Implementing a Clock Synchronization Protocol on a Multi-Master Switched Ethernet Network

Mohammad Ashjaei[1], Moris Behnam[1], Guillermo Rodriguez-Navas[2], Thomas Nolte[1]
[1] Mälardalen University, Västerås, Sweden
[2] Universitat de les Illes Balears, Spain

Abstract

The interest to use Switched Ethernet technologies in real-time communication is increasing due to its absence of collisions when transmitting messages. Nevertheless, using COTS switches affect the timeliness guarantee inherent in potentially overflowing internal FIFO queues. In this paper we focus on a solution, called the FTT-SE protocol, which is developed based on a master-slave technique. Recently, an extension of the FTT-SE protocol has been proposed where the transmission of messages are controlled using multiple master nodes. In order to guarantee the correctness of the protocol, the masters should be timely synchronized. Therefore, in this paper we investigate the possibility of using a clock synchronization protocol, based on the IEEE 1588 standard, among master nodes. Moreover, we evaluate the overhead that is imposed by the clock synchronization protocol to the FTT-SE protocol. Finally, we present a formal verification of this solution by means of model checking technique to prove the correctness of the FTT-SE protocol when the clock synchronization protocol is applied.

1 Introduction

In the context of real-time communication, the usage of switched Ethernet technology is increasing due to its properties such as providing a high throughput and a collision free domain when transmitting messages. These properties can provide possibility to guarantee the timeliness behavior of the traffic. Among several types of switches, Commercial Off-The-Shelf (COTS) switches are more interesting because of their lower maintenance cost/process and wide availability.

Nevertheless, using COTS switches in critical applications is not straightforward as they can generate blocking for urgent messages, inherent in having a FIFO queue inside these switches. Moreover, the switch ports may overflow because of uncontrolled arrival of packets, inducing drop of some packets in a worst-case situation.

One effective solution to eliminate the above mentioned limitations is to use a master-slave technique to control the traffic loaded to the switch. In this paper we focus on the FTT-SE (Flexible Time-Triggered Switched Ethernet) protocol [12] which enforces global coordination of traffic. In order to avoid the potential overflow of the queues inside the switches, the global coordination is performed by a dedicated node called master. This protocol was proposed for a network containing a single switch, while the extension to overcome that restriction was proposed following two different approaches in [13] and [4]. In these both architectures multiple switches are connected together to compose a tree topology. In the former architecture, a single master is attached to the root of the tree and it coordinates the whole traffic in the network. Whilst, in the latter architecture, multiple switches along with multiple master nodes are scheduling the traffic which is more suited for the large networks as it is investigated in [3].

In this paper we focus on the second architecture which comprise several master nodes in a network, namely the multi-master architecture. According to the FTT-SE protocol, all messages should be scheduled within a fixed duration of time, called Elementary Cycle (EC), by the master node. In the multi-master architecture, each master schedules its associated slave node's messages within its local EC. To achieve consistency among the ECs of different masters, all master nodes should be timely synchronized.

In our previous work [4], we have proposed a technique based on using a signaling message broadcasted from the root master node to the children master nodes in order to indicate the start of each EC. Using a signaling message may degrade the performance of the protocol by imposing large timing deviations of ECs due to the switching delays after crossing a number of switches. Therefore, in this paper we investigate the use of a clock synchronization mechanism based on the IEEE 1588 standard, which is a de-facto solution that was proposed for different communication protocols such as the TT Ethernet [2]. Also, we study the effects of this mechanism on the network performance and bandwidth utilization, compared with the signaling method proposed in [4].

Moreover, we formally prove the correctness of the FTT-SE protocol when applying the clock synchronization protocol, using a model checker called UPPALL [5] which has been widely used for real-time systems (see [16], [15]).

This paper consists of the following sections. Section 2 describes the technical background of the FTT-SE protocol, the IEEE 1588 standard and the UPPAAL model checker. Section 3 outlines the need for clock synchronization in the FTT-SE protocol. Section 4 presents the solution based on IEEE 1588. Section 5 sketches the formal verification of the proposed solution. Section 6 summarizes related work and finally Section 7 concludes the paper.

2 Technical Background

In this section we explain some background which we use in the paper, including the multi-master FTT-SE protocol, the IEEE 1588 standard and the UPPAAL model checker.

2.1 The FTT-SE Protocol

FTT-SE [12] is an Ethernet real-time communication protocol that uses a master-slave technique to coordinate all traffic in the network in a flexible and timely manner. Here we describe the multi-master architecture that uses multiple switches, each deploying one master node, as depicted in Figure 1.

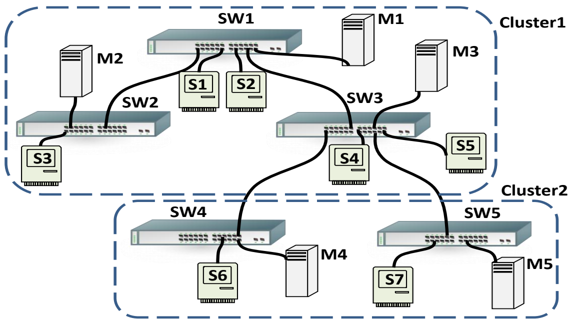


Figure 1. Multi-Master FTT-SE Network

Each switch along with all associated nodes that are directly connected to it is called a *sub-network*, e.g., SW1, M1, S1 and S2 in Figure 1. Moreover, each sub-network is a parent for the lower level sub-networks that are connected to it. A group of sub-networks with the same parent sub-network is called a *cluster*, e.g., Cluster2 in Figure 1. The only exception is the root sub-network that is included in its children cluster. In addition, we distinguish two categories of traffic: the traffic that is transmitted within a sub-network is called *local*, otherwise it is called *global*.

In the multi-master architecture, each master schedule messages on-line according to any desired scheduling policy, e.g., Fixed Priority Scheduling, on a cyclic basis. The basic cycle has a configurable fixed duration and is called Elementary Cycle (EC). Each EC is partitioned into two windows, one to organize the protocol called *initialization time* and another one that is dedicated to data message transmissions which is called *data transmission window* (Figure 2). Moreover, the data transmission window is divided among all traffic types, i.e. local/global and synchronous/asynchronous. The global asynchronous window is further splitted among clusters, e.g. Cluster1 and Cluster2 in Figure 2.

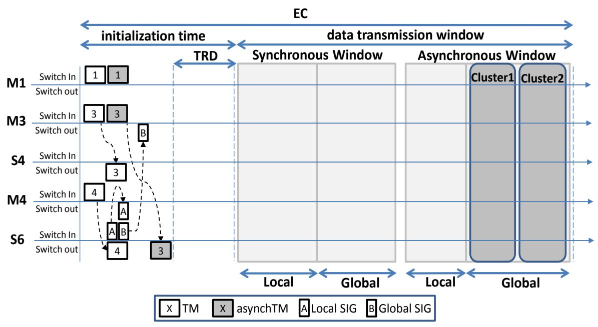


Figure 2. The EC in Multi-Master FTT-SE

Local and global synchronous messages are activated periodically and the scheduler checks every EC whether they fit in the corresponding window. The scheduled messages are encoded in a Trigger Message (TM) that is transmitted, in the beginning of the next EC, to all slave nodes (TM3 from M3 to S4 in Figure 1). The global synchronous messages are scheduled in all master nodes in parallel.

Unlike synchronous messages, activation of asynchronous messages is unknown in advance and can occur at any time. A signaling mechanism [14] allows the slaves to notify the master of pending requests using a Signaling Message (SIG) that is transmitted during the initialization time. For instance, in Figure 2 message A is used as a SIG message for local asynchronous messages sent from S6. The master then schedules the asynchronous messages adequately. To handle the global asynchronous messages, the slave nodes send their SIG messages directly to the master of each cluster, which is responsible for scheduling that traffic (message B in Figure 2). For this purpose, the master of the cluster uses a particular TM, called the asynchTM, that is sent to the children slave nodes right after sending its regular TM in its sub-network (asynchTM3 from M3 to S6 in Figure 2).

The slave nodes receive the TM and the asynchTM, decode them and initiate the transmission of the scheduled messages. Decoding the TM and the asynchTM takes a specific time which is called Turn Around Time (TRD) (Figure 2).

2.2 The IEEE 1588 Standard

IEEE 1588, known as the Precision Time Protocol (PTP), is a clock synchronization protocol especially tailored for networked measurement and control systems [1]. IEEE 1588 supports heterogeneous clock systems with different resolutions and it establishes a master-slave clock synchronization in which each node synchronizes its clock with respect to the reference provided by a single node, called the master.

The IEEE 1588 messaging is illustrated in Figure 3. At the beginning of the algorithm, a particular message, denoted *Sync* message, is transmitted from the master to the slave nodes. The Sync message is time stamped with the starting time t_m. Whenever the slave receives the Sync message, it records the received time t_s. Afterwards, the master sends another message called Follow_up which contains the actual value of the time stamp t_m. Then, the slave calculates the offset from the master using (1), where *pDelay* is the propagation delay of the Sync message.

$$offset = t_s - t_m - pDelay \qquad (1)$$

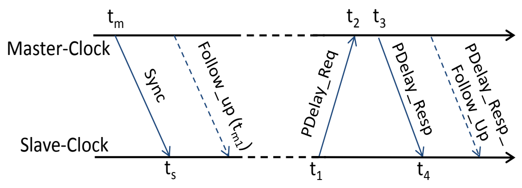


Figure 3. The PTP messaging mechanism

In order to measure the propagation delay, the IEEE 1588 standard mentions two methods: the delay request-response and the peer delay mechanisms. In this paper we use the latter because it does not depend on the Sync message time stamps. The mechanism initiates by sending PDelay_Req from the slave to the master and recording t_1 as sending time stamp. The master receives the request while recording the receiving time t_2 and sends back the response to the slave at time t_3. The slave sends the follow_up message containing the actual value for t_2 and t_3. Finally, the propagation delay is calculated as in Equation (2).

$$pDelay = \frac{(t_2 - t_3) + (t_4 - t_1)}{2} \qquad (2)$$

2.3 UPPAAL

There are different ways to evaluate the correctness of a system using simulation, experiment and formal verification. The formal verification provides a tool to investigate whether the protocol behaves as expected. In the context of formal verification, model checking technique is one of the techniques which guarantees the absence of error in the model. In this paper we have used the UPPAAL model checker [5].

UPPAAL provides the ability to model a system as networks of timed automata where the clock for each timed automaton progresses with the same rate. A timed automaton is a finite state machine which has a clock variable to measure the time progress. Each timed automaton in UPPAAL is called a *template* and it contains number of *locations* and *edges*. An edge is a transition from one location to another, while, a location describes a state in the model. Figure 4 shows a model containing two locations and one edge between them.

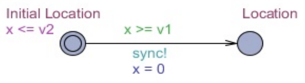


Figure 4. UPPAAL Model Example

Over each edge the defined system variables can be updated. However, the clock variables can only be restarted at edges, e.g., $x = 0$ in Figure 4.

Moreover, two or more actions in different timed automaton can be synchronized by defining *synchronous channels* in the UPPAAL model (sync! in Figure 4 is synchronized with other edges labeled with sync?). Whenever one transition is performed, the other transitions, which are synchronized with that, are updated to a new state in the system. A state is changed in a timed automaton (for both synchronized and regular transitions) whenever a *guard condition* defined for that transition is satisfied. The guard condition can be defined based on an integer, boolean or clock variable, e.g $x >= v1$ in Figure 4. For each location an *invariant* can be defined for the clock variable that limits the amount of time the model can stay in that location, e.g., $x \leq v2$, where $v2 \geq v1$.

The state of a system can stay in a location as long as the edges guard or invariant of the location is not satisfied. However, a particular type of location, namely *committed location*, is available such that the system should immediately leave this location.

3 Problem Formulation

In this section we present the system model and we outline the need for a clock synchronization mechanism in the FTT-SE protocol.

3.1 System Model

In this paper, we assume COTS switches with several parallel prioritized FIFO queues for each output port. We also consider store-and-forward Ethernet and non-blocking message transmission switches. Two different switching delays, namely the store-and-forward (SFD) delay and the switch fabric relaying latency (Δ), is assumed for each switch. The SFD is the time that the switch is required to save the data before forwarding to the right output port.

Furthermore, in order to distinguish between the master-slave notation in the synchronization protocol and the master-slave in the FTT-SE protocol, we use *master-clock* and *slave-clock* nodes for clock synchronization purposes.

We assume the root master as a reference clock. Also, we define ϵ_{Mj} as the clock deviation of master Mj from its parent master during one EC, e.g., ϵ_{M4} is the clock deviation of M4 from M3 in 1 EC in the example that is depicted in Figure 1.

3.2 The Need for Synchronization

According to the FTT-SE protocol, all scheduled messages that are transmitted within an EC should be received within the same EC to override the problem of overflowing in the queues inside the switch. Therefore, the messages that are transmitted across sub-networks should be received before the start of the next EC. This requires that the ECs in all sub-networks are synchronized, otherwise the transmission of messages may face overrun.

In fact, we cannot guarantee the perfect simultaneity of ECs. Therefore, the scheduler of all master nodes should take into account a certain delay ϵ_{Mi} from the beginning and the end of the EC.

Assume that m_1 is a local message that is transmitted within the M1 sub-network and that m_2 is a global message which is sent from S1 to S4. Considering a particular deviation (ϵ_{M3}) between two respective ECs, the overrun occurs for $m2$ as depicted in Figure 5. In other words, the effective EC for data transmission available for both sub-networks is $EC - 2\epsilon_{M3}$ which should be considered by both M1 and M3 schedulers to prevent the overrun.

In our previous work presented in [4] we proposed to use a particular message, called Global Trigger Message (GTM), being broadcasted by the root master and propagated down through the entire network. The remaining masters synchronize upon receiving the GTM and start their local ECs. This mechanism generates a different receiving time of the GTM depending on the corresponding depths in the network hierarchy, that may degrade the network performance. In addition, in case of losing the GTM during the propagation, we need a recovery mechanism to handle this situation, that increases further the overhead of the protocol.

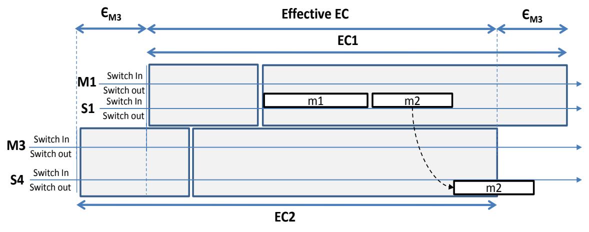


Figure 5. Overrun of a Message

In order to eliminate the above mentioned problems, in this paper we use a clock synchronization protocol based on the IEEE 1588 standard.

Note that, in the FTT-SE protocol, the slave nodes are synchronized with their associated master node when they receive the TM. Therefore, the clock synchronization is not required for the slave nodes.

4 Clock Synchronization Method

In this section we propose an implementation of a clock synchronization protocol in the multi-master FTT-SE network, based on the IEEE 1588 standard. Also, we study the effect of this adaptation on the bandwidth utilization and performance of the network compared with the GTM signaling solution.

4.1 Clock Synchronization in the FTT-SE Protocol

To synchronize the ECs of all master nodes in the network, we propose to reserve a particular bandwidth, called *Sync Guard*, in order to transmit the required messages for clock synchronization as defined in the IEEE 1588 standard. In this approach we assume a single-step clock for nodes which leads to ignore the Follow_Up and PDelay_Resp_Follow_Up messages in the mechanism according to the standard. Moreover, in the multi-master FTT-SE architecture, the master node of each sub-network is the master-clock for its master node of children sub-networks (slave-clock nodes). Therefore, each master node should synchronize with the master node of its parent sub-network. For instance, M3 is a slave-clock node that should be synchronized with M1 as a master-clock node.

The clock synchronization includes two phases: propagation delay measurement and Sync message transmission which are performed in two different ECs, one EC for each phase. In one EC the propagation delay measurement occurs by sending the PDelay_Req from the slave-clock node to the master-clock node. As a response the master-clock node sends back the PDelay_Resp. These two request and response messages are transmitted within the Sync Guard (Figure 6, EC1). The slave-clock node measures the propagation delay according to (2). In the other EC, the Sync message is transmitted from the master-clock node to the slave-clock node within the Sync Guard (Figure 6, EC2).

The procedure of propagation delay measurement is more complex than the Sync message transmission due to the transmission of request and response messages. Moreover, the propagation delay between two master nodes does not vary too much during run time assuming that the architecture remains unchanged. Therefore, the propagation delay measurement is performed periodically every period T_{pd}, not every EC.

This will diminish the overhead of the protocol and the complexity in the FTT-SE protocol.

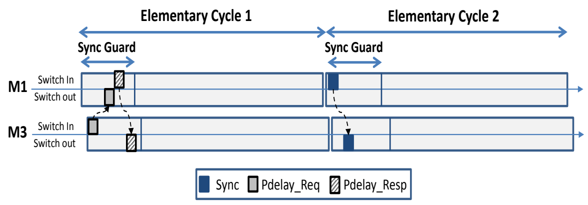


Figure 6. Clock Synchronization Method

However, since the Sync Guard has been assumed within each EC, the Sync message can be transmitted every EC, except the EC in which the propagation delay is measured.

The length of the Sync Guard should be selected enough for both procedures, i.e., the Sync transmission and the propagation delay measurement. In fact, the propagation delay procedure takes more time than the Sync message transmission. Therefore, the Sync Guard size based on the switching delay (two switches between two master nodes) is computed in (3).

$$\begin{aligned}&SyncGuard = C_{req} + 2(SFD_{req} + \Delta) + C_{resp}\\ &+2(SFD_{resp} + \Delta)\end{aligned} \tag{3}$$

where C_{req} and C_{resp} are the transmission times of the Pdelay_Req and Pdelay_Resp messages respectively. Also, the SFD delays of both messages are equal to their corresponding transmission times.

The proposed synchronization protocol affects the bandwidth utilization due to the assignment of the Sync Guard in each EC. The data bandwidth utilization is the percentage of time allocated for data transmission in one EC, which is calculated in (4), where dev is the biggest deviation of ECs with respect to the root master EC and $InitTime$ is the initialization time as shown in Figure 2.

$$util = \frac{EC - SyncGuard - InitTime - 2dev}{EC} \tag{4}$$

In the clock synchronization for the multi-master architecture, each master node is synchronized with the master node of the parent sub-network. Therefore, the worst-case deviation is accumulated in each level of the network hierarchy. However, the worst-case deviation in each level with respect to the master node of the parent sub-network is the summation of the two biggest ϵ because the other smaller deviations are located within them.

In order to compute the worst-case deviation for all ECs, we define $E_l(i)$ that contains ϵ_{Mi} for all master nodes in level l of the network hierarchy. Moreover, to obtain the two biggest deviations we define $E_l^{sort}(i)$ which holds the sorted values of $E_l(i)$ in a descending order. Finally, the worst-case deviation of ECs is calculated in (5).

$$dev = \sum_{level=1}^{l} 2[E_l^{sort}(1) + E_l^{sort}(2)] \tag{5}$$

Note that, the synchronization round in the worst-case is 2 EC as there is an EC in which the propagation measurement is performed. Therefore, the deviation in each level of the network hierarchy should be multiplied by 2 as ϵ is defined for one EC.

Also, the initialization time ($InitTime$) is calculated in (6).

$$\begin{aligned}&InitTime = 2(C_{TM} + C_{asynchTM} + \Delta)\\ &+max\{TRD, N_{max} \times (C_{SIG} + C_{asynchSIG})\}\end{aligned} \tag{6}$$

where N_{max} is the maximum number of nodes in a sub-network. Also, the transmission of SIGs overlaps with the TRD and thus we consider the maximum delay between the TRD and the time to transmit all the SIGs in the network, given by $N_{max} \times (C_{SIG} + C_{asynchSIG})$.

Note that, it is reasonable to assume in the start-up mode of the network that the ECs are not synchronized. Therefore, it takes some ECs to obtain the synchronization among ECs. Meanwhile, the data transmission is paused to prevent any interference with the synchronization protocol signaling.

4.2 Comparative Evaluation

In order to compare the proposed clock synchronization mechanism with the method using GTM, we calculate the data bandwidth utilization for both techniques with respect to the network scale.

In the method using a GTM signal, each master node initiates its local EC by reception of the GTM. Therefore, the deviation of each master from the root master depends on its position in the network hierarchy. This deviation is calculated in (7).

$$dev = (l - 1) \times (C_{GTM} + \Delta) \tag{7}$$

Moreover, the bandwidth utilization for the method using a GTM is computed in (4), where $SyncGuard = 0$. The initialization time is computed in the same way as the clock synchronization method presented in (6), however the transmission time for GTM (C_{GTM}) should be added.

In order to illustrate the effects of both synchronization methods on data bandwidth utilization and maximum deviation of ECs, we generate a particular network example. The network contains several sub-networks where each is limited to include 10 nodes, the network capacity is $100Mbps$ and $\Delta = 5\mu s$. The deviation of ECs using the proposed clock synchronization mechanism depends on the synchronization

Message	Trans. Time (μs)
EC	2000
TM and asynchTM	24
GTM, SIG and asynchSIG	7
PDelay_req and PDelay_Resp	7
TRD	200

Table 1. The Network Example Parameters

accuracy, however we assume $\epsilon = 1\mu s$ for all masters in this example as it is achievable according to the IEEE 1588 standard. Also, the other network parameters are shown in Table 1.

The deviation among ECs for both synchronization methods is illustrated in Figure 7-a. The deviation using the proposed clock synchronization mechanism depends on the clock accuracy, yet it is assumed to be $1\mu s$ in this example. However, this deviation is significantly large in the method using a GTM, in particular for large networks.

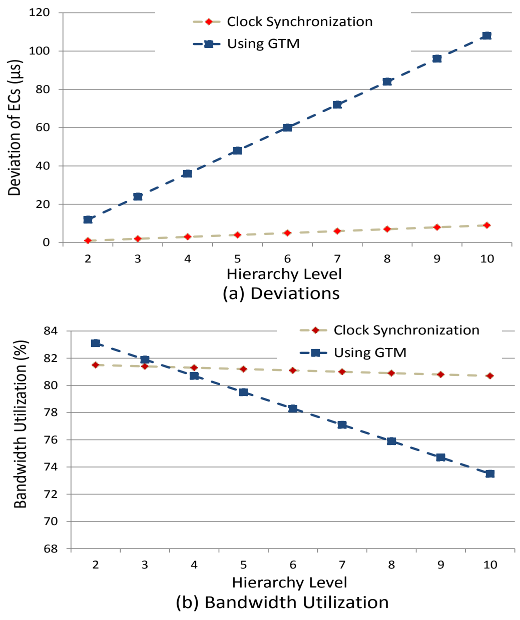


Figure 7. Synchronization Performance

In addition, the data bandwidth utilization for both synchronization methods, using a GTM and the proposed clock synchronization, is depicted in Figure 7-b. As a consequence of the ECs deviation depicted in Figure 7-a, the bandwidth utilization using the proposed clock synchronization decreases slightly by increasing the levels in the network hierarchy.

In contrast, the bandwidth utilization using a GTM decreases sharply. This evaluation shows that, even though there is a Sync Guard consuming the EC in the clock synchronization protocol, it still performs better in particular for a network having more than 3 levels in the network hierarchy. Moreover, although the bandwidth utilization for small networks is less in the method using clock synchronization, the deviation of ECs is very small which enhance the performance of the FTT-SE protocol.

5 UPPAAL Verification Model

Experience shows that reasoning about the temporal behavior of a distributed real-time system is not straightforward. Whenever this system is organized as a multicluster architecture, in which every cluster uses its own scheduler and its own clock-master, then the analysis can hardly be performed without some kind of computer support. In this section we use a number of UPPAAL models (i.e. models based on networks of timed automata) to provide evidence for the analysis discussed in Section 4. The presented models will show three important aspects of FTT-SE. The first model will illustrate the problems associated to having unsynchronized clusters and hence unsynchronized EC. The second model will show how the GTM signaling mechanism may solve the previous problem, but at the cost of lower utilization. The third model will show that implementation of the IEEE 1588 clock synchronization protocol solves the problem with better bandwidth utilization.

The rationale of our models follows the approach described in [15], which encourages a clear separation between the modeling of application aspects and the modeling of temporal aspects.

5.1 Unsynchronized ECs Model

In order to model the behavior in which the ECs of the master nodes are not synchronized, we define 4 automata per each node: 1) `MasterNode` models the operation of a master node, and therefore corresponds to the application aspects; 2) `InitWinTimer` models the waiting time during the initialization time window of each EC; 3) `DataTransTimer` models the timer that measures the data transmission window; 4) `MsgTransmission` models the transmission of a data message, including its associated network delay. Note that the last three automata only model temporal aspects of the system.

Each automata uses an input parameter of type integer (i) for identifying the specific instance of the template. All the automata that relate to the same node, do share the same value of i. This guarantees proper use of the UPPAAL synchronization channels.

The `MasterNode` automaton (depicted in Figure 8) starts in location `Initial` and immediately transits to location `InitWindow` as it is a committed location. This transition activates `InitWinTimer` automaton via channel `initTimerSet[i]`. Then, the automaton stays in location `InitWindow` until the expiration of the timer `InitWindow`, signaled via channel `initTimerExp[i]`. After that, the automaton goes through locations `L1` and `L2` to reach location `DataTransWindow`. During the passed edges two automata are triggered through channels `set[i]` and `InitMsgTrans[i]`. The former starts the timer that measures the data transmission window whereas the latter actually triggers the message transmission.

`MasterNode` stays in location `DataTransWindow` for some time. If the message transmission ends (`sendMsg[i]` is activated) the automaton stays in the same location, but the variable `flag[i]` takes the value 1 (true), meaning that the transmission was successful. As soon as the timer for data transmission window expires (this is signaled by `DataTransTimer` through `expire[i]`), the automaton steps into location `flagCheck`.

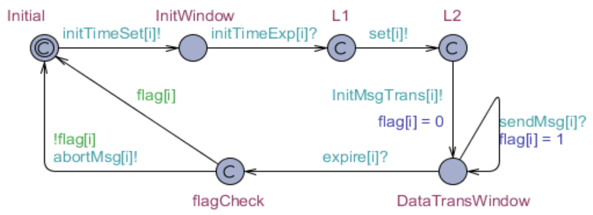
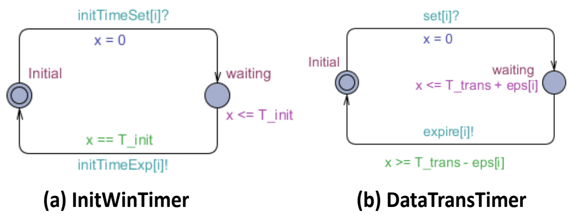


Figure 8. Master Node Timed Automaton

If the automaton reaches location `flagCheck` and the flag is true, it means that the delay for message transmission (`MsgTransmission`) was shorter than the timer for data transmission window (`DataTransTimer`), i.e., the message is transmitted successfully before finishing of the EC. Then, the automaton directly transits to `Initial` location. If the message transmission was not successful during the data transmission window, the flag is false. In this situation the message transmission is aborted through channel `abortMsg[i]`.

`InitWinTimer` automaton, depicted in Figure 9(a), contains two locations, `Initial` and `waiting`, and a clock variable x. The model starts in location `Initial` and transits to location `waiting` whenever it is activated through channel `initTimeSet[i]`. In this transition the clock x is reset to zero. Afterwards, the timed automaton stays in location `waiting` for exactly T_init time units due to the invariant and the guard defined over clock x. Whenever clock x reaches the value T_init, the transition to `Initial` location is initiated and is signaled through channel `initTimeExp[i]`. The value of T_init is the duration of the initialization time within an EC; it is defined in the variable declaration.

The second timed automaton, see Figure 9(b), is called `DataTransTimer` and measures the data transmission window of the EC. Similarly to the previous timed automaton, `DataTransTimer` starts in location `Initial` and goes to location `waiting` when channel `set[i]` is activated. During this transition, clock x is reset. The model stays in location `waiting` for an amount of time that varies within the range $[T\_trans - eps[i], T\_trans + eps[i]]$. The value T_trans is constant and corresponds to the *nominal* data transmission time, which is defined in the variable declaration.



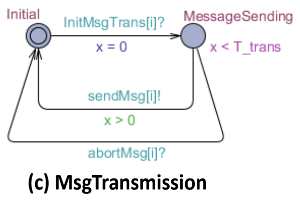


Figure 9. Timer automatons

The deviation with respect to the nominal duration varies in the range $[-eps[i], +eps[i]]$ and is also defined in the variable declaration. In the transition from location `waiting` to location `Initial`, channel `expire[i]` is signaled. This is interpreted by the corresponding `MasterNode` as the end of the EC. Note that as a consequence of the introduced drift, each master may notify the ending of the EC with a certain deviation from the other masters.

The third timed automaton, depicted in Figure 9(c), is called `MsgTrasmission` and models the behavior of a message transmission. It starts in location `Initial` and steps into location `MessageSending` whenever channel `InitMsgTrans[i]` is activated by `MasterNode`. The timed automaton leaves location `MessageSending` after some delay $[0, T\_trans]$, where T_trans is the nominal data transmission window.

For completeness, the variable declaration for this model is shown in Listing 1, except channel declaration due to the space limit.

Listing 1– Variable Declaration

```
const int EC = 500;
const int T_init = 100;
const int T_trans = EC - T_init;
const int N = 2; //number of nodes
int eps[N] = {0, 2};
bool flag[N] = {0, 0};
```

This model allows us to verify that communication is correct only as long as the assumption of having perfectly synchronized clocks holds. This is verified with the following properties, which are satisfied only if $eps = 0$.

```
A[] not deadlock
A[] MasterNode(1).flagCheck imply
        (flag[0]==1)
```

The first property proves that the model does not get stuck in any deadlock which is satisfied for this model. The second property is more important, as it checks whether both masters always successfully transmit within the same EC. When this property is not satisfied, as it may happen if $eps > 0$, it means that Master 1 can finish its EC, i.e. reach location `flagCheck` before the other Master has actually transmitted its data message.

5.2 GTM signaling Model

In order to add the GTM signaling to the previous model, we extend it with an additional timer for keeping track of the GTM transmission delay.

As it is explained earlier, the GTM delay depends on the level of the node in the hierarchy. Thus, the timer for GTM signaling (Figure 10) counts based on (7), where $level[i]$ is the level of node i and sw_delay is the switching delay, both are defined in the variable declaration.

The initialization time automaton is similar to the previous one, as illustrated in Figure 9(a). The timers for data transmission window (`DataTransWin`) and message transmission (`MsgTransmission`) use the same automata as in Figure 9(b) and (c), although now the timer boundaries are defined to count until the end of EC, which in this case is defined as $T\_trans - (level[i] - 1) \times sw\_delay$.

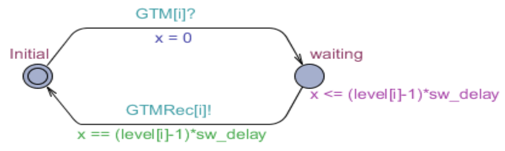


Figure 10. GTM Timer automaton (`GTMTimer`)

The master node automaton is depicted in Figure 11. It is similar to the previous model, except for an additional location, `GTMwaiting` for modeling the GTM delay. After leaving the `Initial` location, it stays in `GTMwaiting` for the time defined in the `GTMTimer` automaton. The other process is the same as in the previous explanation.

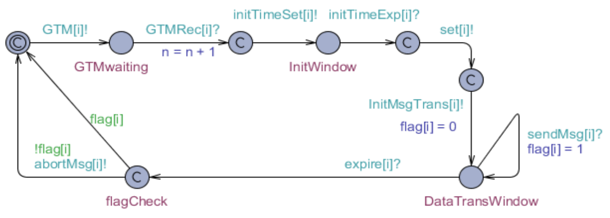


Figure 11. Master Node Timed Automaton with GTM signaling

Formal verification of the properties of this model requires definition of a specific `Observer` automaton; it is depicted in Figure 12(a). Figure 12(b) shows a trivial timed automaton, called `Dummy`, which is required to make `Observer` evolve over time. Note that `Observer` transits to location `check` as soon as n becomes greater than zero. The variable n is increased by any `MasterNode` automaton that passes the GTM delay. `Observer` stays in location `check` until all nodes have passed to `GTMwaiting` location ($n >= N$). The amount of time that `Observer` stays in location `check` gives the time distance between the fastest and the slowest masters, therefore the maximum deviation of ECs.

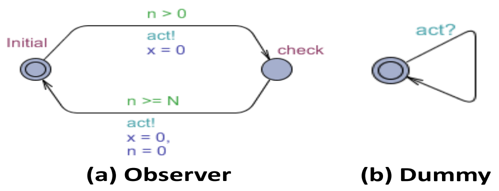


Figure 12. Observer Automaton

The variable declaration is the same as Listing 1, with some additions that are presented in Listing 2.

Listing 2– GTM Synchronization Model

```
const int sw_delay = 2;
const int N = 5;
int eps[N] = {0, 1, 2, 3, 3};
int level[N] = {1, 2, 2, 3, 3};
bool flag[N] = {0, 0, 0, 0, 0};
chan abortMsg[N];
chan GTM[N], GTMRec[N];
```

The verification of successful message transmission is satisfied using the same two properties discussed for the previous model. Moreover, in this model it is also possible to verify the worst-case variation of ECs, with the following property.

```
A[] Observer.check imply (Observer.x
        <= (level[4]-1)  * sw_delay)
```

This property is satisfied when there are 5 nodes in the network ($N = 5$) and it shows that the worst-case deviation is the delay of receiving the GTM suffered by the deepest node in the network hierarchy.

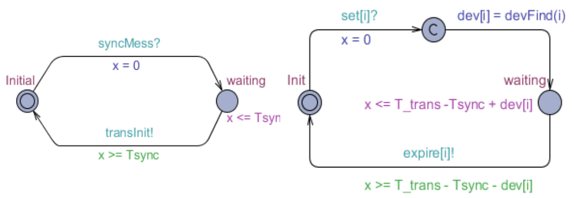
Note that, it is possible to verify larger networks and for doing so we need to change N and $level[N]$ respectively. Therefore, the property would be altered to check if `Observer.x` is equal or greater than $level([N-1]-1) * sw\_delay$.

5.3 Clock Synchronization Model

The third model serves to study the synchronization among ECs when using a particular bandwidth, SyncGuard, at the beginning of an EC. The model is the same as the GTM signaling model, except that we define another timer, called `SyncTimer`, to keep track of SyncGuard, instead of the GTM timer. Another difference is that `SyncTimer` does not have any input parameter and it is unique for all nodes using broadcast channels.

SyncTimer is presented in Figure 13(a). It is activated through broadcast channel syncMess, stays in location waiting for exactly $Tsync$ and signals its expiration via channel transInit.

The data transmission timer (DataTransTimer) in this model differs from the one in the previous models, as shown in Figure 13(b). The basic difference is that the timer deviation is calculated when the timer is initiated (using devFind(i) function). The deviation of each node depends on the position in the level of network hierarchy and is calculated according to (5).



(a) SyncTimer **(b) DataTransTimer**

Figure 13. Timer automatons

The automaton for modeling message transmission (MsgTransmission) is similar to the one in the previous models, but the upper bound now becomes $x < T\_trans - Tsync - dev[i]$, where $dev[i]$ is calculated in the DataTransTimer process.

The master node automaton is depicted in Figure 14. In this model and without losing generality, we assume node(0) as the reference clock. Therefore, node(0) initiates the SyncTimer and the other nodes are waiting in location syncGuard until the SyncTimer expires through channel transInit. This procedure makes all nodes synchronized after the syncGuard window. Therefore, it does not model clock synchronization in itself, but the effect of having clock synchronization. This is a common abstraction technique, see [15]. The other locations and transitions of the master node are similar to the previous node models, and they are therefore not explained.

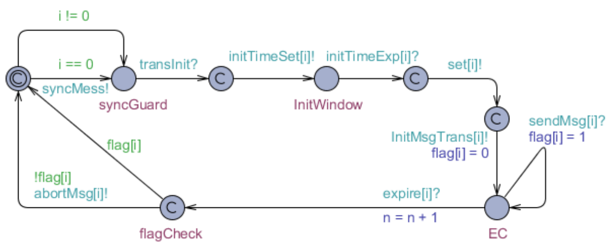


Figure 14. Master Node Timed Automaton with Clock Synchronization

In this model, the same Observer and Dummy automata of Figure 12 are used for measuring the deviation of the ECs. As it is shown in Figure 14, n is increasing when the EC of the node expires. Since the clock synchronization happens at the beginning, the maximum deviation is observed at the end of the EC. Therefore, we increase n when the EC is finished and we measure the difference between the fastest and slowest masters in the Observer automaton.

The variable declaration of this model is presented in Listing 3, except the channel declaration and the variables related to the windows duration which are similar to Listing 1.

Listing 3– Clock Synchronization Model

```
const int N = 5;
const int Tsync = 30;
int eps[N] = {0, 4, 3, 2, 3};
int level[N] = {1, 2, 2, 3, 3};
int dev[N];
bool flag[N] = {0, 0, 0, 0, 0};
```

The verification of successful message transmission is similar to the first model. Moreover, to check the worst-case deviation of ECs the following property should be satisfied. The property is defined for 5 nodes, and in this case $dev[3]$ and $dev[4]$ exhibit the greatest deviations among the nodes, because of their depth in the hierarchy (according to (5)). This property is checked for this model and it is satisfied.

```
A[] Observer.check imply (Observer.x
        <= dev[3] + dev[4])
```

Comparing the GTM signaling verification property and the clock synchronization verification property, the deviation imposed by the latter technique among the master nodes is much smaller than the former technique considering the same architecture as depicted in Listing 3 and 2.

6 Related Work

In the domain of network clock synchronization, different protocols have been established such as the NTP (Network Time Protocol), the IEEE 1588 Precision Time Protocol (PTP) and the IEEE 802.1AS. In the following we refer to some of the more relevant work in which these clock synchronization protocols have been applied to switched Ethernet.

The work in [11] presents the usage of IEEE 802.1AS in the in-vehicle network Audio/Video Bridging (AVB) switched Ethernet. The adaptation of the protocol to support clock synchronization is described, and the influence of different situations on the synchronization accuracy is investigated using the OMNeT++ simulation tool. The results show that the network load does not significantly affect the accuracy, while it can be affected by the selection of the synchronization interval.

A synchronization technique integrating both IEEE 1588 and NTP is presented in [17], implementing it over a switched Ethernet network with traffic smoothing. The traffic smoother classifies different messages including the messages for clock synchronization. In case of synchronization, other data messages are blocked by the traffic smoother.

The PTP is also utilized in the PROFINET protocol with some adaptation to that which is presented in Annex D of the IEEE 1588 standard. In this regard, an architecture to connect the devices with PTP for clock synchronization with the PROFINET infrastructure is proposed in [8] in order to achieve a precise synchronization. In addition, the evaluation of such a clock synchronization protocol over PROFINET IO has been performed using OMNeT++ simulation framework in [9].

UPPAAL is a very well-known model checker based on the theory of timed automata. It has been successfully applied to formally verify a great number of applications, including [7], [10] and [6], to name a few. In this paper we use the modeling patterns that were described in [15], which allow specification of systems with either drifting or synchronized clocks.

7 Conclusion and Future Work

In this paper we present an implementation of a clock synchronization protocol for the multi-master FTT-SE network based on the IEEE 1588 standard. We compare the effects of such a protocol on the performance and the bandwidth utilization, and we compare against our previous solution using GTM signaling. We show that the new solution, using clock synchronization, maintains a worst-case deviation that is significantly smaller than the previous solution, in particular for large networks.

In addition, we approached the problem of unsynchronized ECs using the UPPAAL model checker. Also, we have implemented both synchronization techniques by means of modeling and we have formally proved the efficiency of the clock synchronization compared with the GTM signaling. Our ongoing work aims at performing some further experiments followed by a deployment in an industrial setting.

Acknowledgment

This work is supported by the Swedish Foundation for Strategic Research, via Mälardalen Real-time Research Center (MRTC) at Mälardalen University. Also, it is partially supported by the Spanish Economy and Competitiveness Ministry under grant DPI2011-22992 and by the FEDER.

References

[1] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, 2008.

[2] A. Ademaj and H. Kopetz. Time-triggered ethernet and ieee 1588 clock synchronization. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2007.

[3] M. Ashjaei, M. Behnam, L. Almeida, and T. Nolte. Performance analysis of master-slave multi-hop switched ethernet networks. In *8th IEEE International Symposium on Industrial Embedded Systems (SIES13)*, June 2013.

[4] M. Ashjaei, M. Liu, M. Behnam, A. Mifdaoui, L. Almeida, and T. Nolte. Worst-case delay analysis of master-slave switched ethernet networks. In *2nd International Workshop on Worst-Case Traversal Time (WCTT'12)*. ACM, December 2012.

[5] G. Behrmann, R. David, and K. G. Larsen. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems*, pages 200–236. Springer, 2004.

[6] B. Bordbar, R. Anane, and K. Okano. An evaluation mechanism for qos management in wireless systems. In *11th International Conference on Parallel and Distributed Systems*, 2005.

[7] A. David and W. Yi. Modelling and analysis of a commercial field bus protocol. In *12th Euromicro Conference on Real-Time Systems*, 2000.

[8] P. Ferrari, A. Flammini, D. Marioli, S. Rinaldi, E. Sisinni, A. Taroni, and F. Venturini. Clock synchronization of ptp-based devices through profinet io networks. In *13th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'08)*, 2008.

[9] P. Ferrari, A. Flammini, S. Rinaldi, and G. Gaderer. Evaluation of clock synchronization accuracy of coexistent real-time ethernet protocols. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS'08)*, 2008.

[10] K. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *16th IEEE Real-Time Systems Symposium*, 1995.

[11] H.-T. Lim, D. Herrscher, L. Volker, and M. Waltl. Ieee 802.1as time synchronization in a switched ethernet based in-car network. In *IEEE Vehicular Networking Conference 2011 (VNC'11)*, 2011.

[12] R. Marau, L. Almeida, and P. Pedreiras. Enhancing real-time communication over cots ethernet switches. In *6th IEEE International Workshop on Factory Communication Systems (WFCS'06)*, June 2006.

[13] R. Marau, M. Behnam, Z. Iqbal, P. Silva, L. Almeida, and P. Portugal. Controlling multi-switch networks for prompt reconfiguration. In *Proc. of 9th Int. Workshop on Factory Communication Systems (WFCS12)*, May 2012.

[14] R. Marau, P. Pedreiras, and L. Almeida. Asynchronous traffic signaling over master-slave switched ethernet protocols. In *6th International Workshop on Real Time Networks (RTN'07)*, July 2007.

[15] G. Rodriguez-Navas and J. Proenza. Using timed automata for modeling distributed systems with clocks: Challenges and solutions. *IEEE Transactions on Software Engineering*, 2012.

[16] R. Wang, X. Song, and M. Gu. Modelling and verification of program logic controllers using timed automata. *Software, IET*, 2007.

[17] M. Zhang, S. Shen, J. Shi, and T. Zhang. Simple clock synchronization for distributed real-time systems. In *IEEE International Conference on Industrial Technology 2008*, 2008.