Modeling Time-Triggered Ethernet in SystemC/TLM for Virtual Prototyping of Cyber-Physical Systems

Zhenkai Zhang and Xenofon Koutsoukos

Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN, USA
{zhenkai.zhang,xenofon.koutsoukos}@vanderbilt.edu

Abstract. When designing cyber-physical systems (CPS), virtual prototyping can discover potential design flaws at early design stages to reduce the difficulties at the integration stage. CPS are typically complex real-time distributed systems which require networks with deterministic end-to-end latency and bounded jitter. Time-triggered Ethernet (TTEthernet) integrates time-triggered and event-triggered traffic, and has been used in many CPS domains, such as automotive, aerospace, and industrial process control. In this paper, a TTEthernet model in SystemC/TLM is developed to facilitate the design and integration of CPS. The model realizes all the necessary features of TTEthernet, and can be integrated with the hardware platform model for design space exploration. We validate the model by comparing latency and jitter with those obtained using a commercial software-based implementation. We also compare our model with the TTEthernet modeled in OMNeT++ INET framework. Our model provides startup and restart services that are necessary for maintaining synchronized operations in TTEthernet. We evaluate these services and also the efficiency of the simulation.

Keywords: TTEthernet, SystemC, TLM, Virtual Prototyping

1 Introduction

Cyber-physical systems (CPS) are complex heterogeneous systems whose design flow includes three layers: the software layer, the network/platform layer, and the physical layer [1]. The interactions within and across these layers are complex. The physical layer interacts with the hardware platform through sensors and actuators. Embedded software runs on the hardware platform and communicates via a network to realize the desired functionalities. Due to the high degree of complexity, design flaws often appear at the integration stage. In order to discover potential design flaws at early stages, a virtual prototyping development approach is required.

In virtual prototyping of CPS, modeling the hardware platform in a System-Level Design Language (SLDL) is essential to quickly evaluate the interactions between the platform and the software layer and the physical layer at early design stages. Since CPS are typically distributed real-time systems, the network also plays an important role in design and integration.

In many CPS domains that require known and bounded network latency, such as automotive, aerospace, and industrial process control, time-triggered Ethernet (TTEthernet) has been used for real-time communication. Traditional Ethernet cannot be used, since it suffers from cumulative delay and jitter. TTEthernet integrates time-triggered traffic and event-triggered traffic together, and provides the capability for deterministic, synchronous, and lossless communication while supporting best-effort traffic service of Ethernet at the same time [2].

SystemC, which has become a *de facto* SLDL [3], is proposed to be one main part of virtual prototyping of CPS [4]. It allows system modeling and simulation at various levels of abstraction. In addition, the concept of transaction-level modeling (TLM) is adopted in SystemC to separate the computation and communication. A TLM communication structure abstracts away low-level communication details while keeping certain accuracy. Thus, both the software layer and the network/platform layer can be modeled in SystemC/TLM at early design stages making it suitable for virtual prototyping.

In this paper, we describe a TTEthernet model in SystemC/TLM for virtual prototyping in order to take into account the network effects in a CPS. The model in SystemC/TLM offers many advantages: (1) it is easy to acquire at early design stages; (2) it is scalable to a large number of nodes; (3) the model can be integrated with the hardware platform model in a straightforward manner; (4) it provides efficient and accurate simulation.

The main contribution of this work is a TTEthernet model in SystemC/TLM that realizes all the necessary features for facilitating the design and integration of CPS. The model is validated by comparing latency and jitter with those obtained using a commercial software-based implementation and the model in OMNeT++ INET framework [5]. The model is also evaluated for its startup and restart services and the simulation efficiency.

The rest of this paper is organized as follows: Section 2 gives the related work including TTEthernet and related modeling efforts. Section 3 describes the model in detail. Section 4 validates the model against a real implementation and the model in OMNeT++ INET framework and also evaluates the services in the model and the simulation efficiency. Section 5 concludes this paper.

2 Related Work

Time-triggered architecture (TTA) has been widely used in safety-critical CPS, which require reliable time-triggered communication systems, such as TTP/C, FlexRay, and TTEthernet [6]. Compared to the maximum bandwidth of TTP/C (25Mbit/s) and FlexRay (10Mbit/s), the bandwidth of TTEthernet can reach 100Mbit/s or even 1Gbit/s, making it very attractive in many CPS domains. As

mentioned in [7], there are two versions of TTEthernet. The academic version uses preemption mechanism and only supports time-triggered (TT) and event-triggered (ET) traffic, while the industrial version uses non-preemptive integration of TT and ET and divides ET into rate-constrained and best-effort traffic classes. In [8], the academic version of TTEthernet is introduced to integrate TT and ET traffic together. In [9], an academic version of TTEthernet switch is developed which preempts ET message transmission when a TT message arrives to guarantee a constant transmission delay of TT messages caused by the switch regardless of the load of ET traffic on the network. In [10], a prototypical TTEthernet controller is described and implemented in an FPGA. TTTech Computertechnik AG company issued the TTEthernet specification in [11] and also developed industrial products [12]. Finally, the TTEthernet specification is standardized by SAE in [2].

Modeling TTEthernet has been used to simulate in-vehicle communication systems. In [5], an extension to the OMNeT++ INET framework is made to support simulation of TTEthernet. The model is based on standard Ethernet model in the INET framework. Although the evaluation shows the model is in good agreement with real implementation, the model does not consider different protocol state machines for different roles of synchronization, which results in some services of TTEthernet are simplified or not supported.

In order to support Ethernet networks in the system-level design in a SLDL, various models/approaches have been proposed. In [13], a half-duplex Ethernet based on CSMA/CD MAC protocol is simply modeled using SpecC and TLM techniques. In [14], an Ethernet interface in SystemC/TLM-2.0 is modeled for virtual platform or architectural exploration of Ethernet controllers. Another approach is to integrate network simulators with simulation kernels of SLDLs. In [15], the NS-2 network simulator is integrated into the SystemC/TLM design flow. The advantage of this approach is that network simulators have a good support for almost every commonly used network. However, such an approach requires the integration of two discrete-event simulation kernels, which can greatly reduce the simulation efficiency.

3 Modeling TTEthernet in SystemC/TLM

3.1 Framework

Our TTEthernet model in SystemC/TLM aims at facilitating the design and integration of the network/platform layer in a CPS, especially if the system is a distributed mixed time-triggered/event-triggered real-time system. As shown in Fig. 1, the network/platform layer consists of several computational nodes which communicate with each other through a TTEthernet network. The TTEthernet model includes two separate parts: the TTEthernet controller and the TTEthernet switch. The network is deployed in star topology or cascaded star topology which uses switches to integrate each star topology.

In each node of the system, a TTEthernet controller communicates with other designed hardware components through a memory-mapped bus. Standard

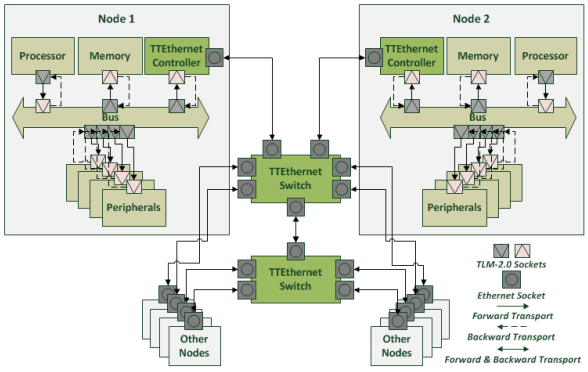

Fig. 1. Network/Platform Layer Design Using TTEthernet Model in SystemC/TLM.

TLM-2.0 sockets are used for this purpose. As a target of TLM, the TTEthernet controller implements a blocking transport interface method for fast but loosely-timed simulation and non-blocking transport interface methods for slow but approximately-timed simulation.

In order to simulate the bidirectional communication link between two ports of the TTEthernet devices, a specific Ethernet socket is used to model the port. As the TLM-2.0 Ethernet socket introduced in [14], our Ethernet socket is a derived class from both initiator and target sockets of TLM-2.0. In order to distinguish different ports of a TTEthernet device, tagged initiator and target sockets are used as base classes of the Ethernet socket. For binding two Ethernet ports, bind() and operator() are overwritten to bind the initiator socket of each port to the target socket of the other port. For invoking transport interface methods, the operator→ distinguishes which socket of a port should be accessed according to the calling method. Since our TTEthernet model uses Ethernet rather than memory-mapped bus, interoperability is not concerned by introducing new transaction type for Ethernet which is similar to the TLM Ethernet payload type introduced in [14].

3.2 Clock Model

In TTEthernet, a synchronized global time is the base for all time-triggered operations. Each TTEthernet device (controller/switch) is driven by a clock having a clock drift. Thus, the clock synchronization service is crucial for the correct operation. In order to simulate its synchronization service, each TTEthernet device needs to have an independent clock with its own drift and offset. However, SystemC uses a discrete event simulation kernel which maintains a global time. If we simulate every tick of a clock with a drift, the simulation overhead will be too large, which can seriously slow down the simulation. Instead, we model the clock as follows: a random ppm value is assigned to each clock in the interval [-MAX_PPM, -MIN_PPM] ∪ [MIN_PPM, MAX_PPM]. According to the time-triggered schedule, the duration in clock ticks from the current time to the time when the next time-triggered action needs to take place is calculated. After that, we can get the duration in simulation time by taking into account its clock

drift: $duration\,in\,simulation\,time = duration\,in\,clock\,ticks \times (tick\,time \pm drift)$, and then we can arrange a clock event with this amount of time by using the notification mechanism of *sc_event* in SystemC.

Because the clock will be adjusted periodically by the synchronization service, the arranged clock event will be affected (its occurrence in simulation time becomes sooner or later). In order to simulate this properly, the arranged clock event and its occurring time in clock ticks is stored in a linked list in an order of occurring time. When a clock event occurs or its time has passed due to clock adjustment, it will be deleted from the linked list and processes pending on it will be resumed. When the clock is corrected, notifications of the arranged clock events are canceled and new simulation times for the notifications of the events are recalculated based on the corrected clock.

A timer model is also built on the clock model, which uses the drift of the clock model to calculate the duration in simulation time and is used for timeout events. In contrast to clock events, timeout events are not affected by clock synchronization and only depend on how many ticks should pass before they occur.

3.3 TTEthernet Traffic Classes

TTEthernet supports three traffic classes: time-triggered (TT), rate-constrained (RC), and best-effort (BE). In order to recognize which traffic class a frame belongs to, either encoding it in the Ethernet MAC destination address or using EtherType field of the Ethernet frame header is feasible [2]. In our model, we employ the destination address divided into two parts to identify critical traffic (CT) including TT and RC. The first part (32 bits) of the destination address shows whether a frame belongs to CT by checking this part against the result of bitwise AND of CT marker and CT mask. The second part (16 bits) gives the CT ID which is used for further checking and scheduling.

TT messages are used for applications with strict requirements like deterministic end-to-end latency and bounded jitter. RC messages, compliant with ARINC 664 standard part 7, are used for applications with less strict requirements, for which sufficient bandwidth should be allocated. BE messages, using the remaining bandwidth of the network, form the standard Ethernet traffic which has no guarantee of delivery and transmission latency.

TTEthernet also has a transparent traffic used for its synchronization protocol. The synchronization message is called protocol control frame (PCF), and has three types: coldstart frames (CS) and coldstart acknowledgment frames (CA) are used for startup and restart services, and integration frames (IN) are used for synchronization service. In our model, the PCF traffic also uses the MAC destination address to encode its identity.

3.4 TTEthernet Device

The TTEthernet controller and switch have several common functions/services. We extract all the common ones, and implement them in a class named *tte_device*.

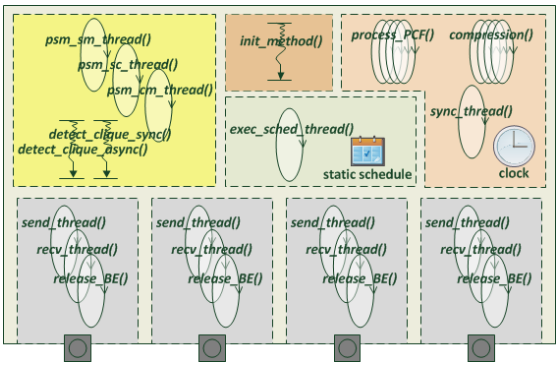

Fig. 2. TTEthernet Device Main Structure.

tte_device is the abstract base class of *tte_controller* and *tte_switch* which has pure virtual functions that need to be implemented by *tte_controller* and *tte_switch* to define different behaviors of these two different devices. Fig. 2 shows the main SystemC processes in *tte_device*. There is an *init_method()* SystemC method process which is sensitive to a power-on event and initializes the device. This is used to model different power-on times given in a configuration file of different devices. After power-on, startup service of TTEthernet will try to bring the device into synchronized operation mode.

Ports: An Ethernet socket is used to realize the functions of Ethernet ports. The TLM-2.0 transport interface methods are implemented to transmit standard Ethernet frames. Ethernet socket has both blocking transport interface and non-blocking transport interface. Due to the star or cascaded star network topology of TTEthernet, the collision domain is segmented and only two TTEthernet devices which are directly connected may contend for the use of the medium. We model TTEthernet working in full-duplex mode so that collisions become impossible; moreover, the non-preemptive integration of TT and ET is used that is compliant with the products in [12]. Thus, the efficient blocking transport method becomes accurate enough to model the communication between two TTEthernet devices.

Each TTEthernet device (controller/switch) can have several Ethernet ports according to its configuration. For a controller, multiple ports represent redundancy which send the same frame in order to realize fault-tolerance. For a switch, each port can be connected to a controller or a switch to create a separate collision domain. Each port is associated with three dynamic thread processes which are *send_thread()*, *recv_thread()*, and *release_ET()*. The *send_thread()* and *recv_thread()* processes with the scheduler model the data link layer of TTEthernet, which uses TDMA MAC protocol. The *send_thread()* process is responsible for starting a frame transmission, and is controlled by the scheduler process and the *release_ET()* process via events. The *release_ET()* process knows the schedule and is responsible for signaling the *send_thread()* process to send an ET frame if there is enough gap for this frame before next TT frame dispatching time comes. The *recv_thread()* process waits for an incoming frame delivered by the *b_transport()* method registered to the Ethernet socket. When a frame is transmitted through the *b_transport()* method, it will be processed by the

recv_thread() process. According to the analysis of the destination address, either a PCF handler process will be dynamically spawned, or one of the traffic processing functions (TT, RC, or BE) will be called. The traffic processing functions are pure virtual functions which need to be implemented by different TTEthernet device to realize different behaviors.

Scheduler: Every TTEthernet device sends packets according to a static schedule that relies on synchronized global time. The static schedule is generated by an off-line scheduling tool and used by the TTEthernet device through a configuration file. We use the off-line scheduling tool provided by TTTech [12], which guarantees two TT frames never contend for transmission.

The *exec_sched_thread()* process implements the function of the scheduler and is responsible for signaling the *send_thread()* processes of the ports to start a TT frame transmission according to the static schedule. It pends on a synchronization event occurring when the device enters the synchronized states, and starts executing the schedule when the event happens. If the device goes out of the synchronized states, it also signals the *exec_sched_thread()* process to stop executing the schedule. If the device is a synchronization master, the scheduler process also signals *send_thread()* processes to send out an integration PCF when PCF's dispatching time is reached (dispatching time is 0 in our model).

Protocol State Machine: Each TTEthernet device executes exactly one of the protocol state machines to maintain its role for synchronization, which are formulated in [2]. All TTEthernet devices can be classified into three different roles: synchronization masters (SMs), synchronization clients (SCs), and compression masters (CMs). Startup service of the protocol state machines tries to establish an initial synchronized global time to make devices operate in synchronized mode. When a device detects it is out of synchronization, restart service of the protocol state machines will try to resynchronize itself.

The model has three SystemC thread processes to realize different protocol state machines respectively, which are *psm_sm_thread()* for SM, *psm_sc_thread()* for SC, and *psm_cm_thread()* for CM, as shown in Fig. 2. Each state has its own *sc_event* object which is pended on by the state. If a state has a transition fired because of timeout, it also sets an event's notification by using the timer model, and pends on the event "OR" list of its own *sc_event* object and the timeout *sc_event* object. The *sc_event* object will be notified when any one of transitions of this state is enabled, and corresponding transition flag will be set showing the guard of this transition is met. By checking the flags in an order that is defined in [2], priorities of concurrent enabled transitions are enforced in the protocol state machines, which guarantees determinism. Since concurrent *sc_event* notifications will not queue up, events enabling concurrent transitions will not queue up during execution of the protocol state machines.

Clique detections are used in TTEthernet to detect clique scenarios where different synchronized time bases are formed in a synchronization domain. When cliques are detected, protocol state machines will try to reestablish synchronization. The *detect_clique_sync()* method process is responsible for synchronous clique detection and is sensitive to an event that will be notified when the accep-

tance window for receiving scheduled PCFs is closed. The *detect_clique_async()* method process is responsible for asynchronous clique detection and is sensitive to an event that will be notified when the acceptance window for receiving scheduled PCFs is closed in CMs or when the clock reaches the dispatching time in SMs or SCs.

Synchronization Service: When operating in synchronized mode, TTEthernet uses a two-step synchronization mechanism: SMs dispatch PCFs to CMs, and CMs calculate the global time from the PCFs (i.e. "compress") and dispatch "compressed" PCFs to SMs and SCs. SMs and SCs receive "compressed" PCFs and adjust their clocks to integrate into the synchronized time base.

When a PCF arrives, a dynamic PCF handler process (*process_PCF()*) will be spawned to cope with this PCF. Concurrent PCF handler processes may exist due to multiple PCFs arriving with small time difference. Permanence function [2] is used to reestablish the temporal order of the received PCFs. The *process_PCF()* implements the permanence function by using the timer model. By checking the PCFs, the process also enables some transitions whose guards only count on PCFs in the protocol state machines.

If the TTEthernet device is a CM, a dynamic compression process (*compression()*) may be spawned if there is no process handling corresponding integration cycle of the PCF. The integration cycle filed of a PCF shows which round of synchronization this PCF belongs to. Compression function [2] is used to collect PCFs having the same integration cycle number within a configurable interval and compress these PCFs for calculating the synchronized global time. The *compression()* also uses the timer model to realize all the time delays needed by its collection and delay phases.

When the acceptance window for receiving PCFs ends, the *sync_thread()* process will be resumed to calculate the clock correction from the PCFs that are in-schedule. After a fixed delay (at least greater than half of the acceptance window), the clock will be adjusted by the calculated correction value.

3.5 TTEthernet Controller & Switch

Both the TTEthernet controller and switch are derived from TTEthernet device, and implement the pure virtual functions to realize different behaviors of processing the traffic.

The TTEthernet controller also acts as a TLM-2.0 target which receives transactions containing Ethernet frames via a target socket. Extensions are made to the generic payload to show which traffic class the Ethernet frame belongs to. Each traffic class has its own transmission and reception buffers. In the case of a write command, the controller puts the extracted Ethernet frame into the corresponding traffic transmission buffer. When the controller receives a frame, it will signal the processor to read it via interrupt, and it puts the received frame into a transaction in response to a read command and sets the traffic class extension.

The TTEthernet switch uses critical traffic table and schedule table to route and forward CT (TT and RC) frames, and uses static/dynamic routing tables

for BE frames. It also acts as a temporal firewall for TT traffic to segregate faulty controllers if they are babbling. For RC traffic, it uses token bucket algorithm to enforce a bandwidth allocation gap between two consecutive RC frames.

4 Experimental Results

In this section, we compare our TTEthernet model with the model in OMNeT++/INET framework [5] and a real TTEthernet implementation from TTTech for validation. We also evaluate the startup and restart services as well as the efficiency of the simulation.

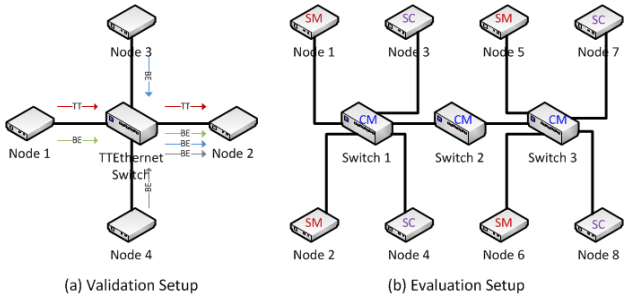

Fig. 3. Experiment Scenarios.

4.1 Validation

We set up a star topology which has four nodes connected to a central TTEthernet switch with 100Mbit/s links as shown in Fig. 3 (a). Node 1 sends both TT traffic and BE traffic to Node 2, and both Node 3 as well as Node 4 send only BE traffic to Node 2. All the traffic goes through a TTEthernet switch. The communication period is 10ms, and the time slot is 200μs. The maximum clock drift is set as 200ppm for the models. Node 1 sends a TT frame at 1ms offset of each period. The configuration files including their corresponding XML files for the nodes and switch are generated by the TTTech toolchain [12]. From the generated XML files, we extract parameters such as critical traffic table and schedule table to configure our model and the model in OMNeT++. In this setup, the switch dispatches the TT frame sent by Node 1 at 1.4ms offset of each period. The metrics we measure are average end-to-end latency and jitter of TT frames which are important factors for real-time communication systems. We measure these metrics for different TT frame sizes under full link utilization of BE traffic. Fig. 4 shows the results of our model in SystemC/TLM, the model in OMNeT++ INET framework, and the software stack implementation in Linux from TTTech [12].

From the figure we can see the model in SystemC/TLM and the model in OMNeT++ INET framework give very similar results. In [16], the method of measuring end-to-end latency of software-based implementation of TTEthernet

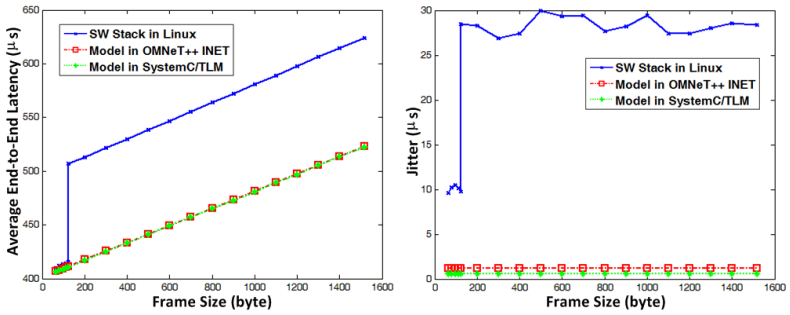

Fig. 4. Average End-to-End Latency and Jitter of Different Frame Sizes.

is stated. According to [16], the measured latency gap (90μs) between frame size of 123 and 124 bytes of the software-based implementation is caused by the measuring port driver configuration. The measured jitter of the software-based implementation is bounded by 30μs [16]. The hardware-based implementations will bound the jitter more tightly [12].

4.2 Evaluation

We set up the network as shown in Fig. 3 (b) to evaluate the startup and restart services implemented in our model. In this cluster, Node 1, Node 2, Node 5, and Node6 are SMs; Switch 1, Switch 2, and Switch 3 are CMs; the rest are SCs. The integration cycle is 10ms, and the parameters are generated by the TTTech toolchain [12].

With different power-on times, we record the time when every powered device in the cluster enters its synchronized state in Tab. 1. Since different power-on times of Switch 2 may cause cliques in the cluster, we also record the time when every powered device is resynchronized back to its synchronized state due to clique detection and restart service. Since in this setup SCs only passively receive PCFs, we set the power-on times of them as 0s.

Table 1. Startup and Restart Service Evaluation

N1 & N2 & N5 & N6	SW1 & SW2 & SW3	Sync	Resync
0s/0s/0s/0s	0s/0s/0s	29.834ms	-
0.1ms/1ms/0.5ms/1.2ms	1.1ms/0.8ms/1.5ms	30.845ms	-
2ms/4ms/8ms/6ms	30ms/10ms/40ms	79.856ms	-
0s/0s/0s/0s	0s/30ms/0s	38.677ms	-
0s/0s/0s/0s	0s/50s/0s	29.776ms	50.0256s

When every device approximately starts at the same time, the devices will be synchronized quickly which are shown in the first two cases. When the CMs are powered later than the SMs, the time when every one is synchronized will be delayed as shown in the third case. In the fourth case, Node 1, Node 2, and Switch 1 establish a synchronized time base at about 29.8ms; likewise, Node 5, Node 6, and Switch 3 establish the other synchronized time base. Switch 2 which is the CM connecting with the other two CMs is powered just a little bit

later than the time when the two synchronized time bases are established. Since during this small time interval the clock drifts have not caused the two time bases to differ too much, Switch 2 will join in the synchronization quickly and the two time bases will be merged into one time base. In the fifth case, two separate synchronized time bases are established before Switch 2 is powered as well. However, this time Switch 2 is powered much later (about 49.07s) than the time when the two time bases are established. The clock drifts have caused the two subsets of devices not to be synchronized over subset boundaries. When Switch 2 is started, asynchronous clique detection and restart service implemented in our model result in a new synchronized global time, and at 50.0256s every device is synchronized.

Finally, we evaluate the scalability and simulation efficiency of our approach. We set up the evaluation using a central switch, and all the nodes are connected to the switch. The simulation time is 1000s, and increasingly add a pair of nodes into the network. Each pair of the nodes, such as Node 1 and Node 2, communicates with each other using TT, RC, and BE traffic. Each node sends out a TT frame, a RC frame, and a BE frame every 10ms. Thus, there are $300,000 \times number\ of\ nodes$ frames totally. The result is shown in Fig 5.

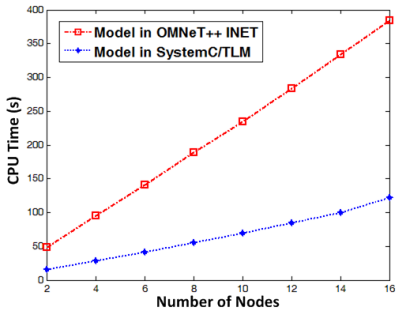

Fig. 5. Used CPU Time of Different Number of Nodes.

From the results we can see the model in SystemC/TLM has good simulation efficiency when the number of nodes increases. The simulation speed of the model in OMNeT++ INET framework is also evaluated under the same computation environment (2.50GHz dual-core CPU and 6GB memory). We simulate the same topology and traffic by using the fastest mode in OMNeT++ to get rid of the influence of animation and text outputs.

5 Conclusions

Due to the complex interactions between different layers of a CPS, virtual prototyping has become an important approach to discover potential design flaws before the last integration stage. SystemC/TLM has been adopted for virtual prototyping because of its capability of modeling both the software layer and the network/platform layer of a CPS. TTEthernet has been used in many CPS domains and provides bounded end-to-end latency and jitter.

In order to take into account the network effects caused by TTEthernet when designing a CPS, a model in SystemC/TLM is proposed in this paper. The developed model considers all the necessary features of TTEthernet and can be integrated into the hardware platform model in a straightforward manner. We validate the model against the model in OMNeT++ INET framework and the software-based implementation from TTTech, and evaluate the startup and restart services which are used to maintain the synchronization by powering on the devices at different times.

The future work focuses on integrating this model into a mixed TT/ET distributed CPS simulation framework and using timed automata to verify this model.

Acknowledgments. This work has been partially supported by the National Science Foundation (CNS-1035655).

References

1. Sztipanovits, J., Koutsoukos, X.D., Karsai, G., Kottenstette, N., Antsaklis, P.J., Gupta, V., Goodwine, B., Baras, J.S., Wang, S.: Toward a Science of Cyber-Physical System Integration. Proceedings of the IEEE **100**(1) (2012) 29–44
2. SAE Standard AS 6802: Time-Triggered Ethernet (2011)
3. IEEE Standard 1666-2011: Standard SystemC Language Reference Manual (2011)
4. Mller, W., Becker, M., Elfeky, A., DiPasquale, A.: Virtual Prototyping of Cyber-Physical Systems. In: ASP-DAC '12. (2012) 219–226
5. Steinbach, T., Kenfack, H.D., Korf, F., Schmidt, T.C.: An Extension of the OMNeT++ INET Framework for Simulating Real-time Ethernet with High Accuracy. In: SIMUTools '11. (2011) 375–382
6. Kopetz, H., Bauer, G.: The Time-Triggered Architecture. Proceedings of the IEEE **91**(1) (2003) 112–126
7. Steiner, W., Bauer, G., Hall, B., Paulitsch, M.: Time-Triggered Ethernet: TTEthernet (Nov. 2010)
8. Kopetz, H., Ademaj, A., Grillinger, P., Steinhammer, K.: The Time-Triggered Ethernet (TTE) Design. In: ISORC '05. (2005)
9. Steinhammer, K., Grillinger, P., Ademaj, A., Kopetz, H.: A Time-Triggered Ethernet (TTE) Switch. In: DATE '06. (2006) 794–799
10. Steinhammer, K., Ademaj, A.: Hardware Implementation of the Time-Triggered Ethernet Controller. In: IESS '07. (2007) 325–338
11. Steiner, W.: TTEthernet Specification (2008)
12. TTTech Computertechnik AG: TTEthernet Products. `http://www.tttech.com/en/products/ttethernet/`
13. Banerjee, A., Gerstlauer, A.: Transaction Level Modeling of Best-Effort Channels for Networked Embedded Devices. In: IESS '09. (2009) 77–88
14. GreenSocs Ltd: Ethernet Communication Protocol using TLM 2.0. `http://www.greensocs.com` (2010)
15. Bombieri, N., Fummi, F., Quaglia, D.: TLM/Network Design Space Exploration for Networked Embedded Systems. In: CODES+ISSS '06. (2006) 58–63
16. Bartols, F., Steinbach, T., Korf, F., Schmidt, T.C.: Performance Analysis of Time-Triggered Ether-Networks Using Off-the-Shelf-Components. In: ISORCW '11. (2011) 49–56