offset} = -\texttt{localTime}()$

*Then, with period `serverActivationPeriod`.*

(a) Get and send to the controller all packets received since the last activation phase on the command VLs.
(b) Get all packets received since the last activation phase on the synchronization VLs.
(c) Discard all packets coming from links marked as to be ignored.
(d) Transmit to the controller the notifications of the received packets.
(e) Inform the controller of the synchronization VLs from which the server received either more than one packet, or a packet of unknown type (neither INIT nor TIME) or with a bogus timestamp.
(f) Execute the commands received from the controller. Commands can deal with the configuration (several are allowed simultaneously):
  - `DISABLE vl_id`: ignore packets coming from the given VL.
  - `ENABLE vl_id`: take into account the packets coming from the given VL.

  Other commands control the execution (only one is allowed at each activation period):
  - `RESTART`: continue (or restart after `STOP`) the complete execution of the algorithm.
  - `STOP`: stop sending packets and stay in initial mode without modifying the parameters.
  - `TERMINATE`: definitely terminate and stop interacting with the controller.
(g) Send an INIT packet containing `currentTime()`.
(h) Finally, two cases:
  - if `numberOfServersQuorum` TIME packets have been received since (re)boot and the function is in operational mode, for each TIME packet containing the timestamp $\texttt{h}_\texttt{i}$ received at local time $\texttt{t}_\texttt{i}$, we estimate the current time of the server $\texttt{s}_\texttt{i}$ which sent the packet as:

$$\begin{aligned} \texttt{t}_{\texttt{s}_\texttt{i}} = \texttt{h}_\texttt{i} \quad &+ \quad \texttt{minimumDelay}(\texttt{s}_\texttt{i}, \texttt{s}) \\ &+ \quad \texttt{localTime}() - \texttt{t}_\texttt{i} \end{aligned} \tag{1}$$

    The offset of the server is set to the *mean* of these estimations, minus the local time. Only the last packet received from a given server is kept.
  - if $\texttt{numberOfServers} - 1$ INIT packets have been received since (re)boot and the function is in initial mode, the algorithm is the same as in operational mode, but the offset of the server is set to the *max* of the estimated times instead of the mean.

  Ultimately, in both cases the server goes to operational mode at the next activation.

*2) Algorithm for servers in operational mode:* : every `serverActivationPeriod`, the server performs the following tasks:

(a) some housekeeping: same as items (a) to (e) in the initial mode algorithm.
(b) Inform the controller if less than $\texttt{numberOfServers} - 1$ TIME packets with valid timestamps have been received.
(c) Execute the commands received from the controller. These are the same as in item (f) in the initial mode algorithm.
(d) Update internal parameters with the received TIME packets. First, compute the current reference time `referenceTime` as the *mean* of the local current time and the estimated current times of the other servers (which are computed with Equation 1).
(e) Update the `coefficient` and `offset` parameters of the server according to Equations 2, 3, and 4 below, and send a TIME packet containing `currentTime()`.

$$\texttt{newCoefficient} \quad = \quad 1 + \frac{\texttt{referenceTime} - \texttt{currentTime}()}{\texttt{serverActivationPeriod}} \tag{2}$$

$$\texttt{offset} \mathrel{+}= \texttt{localTime}() \times (\texttt{coefficient} - \texttt{newCoefficient}) \quad (3)$$

$$\texttt{coefficient} = \texttt{newcoefficient} \quad (4)$$

*3) Algorithm for clients in initial mode:* During function initialization, clients acts as if they had received a `STOP` command from the controller. Each client waits for a `RESTART` command to execute its algorithm in nominal mode.

Then, with a period of `clientActivationPeriod`:

(a) Gather all packets received since the last activation.
(b) Discard packets from links to ignore and packets of type INIT.
(c) Notify the controller of VLs with bogus behaviors: more than one packet received, unknown packet type, or negative timestamp.
(d) Execute configuration and execution commands. These are the same as for servers.
(e) If `numberOfServersQuorum` TIME packets have been received:
- Compute `offset` and `coefficient` as for servers in initial mode (see III-B1).
- Notify the operating system that the time reference service is available.
- Switch to operational mode at the next activation period.

*4) Algorithm for clients in operational mode:* With period `clientActivationPeriod`:

(a) Perform the housekeeping items (a) to (c) of the previous algorithm.
(b) Inform the controller if less than `numberOfServers - 1` TIME packets with valid timestamps have been received.
(c) Execute configuration and execution commands.
(d) Update internal parameters with the received TIME packets. First, compute the current reference time `referenceTime` as the mean of the estimated times of the servers using the formula in Equation 1. Then update the `offset` and `parameters` according to equations 5, 6, and 7 below.
In this step the difference between a server and a client is that the latter does not consider its current time when computing the mean of the estimated times of the servers.

$$\texttt{newCoefficient} = 1 + \frac{\texttt{referenceTime} - \texttt{currentTime}()}{\texttt{clientActivationPeriod}} \quad (5)$$

$$\texttt{offset} \mathrel{+}= \texttt{localTime}() \times (\texttt{coefficient} - \texttt{newCoefficient}) \quad (6)$$

$$\texttt{coefficient} = \texttt{newcoefficient} \quad (7)$$

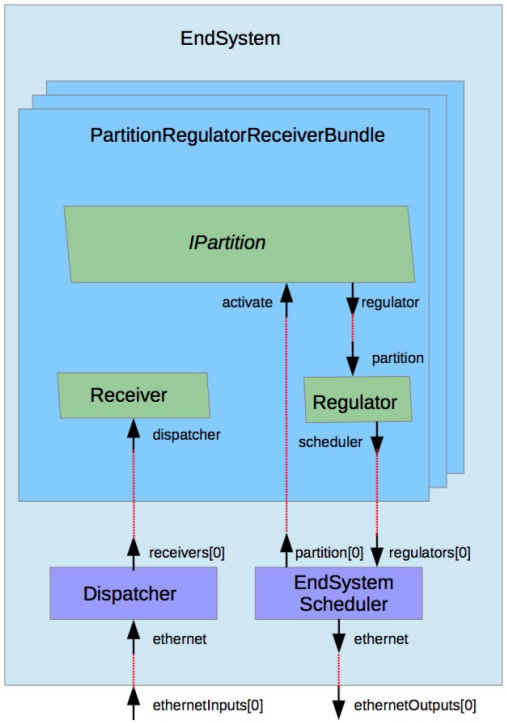


Fig. 3. A664 EndSystem

## IV. Experiments

In order to validate our proposal, we modeled A664-P7 and the proposed time reference function within OMNeT++. In this section, we will first describe the OMNeT++ models implementing our proposed reference time service (including servers, clients, network and protocols). Then, we will detail the results of experiments on a realistic network instance.

### A. Setup

*A664 architecture:* An A664 system consists of nodes (`A664Node`) interconnected by Ethernet links. A node is either a terminal device (EndSystem) or a switch.

As shown in Figure 3, a node has one or more Ethernet ports, this parameter is noted `numberOfEthernetPorts`. A terminal device has a value of 1 for this parameter. We distinguish input ports (`ethernetInputs`) from output ports (`ethernetOutputs`). For each input port, a node has a `Dispatcher` responsible for processing an incoming packet on the Ethernet link connected to this port. Similarly, for each output port, a node has a scheduler (the OMNeT++ *moduleinterface* : `IScheduler`) responsible for sending the packets on the Ethernet link connected to this port.

A terminal device has a unique scheduler (`EndSystemScheduler`), a local clock (*moduleinterface* `IClock`) and 0 or more modules (`PartitionRegulatorReceiverBundle`) including:

- A *moduleinterface* `IPartition` in charge of manipulating packets.

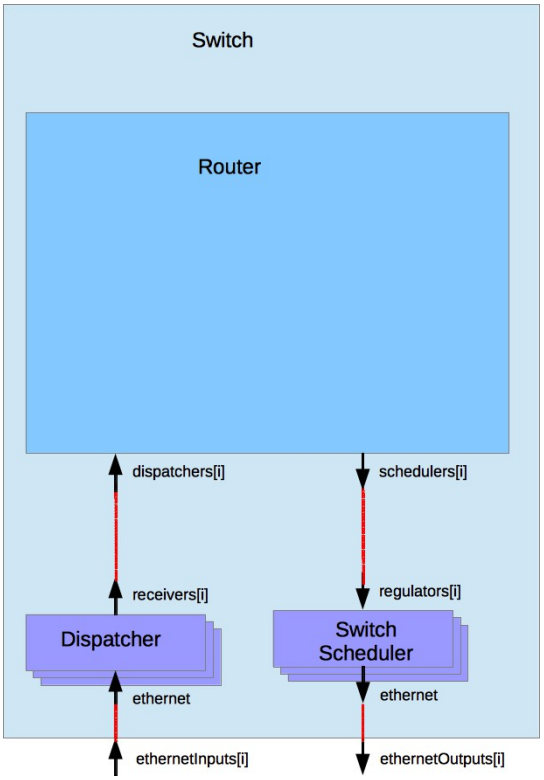


Fig. 4. A664 Switch

- A `Regulator` which is connected to the scheduler and which manages the virtual link used for the packets sent by this partition.
- A `Receiver` which is connected to the `Dispatcher` and which filters packets according to the virtual link numbers (parameter `virtualLinkNumbers`). The receiver is responsible for timestamping the incoming packets and to keep them until the corresponding partition is activated.

In order to allow a partition to transmit without delay, the scheduler of the A664 end-system determines at startup a static scheduling of the use of the Ethernet link by the virtual links. Hence, we can anticipate and prepare a time packet embedding the exact departure time to avoid a latency that would weaken the precision.

Packets `A664Packet` exchanged in an A664 network carry the corresponding virtual link number and priority.

As illustrated in Figure 4, a switch has specific schedulers (`SwitchScheduler`), and a router that, depending on a routing table, will route the packets (and duplicate them if necessary). Such schedulers take into account the priorities of the virtual links to decide which packet to send first. However, they are not preemptive: once the sending of a packet has started, it will not be preempted for sending a packet of higher priority.

*Clocks:* Each terminal device has a local clock. Three clocks were implemented:

- `PerfectClock` is a perfect clock (it provides the simulation time of OMNeT++), without parameters.
- `FixDriftClock` is a clock with a fixed drift; this is determined randomly at startup using the `driftMin` and `driftMax` parameters. A reasonable `driftMax` value is 50 ppm (parts-per-million).
- `ChangingDriftClock` is a clock with a drift that dynamically changes according to the period indicated by the `changeDriftPeriod` parameter; the chosen value is random between the `driftMin` and `driftMax` parameters.

*Time reference function:* A specific type of packets, `TimeReferencePacket`, subtype of `A664Packet` is used by the protocol. It has a version number (on 1 byte), a type (on 1 byte) and a time represented by the OMNeT++ type `simtime_t` on 8 bytes.

A time reference server is modeled by a `TimeReferenceServerPartition` that is responsible for computing and giving access to the server's current reference time based on received packets. It is also responsible for managing the state of the server (and therefore the type of packets sent). A time reference client is modeled by a `TimeReferenceClientPartition`. Each partition has a `minimumDelays` table to get the minimum transfer time of a packet coming from another server.

To simulate a reboot of a server, the corresponding partition uses `minimumResetTime` and `maximumResetTime` parameters in order to, if they are not zero, compute a random time after which the partition restarts. Also, for simulating a server failure, the partition uses the `minimumFreezeTime` and `maximumFreezeTime` parameters in order to, if they are not zero, compute a random time at which the partition will send packets containing the same time value.

Finally, `RandomTransmittingPartition` was implemented for a random broadcast of packets. Each packet is sent with a probability set by the `frequency` parameter, and its size is determined randomly using the `sizeMin` and `sizeMax` parameters .

*A380 simulation:* The simulations and corresponding measurements are performed on a network similar to the network of the A380 introduced in Figure 1. It consists of 9 interconnected switches with 2 or 3 end-systems on each switch; each of these devices is either a time reference client, a time reference server (`ts_1`, `ts_2`, `ts_3` and `ts_4`) or an end-system with a `RandomTransmittingPartition`. The connections are 100 Mb/s Ethernet links. The precision of the simulation time was configured to one nanosecond.

The `mbl_2` device has 7 additional random broadcast partitions, making a total of 8. These 8 partitions are configured with a maximum packet size and minimum periodicity. The simultaneous activation of the 8 partitions almost saturates an Ethernet link of 100Mb/s. The virtual links coming from this device are such that this saturation influences the exchanges between servers `ts_2` and `ts_1`, increasing transmission delays. The different simulation configurations we ran, activated 0, 2, 4, 6 or 8 of these partitions in order to simulate the different load cases of the network. The diagram in Figure 5

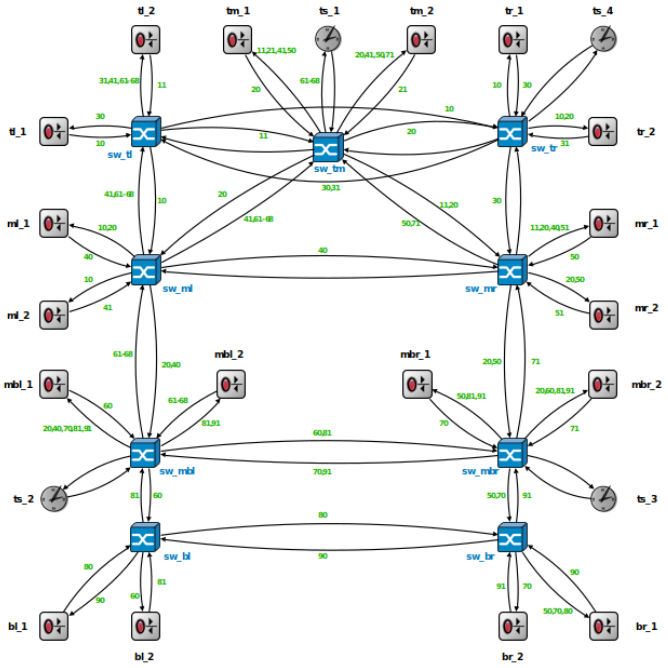


Fig. 5. A380 simulation

shows the whole structure together with the virtual links used by the random broadcast partitions.

### B. Results

We experimented many configurations for which we measured, every 10 ms:

- `referenceTimeAverage`: the reference time that is the average of all servers corrected current times.
- `referenceTimeMaximumDifference`: the maximum difference between servers current times, this precision measurement will be drawn in blue in the following graphs.
- `clientsWorstPrecision`: the biggest difference between a client corrected time and `referenceTimeAverage`, drawn in red.

The activation period for server and client partitions is 128ms. The `maximumTimeDifference` parameter is set to 1ms. The time packets have a size of 30 bytes, their transmission on a 100Mb/s Ethernet link lasts 2.4 $\mu$s.

*Perfect clocks with random traffic:* In this first configuration, all servers and clients have perfect clocks. We activate the random transmission partitions of all terminal devices (except partitions 2 to 8 of `mbl_2` which would completely saturate the network). Time packets are sometimes delayed, despite their high priority.

In this case, Figure 6 reveals an excellent precision of 3 $\mu$s for servers and 7 $\mu$s for clients.

*Fix drift clocks with random traffic:* We keep a random traffic but we consider that servers clocks have fix drifts. The clock of `ts_1` has a drift of -40ppm, `ts_2` of -50ppm, `ts_3` of 30ppm and that of `ts_4` is 10ppm.

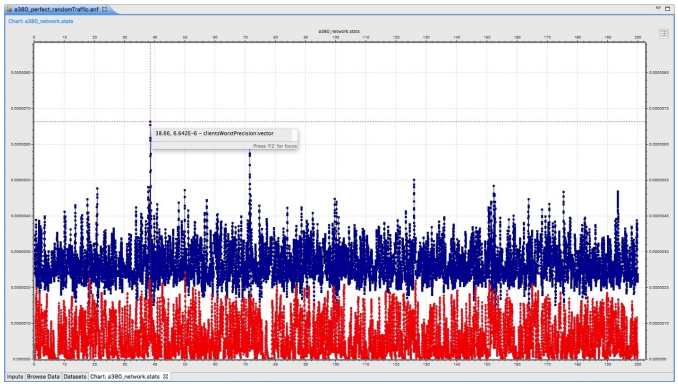

Fig. 6. Perfect clocks with random traffic

From Figure 7, we notice that because of the servers clocks drifts, we loose much precision. In this case, we have a precision of 120 $\mu$s for servers and 90 $\mu$s for clients.

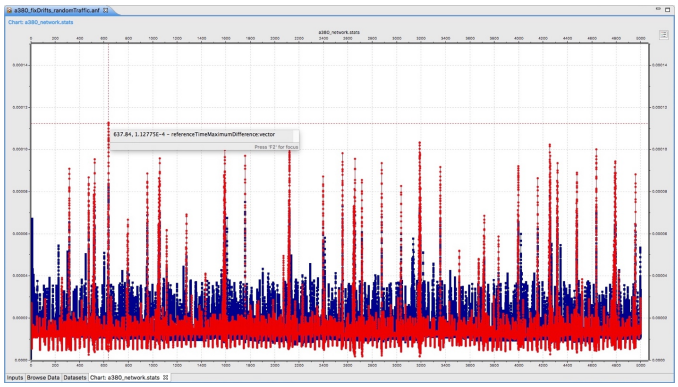

Fig. 7. Fix drift clocks with random traffic

*Fix drift clocks with medium traffic:* We keep Fix drift clocks and we activate 4 of the 8 partitions of `mbl_2` creating a 50% load on the corresponding virtual link. In such a configuration a time packet can be delayed by 350 $\mu$s. From Figure 8, we observe that the chaotic aspect introduced by the traffic dominates the one of drifts. In this case, we have a precision of 250 $\mu$s for servers and 200 $\mu$s for clients.

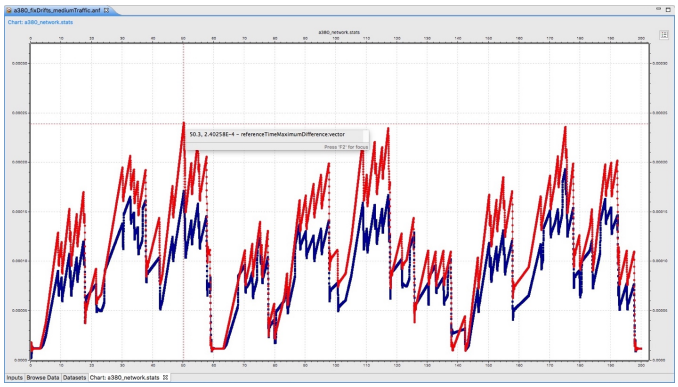

Fig. 8. Fix drift clocks with medium traffic

*Fix drift clocks with max traffic:* We activate all transmitting partitions of `mbl_2` to saturate the network. This configuration must give the worst case precision measurements. From Figure 9, we observe that the precision is unchanged, with a worst case precision of 250 $\mu$s for servers and 200 $\mu$s for clients. However, it is clear that the average precision is worse.

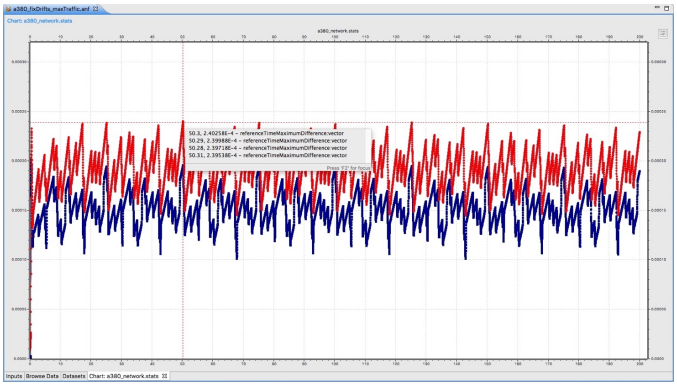

Fig. 9. Fix drift clocks with max traffic

## V. Discussion

*1) Worst case precision:* The precision of 100 $\mu$s cannot be reached in the worst case on a 100 Mb/s bus due to the jitter variations in packets exchanged by servers. Indeed, a packet received by a switch may be sent with a fixed and known processing time to next or last node. But, even with a high priority, it may also have to wait when an other packet is already being transmitted on the link to the destination. With a maximum A664-P7 frame size equal to 1538 bytes, it means that the insertion of a TIME packet on an ethernet link by a switch, even if it is the only one with a high priority, may be postponed up to 123.04 $\mu$s on a 100 Mb/s ethernet link. Considering a case where all packets received by a time server have been delayed by around 120 $\mu$s for each switch passed through, it is clear that this particular time server will compute its new reference time with an error well above 100 $\mu$s, the exact computation of this error being only possible on a known network configuration.

*2) Improvements:* The version of the algorithm presented in this article computes the slope and intersect of the `currentTime()` function in order to fix the current time as fast as possible. However, this can lead to wrong fixes when a packet has been delayed in the switches and is older than we believe. This can also lead to a "vibrating" `currentTime()` function which alternates between a large and a small slope.

To solve this issue, we developed a version of the algorithm in which the slope and intersect are computed using linear regression over the last $k$ steps. For $k = 1$, there is no change, the new slope and intersect are computed from the previous ones. When $k$ is too large, the function is too slow to fix the current time. However, for $k$ around 5, the average precision is improved. For instance, on a 1 Gb/s network, the average precision went from 11.7 $\mu$s to 4.2 $\mu$s with linear regression. In the same time, the worst case precision was only improved from 34 $\mu$s to 24 $\mu$s. Another improvement consists in computing a first linear regression, then computing it again while ignoring the data from the servers that are too far from the first computed line. This eliminates packets that have been delayed too much, or which come from servers with an altered current time. This improved the average precision to 0.2 $\mu$s in our case, but it did not change the worst case precision.

This improvement was not integrated in the final version of the function because it made the algorithm more complex without much improving of the worst case precision, and average precision was not a primary goal. However, it may be useful in a less critical context where a better average precision is desired.

## References


[1] D. L. Mills, *Computer network time synchronization. The network time protocol on earth and in space. 2nd ed.* CRC Press, 2011.

[2] IEEE, "Ieee standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. 1–300, July 2008.

[3] R. Holler, T. Sauter, and N. Kero, "Embedded synutc and ieee 1588 clock synchronization for industrial ethernet," in *EFTA 2003. 2003 IEEE Conference on Emerging Technologies and Factory Automation. Proceedings (Cat. No.03TH8696)*, vol. 1, Sept 2003, pp. 422–426.

[4] A.-D. D. Ethernet and U. N. committee, *Time-Triggered Ethernet AS6802*. SAE International, NOV 2016.

[5] A.-D. T.-T. F. committee, *TTP Communication Protocol AS6003*. SAE International, AUG 2011.

[6] F. Consortium, *FlexRay Communications System Protocol Specification V 3.0.1*. FlexRay Consortium, OCT 2010. [Online]. Available: https://svn.ipd.kit.edu/nlrp/public/FlexRay/FlexRay%E2%84%A2%20Protocol%20Specification%20Version%203.0.1.pdf

[7] J. L. Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, Apr. 1988. [Online]. Available: http://dx.doi.org/10.1016/0890-5401(88)90043-0

[8] S. I. Kazi, "Architecting of avionics full duplex ethernet (afdx) aerospace communication network," *IJECT*, vol. 4, no. 4, April-June 2013. [Online]. Available: http://www.iject.org/vol4/spl4/c0140.pdf

[9] M. Boyer, L. Santinelli, N. Navet, J. Migge, and M. Fumey, "Integrating end-system frame scheduling for more accurate afdx timing analysis," in *Proceedings of the 6th Embedded Real Time Software and System Congress (ERTS2 2014)*, 2013.

[10] J. Grieu, "Analyse et évaluation de techniques de commutation ethernet pour l'interconnexion des systèmes avioniques," Ph.D. dissertation, Institut National Polytechnique de Toulouse, SEP 204.

[11] H. Butz, "The airbus approach to open integrated modular avionics (IMA): Technology, methods, processes and future road map," in *First International Workshop on Aircraft System Technologies (AST 2007)*, 2007.

[12] J.-B. Itier, "A380 integrated modular avionics," 2007. [Online]. Available: http://www.artist-embedded.org/docs/Events/2007/IMA/Slides/ARTIST2_IMA_Itier.pdf