ABSTRACT
Cluster simulation is a popular method for supporting system integration in various distributed applications by simulating the environment of a subsystem under test. Particularly in real-time systems, the timing requirements of transmission and reception must be fulfilled, which is not easy to achieve. In this paper, we contribute a scheme for cluster simulation of real-time Ethernet (RTEthernet) based distributed systems. It relies on the discrete event-based simulation framework OMNeT+++, interconnected with an ARM-based co-processor. Our approach allows coupling a real-world RTEthernet subsystem with virtual components running in the discrete simulation, that realise the required behaviour for the subsystem. We have evaluated the performance limits of our approach regarding latency and jitter, when running the simulation on a Linux system with the real-time Kernel patch. The results show that the timing requirements for the cluster simulation of small RTEthernet networks can be achieved.

Categories and Subject Descriptors
I.6.3 [Simulation and Modeling]: Applications

General Terms
Measurement, Performance, Experimentation

Keywords
Real-time Ethernet, Cluster Simulation, OMNeT++

1. INTRODUCTION
While simulation is already established in the design and reconfiguration phase of large distributed real-time systems, it is equally useful during the integration and setup phase. Usually, when a system is being integrated, parts of the network cluster must be tested, while its environment is not available in hardware. These cluster simulations generally use real-time simulation platforms that are specifically designed and require expensive hardware. They are specialised for a specific use-case and are inflexible to adapt to varying conditions. Further, such systems are designed for a dedicated communication protocol and thus are not feasible for design changes on the protocol itself.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

OMNeT++ 2013 March 5th, Cannes, France. Copyright 2013 ACM ...$10.00.
haviour has to be done by analysing the received frames.

**Hardware-in-the-Loop Simulation:** When the reaction of the simulated nodes on the real environment of the SUT shall be monitored, the simulator has to be connected to the sensors and actuators of the SUT. The SUT is connected to the HIL simulator via the communication interface and an additional environment interface. This enables also a verification of the actual technical behaviour but requires high computation performance.

The two depicted approaches allow verifying the behaviour of a SUT or a single participant inside a distributed real-time system. But at the same time they require expensive high performance hardware and real-time operating systems to fulfil the requirements of a real-time simulation. In contrast to the presented solutions, our approach is based on off-the-shelf hardware and a flexible event-based simulation environment. Our implementation can be used variably for different purposes, such as early network protocol analyses, or prototype application implementations. On the other side, the previously depicted (commercial) cluster and HIL simulations are often limited to special use-cases, to be able to meet the demanded timing requirements. Hence, they have higher simulation performance and timing precision.

### 2.2 Real-time Ethernet and TTEthernet

The advent of more and more bandwidth demanding real-time applications has paved the path for Ethernet-based real-time protocols. The main challenge for real-time communication over Ethernet is to reduce and limit latency and jitter. Time-triggered systems achieve a very low jitter by preventing frames from concurrently accessing the same line card. They operate synchronised with a shared time-base. According to a coordinated time division multiple access schedule, each sender is allowed transmitting data only in its offline assigned time slots.

Our cluster simulation approach relies on the TTE4INET model [4] for OMNeT++ and a RTEthernet stack for microcontrollers [2], which are both implementations of the TTEthernet protocol [8]. TTEthernet is able to transmit time critical and best effort messages on the same physical infrastructure and provides three different message classes. **Time-triggered** (TT)-messages have the highest priority and the strictest timing constraints. To allow critical event-triggered messages, **rate-constrained** (RC)-messages have the second highest priority. Rate-constrained transmission is done in a *bandwidth limiting* manner. **Best-effort** (BE)-messages conform to standard Ethernet, have the lowest priority and no guaranteed transmission. The required system wide time base for time-triggered communication is accomplished by a fail-safe clock synchronisation protocol. The timing requirements for nodes in a TTEthernet networks rely on the synchronisation accuracy, which has to be in the lower microseconds range.

To be able to connect the simulation to a physical RTEthernet network, the simulation platform must provide complete support of the used protocol. In our case, it is essential that the platform has to be synchronised to the network, in order to transmit TT-messages in the correct time slot. Otherwise, they will be dealt as frames sent by a faulty device and will be dropped by the network. Also the distinction between critical and best-effort messages must be achieved, since low priority messages are not allowed interfering with high priority critical messages. In order to react predictably to received frames, the latency between the actual reception of frames at hardware level and the simulation environment has to be bounded.

### 2.3 OMNeT++ Related Work

During the development of a simulation model for the Stream Control Transmission Protocol (SCTP), an external network interface module was developed [6] that allows communication from the simulation environment to real networks. To receive external messages, the packet capture library libpcap was used, which allows creating filters to capture messages. To send messages from the simulation to physical nodes, raw IP sockets were used. In addition, the simulation was synchronised with the real time by obtaining the time of the next scheduled event. In contrast to this approach, we are using dedicated hardware for the transmission and reception of frames, since the accuracy of the operating systems build-in scheduler is not sufficient.

Another approach was used for the simulation of smart grid communication systems [1]. In order to avoid delays in the transmission of messages to and from the simulation, a new simulation kernel has been implemented that uses two processes on distributed CPU cores. One process involves the simulation with the event scheduling. The second process is responsible for the communication with the external systems. The communication between these processes is realised with the standard Message Passing Interface (MPI). The application of MPI enables to transfer critical simulation sections to dedicated cores. This allows a better performance and at the same time a higher precision, which makes it very attractive for our scheme.

In contrast to these approaches, our concept is forced to reliably provide real-time capabilities, since a delayed frame transmission results in incorrect behaviour of the SUT and the simulation.

### 3. CONCEPT & ARCHITECTURE

Cluster simulation of RTEthernet protocols requires high temporal precision regarding the transmission and reception of critical messages. Therefore, an accurate knowledge of latency and jitter of the simulation hardware is of great importance.

#### 3.1 Description of the Developed Platform

Our concept is based on the components shown in Fig. [1] Since OS-based RTEthernet stacks do not supply the required temporal precision in transmission, we are using an implementation for an ARM9-based system-on-chip [2]. The
platform provides separate communication channels, which are working independently of the ARM-CPU. This architecture permits parallel processing of frames and applications. In order to allow communication between both platforms, the host is connected via virtual dual-port-memory (DPM) to the microcontroller. Memory areas and registers of the microcontroller can be mapped to the memory of the Linux system. Handshake registers and an interrupt line control the synchronous data exchange.

In our approach we use a Linux operating system with an open source Userspace I/O framework to control the PCI-based DPM device. This mainline Kernel module allows the user to write a device driver running primarily in Userspace. The only task to be implemented in Kernelspace is the handling of interrupts. A small Kernel module containing a minimal ISR is required, that acknowledges or disables the interrupt and triggers the Userspace process. In general, Linux is not suitable for real-time applications; therefore we use the RT Kernel patch [5].

3.2 Overview on the Architecture

**TimeStampping of Frames:** For the classification of messages in RTEthernet networks, precise information about the reception of periodic time-triggered messages is required to determine whether a frame was received in time. Therefore, timestamps of received messages must be taken at the earliest possible time, which is at the receiving hardware unit. The microcontroller features a timestamping unit (see TU in Fig. 1), which stamps all frames with a resolution of $10^{-6}$s and appends these time stamps to the frames.

**Simulation with a Real-Time Scheduler:** OMNeT++ provides three different schedulers that are responsible for assigning events to the appropriate module at the scheduled point in simulation time. The default scheduler is the cSequentialScheduler. This scheduler selects the first element in the event queue and then executes the associated routine. Thus, simulation time continues without any relation to the real time. The second scheduler is the cRealTimeScheduler, which uses the system clock in wait calls to synchronise simulation time to real time. cSocketRTScheduler is the third scheduler and extends the cRealTimeScheduler. It waits for incoming messages from an external device during wait calls. As all these schedulers handle the events sequentially, deviations between the simulation time and the real time may occur, depending on the computing time required for the simulation. Furthermore, communication with external devices, such as network interface cards, can result in additional delays since the scheduler has to wait, until the device has finished its processing. To avoid these delays, the simulation and the communication module are distributed on separate CPU cores to run in independent parallel processes (see Simulation and DPMCommunicator in Fig. 1).

Time-triggered Ethernet expects a synchronised time of all components that receive and send time-triggered messages. Therefore, the scheduler of the simulation requires a view on the synchronised hardware clock of the system-on-chip to receive and send critical messages. The microcontroller allows for mapping its time registers via DPM to the hosts memory, which permits applying the synchronised clock to the simulation time. We implemented a new cDpmRtScheduler that extends the cRealTimeScheduler such that the simulation time will be synchronised to the real time of the microcontroller.

**Sending Real-World Frames via the External Microcontrollers Scheduler:** The demanded temporal requirements cannot be fulfilled when utilising the Linux network stack for Ethernet, because of its completely fair frame processing. Therefore we are using the microcontrollers internal high-resolution scheduler for sending frames from the simulation to the real-world network. This approach allows us to schedule the transmission of messages to external receivers with a resolution of $10^{-6}$s. If a message object is addressed to an external node, it is first translated into a network compliant message format and afterwards sent by the MPI to the DPMCommunicator process. This process copies the message to the microcontrollers memory and then triggers an ISR on the microcontroller with an interrupt. The message is inserted in the scheduler of the microcontroller and is sent at the predefined point in time.

**Reception of Real-World Frames in the Simulation:** Since frames in time-triggered Ethernet are prioritised according to their message class, the forwarding of frames to the simulation must be prioritised, too. We achieved these priorities by assigning each type of frame a handshake register on the microcontroller. Handshake registers are used to realise the prioritised synchronous data transfer to the host PC. When a frame is received at the microcontroller, an interrupt is generated on the host and it is informed only about the frame with the highest priority. This guarantees that critical frames have always precedence.

The transfer of the frames through the DPM results in a dynamic delay which must be added to the frames time stamp to specify the correct receive time in the simulation. Eq. 1 shows the calculation of the receive time,

$$t_{\text{Sim}} = t_{\text{Frame}} + t_{\text{Controller}} + t_{\text{Host}}$$

where $t_{\text{Sim}}$ is the corrected receive time of the frame in the simulation and $t_{\text{Frame}}$ denotes the delay of reception, copying the incoming frame to the memory and triggering the host with an interrupt. $t_{\text{Host}}$ identifies the transmission delay from the microcontroller to the simulation.

**Simulating with the RT Kernel Patch:** The RT Kernel patch allows assigning RT priorities for applications and ISR-threads. In our approach it is used to prioritise the simulation, the DPMCommunicator module and the DPMISR (see Microcontroller ISR in Fig. 1). These modules are assigned with a higher priority than the rest of the hosts system, to avoid preemption, which would result in additional jitter. Further inaccuracies may be caused by the system management module, when a system management interrupt takes CPU time and all processes including the OS are preempted. Thus, these functions have to be disabled.

4. **EVALUATION & RESULTS**

Our approach is evaluated in a test environment to determine the temporal behaviour. It consists of one physical node that periodically sends TT-messages and one virtual component running the TTE4INET simulation model [4] which receives these messages. As TT-messages have a fixed configured time slot for transmission, the focus of our test setup is based on the analysis of the latency and jitter.

**Latency Measurement between Host and Microcontroller:** To analyse the latency of our approach, the physical component generates 1000 TT-messages per Ethernet frame size in the range between minimum (64 byte) and maximum (1518 byte) frame sizes. The virtual component
records a timestamp for every received message in the simulation. The latency is calculated using the difference between this timestamp and the timestamp that was recorded by the microcontroller. Fig. 2 shows that the cost of our approach is almost linear and has a calculated slope of 0.49 µs per byte. The static part of the latency is measured with 186.4µs and has its origin in the processing of the messages in the microcontroller, the interrupt processing in the Linux Kernel and the bandwidth of the used DPM PCI-Card.

**Jitter Analysis:** The test setup for the jitter analysis is similar to the previously presented latency measurement. Here, the virtual component calculates the difference between the measured cycle times. The measurements show that incoming TT-messages have a maximum jitter of 37µs in the simulation, which is caused by the host system. Fig. 3 depicts two distribution areas of the measured latency. The first area exists around the target point of the cycle that is the zero point and the second one is about 10 µs below zero. The jitter has its origin in the best-effort behaviour of the used off-the-shelf hardware. Thereby, more latency and jitter sources are present such as the DMA bus mastering which inserts wait cycles on the bus and causes latency and jitter to the host.

**Discussion of the Results:** Currently we are using the presented approach to develop and test RTEthernet nodes. The latency and jitter performance is sufficient to obtain reliable results. The simulation of applications that have latency requirements in the range of 230 µs is feasible for minimal sized frames. Furthermore, synchronisation protocols such as used in TTEthernet are able to compensate the jitter to perform a successful synchronisation.

5. CONCLUSION & OUTLOOK

In this work, we showed a concept for cluster simulation of RTEthernet systems. Our platform is based on a standard PC with the RT-Linux Kernel running the RTEthernet models in the OMNeT++ simulation framework and an ARM9 microcontroller as a co-processor.

Our evaluation shows that the platform offers sufficient performance for latency requirements of distributed real-time systems in the range of 230 µs, which is limited and has a linear dependency on the frame size. The observed jitter is below 40 µs.

6. REFERENCES


