Evaluating Requirements of High Precision Time Synchronisation Protocols using Simulation

Lazar T. Todorov, Till Steinbach, Franz Korf, Thomas C. Schmidt
HAW-Hamburg, Department Informatik
Berliner Tor 7, D-20099 Hamburg, Germany
{lazartodorov.todorov, till.steinbach, korf, schmidt}@informatik.haw-hamburg.de

ABSTRACT

High precision time synchronisation protocols are used in distributed real-time systems such as trains, planes, cars or industrial installations. In time-triggered systems, with a coordinated time division multiple access media allocation strategy, the achievable precision of time synchronisation among sending participants determines the quality of communication and the available bandwidth. The simulation of time synchronisation protocols allows to find problems at the earliest time – in general, during the design and configuration – of a synchronised distributed system.

In this work we show a concept for the simulation of distributed real-time synchronisation protocols that uses discrete event-based simulation. Our model for the OMNeT++ Framework is adaptable and thus allows for providing highly accurate results or fast simulations. The precise simulation of a real-time synchronisation protocol usually consumes considerable simulation time. This paper presents an approach to speed up accurate simulation, based on recordings of previous runs. We evaluate typical real-world use cases for the introduced concept by simulating the AS6802 standard for time synchronisation. Our results show that the simulation can help to reduce the effort of determining configuration parameters for clock synchronisation protocols. We further quantify the performance increase of our evolutionary approach.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; C.2.2 [**Computer-Communication Networks**]: Network Protocols—AS6802; C.4 [**Performance of Systems**]: Fault tolerance

General Terms

Measurement, Performance, Reliability

Keywords

Network Synchronisation Simulation, AS6802, OMNeT++

OMNeT++ 2013 March 5th, Cannes, France.

1. INTRODUCTION

High precision time synchronisation protocols are of vital importance for most distributed real-time systems. Especially for systems that communicate time-triggered, with a coordinated time division multiple access media allocation strategy, the achievable precision of the time synchronisation among sending participants determines the quality of communication and the amount of bandwidth that is usable. Most of the currently utilised protocols use a hierarchical synchronisation model with synchronisation masters and clients. For failsafe operation they provide redundancy or failure recovery strategies. Although most synchronisation protocols are verified, the simulation is important for the design and configuration of a synchronised distributed system. It covers aspects of the utilised hardware that influence the time from start-up until the synchronisation is stable or the precision of the globally established time base.

In this work we provide a concept for the simulation of distributed real-time synchronisation protocols using discrete event-based simulation in OMNeT++. We provide a model for the simulation of distributed clocks that is adaptable, and thus allows us to provide highly accurate results or fast simulations. Based on the simulation of the AS6802 standard for the time-synchronisation in OMNeT++ [9] we give typical real-world use cases for the introduced concept.

Simulating a distributed real-time synchronisation protocol consumes a lot of simulation time. Thus we introduce a so called evolutionary approach that divides the model of the clock synchronisation and the model of the network technology itself. This allows us to record the time-consuming simulation of clock drift and synchronisation. Based on the recordings following simulations can be significantly accelerated. The approach works as follows: During the first simulation of a network the results of the real-time synchronisation protocol are recorded. Subsequent simulation runs of the same network exclude the real-time synchronisation. Instead a modified clock model is being used: Based on the results of the first simulation it emulates the clock behaviour similar to a simulation that included the real-time synchronisation protocol. The synchronisation results of the first run can be divided into independent and self-contained parts. To provide a wide range of time behaviour the clock model will link up these subparts at random. This kind of speed up is in particular useful in situations where the simulation must run in real-time [7].

Time synchronisation protocols, which provide accuracy below 10 µs, require coordination of hardware and software components. For example the IEEE 1588 Precision Time

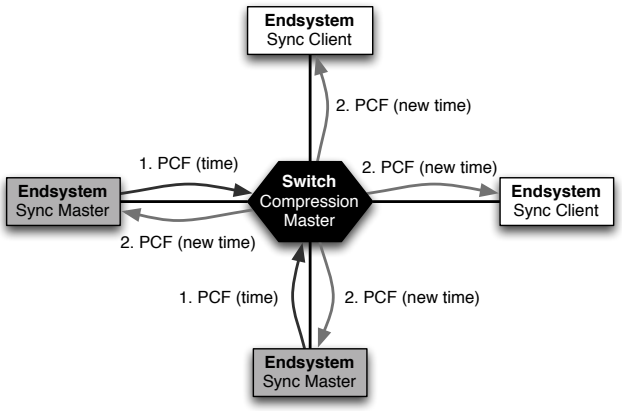

Figure 1: Two step synchronisation approach: Protocol Control Frames of type IN are sent by the synchronisation masters to the compression master, the compressed time is then calculated and propagated to all of the systems participants.

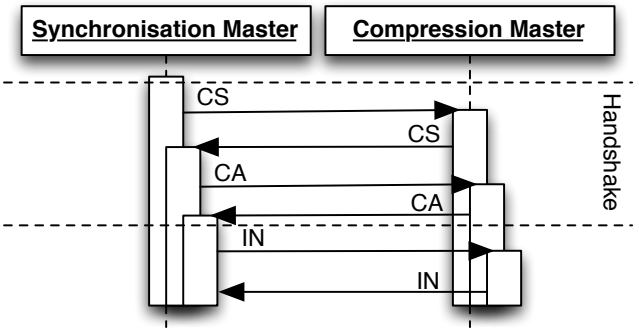

Figure 2: Startup sequence: After initial synchronisation has been done by the handshake protocol, the synchronisation master sends the first integration frame to the compression master. The compression master generates a new integration frame in response and sends it back to the synchronisation master.

Protocol (PTP) [3] timestamps the protocol relevant frames in hardware while the protocol itself can be implemented at the network layer of the protocol stack. This paper shows how event-based simulation can be used to discover which time synchronisation accuracy can be achieved for a given hardware and synchronisation protocol. Vice versa for a given synchronisation protocol and accuracy requirements a simulation based approach can extract the corresponding hardware requirements.

Safety critical real-time systems are a typical field of application for time-triggered systems. Such systems must handle rare events and alarm floods that might lead to bandwidth peaks. Due to the safety requirements, redundancy must be provided by the time synchronisation protocol. Hence failover situations must be analysed.

The paper is organised as follows: In Section 2, we introduce AS6802 time-synchronisation and present preliminary and related work. Section 3 presents the concepts and architecture of the developed clock and synchronisation model. Based on an evaluation the use-cases are shown in Section 4. Finally, Section 5 concludes our contribution and gives an outlook on future research.

2. BACKGROUND & RELATED WORK

There are several clock synchronisation protocols for distributed real-time systems. In this work we show our simulation concept based on the AS6802 clock synchronisation protocol for time-triggered Ethernet (TTEthernet).

2.1 AS6802 Clock Synchronisation Protocol

The TTEthernet (AS6802) specification [11] was standardised in 2011 by the Society of Automotive Engineers (SAE) [12]. It is a compatible extension of IEEE 802.3 standard switched Ethernet [4] and uses topologies formed of full-duplex links. For supporting time-triggered (TT) communication TTEthernet defines a fault-tolerant master-slave time synchronisation protocol for establishing and maintaining a synchronised time in the network.

The synchronisation of local clocks is done in a periodic manner. This routine is called *integration cycle*. The synchronisation is realised by periodic exchange of special Ethernet frames between all devices at the beginning of each integration cycle. These frames are called *Protocol Control Frames* (PCF). The Protocol defines three types of PCF frames: *coldstart frame* (CS), *coldstart acknowledge frame* (CA) and *integration frame* (IN). Depending on the system timing requirements, each device can be configured as *synchronisation client* (SC), *synchronisation master* (SM) or as *compression master* (CM). Typical network configurations use a switch as compression master and end systems as synchronisation masters or clients.

To keep the local clocks synchronised, AS6802 specifies a two-step synchronisation protocol, as depicted in Figure 1. First, each synchronisation master sends an IN frame to all compression masters. Each compression master calculates an average value from the relative arrival times of these IN frames and corrects its local clock by this value. In the second step each compression master sends an IN frame with its new local time to all synchronisation masters and clients. Based on the received IN frames all synchronisation masters and clients calculate a clock correction value, which is added to the local clock.

For initial synchronisation at startup time or during the restart of devices, AS6802 defines a fault-tolerant handshake protocol (see Figure 2). After power on, each synchronisation master sends a coldstart frame (CS) to the compression master. The compression master relays the CS frame back to the synchronisation masters. After reception of the CS frame the synchronisation master sends a coldstart acknowledge (CA) frame to the compression master. The compression master relays the CA frame back to the synchronisation masters. After reception of a CA frame a synchronisation master waits for preconfigured timeout duration. Then the synchronisation master sends integration (IN) frames to the compression master according to the synchronisation approach given above (see Figure 1). Synchronisation clients are not involved in this initial synchronisation sequence. As shown in figure 1 they only receive IN frames from a compression master to correct their local clock.

To understand the influence of clock drift on the time synchronisation quality, a closer look on the synchronisation

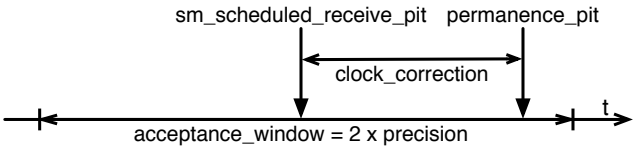

Figure 3: Clock correction in synchronisation master and client.

protocol is necessary. For simplicity we consider the clock synchronisation of master and client only. To re-establish the dispatch order of messages and to provide remote clock reading during synchronised operation, AS6802 defines the *permanence point in time* concept (PPT), which is based on a transparent clock mechanism. All network devices that impose a delay on the transmission, reception, or forwarding add their delay to the *pcf_transparent_clock* field in the PCF. The worst case delay (*max_transmission_delay*) between any synchronisation master (SM) and the compression master (CM) can be calculated offline. The *precision* is a key synchronisation parameter that is determined offline. It defines the worst-case deviation of any two correct clocks in the network. *max_transmission_delay* and *precision* are defined for the whole network.

After reception of a PCF each device calculates the *permanence_delay* according to Equation 1:

$$\begin{aligned} permanence\_delay = max\_transmision\_delay \\ -pcf\_transparent\_clock \end{aligned} \quad (1)$$

The PCF is delayed for this time. Afterwards the *permanence point in time* (PPT) is reached (see Equation 2):

$$permanence\_pit = receive\_pit + permanence\_delay \quad (2)$$

To re-establish the dispatch order of PCFs the PPT calculation is done for every PCF. The offset between any two PPTs is equal to the offsets between the send points in time of the corresponding PCFs. Thus, the PPT concept is used for remote clock reading and calculation of clock correction offsets in all devices.

During the first step of the synchronisation protocol, each synchronisation master (SM) sends a PCF (IN frame) to the compression master (CM) (see Figure 1). Based on the PPTs of all PCFs the CM corrects its local clock according to the local clocks of all SMs. The new local time is utilised by the CM to calculate the dispatch point in time of the PCF that will be send back to all SMs. For all SMs the receive point in time (*sm_scheduled_receive_pit*) is calculated offline as follows:

$$\begin{aligned} sm\_scheduled\_receive\_pit = \\ 2 * max\_transmission\_delay \\ +compression\_master\_delay \end{aligned} \quad (3)$$

An SM opens an acceptance window of size $2 \times precision$ around *sm_scheduled_receive_pit* and monitors the PPT of the PCF signalled within this window (see figure 3). Since *max_transmission_delay* and *compression_master_delay* are constant, the position of the PPT within the acceptance window depends on the dispatch point in time of the PCF defined by the CM. A PCF with a PPT outside this window points to a SM error and will be forwarded to an error handling mechanism.

The *clock_correction* offset is calculated as follows:

$$\begin{aligned} clock\_correction = permanence\_pit \\ -sm\_scheduled\_receive\_pit \end{aligned} \quad (4)$$

The *clock_correction* values are added to the *local_clock* variable at the offline configured point in time.

From the equations shown above, we can conclude that in an ideal environment without considering the effect of rounding and digitalisation errors, the quality of synchronisation directly depends on the accuracy that can be achieved with the hardware devices.

2.2 Related Work

Praus et. al. [10] introduced a framework for the simulation of fault tolerance in the clock synchronisation of industrial automation networks. Similar to our approach, their implementation of the protocol is based on the OMNeT++ INET Framework. The model divides the clock synchronisation in software and hardware parts where separate modules are used for each component.

Liu and Yang showed a simulation of the IEEE 1588 PTP synchronisation protocol [3] in OMNeT++ [8]. The authors conclude that the simulation can show the synchronisation performance and time to converge. Giorgi and Narduzzi [2] use an OMNeT++ based model to assess the suitability of IEEE 1588 PTP in relation to network topologies and operating conditions.

Ferrari et. al. [1] analysed the synchronisation accuracy of concurrent real-time synchronisation protocols in simulation. The work shows that the simulation can make the tradeoff between performance reduction and the sharing of the same physical infrastructure visible and allows for tuning of network parameters.

3. CONCEPT & ARCHITECTURE

One key element for the simulation of a distributed synchronisation algorithm is the model of the local clock. It defines the accuracy of the simulation results while at the same time majorly influences the simulation performance.

The most important attribute of a clock is the durability of the length of each clock tick. All clocks have a certain inaccuracy called frequency drift [5,14]. Frequency drift may occur due to the physical architecture of the clock or due to environmental conditions, in particular temperature changes. These drifts may have significant impact on the overall synchronisation behaviour and must be carefully included in the clock model.

An accurate model of a local clock would model the drift by simulating each tick of the clock as a separate event. Such a model allows to adapt the tick length directly related to the simulated environmental conditions and provides the most realistic simulation results. At the same time, this approach extraordinarily slows down the simulation, as it requires to simulate a significant amount of events that are not directly associated with the communication protocol.

The International Telecommunication Union defines *frequency stability* as the frequency change within a given time interval caused by spontaneous and/or environmentally influence [5]. According to the frequency stability a fast clock model simulates several ticks in one step. The frequency stability or clock drift is then simulated by a drift factor that is assumed constant for a configurable interval. Equation 5

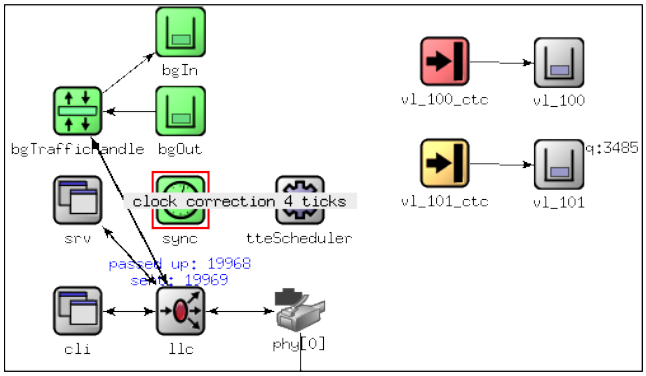

Figure 4: Simulated Modules in an endsystem with synchronisation.

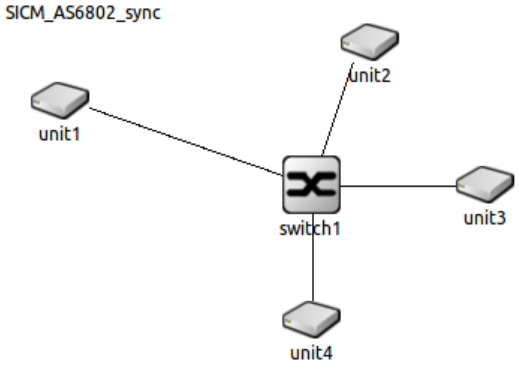

Figure 5: AS6802 network example.

shows the formula for the calculation of a clock event after δ ticks:

$$t' = t + \delta * (\Delta t_{Tick} + \Delta t_{Drift}) \quad (5)$$

In this model, t' defines the simulation time of the clock event after δ ticks, by using t as the current simulation time, Δt_{Tick} as the fixed time for one tick (without clock drift), and Δt_{Drift} as the average drift for a configurable amount of time.

The overhead for calculating the clock drift is then configurable by the amount of time the drift is assumed to remain constant. Previous work has shown that the precision of this approach mostly depends on the frequency stability of the clock and the number of events that are scheduled during the constant time interval [13]. We found that a valid and sufficiently accurate simulation for time-triggered systems under normal conditions can be achieved with an interval of one communication cycle, as the variance of the average drift is very low [14]. Our approach does not simulate ageing processes, but the frequency drift of different ageing and environmental conditions can be simulated by changing Δt_{Drift}.

To support an adaptable solution we made the clock model exchangeable. This way the simulation developer can choose from different abstraction levels of clock models to achieve the desired precision in the simulation.

We use exchangeable clock modules to realise an evolutionary approach. In the beginning the synchronisation behaviour of a distributed system is simulated using the complex and simulation time consuming time synchronisation protocol. During this simulation *clock_correction* and Δt_{Drift} will be recorded for each device and each integration cycle. Using these results future simulations of the same network will run without simulating the time synchronisation protocol but achieve the same temporal behaviour as a simulation including time synchronisation. In detail, for each integration cycle one data set of the first simulation run is mapped. Instead of using the clock correction of the time synchronisation protocol, the recorded *clock_correction* value is added to the *local_clock*. Since the PCFs are sent at a preconfigured time of each integration cycle, the required bandwidth for synchronisation frames is still reserved, but the PCF is not simulated. For each integration cycle the *clock_correction* and Δt_{Drift} data set is selected out of the data stored during the first simulation.

Assuming we have several devices with a given clock drift and we build a network with a given slot length for time-triggered transmission. This slot shall have a minimal length equal to the acceptance window. To calculate the minimal length of this window we need to calculate the maximal possible synchronisation accuracy. It is the minimal possible *precision* value that can be achieved with the hardware devices. Since the precision is defined as the worst case deviance between any two correct local clocks in the network, we choose the two clocks with the largest drift value *x'* and *y'*. Therefore we can calculate the *precision* as follows:

$$precision = (x' + y') \times integration\_cycle\_duration \quad (6)$$

In our implementation each device has a configurable parameter *max_clock_drift*. Thus we can simulate different topologies and configuration options with different hardware clocks and analyse whether the devices meet the requirements or determine the requirements for a network. For example the minimum length of the slot for TT frames that can be achieved with the given devices.

Our simulation model consists of several submodules (see Figure 4). The scheduler hosts the local clock. It offers an interface to register tasks and to adjust the clock (clock correction). The speed as well as the drift of the clock is configurable, according to the abstraction level of the chosen clock drift implementation. The sync module hosts the logic for the synchronisation protocol. Incoming and outgoing frames are routed through buffers for best-effort or real-time frames. These buffers interact with the physical layer. In our implementation of the synchronisation according to the AS6802 specification, the protocol logic is implemented in an automaton pattern. This allows us to reproduce the specified behaviour while keeping the implementation platform independent. Later on, the simulated protocol can be transferred in a real-world prototype without extensive changes in the program code.

4. USE-CASES & RESULTS

In the evaluation we show how simulation parameters and timing requirements are obtained from the simulation and how our evolutionary approach accelerates the simulation of large synchronised networks.

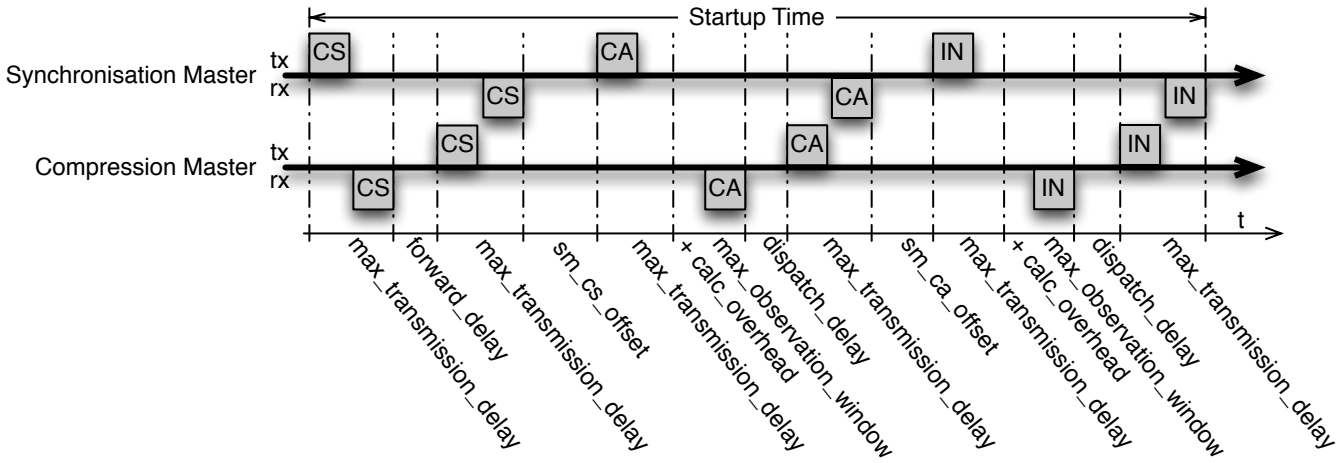

Figure 6: Estimation of startup time (time to sync).

		example 1			example 2			example 3		
Device	**sync config**	**drift change**	**max drift**	**time to sync [µs]**	**drift change**	**max drift**	**time to sync [µs]**	**drift change**	**max drift**	**time to sync [µs]**
Unit 1	SM	±100ps	20ps	209.843830	±100ps	20ps	209.843830	±20ps	1ps	209.839974
Unit 2	SC	±150ps	3ps	209.923832	±150ps	3ps	209.923832	±23ps	7ps	209.919979
Unit 3	SM	±300ps	34ps	209.923833	±300ps	34ps	209.923833	±15ps	2ps	209.839974
Unit 4	SM	±200ps	20ps	209.923843	±212ps	20ps	209.923843	±22ps	7ps	209.919978
Switch 1	CM	±70ps	14ps	202.883831	±70ps	14ps	202.883831	±15ps	4ps	202.879975

Table 1: Results of the hardware configuration analysis for the example configurations.

4.1 Hardware Configuration Analysis

In our first use case we show the simulation of three different hardware configurations. The analysis uncovers whether the configuration meets the given network requirements. The global parameters are:

- *clock_tick*: 80 ns
- *precision*: 6400 ns
- *cycle_duration*: 12500 ticks = 1 ms

We use a simple network as depicted in figure 5. The network consists of four end systems and one switch connected in a star topology. Unit 1, unit 3, and unit 4 are configured as synchronisation master, unit 2 as synchronisation client and switch 1 as the compression master. The scale exponent of OMNeT++ is configured to picoseconds time resolution.

We calculate the expected *precision* offline. In example 1 and 2 the units with the largest drift values are unit 3 and unit 4, in the third case unit 2 and unit 4:

$$\begin{aligned} max\_clock\_deviation(ex1) &= unit3 + unit2 \\ &= 300ps + 200ps \\ &= 500ps \end{aligned} \tag{7}$$

$$\begin{aligned} max\_clock\_deviation(ex2) &= unit3 + unit2 \\ &= 300ps + 212ps \\ &= 512ps \end{aligned} \tag{8}$$

$$\begin{aligned} max\_clock\_deviation(ex3) &= unit2 + unit4 \\ &= 23ps + 22ps \\ &= 55ps \end{aligned} \tag{9}$$

With a cycle of 1 ms (equal to 12500 ticks), the calculated worst case deviation is:

$$precision(ex1) = (12500ticks \times 500ps) = 6250ns \tag{10}$$

$$precision(ex2) = (12500ticks \times 512ps) = 6400ns \tag{11}$$

$$precision(ex3) = (12500ticks \times 55ps) = 687.5ns \tag{12}$$

These values are within our required value and we expect that the network will enter synchronised state. In order to check this hypothesis, we run the simulation for 120s and measure the time until a tentative synchronisation is achieved by the protocol startup service (*time to sync*)[1]. This means we measure the time from synchronisation start until the first integration cycle after the first synchronised state (see Table 1).

The best-case startup time (time to sync) can be calculated as the sum of synchronisation timing parameters (see Figure 6). Equation 13 shows the calculation of the best-case startup time:

$$\begin{aligned} time\_to\_sync &= (6 * max\_transmission\_delay \\ &+2 * (max\_observation\_window + calc\_overhead) \\ &+2 * dispatch\_delay + forward\_delay \\ &+sm\_cs\_offset + sm\_ca\_offset \\ &= 6 * 87ticks + 2 * (80ticks + 5ticks) + 2 * 161ticks \\ &+0ticks + 504ticks + 1100ticks) * 80\frac{ns}{tick} \\ &= 2681ticks * 80\frac{ns}{tick} = 209.44\mu s \end{aligned} \tag{13}$$

[1]The protocol designers claim that the protocol provides defined worst-case synchronous startup time [6].

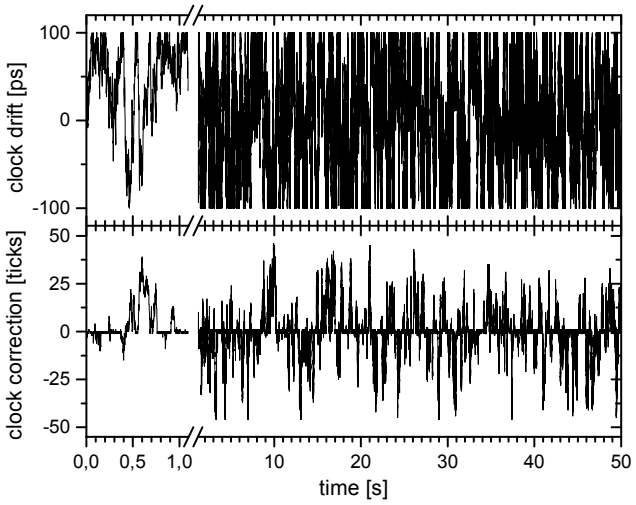

Figure 7: Clock correction for a single clock (unit1).

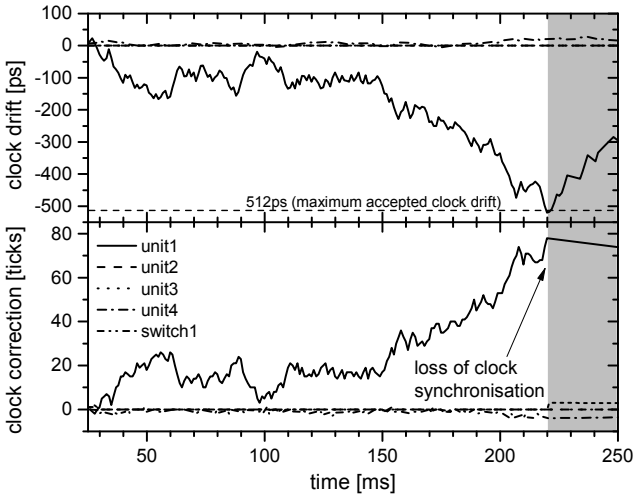

Figure 8: Protocol simulation with synchronisation loss due to high clock drift.

The expected startup time slightly differs (<550 ns) from the actual times simulated (see Table 1). The divergence is due to the accumulated imprecision of all calculations done in the devices. The smallest schedulable unit is one tick, thus all calculations are done in ticks as well. The roundoff error is at maximum 80 ns (1 tick) per calculation.

The protocol behaviour during the synchronised state is shown in figure 7. The upper graph shows the clock drift of one unit (unit1) in the sample network, while the lower graph shows the clock correction values. In the zoomed interval it can be observed how the protocol corrects the clock drift by correcting the local clock in each integration cycle. The more the clock drifts in one direction the more ticks are required to correct the clock.

4.2 Timing Requirements Analysis

The second use case represents a configuration with two very inaccurate clocks (see Table 2). Equation 14 and 15 calculate the achievable *precision*.

$$max\_clock\_deviation(ex4) = unit4 + unit1 = 700ps + 800ps = 1500ps \quad (14)$$

$$precision(ex4) = (12500ticks \times 1500ps) = 18750ns \quad (15)$$

Since the calculated value is rather high, the analysis of clock synchronisation is of interest to analyse the stability of operation.

Device	sync config	max drift	drift change	time to sync [µs]
Unit 1	SM	±800ps	30ps	-
Unit 2	SC	±0ps	0ps	-
Unit 3	SM	±0ps	0ps	-
Unit 4	SM	±700ps	3ps	-
Switch 1	CM	±0ps	0ps	-

Table 2: Configuration of example 4.

The simulation provides the number of stable cycles before entering unsynchronised operation. According to the simulation results (see Figure 8), synchronisation will be lost in cycle 220. In cycle 220 unit1 reaches a clock drift peak value of 520 ps seconds which is greater than our limit of 512 ps. The given precision of 6400 ns is equal to $12500 \times 512ps$ deviance. This is the reason for losing synchronisation. The simulation discovers that the given configuration does not meet the requirements.

4.3 Simulation of Large Networks

Based on our evolutionary approach, the last use-case shows how to simulate large networks without reducing the quality of the simulation. The network parameters are equal to example 3. We will use two configurations with different numbers of devices to illustrate the achievable acceleration.

First we run a simulation that includes the time synchronisation protocol (based on PCF). This run records *clock_correction* and Δt_{Drift} values. Afterwards we run the simulation again without the time synchronisation protocol, based on the *clock_correction* and Δt_{Drift} values recorded during the first run. Due to the Simsec/sec ratio (see Table 3) we can conclude that the separate simulation of PCF traffic leads to a significant acceleration of the simulation. In our tests we were able to run the simulation over 40 times faster, allowing us to precisely simulate even large networks on a standard consumer computer in real-time.

network	nodes	Simsec/sec	accel. factor
with PCF	5	≈0.67	-
without PCF	5	≈8.5	≈12.68
with PCF	9	≈0.22	-
without PCF	9	≈4.4	≈20.00
with PCF	23	≈0.04	-
without PCF	23	≈1.63	≈40.75

Table 3: Simulation results of large networks with and without the evolutionary approach.

5. CONCLUSION & OUTLOOK

Real-time clock synchronisation protocols gain importance in several application domains. Especially during design and configuration, simulation is a strong tool to obtain system parameters and network metrics. This paper presents a simulation concept for real-time clock synchronisation protocols based on the AS6802 specification.

By showing different examples in our evaluation, we introduce relevant use-cases for the proposed simulation concept. With given parameters the simulation can obtain central metrics of the synchronisation protocol such as startup duration (time to sync) or synchronisation precision. Further it can assess the required precision of local clocks for a set of given system accuracy requirements.

By separating the simulation of clock synchronisation and operation, we introduce an evolutionary approach, that speeds up the simulation of synchronised distributed systems significantly. To achieve this, the clock synchronisation is simulated while the clock behaviour and clock-correction is recorded. Afterwards the synchronisation can be omitted while the local clocks of all simulated devices are fed with the recorded values. This provides a performant simulation without losing accuracy.

For our future work we plan more detailed clock models. A model where the simulated drift of the clocks of all devices is artificially synchronised with each other will improve simulation coverage. Such a module allows us to specifically generate worst case scenarios where the synchronisation has the largest difference in clock drift. Further improvements of the clock model will contain temperature and voltage models to simulate local clocks in changing environments – such as in-car scenarios – most accurately.

The implementation of the synchronisation model is currently prepared for OpenSource publication. A first beta release can be found on the project webpage (`http://core.informatik.haw-hamburg.de`).

Acknowledgments

This work is funded by the Federal Ministry of Education and Research of Germany (BMBF) within the RECBAR project.

6. REFERENCES

[1] P. Ferrari, A. Flammini, S. Rinaldi, and G. Gaderer. Evaluation of clock synchronization accuracy of coexistent Real-Time Ethernet protocols. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 87–91, Piscataway, NJ, USA, Sept. 2008. IEEE Press.

[2] G. Giorgi and C. Narduzzi. Modeling and Simulation Analysis of PTP Clock Servo. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) 2007*, pages 155–161, Piscataway, NJ, USA, Oct. 2007. IEEE Press.

[3] Institute of Electrical and Electronics Engineers. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. Standard IEEE Std. 1588, IEEE, 2002.

[4] Institute of Electrical and Electronics Engineers. IEEE 802.3: LAN/MAN CSMA/CD Access Method. Standard IEEE 802.3-2005, IEEE, 2005.

[5] International Telecommunication Union - Telecommunication Standardization Sector. G.810 Definitions and Terminology for Synchronization Networks. Technical Report Series G: Transmission Systems and Media, Institution, Geneva, Switzerland, Aug. 1996.

[6] M. Jakovljevic. Deterministic Ethernet: SAE AS6802 "Time-Triggered Ethernet. SAE International, Nov. 2011.

[7] O. Karfich, F. Bartols, T. Steinbach, F. Korf, and T. Schmidt. A Hardware/Software Platform for Real-time Ethernet Cluster Simulation in OMNeT++. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, 2013. to appear.

[8] Y. Liu and C. Yang. OMNeT++ based modeling and simulation of the IEEE 1588 PTP clock. In *International Conference on Electrical and Control Engineering (ICECE), 2011*, pages 4602–4605, Piscataway, NJ, USA, Sept. 2011. IEEE Press.

[9] OMNeT++ Community. OMNeT++ 4.2.2. http://www.omnetpp.org.

[10] F. Praus, W. Granzer, G. Gaderer, and T. Sauter. A simulation framework for fault-tolerant clock synchronization in industrial automation networks. In *IEEE Conference on Emerging Technologies and Factory Automation 2007*, pages 1465–1472, Sept. 2007.

[11] SAE. Time-Triggered Ethernet AS6802. SAE Aerospace, Nov. 2011.

[12] SAE - AS-2D Time Triggered Systems and Architecture Committee. Time-Triggered Ethernet (AS 6802), 2009.

[13] T. Steinbach, H. Dieumo Kenfack, F. Korf, and T. C. Schmidt. An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 375–382, New York, Mar. 2011. ACM-DL.

[14] D. B. Sullivan, D. W. Allan, D. A. Howe, and F. L. Walls, editors. *Characterization of Clocks and Oscillators*. National Institute of Standards and Technology, Boulder, Colorado, 1990. technical note 1337.