Performance Analysis of Time-Triggered Ether-Networks Using Off-The-Shelf-Components

Florian Bartols, Till Steinbach, Franz Korf, Thomas C. Schmidt
Department of Computer Science
Hamburg University of Applied Sciences, Germany
{florian.bartols, till.steinbach, korf, schmidt}@informatik.haw-hamburg.de

***Abstract*—The performance analysis and validation of distributed real-time systems poses significant challenges due to high accuracy requirements at the measurement tools. A fully synchronized time-scale at ultrafine granularity is not easy to generate. Even though there are several analyzer tools for standard switched Ethernet, these tools cannot be applied in time-triggered networks, since they do not meet the requirements of synchronized packet generation. This paper introduces a low cost and lightweight approach to measure end-to-end latency of time-triggered Ethernet traffic with off-the-shelf components. By using standard computer hardware and a real-time Linux Kernel, it is shown that measurement can be achieved in a resolution of microseconds. Furthermore, a validation with an Ethernet performance analyzer and a mathematical framework is presented to check the given results.**

***Index Terms*—real-time Ethernet, TTEthernet, time-triggered, benchmarking, end-to-end latency, components-off-the-shelf, real-time Linux;**

I. Introduction

Today's vehicles and airplanes are complex distributed real-time (RT) systems with a high demand of broadband communication links at guaranteed transmission performance. The amount of industrial plants with comparable characteristics grows continuously. These systems require control units that are connected across a backbone network, which needs to carry heavy load, while keeping message delays predictable. Due to the flexible and highly scalable protocol, backbone networks of industrial plants are often based on real-time Ethernet.

Standard switched Ethernet is a technology that allows to increase the amount of traffic simultaneously transferred by using segregated communication in groups. However, due to its random access and best-effort approach, it does not provide reliable temporal performance bounds. There are many attempts to overcome those obstacles like token-based, bandwidth-limiting or time-triggered Ethernet extensions.

Time-triggered Ethernet (TTEthernet) [1], the protocol applied in this work, supports several traffic classes with different qualities for the various real-time related metrics. TTTech [2] developed in collaboration with Honeywell [3] the TTEthernet specification that is currently proposed for standardization by the Society of Automotive Engineers [4]. Even though the results of this paper are based on TTEthernet, they are mostly transferable to other time-triggered Ethernet protocols, as well.

Time based Ethernet technologies require time measuring devices at Ethernet frame level. The information provided by these instruments are used for many development tasks like validating hardware and software, or analysis of protocols and network utilization. Many analyzers like oscilloscopes with Ethernet support can be used for this kind of temporal measurement. For cost and flexibility reasons, analyzers based on standard embedded computers and real-time operating systems (OS) gain importance. Concerning this kind of RT network analysis, the contribution of this paper is a lightweight analyzer that supports time measurement in the range of microseconds at a price below 500$. It is based on an embedded PC and a Linux OS with RT Kernel patch. Additional hardware like probes is not required.

The analyzer presented here can be used as a packet generator, as well, which is due to the flexibility obtained from the Linux OS running on an embedded PC. An application running in Userspace generates the packet stream. The network driver provides the time measurement techniques. Hence this approach supports any protocol-specific component residing between application and network driver. In particular, the analyzer supports the Linux TTEthernet protocol stack.

In this paper we measure end-to-end latency based on time stamps which are added to the payload of Ethernet frames. To avoid jitter and time lapses, these stamps are injected just before a frame is sent from the generating device and immediately after a frame has been received. The accuracy of this approach is in the range of a few microseconds and bases on the 64-bit CPU cycle counter register of Intel x86 CPUs. To avoid unexpected jitter, a Linux OS with RT Kernel patch is used. Moreover, system management interrupt (SMI) and CPU throttling are disabled.

In distributed measurement approaches, where one node is responsible for keeping the send time and another the receive time, clock synchronization is required for measurement of end-to-end latency. Since the analyzer given in this paper is realized on a single embedded PC with two separate ports for sending and receiving packets, a synchronized clock is guaranteed by design.

The remainder of this paper is organized as follows. In section II, related work, different time measurement concepts and the basic requirements for time measurement are introduced. Section III illustrates background information concerning RT Ethernet and TTEthernet. The implementation of the off-the-shelf analyzer is outlined in section IV. Section V evaluates the presented approach by comparing the results with a hardware analyzer and a mathematical model. In section VI results

for different measurements are depicted. Finally, section VII concludes and gives an outlook.

II. Related Work

In the field of real-time performance analysis, hardware- and software-based measurement approaches exist, whereat hardware approaches, using additional equipment like probes are more precise. If timestamps of Ethernet frames collected at different network ports are to be compared, the corresponding clocks must be synchronized. Analyzers for time measurement are divided into three groups. The first supports only one network interface such that time stamps of frames related to different network ports cannot be compared. Comparable to the approach described in this paper, analyzers belonging to the second group are based on one physical clock for all network ports, so that time analysis between frames of different network ports is made possible without clock synchronization. The last group stamps Ethernet frames in a distributed way using several clocks that are synchronized.
The IXIA 1600T [5] uses a hybrid approach with a granularity of $10ns$. Base measurement units consist of one local clock and multiple Ethernet ports. For large scenarios these base units can be connected.
A non commercial approach that uses several FPGA-based probes with local clocks and a clock synchronization protocol is presented in [6]. The synchronization accuracy between these probes is below $100ns$. In relation to the approach presented in this paper, [6] provides a higher accuracy for 1000$. On the other hand the amount of work for assembling, evaluating and setting up the probes must be taken in account, too.

For validating hard real-time communication over switched Ethernet, a distributed software approach was implemented in [7], where end-to-end latency between two nodes with an accuracy of $10\mu s$ is measured. The time synchronization protocol is based on an additional "black" cable connecting both nodes. Moreover, in a distributed measurement the node which is responsible for latency calculation must have access to the send- and receive timestamp of a frame. Modifying the payload of the network frame solves this problem by adding a timestamp while sending and receiving a packet. Similar to the approach given in this paper the time measurement described in [7] applies this technique.

The round-trip-time for real-time applications with 802.1Q Ethernet is measured in [8]. This is achieved with a software implementation that uses Linux raw sockets to access a frame directly. In comparison, the approach presented in this paper improves accuracy. It collects time stamps in the network driver just a few machine instructions before a frame will be sent and a few machine instructions after a frame has been received.

The utilization of Wireshark [9] based measurements is discussed in [10]. There are different test setups provided for using Wireshark and the results are compared to each other. Since this approach consists of one network port only, it cannot be applied for distributed measurement. The accuracy of the measurement results given in this work is improved due to using the RT Kernel patch for Linux and timestamps taken at network driver level.

III. Time-Triggered Ethernet

In achieving real-time behavior on Ether-Networks, the main challenge is to prevent RT packets from simultaneously traversing a switch interface or line card. In general, the strategies that allow Ethernet based RT communication can be divided in three different approaches. In *token-based* systems, a token is passed to each participant. Only the node that owns the token is allowed to transmit data. There have been efforts to use token based protocols for RT communication [11]–[13]. A commercial product based on a token access policy is Ethercat [14], which was developed for process automation.

The second class of RT capable Ethernet is given by *bandwidth limiting protocols* such as the Avionics Full-Duplex Switched Ethernet (AFDX) [15]. AFDX is used as communication protocol in airplanes like A380, the new flagship product by Airbus. In bandwidth limiting protocols, the maximum bandwidth is limited at each sender. Specialized switches survey the input ports to ensure that each sender transmits only within its assigned bandwidth.

The final group of relevant solutions is given by the *time-triggered protocols*. In time-triggered protocols, all senders operate in a synchronized way. According to coordinated time division multiple access (TDMA), scheduling defines permissible transmission times for all senders. This approach is well known within the automotive industry and based on previous experiences with TTCAN [16] and FlexRay [17]. In process automation, these protocols dominate deployment. Examples are Profinet [18] or SynqNet [19].

TTEthernet [1] the technology focused in this paper is a time-triggered protocol. It was developed by TTTech in collaboration with Honeywell and can be seen as a new approach to transmit critical and best-effort traffic on the same physical infrastructure. Therefore the protocol defines three different traffic types, which have individual requirements in timing and transmission.

- **time-triggered (TT)-traffic** has the highest priority, the strictest timing requirements and is used for hard RT communication. Sending and receiving is done in a time-triggered way.
- **rate-constraint (RC)-traffic** uses a guaranteed bandwidth for transmission and complies to AFDX standard [15].
- **best-effort (BE)-traffic** conforms to standard Ethernet traffic and has the lowest priority and thus, there is no guaranteed transmission.

Figure 1 shows a sample application in the automotive context where different traffic classes coexist on the same link. While time-triggered traffic e.g. for chassis information is transferred in periodic cycles with the highest priority, rate-constrained messages are transferred in bandwidth gaps. Afterwards free bandwidth is filled with best-effort traffic.

In TTEthernet each device has a schedule for time-triggered messages. This enables a coordinated TDMA network access

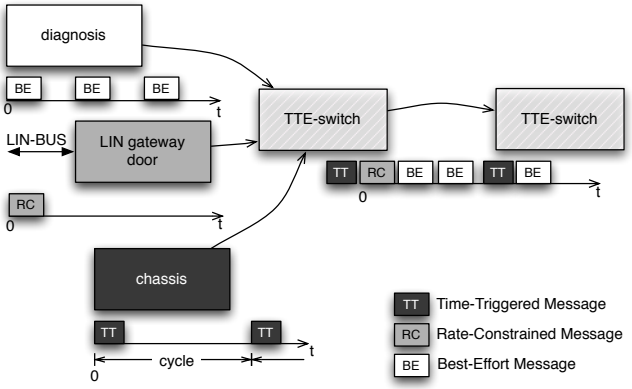

Fig. 1. TTEthernet in-vehicle sample application presenting the usage of the three traffic types.

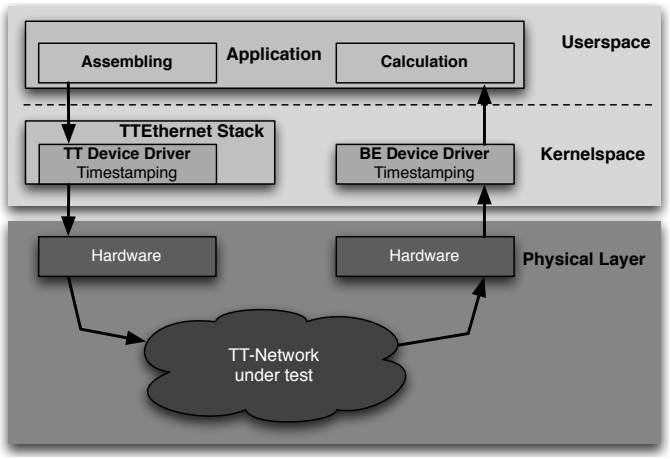

Fig. 2. The conceptual overview of the presented approach in three layers.

policy and ensures that critical traffic is never delayed due to busy linecards.

To ensure a system-wide time base, TTEthernet provides a two-step time synchronization protocol for all participants. Synchronization-Masters are responsible for synchronization initiation and send in a first step a synchronization message to a Compression-Master, which calculates an average of all Synchronization-Masters time bases. In a second step the Compression-Master sends the synchronization message to the Synchronization-Masters and to the remaining participants, formally known as Synchronization-Clients.

In TTEther-Networks TT-frames, which are sent outside their configured time slot, will be dropped in the TTEthernet switch [1]. These frames are seen as frames sent by a faulty device. This kind of error detection is necessary to avoid failures caused by a babbling idiot. As in this paper latency measurement of critical traffic is presented, one have to keep this behavior in mind. A measurement device has to be synchronized to the system-wide time base. For this reason standard tools and analyzers, like the presented IXIA device cannot be used, due to inability of synchronized packet generation.

IV. Implementing an off-the-shelf measurement facility

A. Conceptual Overview

In general, operating systems provide tools for latency measurement (i.e. ping, traceroute, etc.) in best-effort networks, where precision of milliseconds is sufficient.

Most real-time analysis environments pose on the precision in microseconds, which could not be fulfilled with the given OS tools. The suggested solution is time measurement on a low-level base in the OS. Tagging send and receive time should be done very close to the hardware. When using a computer system with an OS, time tagging has to move in the device driver, so that process scheduling and OS-functions take minimal effect on the measurement. Figure 2 presents the concept of the discussed approach.

A Userspace application (see figure 2) is responsible for frame assembling and for the calculation of the measured end-to-end latency after reception. The TTEthernet stack, which is provided by TTTech, ensures that generated frames are sent in-schedule and will not be dropped inside the TTEthernet switch. Modified device drivers, developed and discussed in this paper, are adding current timestamps to the frame while sending and receiving.

The application and the proper Kernelspace processes are scheduled with Linux RT-priorities by using the RT Kernel patch [20].

B. Measurement in Kernelspace

Any tool running in Userspace cannot meet high precision requirements, because the scheduler and hardware-layer could produce overhead on timing measurement, especially when events generated by hardware, interrupts the running process or process scheduling occurs. Usually, depending on implementation, these tools use OS functions to get the current system time. As a consequence these measurement-tools produce higher jitter.

Based on higher level network protocols, latency measurement could only be realized in systems which support IP/ICMP. The given OS tools measure the round-trip-time of a connection. To be predictable, frame routing of critical traffic in RT-Ether-Networks is done in a static way. Thus, these routes can be unidirectional or the return path differs. Hence, the round-trip-time is not applicable for end-to-end latency measurement.

C. Increasing the Resolution by the Real-time Kernel Patch

In general Linux does not fulfill RT requirements. To enable RT characteristics in a Linux-based system, there are some implementations in use, which are operating with a Coop-Kernel [21]. A separate RT-Kernel handles RT-tasks and is responsible for scheduling the RT-tasks and the standard Linux-Kernel, which has the lowest priority. Non RT applications, like logging functions, can run within the standard Kernel. Examples can be found in [22], [23].

The RT Kernel patch by Ingo Molnar [20] follows another approach. This patch enables RT characteristics to the standard Linux Kernel, by adding complete Kernel preemption, optimized RT interrupt handling [24] and RT scheduling (FIFO or Round-Robin) of RT-Kernel and RT-Userspace tasks.

The RT patch enhances RT-priorities also to Kernelspace processes so that they are preemptible by high prior Userspace processes, which is generally not the case in standard Linux.

When hardware interrupts a running process, the assigned interrupt service routine (ISR) handling is outsourced in a kernel process with RT-priority. In this case the ISR preempts all running processes to only start a process, which is much faster.

D. Accessing the Current System Time

The knowledge of current system time is essential for precise measurement. It can be achieved by using timing hardware or software functions. In Linux-based systems, the Kernel provides three different approaches of retrieving current system time to measure time periods [25].

jiffies counter depends on Linux timekeeping architecture frequency. Usually, in Kernel 2.6.x, the default value is $1000Hz$. It is a 64 bit (platform independent) value and has a resolution in milliseconds.

get_cycles() returns the current value of a 64 bit CPU counter register. This register increments every CPU cycle and thus the value depends on its frequency and CPU design. Using this function on a system which has no counter register implemented, it returns 0.

do_gettimeofday() returns a data structure with current value in seconds and microseconds of the system. Depending on system implementation it uses the jiffies counter or a hardware counter. The resolution is "near microseconds" when a hardware counter is used.

When external hardware - like timers are used, it must be ensured that a fast link between these devices and the CPU exists. The specific link delay depends on the bus architecture and bus speed.

do_gettimeofday() produces overhead by using functions to convert CPU-cycles in seconds and microseconds. The jiffies counter is no option in this approach, due to its low resolution in milliseconds.

In this measurement approach the get_cycles()-function is used. It produces the slightest overhead, because of direct hardware access and has the highest resolution, which only depends on CPU-frequency. To get the delay ($t_{latency}$) in human readable form, the latency which is represented as $\Delta cycles_{CPU}$, will be transformed according to equation 1. This calculation can be done in Userspace.

$$t_{latency} = \frac{1}{f_{CPU}[GHz]} * \Delta cycles_{CPU} \tag{1}$$

The measurement is running on an Intel Atom architecture and therefore the count register increments on a constant rate [26]. To make sure CPU-throttling takes no effect on the measurement, all ACPI-functions have to be disabled.

E. Modifying Ethernet Frames

To get the interval between sending and receiving point in time, these points must be assigned to an Ethernet frame. One solution is storing timestamps and frame sequence numbers in Kernelspace, whereas another solution is adding timestamps of sending and receiving point in time to the current frame. Due to the lightweight approach, storing information in Kernelspace is not the preferred way. Instead modifying the frames is the aimed method to assign timestamps to a frame. There is another advantage in this approach. Measurement functionality can be done promptly without using the long way of storing and reading information out of Kernelspace.

Ethernet frames are tagged with the start value shortly before they are copied to the hardware controller. On the receiving side, getting the timestamp is accomplished as soon as the hardware interrupts. Afterwards, the value is copied when the frame data is completely accessible in Kernelspace.

In a Userspace application the TT measurement frames are assembled with Linux raw sockets. Raw sockets are used to bypass the layer three and four handlers and thus allow direct access to the whole Ethernet frame. Figure 3 depicts a complete measurement frame.

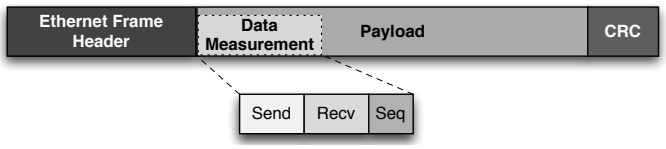

Fig. 3. Complete TT-measurement frame, where Send is the send point in time, Recv is the receiving point in time and Seq is the sequence number.

The first bytes represent the sending point in time in CPU-cycles and are followed by the receive point in time. Additionally, a sequence number is added to get information about data flow and to identify a frame, when external tools are used for validation purpose. For this work the remaining payload does not contain useful data. Anyway, the concept can be expanded to a quality of service protocol (QoS) inside a RT application.

The result is then produced in the same application, by accessing Ethernet frames via raw sockets.

F. RT-Priorities

As the real-time Kernel patch allows assigning RT priorities for applications and ISR-threads, they have to be adapted for this approach. The involved measurement applications and ISRs have to be assigned with higher priorities than others, in order to avoid preemption and resulting inaccuracy.

G. Disabling the System Management Interrupt

Under unfavorable circumstances problems may occur due to the system management module while a system management interrupt takes CPU time and preempts all running processes including the OS. Usually, these types of interrupts are only executed if system safety functions (shutdown on high CPU temperature) are handled. Thus, the system management

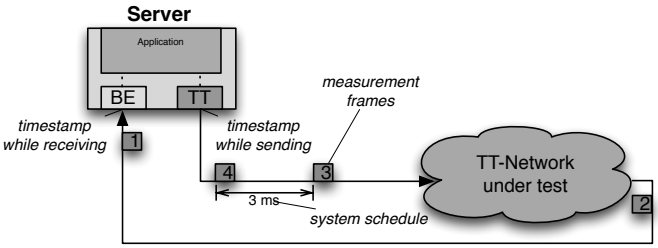

Fig. 4. The test setup of the used example measurement consists of a $3ms$ cycle.

interrupt has to be disabled. If deactivation of the system management interrupt is disallowed, Linux provides a kernel module to track the elapsed time [27].

H. Synchronization

In contrast to performance analyses of best-effort traffic, packet generation has to be synchronized with the schedule for time-triggered communication. Packages sent out of schedule will be dropped by the TTEthernet switch. To achieve in-schedule frames, a time-triggered protocol stack implementation has to be applied. In this paper the stack of a TTEthernet evaluation system by TTTech is used. It is compiled for the RT Linux Kernel and provides adapted device driver to comply with the protocol stack. The addressed driver modification to insert timestamps has to be adjusted to it.

V. Measurement Validation

To evaluate the presented measurement approach, a sample configuration test setup is measured and verified against an Ethernet analyzer and a mathematical framework.

A. Measurement Test Setup

The test setup for a sample measurement is depicted in figure 4. An end system called server sends 100 measurement frames within the system cycle configuration. A TTEthernet schedule with a $3ms$ cycle duration is used. All measurement frames are transmitted at a predefined point in time and are sent via the TT device driver and received via the BE device driver. The reception is done in an event triggered way to avoid further delays. Additionally, the BE device has to be configured in promiscuous mode to receive all frames, because of the TT frame destination address format.

The server is an embedded PC *IBX 530* by iEi technology [28], composed of an Intel 1.6 GHz Atom CPU and two Realtek 8168 network interface card (NIC) chipsets. An Ubuntu Linux 8.04 LTS distribution with the preliminary discussed real-time patch for Kernel $2.6.24$ is used.

This test setup allows end-to-end latency measurement of any TT-Ether-Network. A TTEthernet switch is used to represent a "Time-Triggered Ether-Network under test" and measure its latency. To keep close to standard Ethernet benchmarking, frames are transmitted according to RFC-2544 [29].

The result of the sample measurement is depicted in figure 5. A gap between 127 and 128 byte frame size can be seen

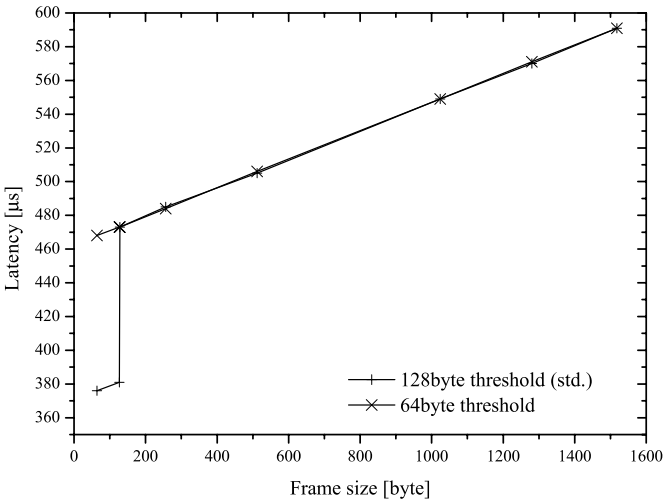

Fig. 5. Measurement results of standard configuration with 128 byte threshold in comparison with the modified configuration with 64 byte threshold.

where the latency jumps up of $90\mu s$. This gap is a result of the used configuration in the BE-device driver. The jitter (in this paper described as distance of minimum and maximum delay for a defined frame size within a measurement) is also related to this gap. On the interval between 64 and 127 byte frame size the jitter bounds in $10\mu s$ whereas on the interval between 128 and 1518 the jitter bounds in $30\mu s$.

The results of this measurement depend on the used system schedule configuration. TT-frames are delayed for a pre-configured amount of time in the TTEthernet-switch to avoid inaccuracy in transmission. This means TT-frames are allowed to be received in a window within the system cycle. In this 3ms cycle configuration TT-frames arrive on $800\mu s$ on the incoming port at the TTEthernet switch and are relayed at $1150\mu s$. This relay delay will be defined during network configuration. The delta of $350\mu s$ can be seen as a static switch relay delay. The propagation delay of an Ethernet frame for 100Mbit/s Ethernet is $0.08\frac{\mu s}{byte}$. It is a linear function depending on frame size and can be seen as a predictable delay.

The gap of $90\mu s$ in figure 5 is a result of the used BE driver configuration. In standard Linux Kernel configuration, the NIC copies the frame data to host memory, when a frame is completely received [30]. After modifying the configuration with a hard threshold at 64 byte, where data is copied to the host, a linear gradient is achieved (see figure 5). Therefore an assigned hardware and driver delay (t_{HWD}) for the used Realtek network hardware is assumed, which can be calculated with equation 2.

$$t_{HWD} = t - t_{PD} - t_{SD} \tag{2}$$

Where t is the measurement result, t_{PD} the predictable propagation delay of 100 Mbit/s and t_{SD} is the static switch configuration delay.

This delay has to be subtracted from every measured result to get a fair and precise end-to-end latency measurement.

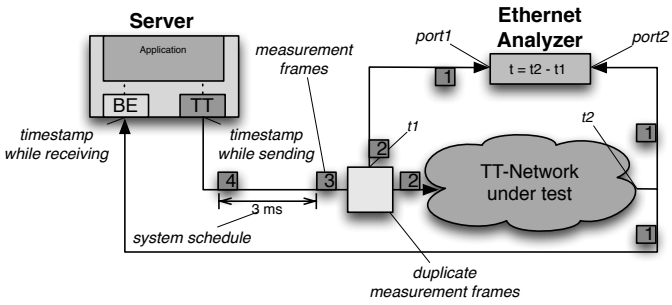

Fig. 6. The test setup for approach validation with an Ethernet performance analyzer.

For jitter bounds measurements, frame sizes $\leq$ 127 byte are adequate. The jitter complies with the given synchronization precision of the used network under test which is in single-digit microseconds range.

B. Validating the Measurement Approach

To validate the results of the measurement setup a test setup using a standard hardware analyzer for Ethernet is built up and a mathematical framework of the TTEthernet forwarding components [31] is used.

1) Hardware Sniffer: For validation with hardware an Ethernet performance analyzer by IXIA [5] is used. This device has several Ethernet ports and a synchronized time base in nanoseconds. In general, one port is used as a packet generator and another as a consumer. Since the IXIA 1600T does not support the TTEthernet synchronization to generate in-schedule frames, this approach cannot be used. Instead a further test setup given in figure 6 is applied. Basically, it is the same way as presented in [6] to measure RT-Ether-Networks. The server is still used as an in-schedule-producer and both ports on IXIA as consumer.

The first port is located right before the TT-network-under-test, the second behind it, so it is possible to measure the distance of one frame. Additionally, a standard Ethernet switch to duplicate measurement frames had to be included. This is realized due to Multicast destination addresses, a frame will be relayed on all connected ports except the one which received it. For precise measurement it has to be ensured that the frame will be relayed on all linecards simultaneously. The results of the validation are comparable to the results of the discussed approach (see table I) apart from the additional delay of the duplication switch (approx. $9\mu s$). A further validation improvement can be an Ethernet tap, which is used in [6] and has a delay less than $10ns$.

2) Mathematical Framework: Using the mathematical framework of TTEthernet shown in [31], the approach presented in this paper will be verified again. This framework allows calculating latency, bandwidth and jitter independently of the hardware implementation. To calculate the switch latency equation 3 is used:

$$t_L = t_{WD} * l_W + n_s * l_F * t_b + \sum_{i=1}^{n_s} t_{SD_i} \quad (3)$$

The total latency t_L consists of the wire delay $t_{WD} * l_W$ for all wires between sender and receiver. The time for the message transmission for each switch $n_s * l_F * t_b$ (n_s number of switches, l_F length of a frame and t_b propagation delay) and the scheduled delays in each switch $\sum_{i=1}^{n_s} t_{SD_i}$. The delay t_{SD_i} can be calculated as the difference of send point in time and receive point in time for the switch. Since the wires in the experiment were very short ($0.5m$) the signal propagation delay $t_{WD} * l_W$ is insignificant ($10\ \frac{ns}{m} * 0.5\ m = 5ns$).

Equation 4 and 5 provide the frame propagation delay for maximum and minimum payload.

$$\begin{aligned} t_{FPD_{max}} &= l_F * t_b \quad (4) \\ &= 1518byte * 8\frac{bit}{byte} * 0.01\ \frac{\mu s}{bit} \\ &= 121.44\mu s \\ t_{FPD_{min}} &= 64byte * 8\frac{bit}{byte} * 0.01\ \frac{\mu s}{bit} \quad (5) \\ &= 5.12\mu s \end{aligned}$$

The last part for the equation is the sum of relay delays of all switches $\sum_{i=1}^{n_s} t_{SD_i}$. It only consists of the delay that is based on the schedule of our switch under test t_{SD_0}. In our experiment it is $350\mu s$. According to equation 3 the switch latency can be calculated:

$$\begin{aligned} t_{L_{max}} &= t_{WD} * l_W + n_s * l_F * t_b + \sum_{i=1}^{n_s} t_{SD_i} \quad (6) \\ &= 10\ \frac{ns}{m} * 0.5m + 1 * 121.44\mu s + 350\mu s \\ &= 471.445\mu s \\ t_{L_{min}} &= 10\ \frac{ns}{m} * 0.5m + 1 * 5.12\mu s + 350\mu s \quad (7) \\ &= 355.125\mu s \end{aligned}$$

As expected a linear dependency on the frame length can be seen. The results of the mathematical model comply with the measured latency (see table I).

To give an overview of the realized results table I shows the measured latencies for a minimal The difference of $9\mu s$ relies on the used duplication switch.

	measurement approach	**hardware sniffing**	**mathematical model**
min. framelength	$355\mu s$	$364\mu s$	$355.125\mu s$
max. framelength	$471\mu s$	$479\mu s$	$471.445\mu s$

TABLE I
ALL VALIDATION RESULTS IN COMPARISON

VI. Measurement Results

The results of the measurement based on the setup given in figure 4 will now be presented. First, the effect of the employed TTEthernet schedule on message transmission is depicted. Second, the TTEthernet switch is compared to a standard consumer components-off-the-shelf (COTS) switch, which does not provide priority-driven transmission.

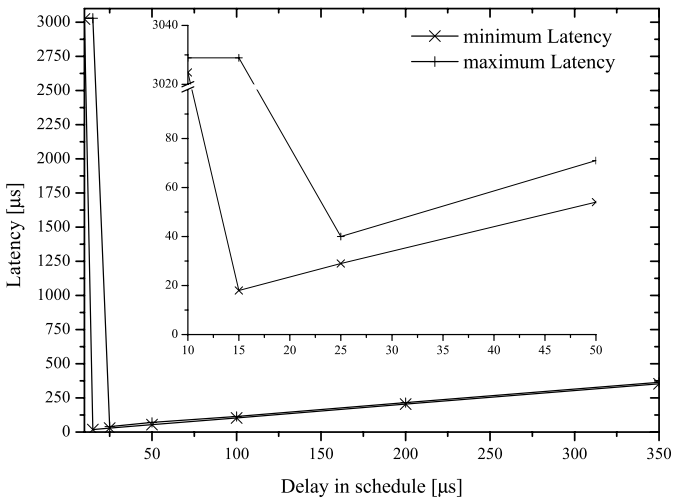

Fig. 7. Minimum and maximum latency for a 64byte frame with different delays in schedule.

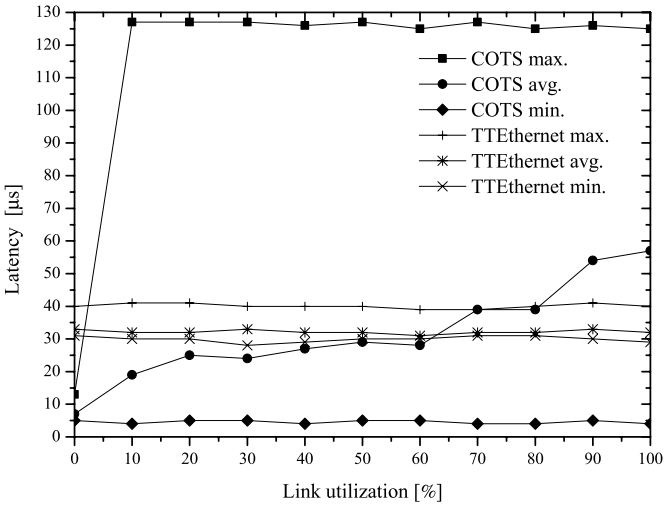

Fig. 8. Latency comparison 100Mbit COTS switch and TTEthernet switch, time-triggered traffic with minimum payload and varying best-effort link utilization.

A. Influence of Scheduling

The main variable for the latency of a time-triggered message is the schedule of the TTEthernet switch. The schedule determines when a time-triggered message is forwarded. To demonstrate the impact of the schedule on the latency, time-triggered messages with minimum frame size are transferred while the switch is configured with schedules with relay delays between $10\mu s$ and $350\mu s$.

Figure 7 shows the minimum and maximum latency for a measurement of 100 frames. In general, the scheduled delay has a linear effect on the latency. Jitter is within our expected range of $10\mu s$.

For short relay delays, the relaying is not reliable anymore. Since the synchronization has a finite precision, the sender misses its deadline and the frame is forwarded in the next cycle. For the schedule with $25\mu s$, the deadline is reached reliably. The schedule with $15\mu s$ is a turning point where the deadline is only reached most of the time. For smaller relay delays, the deadline is always missed.

For the turning point ($15\mu s$), packet loss is experienced. Packet loss happens when a frame misses and the following frame reaches the deadline. Then the following frame overwrites the message in the buffer and only the new frame is relayed. For the results in this paper, the switch was configured to relay frames tolerantly. With an acceptance window for incoming time-triggered frames the switch may be configured to drop messages that arrive too late to meet their deadline.

B. Comparing a Standard and a TTEthernet Switch

TTEthernet provides a reliable transmission with predictable latency for critical traffic. The behavior of a TTEthernet switch is compared to a standard COTS switch when the link utilization of best-effort traffic increases. In this measurement, the transmission of critical traffic is accomplished with 100 frames for each measurement, 64 byte frame size and a $25\mu s$ relay delay. The link utilization is applied with full size best-effort frames. Figure 8 presents the minimum, maximum and the average latency as a function of link utilization.

The results for the TTEthernet switch remain constant, whereas the maximal values at the COTS switch reach $126\mu s$ with 10% utilization. The COTS switch queues frames independently of the message priority. Critical traffic will be delayed for the amount of a full size best-effort frame duration, which is $122\mu s$. Since the COTS switch does not provide priority-driven transmission of critical traffic, the probability of collisions on outgoing linecards increases with the link utilization. This effect can be seen in COTS average latency plotted in figure 8, which increases in dependency of link utilization.

Figure 9 shows the latency distribution for the TTEthernet and COTS switch. This analysis reveals the characteristic differences of the switches. The decision whether the forwarding matches the requirements of the application can be supported by latency distribution graphs.

For the TTEthernet switch, the time-triggered messages latency is within the expected precision. Due to queuing, the latency of the COTS switch is widely distributed and therefore a predictable message transmission of critical traffic cannot be derived.

VII. Conclusion & Outlook

In this paper, an approach to measure end-to-end latency of time triggered Ether-Networks with off-the-shelf-components was presented for costs less than 500$. This lightweight synchronized packet generation tool for analyzing and verifying RT-networks is realized by a RT Linux Kernel, modified device drivers and an implementation of a time-triggered Ethernet protocol stack, developed by TTTech.

The measured end-to-end latency of TT-Ether-Networks under test depended on the used system-cycle, hardware driver configuration and the frame size. The resulting hardware and

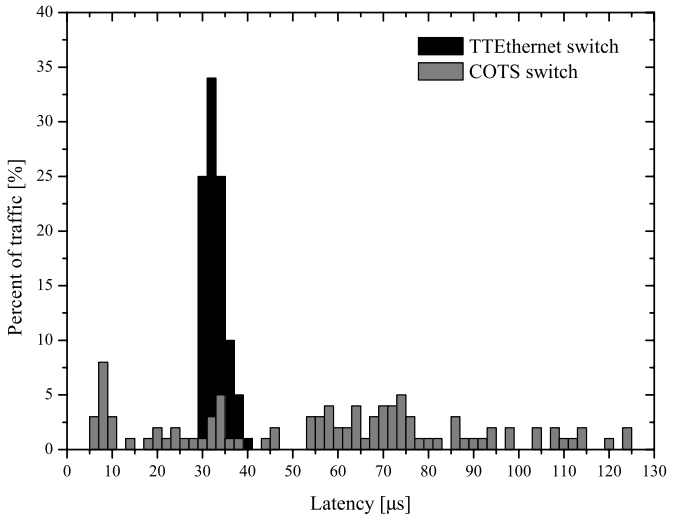

Fig. 9. Latency distribution comparison 100Mbit COTS switch and TTEthernet switch.

driver delay had to be subtracted from the measured latency in order to be precise.

For verification purpose, a compatible test setup with a hardware sniffer and a mathematical framework was applied. The results of the approach discussed in this paper confirm to the direct measurements performed in hardware, as well as to values calculated from the framework.

The presented measurement results of TTEthernet and COTS hardware clearly demonstrate the suitability of the provided approach for evaluating of time-triggered EtherNetworks.

Generally, the results promise exact end-to-end latency measurement, jitter measurement can only be realized by using frame sizes below 128 byte. Furthermore, non-maskable interrupts can corrupt this measurement approach, because they have to be executed immediately in critical and unfrequented situations like parity error on the memory bus occurs.

In future work, we will validate our approach using different hardware and hope to overcome the issues imposed by the driver and network chipset. Hardware with a specifically low receive and copy delay should be favored. Furthermore, we plan to adapt our approach to an ARM9-based microprocessor without operating system, which provides a CPU counter register, too. Due to control of all I/O and ISR executions, a higher precision should be achievable from this approach.

VIII. Acknowledgment

The authors would like to thank Andreas Büker and Fujitsu Technology Solutions GmbH for supporting the validation with their hardware and their knowledge of Ethernet measurement.

References

[1] W. Steiner, "TTEthernet Specification," TTTech Computertechnik AG, Nov. 2008. [Online]. Available: http://www.tttech.com

[2] TTTech Computertechnik AG, "," Wien. [Online]. Available: http://www.tttech.com

[3] Honeywell International. [Online]. Available: http://www.honeywell.com

[4] SAE - AS-2D Time Triggered Systems and Architecture Committee, "Time-Triggered Ethernet (AS 6802)," 2009. [Online]. Available: http://www.sae.org

[5] Ixia, "Ixia - 1600T Traffic Generator / Performance Analyzer." [Online]. Available: http://www.ixiacom.com

[6] P. Ferrari, A. Flammini, D. Marioli, and A. Taroni, "A Distributed Instrument for Performance Analysis of Real-Time Ethernet Networks," *Industrial Informatics, IEEE Transactions on*, vol. 4, no. 1, pp. 16–25, Feb. 2008.

[7] J. Loeser and H. Haertig, "Low-latency hard real-time communication over switched Ethernet," in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, Jun. 2004, pp. 13–22.

[8] R. A. de M. Valentim, A. H. F. Morais, G. B. Brandão, and A. M. G. Guerreiro, "A performance analysis of the Ethernet nets for applications in real-time: IEEE 802.3 and 802.3 1 Q," in *6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008.*, Jul. 2008, pp. 956–961.

[9] Wireshark, 2010. [Online]. Available: http://www.wireshark.org

[10] I. Schafer and M. Felser, "Precision of ethernet measurements based on software tools," in *IEEE Conference on Emerging Technologies and Factory Automation, 2007. ETFA.*, Sep. 2007, pp. 510–515.

[11] J. K. Strosnider, T. Marchok, and J. Lehoczky, "Advanced real-time scheduling using the IEEE 802.5 token ring," in *Real-Time Systems Symposium, 1988., Proceedings.*, Dec. 1988, pp. 42–52.

[12] L. Yao and W. Zhao, "Performance of an extended IEEE 802.5 protocol in hard real-time systems," in *INFOCOM '91. Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s., IEEE*, Apr. 1991, pp. 469–478.

[13] A. P. Jayasumana and G. G. Jayasumana, "On the use of the IEEE 802.4 token bus in distributed real-time control systems," *Industrial Electronics, IEEE Transactions on*, vol. 36, no. 3, pp. 391–397, Aug. 1989.

[14] EtherCAT Technology Group, "EtherCAT." [Online]. Available: http://www.ethercat.org

[15] Aeronautical Radio Incorporated, "Aircraft Data Network," ARINC, Annapolis, Maryland, Standard 664, 2002.

[16] International Organization for Standardization, "Road vehicles – Controller area network (CAN) - Part 4: Time-triggered communication," ISO, Genf, ISO 11898-4:2004, 2004.

[17] FlexRay Consortium, "Protocol Specification," FlexRay Consortium, Stuttgart, Specification 2.1, Dec. 2005.

[18] PROFIBUS & PROFINET International, "Profinet," Karlsruhe. [Online]. Available: http://www.profibus.com/pn/

[19] SynqNet Interest Group, "Synqnet." [Online]. Available: http://www.synqnet.org/

[20] T. Ts'o, D. Hart, and J. Kacur, "RT-Kernel Wiki," 2010. [Online]. Available: https://rt.wiki.kernel.org/index.php/Main_Page

[21] D. Abbot, *Linux for embedded and real-time applications - 2nd Edition*. Butterworth Heinemann, May 2006.

[22] RTAI Team, "RTAI - the RealTime Application Interface for Linux from DIAPM," 2010. [Online]. Available: http://www.rtai.org

[23] Xenomai, "Real-Time Framework for Linux," 2010. [Online]. Available: http://www.xenomai.org

[24] K. Yaghmour, J. Masters, G. Ben-Yoseff, and P. Gerum, *Building Embedded Linux Systems - 2nd Edition*. O'Reilly Media, Inc., Aug. 2008.

[25] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers, Third Edition*. O'Reilly Media, Inc., 2005.

[26] Intel, "Intel 64 and IA-32 Architectures Software Developer's Manual," Intel, Tech. Rep., Sep. 2010.

[27] Jon Masters, "SMI Detector - A simple module for detecting System Management Interrupts," 2009. [Online]. Available: http://lkml.org/lkml/2009/4/2/426

[28] IEI Technology Corp. [Online]. Available: http://www.ieiworld.com/

[29] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," IETF, RFC 2544, Mar. 1999.

[30] Realtek Semiconductor Corp., "Integrated Gigabit Ethernet Controller For PCI-Express Applications - Registers Datasheet," Realtek Semiconductor Corp., Hsinchu, Taiwan, Tech. Rep., Apr. 2006.

[31] T. Steinbach, F. Korf, and T. C. Schmidt, "Comparing Time-Triggered Ethernet with FlexRay: An Evaluation of Competing Approaches to Real-time for In-Vehicle Networks," in *8th IEEE Intern. Workshop on Factory Communication Systems*. Piscataway, New Jersey: IEEE Press, May 2010, pp. 199–202.