A Simulation Framework for Fault-Tolerant Clock Synchronization in Industrial Automation Networks

Fritz Praus, Wolfgang Granzer, Georg Gaderer, Thilo Sauter
Research Unit for Integrated Sensors Systems, Austrian Academy of Sciences
Viktor Kaplan Strasse 2, A-2700 Wiener Neustadt, Austria
{Fritz.Praus,Wolfgang.Granzer,Georg.Gaderer,Thilo.Sauter}@oeaw.ac.at

Abstract

Many applications such as distributed measurements or real-time networks benefit from a common notion of time. Protocols providing high precision and simple clock synchronization are necessary to achieve such a common time base. However, most of the available protocols are lacking with regard to fault tolerance and performance in case of a fault. The project IMAGINE (Introduction of Master Group Based Industrial Ethernet) overcomes these limitations by introducing a fault-tolerant IEEE 1588 master group. A proof of concept for a large- or even medium- scale network is, however, very difficult to obtain under laboratory conditions. Therefore, a simulation framework has been developed, which is presented in this paper.

1. Introduction

Synchronization of processes is an essential requirement in distributed systems. One possibility to do so is clock synchronization, and various technologies to synchronize clocks exist today. Key properties of a clock synchronization approach are its achievable precision and accuracy. Precision refers to the synchronization of the clocks with respect to each other whereas accuracy is defined as synchronization of nodes with respect to an external reference (e.g., GPS time). Currently, an accuracy of nanoseconds or even better is possible. Still, it is desirable that these highly precise clock synchronization protocols are kept as simple and robust as possible. On the one hand, the protocol shall be fault-tolerant in that synchronization must be possible in a reasonable time even when key components of the network (e.g, a synchronization master) fail. On the other hand, resources requirements (e.g., regarding network bandwidth, node capacity) shall be as small as possible especially in large networks.

As shown in Section 2, none of the available clock synchronization protocols satisfy the above mentioned requirements. The research project IMAGINE (Introduction of Master Group Based Industrial Ethernet)[1] tries to overcome these problems by introducing a new approach. Particular attention is paid to improving the robustness of standard synchronization methods popular in automation. Evaluating and judging these newly defined algorithms raises the question of performance analysis under realistic operating conditions.

Analytical methods allow to describe problems in a very exact, provable, and predictable form. However, it is very difficult or nearly impossible to analyze the dynamical behavior of complex systems. Quite often, it is only feasible to describe steady states of such systems.

Prototyping allows to study the real-world behavior of various systems. Results of such an analysis can be achieved very fast and the basic functionality of a system can be shown. It may however be difficult to perform such an analysis under laboratory conditions in a reproducible way. Moreover, large-scale analysis with more than a few participating nodes is neither easily manageable, nor can it be implemented in a cost-efficient way.

Simulation is an approach mainly targeted to analyzing dynamical systems. Abstract models of a particular system are developed and evaluated using a simulator. It is thus possible to simulate large-scale installations consisting of thousands of nodes. Compared to prototyping, a simulation is not bound to real-time. To simulate the behavior of a system over years, a simulator needs only a fraction of time that would be needed using prototyping. Obviously, the effective duration of the simulation depends on the available system resources (i.e., computational power and memory size) of the simulation host system as well as on the level of detail of the simulation model.

For the performance analysis of large-scale and heterogeneous systems which are within the focus of IMAGINE, simulation is the method of choice. Nevertheless, the selection and implementation of suitable simulation strategies is by no means straightforward. The purpose of the paper is thus to discuss the simulation framework which has been developed within the scope of the project.

[1]The work presented in this paper was funded by the Austrian Research Promotion Agency (FFG) under the BRIDGE project 810092.

1-4244-0826-1/07/$20.00 © 2007 IEEE

The remainder of the paper is structured as follows: After this short introduction, an approach to fault tolerant clock synchronization is presented. Then, the demands on the simulation framework are discussed (Section 3) as well as an introduction into the functionality of the used simulator is given (Section 4). In Section 5, the simulation framework itself as well as the developed components are described in detail. Finally, a conclusion and an outlook to further work will round up the proposed approach.

2. Approach

In the Internet domain, the oldest and probably best known protocol for clock synchronization is the Network Time Protocol (NTP). This protocol has been designed to synchronize clocks with special attention to fault tolerance. Due to the potential instability of the Internet, specially designed democratic algorithms for assembling a fault tolerant local clock value are used. However, due to the design of NTP, the achievable accuracy is only in the range of several milliseconds. Furthermore, the power-up behavior is poor due to the self-adjusting internal protocol parameters of the nodes.

Other protocols like the democratic, interval-based Synchronized Coordinated Universal Time (SynUTC) algorithms [16, 14] use the so-called Marzullo function as convergence algorithm. The main advantage of these democratic algorithms is the increased fault tolerance, since it can be shown that in a network with $2F+1$ nodes, F nodes may be faulty without influencing other nodes. Furthermore, the overall accuracy of a democratic ensemble increases with the number of participating nodes (similar functionality can be found in stratum groups of NTP).

For calculating the so-called ensemble time, knowledge about the clock value of all other nodes is required. This is a significant drawback since the network load will increase significantly. Moreover, from the system design viewpoint, the node complexity increases in such systems as they have to handle multiple connections in order to keep track of $2F$ foreign clocks. Therefore, in the field of industrial and home automation, where nodes have to be as simple and efficient as possible connecting to one single, precise reference clock is preferred.

A key player in this area is the well-established clock synchronization protocol IEEE 1588[4] introduced in 2002. This standard defines the Precision Time Protocol (PTP) and specifies clock synchronization for highly accurate applications. It is held – compared to other protocols like NTP – as simple as possible. IEEE 1588 is currently enhanced to version 2 and the new standard is expected to be published in May 2007. IEEE 1588 synchronizes clocks using a master/slave approach. Once turned on, a node enters a listening state. During this period, nodes try to catch synchronization messages being broadcasted by other nodes. These synchronization messages (containing the accurate time) are evaluated based on the so-called Best Master Clock (BMC) algorithm. It elects a master out of all nodes (including the node itself) by verifying the accuracy of each clock.

The problem of the BMC algorithm is also its strength. If, for example, a master node fails to work properly, it will be replaced by another, master-capable IEEE 1588 node. As all nodes in this case would always wait for at least `PTP_SYNC_RECEIPT_TIMEOUT` (which is itself defined as ten times the value of `PTP_SYNC_INTERVAL`, being 2 seconds as default), the re-election can take place in earliest 20 seconds. This `PTP_SYNC_RECEIPT_TIMEOUT` is a compromise [6] between the ability of a node to determine a new master and to react (or not react) on simple, undelivered packets. Besides, the value does not include possible collisions occuring when two nodes decide on their own to become a new master. Therefore, in other applications like telecommunication industry even longer periods (for master re-election) are used in order to ensure stability of a communication link.

IMAGINE tries to overcome this lack of fault tolerance of PTP by introducing three different *IMAGINE stratum levels* (Figure 1). IMAGINE Stratum-0 provides a global reference time by providing an interface to highly accurate clock references (e.g., GPS receivers or atomic clocks). IMAGINE Stratum-1 consists of highly accurate nodes which are potential masters. This so-called master group combines the clocks of all master group members to one democratic ensemble time via algorithms developed in [15]. This ensemle time is used to synchronize the fully IEEE 1588 compatible slaves of IMAGINE Stratum-2. The synchronization is done via the so-called master group speaker, which is in the actual implementation a IMAGINE Stratum-1 enabled switch [7].

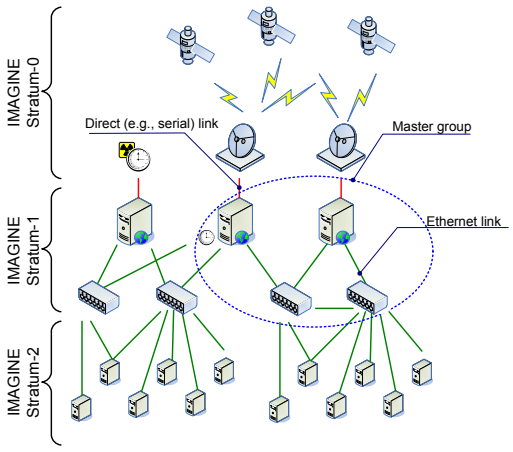

Figure 1. IMAGINE System Concept with Master Group and IEEE 1588 Nodes

3. Simulation Framework Requirements

In general, several requirements to a suitable simulation framework can be formulated: The simulation framework shall offer proper hard- and software models to allow from their *granularity* a detailed simulation of the behavior of interest. For example, to simulate the behavior of a few thousand nodes, the exact state of each flip-flop of each node is not of particular interest. Nevertheless, the level of detail of the simulation models must be appropriate to provide a meaningful simulation. Moreover, such a system has to provide the possibility to replace some of the black boxes with more fine-grained simulation elements.

Therefore, all simulation (sub-) models should be *exchangeable*. It must be possible, for example, to replace a black box with a more hardware focused model without interfering with the rest of the simulation framework. To support a hardware/software co-simulation, it is desirable that the software modules used later on for actual prototype implemenation can be integrated directly into the simulator without changes. To achieve this, the communication interface of each protocol stack, which is in general an operating system interface, has to be replaced transparently by simulator interfaces.

The overall simulation system has to be efficient in terms of *performance*. As it is the goal to simulate networks consisting of several thousand nodes, a continuous time driven simulation would not end in reasonable computation times. Therefore, a simulation concept based on discrete event simulation has to be used. A Discrete Event System (DES) is a system where state changes (events) happen at discrete instances in time and events take zero time to happen. It is assumed that nothing (i.e., nothing of interest) happens between two consecutive events (in contrast to continuous systems where state changes are continuous). Obviously, choosing a discrete event approach influences the design of the simulation models significantly. Consider, for example, a model of an oscillator. It is easy to describe in a continuous time model but cannot be trivially adapted to be suitable in a DES.

As mentioned in Section 1, DES only needs a fraction of time that would be needed using prototyping. Nevertheless, the granularity of the simulation models should be chosen in such way that it is possible to simulate over several years without exhausting the available resources.

4. Simulator Choice

In IMAGINE, the DES simulation environment OMNeT++ [18] is used ($OMNeT++$ 3.3 including $INET-$ 20061020 framework). In addition to OMNeT++ which is free for academic and non-profit use, a commercial version called OMNEST is also available. OMNeT++ provides a component-based, modular and open architecture for discrete event simulation. In particular, it consists of a simulation kernel and various utility classes being responsible for, e.g., random number generation, statistics collection, or topology discovery. The actual functionality, ranging from simple protocol stacks up to complex components is handled by various simulation models which are implemented as so called OMNeT++ modules.

The simulation kernel maintains a set of upcoming events called Future Event Set (FES). This total ordered list is processed by selecting the next event in time, processing the action to be taken at this event and deleting the current event. Note that the processing of an event may also insert future events. The simulation is finished as soon as the FES consists of an empty list.

The timescale for the simulation core is called *simulation time*. It is used to schedule future events as well as to timestamp the arrival time of an event. Our framework uses this simulation time as the global reference time to represent the real time (e.g., Coordinated Universal Time (UTC) or International Atomic Time (TAI)) of our simulation. However, this is in contrast to the node's *local time* which represents the value of the simulation clock at each node. Therefore, each node has its own local time.

In OMNeT++ , the IEEE 754 `double` datatype is available for the simulation time. Its unit may be interpreted freely but its size is according to IEEE 754 11 bits for the exponent and 52 bits for the mantissa. Thus, it is possible to distinguish between 2^{52} events (i.e., points in time) without loss of precision. With a unit of ns this allows a simulation time of $\sim$ 52 days. For long-term simulation beyond this limit, a patched OMNeT++ version ($OMNeT++ -3.3\_longdouble$) with higher precision has been developed and is currently under evaluation. Using the IEEE 754 `long_double` datatype allows a simulation of $\sim$ 600 years in ns resolution but its use is limited to operating systems providing the high resolution datatype (e.g., Linux). For simulation of high-precision clock synchronization in subnanosecond resolution, the unit of the OMNeT++ simulation time has to be interpreted accordingly (e.g., $10ps$). Note that this limits the possible simulation duration.

As mentioned above, the behavior, i.e., the glue logic of the simulation models are represented by OMNeT++ modules which are implemented as C++ objects. To communicate between these OMNeT++ models, two different types of hierarchically nested intermodule communication are intended:

- On the one hand, communication which needs a finite amount of time relevant for the simulation, is represented as OMNeT++ messages. The models communicate by passing these messages to each other using gates and channels. Processing is handled by the OMNeT++ models being implemented as C++ objects in the methods `handleMessage()` and/or `activity()`.
- On the other hand, not every communication takes a finite amount of time which is of particular interest for the simulation. A register access, for example,

may be interesting in the simulation of VHDL code for hardware design but not in a simulation framework for protocol verification. Therefore, models may also communicate using standard programming (e.g., C/C++) interfaces.

As mentioned in Section 3, existing synchronization stacks targeted to run in the prototype implementation as well as OMNeT++ modules shall be reused and integrated into the simulation framework. Modifications to this software shall be avoided. An existing clock synchronization stack, for example, shall not be modified but nevertheless run smoothly in the simulation environment. Obviously, concepts dealing with OMNeT++ message passing mechanism have to be established.

For a more detailed description of the OMNeT++ simulation concepts see [3].

5. Simulation Framework

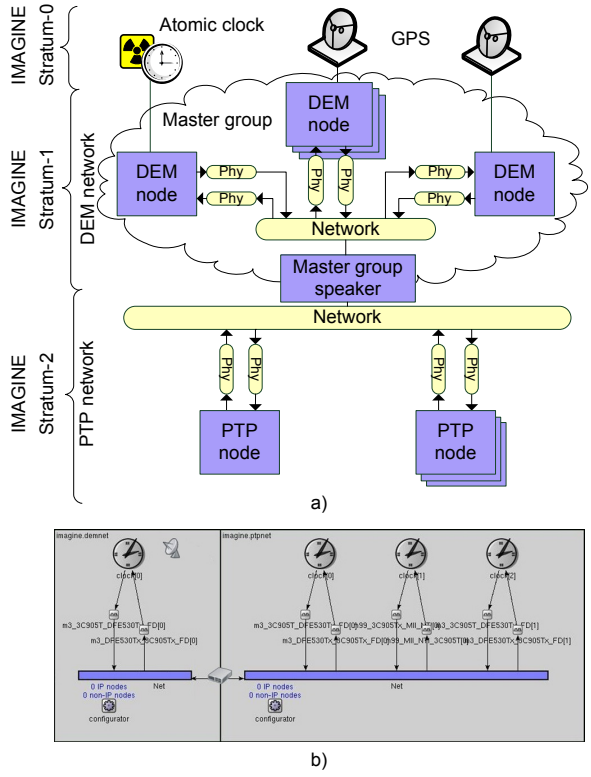

Figure 2. Simulation Framework.

As shown in Figure 2a (abstract model) and Figure 2b (screenshot), the topology of an IMAGINE network is divided into three different IMAGINE stratum levels that consist of various nodes like GPS receivers and atomic clocks (IMAGINE Stratum-0) as well as different synchronization nodes (IMAGINE Stratum-1 and Stratum-2). The synchronized nodes of level 1 and 2 are interconnected by broadcast networks, being divided into a democratic network (IMAGINE Stratum-1) and a PTP network (IMAGINE Stratum-2). The simulation of these networks consists of different simulation models which are implemented as a mixture of OMNeT++ modules and C/C++ objects. On the one hand, a model for the network itself (cf. Section 5.1) as well as a model for simulating the physical network medium (cf. Section 5.2) have been defined. On the other hand, various modules for simulating clock synchronization nodes (including support for democratic synchronization and synchronization using the PTP protocol; c.f. Section 5.3) have been implemented.

5.1. Network

The topology of the simulation is split into two different networks. One consists of a set of clock synchronization nodes (master group) located at the *democratic network* (IMAGINE Stratum-1). The purpose of this network is to serve as a fault-tolerant master group for a set of PTP slaves: Multiple nodes shall establish a common reference time via democratic clock synchronization algorithms and possible global reference time connections (to IMAGINE Stratum-0). This common reference time shall appear to a set of PTP devices as a single PTP master.

The second level is a set of PTP devices located at the *precision time protocol network* (IMAGINE Stratum-2). They act as PTP slaves and are connected to the democratic network via the master group speaker. However, the framework is not limited to such relatively flat networks. The topology is freely combineable, and so hierarchical networks such as presented in [8] are also possible.

In our simulation model, the networks represent ideal networks with no delay and no transmission errors. They merely serve as pure interconnection dummies which are required for the simulator. The real-world parameters considering transmission delay, transmission errors and asymmetry of the receive and send path are handled at the physical layer model of each node.

5.2. Physical Layer

The delay variation of the physical layer (PHY) and cabling plays an important factor for precise clock synchronization since it influences and limits the common accuracy. In particular, parameters such as propagation delay, network jitter and asymmetry have to be considered for precise simulation. Note, that in case of hardware timestamping is provided, any sort of constant delay in the communication path does not influence the accuracy since it can be considered in the calculations for the common notion of time.

The normally distributed jitter between an end-to-end communication path is caused by the physical layer devices and the cable. In the simulation, it is handled by two PHY models, which are located at each node in the send and receive path. The PHY accepts packets from the network, delays and probably drops them and finally passes them to its clock synchronization cell. Each PHY model consists of the parameters $delay\_mean$, being used for modeling the propagation delay in the path, $delay\_stdd$,

being used for modeling the network jitter in a path and tx_error, being used for modeling the error rate, i.e., probability to drop a packet. The concrete values for the parameters are based on measurements taken from [9]. Since two PHY models with different parameters are used in receive and send path, it is possible to model asymmetry in the delay, which plays an important role for precise clock synchronization [11].

5.3. Clock Synchronization Node

Figure 3a shows the abstract model of the clock synchronization node. Figure 3b shows a screenshot of the corresponding OMNeT++ model. It consists of a software part (network stack c.f. Section 5.3.1 and clock synchronization stack c.f. Section 5.3.2) and a hardware part (synchronization cell c.f. Section 5.3.3). This general model of a node is used for democratic clock synchronization as well as for synchronization using the PTP protocol. The only difference is the clock synchronization stack which can be exchanged arbitrarily.

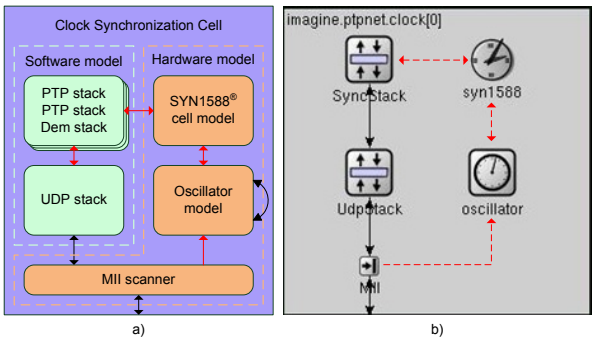

Figure 3. Clock Synchronization Node.

5.3.1. Network Stack

The network stack (being UDP stack in Figure 3, 4) provides a realistic simulation of the network traffic and offers interfaces for sending and receiving messages via a network. In IMAGINE, the INET framework for OMNeT++ [2] is used. It is suited for simulations of wired, wireless, and ad-hoc networks. The INET Framework provides IP, UDP/TCP, 802.11, Ethernet and several other protocols. Since the clock synchronization stacks in IMAGINE rely on UDP communication only, this minimal subset of the INET framework is used: The network stack receives packets from the Media Independent Interface (MII) scanner (Section 5.3.3) relying on the OMNeT++ communication flow and handles them according to their protocol types. Synchronization packets are passed to the clock synchronization stack using the standard C `recvfrom()` function call. For sending time synchronization packets the clock synchronization stacks call the standard C `sendto()` function of the network stack, which processes and passes them to the MII scanner.

5.3.2. Clock Synchronization Stack

The Clock Synchronization Stack (CSS) performs adjustments to the local clock of the node. As mentioned before, one design goal of IMAGINE is to create an universally applicable simulation framework, where various components/models can be integrated, exchanged, evaluated, and tested easily. For this reason, simple and clear interfaces have been defined in IMAGINE and it is possible to e.g., use democratic as well as PTP CSS. A CSS shares an interface to the network stack for exchanging clock synchronization packets as well as an interface to the (simulated) cell model.

PTP Stack

The PTP stack performs precise clock synchronization (IMAGINE Stratum-2) of the PTP slaves in the PTP network with the master group speaker located between the democratic network and the PTP network. In particular, the following stacks are evaluated and used in IMAGINE:

- PTP stack from Oregano Systems [12]: It performs synchronization conforming to the IEEE 1588 standard. Some of the key features are: Full Standard IEEE 1588 support, Hardware, Software and On-the-Fly timestamping, and rate as well as state synchronization.
- PTPd daemon [5]: It is freely available under a BSD-style license and provides a complete implementation of the IEEE 1588 specification for a standard (non-boundary) clock.

Democratic Clock Synchronization Stack

Using democratic clock synchronization, a set of homogeneous nodes is able to establish an ensemble time based on a peer-to-peer basis (IMAGINE Stratum-1). In IMAGINE, multiple nodes form the democratic network, establish a reference time with an optional external synchronization to e.g., a GPS time and form a fault-tolerant master group for synchronizing high precision PTP slaves. Currently, existing and well established democratic clock synchronization algorithms [19] are being investigated, adapted, and reused. The final democratic clock synchronization stack is still under development, however plans exist to create a framework with exchangeable synchronization algorithms such as the Fault Tolerant Average (FTA), Fault Tolerant Interval (FTI), Marzullo, or the clock rate synchronization algorithm.

Integration of a CSS into the IMAGINE simulation framework is straightforward: The stack remains untouched, only its interfaces are adapted. The system calls of the stack are mapped to the OMNeT++ interface methods as well as to the interface methods of the other simulation models. On the one hand, this includes a mapping of the network interface functions to the functions provided by the INET framework. On the other hand, this includes the simulated access to the hardware described in Section 5.3.3.

5.3.3. Synchronization Cell Model

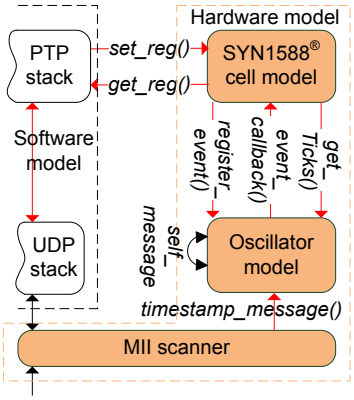

Figure 4. Hardware Model

To be able to synchronize high accuracy clocks, hardware support is necessary. In IMAGINE, a hardware clock called SYN1588 Ⓡ cell is integrated into each clock synchronization node. This SYN1588 Ⓡ cell provides necessary hardware mechanisms to perform highly precise clock synchronization according to the IEEE 1588 standard. However, since the requirements of PTP regarding hardware support are similar to requirements of the democratic synchronization protocol, the SYN1588 Ⓡ cell can also be used for democratic clock synchronization.

In order to perform a realistic and meaningful simulation of clock synchronization algorithms, even those underlying hardware components have to be emulated in software. The main benefit of such a detailed simulation (including an emulation of the hardware) is that our simulation can use the same synchronization algorithms which are being used in the prototype implementation. Therefore, it is possible to compare the simulation results with the prototype implementation directly.

The proposed hardware model which emulates the functionality of the synchronziation cell can be seen in Figure 4. This simulation model consists of three different building blocks.

Oscillator Model

The simulation framework of OMNeT++ provides a timescale called simulation time. This simulation time is used to schedule network messages as well as timing events. Obviously, our real-world synchronization cell does not have such a global reference time. It rather uses the ticks of the integrated oscillator for calculating its local time. To be able to simulate these tick events, a software oscillator model providing a conversion between simulation time and the current ticks[2] has been implemented. The behavior of a real-world oscillator differs from that of a perfect one which means that the frequency of an oscillator varies over time. Therefore, calculating the current ticks using a fixed tick period is not appropriate.

To be able to simulate these frequency deviations in our simulation, a common oscillator model is used [17]. Using this model, the frequency at a given time can be calculated as $f(t) = f_0 + \Delta f_0 + a(t - t_0) + \Delta f_n(t) + \Delta f_e(t)$ with f_0 as the start frequency, Δf_0 as the frequency offset at start time t_0, a the ageing factor, $\Delta f_n(t)$ a jitter noise term and $\Delta f_e(t)$ an environmental term. This can be used to establish a relationship between simulation time and elapsed ticks.

The oscillator model has been implemented as an OMNeT++ simple module including an interface to the synchronization cell. Using this interface, the synchronization cell model can register timer events at a given amount of ticks. For each registered timer event, the corresponding simulation time is calculated using the above mentioned equation. This calculated simulation time is used to schedule a OMNeT++ self message[3]. Additionally, the event is stored together with the callback function in a local event queue which is invoked by the oscillator model after the given event has occurred.

MII Scanner

To be able to precisely measure the network link delay between clocks as specified in the IEEE 1588 standard, incoming and outgoing synchronization messages have to be timestamped. To reduce the error introduced by fluctuations in the time taken to traverse the software protocol stack, the timestamp shall be generated at the physical layer. Hardware assistance is necessary for achieving this.

The required hardware support is provided by the MII scanner. It is responsible for detecting valid synchronization messages (i.e., PTP messages) by parsing the message content of all incoming and outgoing network messages. Valid synchronization messages (in the case of a PTP node, this is an UDP message with destination port 319 or 320) are timestamped and their sequence ID as well as their source universally unique identifier (UUID) are extracted and passed to the SYN1588 Ⓡ cell . They are used to associate the correct timestamp with the corresponding network message later on during processing of the message.

SYN1588 Ⓡ cell Model

The main component of synchronization cell is a Clock Synchronization Core (CSC) called SYN1588 Ⓡ cell . It provides the necessary hardware functionality to perform a high precision clock synchronization using democratic or PTP based algorithms[4]. Our simulation model emulates the behavior of the CSC [13, 10]. The CSC is configured and controlled via various 32-bit registers. Currently, the following subset of the features of the CSC are supported

[2]The term current ticks is used as a synonym for the amount of elapsed ticks since the oscillator is running.

[3]An OMNeT++ self message is a message being sent to the module itself at a later point of simulation time. Using this mechanism, timer events can be simulated.

[4]As mentioned above, the functionality of the CSC also satisfies the requirement of democratic clock synchronization.

by our simulation model:

The core component of the CSC is an *adder-based hardware clock* with a total register length of 104 bits. The upper 64 bits are compatible to the IEEE 1588 standard (i.e., 32 bits for seconds and 32 bits for nanoseconds). The lower 40 bits (32 bits subnanoseconds and 8 bits ultrafractional part) are used to increase the accuracy of the synchronization process and thus allow high-precision clock synchronization.

The hardware clock is configured and controlled via shadow registers. These shadow registers can be used to set the current time directly as well as to set a stepsize which is added to the current time value with every tick event of the oscillator. The contents of the shadow registers can be loaded directly or after a defined future point in time has elapsed (apply timer). Furthermore, it is possible to load the time value and the stepsize simultaneously or separately.

Due to the length of the timer register (104 bits), it is not possible to simply simulate the hardware clock using standard C/C++ datatypes. The 64 bit precision of unsigned double integers (on x86 systems) is not sufficient and so, the Gnu Multiprecision Library (GMP) [1] has been used. GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating point numbers. There is no theoretical limit in the precision of used operands and at the same time it is very fast, since the used algorithms are highly optimized.

Based on this hardware clock, the CSC provides various *timers*. To be able to schedule future events (e.g., send a synchronization message), the PTP stack as well as the democratic stack use period timers to implement timeouts. Therefore, period timers are also supported by our simulation model. Other timers (e.g., timers to activate external signals) are currently not implemented, since they are not required by the used software stacks.

As mentioned above, the MII scanner passes the sequence ID as well as the source UUID of detected PTP messages to the CSC to perform *hardware timestamping*. The CSC stores these values together with the respective timestamp. Currenty, two different First In First Out (FIFO) buffers are used, one for incoming and one for outgoing frames. Each entry consists of the PTP source UUID and sequence ID as well as the upper 96 bits of the timestamp.

Both FIFO buffers can be accessed via registers. Additionally, an interrupt service which indicates a received or sent network packet is available.

6. Preliminary Simulation Results and Future Work

As a first application, the open source PTP daemon has been evaluated. A simple simulation network consisting of a simple PTP network has been set up. This PTP network contains 3 PTP slaves and 1 PTP master providing the global reference time. The democratic network has been left out for simplification. The following parameters have been used for the 4 nodes:

- Network and PHY (based on [9]): CAT-5, full-duplex, 3 meter cable length, Src. 3Com 3C905T and Dst. D-Link DFE530Tx network interface cards ($delay\_mean$ = 303.50 ns, $delay\_stdd$ = 0.92 ps, $tx\_error = 10^{-10}$), Src. D-Link DFE530Tx and Dst. 3Com 3C905T network interface cards ($delay\_mean$ = 330.44 ns, $delay\_stdd$ = 0.98 ps, $tx\_error = 10^{-10}$)

- Oscillator: frequency $f_0 = 10Mhz$, class = $100ppm$, ageing factor $a = 10^{-10}Hz/s$, phase noise $\Delta f_n(t) = 70ps$ (normal distributed), environmental factors (e.g., temperature) $\Delta f_e(t)$ ignored

Figure 5 shows the startup behavior of the network.

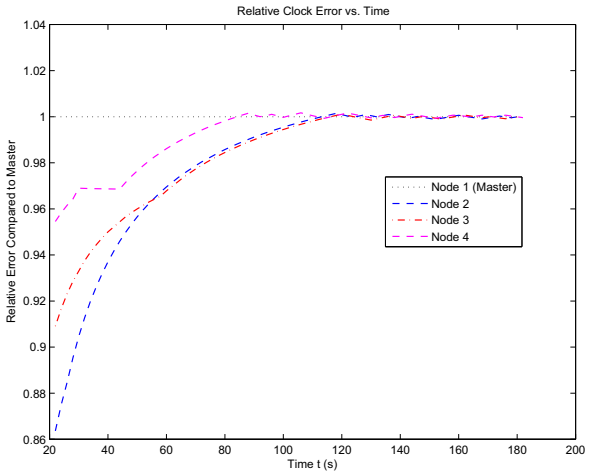

Figure 5. Open Source PTP Stack Evaluation.

In this paper a simulation framework for distributed clock synchronization is presented. It offers the possibility to simulate, evaluate, and compare different clock synchronization methods. In particular simulation of large-scale fault-tolerant networks consisting of resource limited devices as well as more powerful nodes is possible.

Nevertheless, further research is necessary to improve and refine the simulation. These steps include an advanced model and detailed dissemination of the oscillator and a detailed evaluation regarding precision, start-up, and fault-tolerant behavior of the different synchronization stacks. The PTP stacks and the democratic stacks will be compared and measurements will be taken. Finally, the implemented algorithms will be transfered to hardware (i.e., fault-tolerant switches, SYN1588 ® cell) so that prototype implementations will be available.

7. Acknowledgments

The authors wish to thank Patrick Loschmidt not only for his useful hints, but also for many helpful discussions.

References

[1] *GNU Multiple Precision Arithmetic Library (GMP)*. http://gmplib.org/.

[2] *INET Framework for OMNeT++/OMNEST release 2006-10-12*. http://www.omnetpp.org/doc/INET/.

[3] *OMNeT++ version 3.2*. http://www.omnetpp.org/doc/manual/usman.html.

[4] *1588 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systemsn*. IEEE Instrumentation and Measurement Society TC9, 2002.

[5] K. Correll, N. Barendt, and M. Branicky. Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol. In *Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2006.

[6] J. C. Eidson. *Measurement, Control and Commmunication Using IEEE 1588*. Springer Verlag London 2006 Limited, 2006.

[7] G. Gaderer, P. Loschmidt, and T. Sauter. IEEE 1588 Real-Time Networks with Hybrid Master Group Enhancements. In *Proceedings of the 4th International Workshop on Real Time Networks*, July 2005.

[8] G. Gaderer, P. Loschmidt, and T. Sauter. Quality monitoring in clock synchronized distributed systems. In *6th IEEE International Workshop on Factory Communication Systems*, pages 13–21, June 2006.

[9] M. Horauer. *Clock Synchronization in Distributed Systems*. PhD thesis, Technical University of Vienna, 2004.

[10] P. Loschmidt, G. Gaderer, and T. Sauter. Synchronized Access to Sensor Networks. In *Proceedings of the IEEE Sensors 2006*, page 218, Daegu, October 2006. IEEE.

[11] T. Müller, A. Ockert, and H. Weibel. PHYs and Symetrical Propagation Delay. In *Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2004.

[12] Oregano Systems. *Precision Time Protocol, preliminary data sheet*, February 2006.

[13] Oregano Systems. *SYN1588 enabled IEEE 1588 compliant clock sychronization, SYN1588Clock_L IP Core, brief data sheet*, September 2006.

[14] U. Schmid. An Annotated Bibliography on Clock Synchronization in Distributed Systems. Technical Report 183/1-45, Vienna University of Technology, Department of Automation, Dec. 1994.

[15] U. Schmid and K. Schossmaier. Interval-based Clock Synchronization. In *Journal of Real-Time Systems*, volume 2, pages 173–228, March 1997.

[16] U. Schmidt. Orthogonal accuracy clock synchronization. *Chicago Journal of Theoretical Computer Science*, pages 3–77, 2000.

[17] D. B. Sullivan, D. W. Allan, D. A. Howe, and F. Walls. Characterization of clocks and Oscillators. Technical Note 1337, NIST, 1190.

[18] A. Varga. Using the OMNeT++ discrete event simulation system in education. *IEEE Transactions on Education*, 1999.

[19] B. Weiss, G. Gridling, U. Schmid, and K. Schossmaier. The SimUTC fault-tolerant distributed systems simulation toolkit. In *7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, October 1999.