# SEtSim: A Modular Simulation Tool for Switched Ethernet Networks

Mohammad Ashjaei, Moris Behnam, Thomas Nolte

*Mälardalen Real-Time Research Center (MRTC), Mälardalen University, Västerås, Sweden*

*P.O. Box 883, SE-721 23 Västerås, Sweden*

*E-mail address: mohammad.ashjaei@mdh.se*

*Corresponding author. Tel.:+46 21 101772; Fax:+46 21 103110*

**Abstract**

Using high bandwidth network technologies in real-time applications, for example in automotive systems, is rapidly increasing. In this context, switched Ethernet-based protocols are becoming more popular due to their features such as providing a collision-free domain for transmission of messages. Moreover, switched Ethernet is a mature technology. Several protocols based on switched Ethernet have been proposed over the years, tuned for time critical applications. However, research for improving the features and performance of these protocols is still on-going. In order to evaluate the performance of early stage proposed protocols, the mathematical analysis and/or experiments are required. However, performing an experiment for complex network topologies with a large set of messages is not effortless. Therefore, using a simulation based approach for evaluating a protocol's performance and/or properties is highly useful. As a response to this we have developed a simulator, called SEtSim, for switched Ethernet networks. SEtSim is developed based on Simulink, and it currently supports different network topologies of the FTT-SE protocol as well as Ethernet AVB protocol. However, the kernel of SEtSim is designed such that it is possible to add and integrate other switched Ethernet-based protocols. In this paper, we describe the design of SEtSim and we show its scalability.

*Keywords:* Switched Ethernet, Real-Time Networks, Simulation, Evaluation, Simulink, FTT-SE Protocol, Ethernet AVB

## 1. Introduction

Recently, there has been a growing interest in using Switched Ethernet for hard real-time distributed systems as it provides means to improve the global throughput of time-critical message transmissions compared with other real-time network technologies. Switched Ethernet provides traffic isolation and it eliminates the impact of the non-determinism due to the CSMA/CD arbitration, that the original Ethernet was suffering from. Nevertheless, using Commercial Off-The-Shelf (COTS) switches in time-critical applications is not adequate as it has the following limitations: (i) the queues inside the COTS switches are limited in size and may overflow due to an uncontrolled packet arrival, which may lead to drop of some packets, and (ii) COTS switches typically have FIFO queues, that can generate long blocking times for urgent packets. Normally, this problem can be solved using separated VLANs for different priorities, however the number of priority levels still is limited to the number of parallel FIFO queues for each port.

There have been many works addressing the adequacy of switched Ethernet for real-time communication. Many solutions that have been proposed are using enhanced switches such as EtheReal [1] and the EDF Scheduled Switch [2], both reserving a channel for traffic transmission. In addition, recent technologies optimized for quick forwarding have been proposed, such as Ethernet AVB (Audio and Video Bridging) that has gained some momentum in the automotive industry.

Moreover, Avionics Full Duplex Switched Ethernet (AFDX) [3] is developed as a network specification with enhanced forwarding which, unlike the Ethernet AVB protocol, does not rely on a clock synchronization protocol. AFDX has been used mostly in avionics.

Despite the performance improvements offered by using these enhanced switches, their usage result in a high cost and a lower availability compared to COTS switches. Therefore, other protocols have been developed based on overlay protocols that control the traffic loaded to COTS switches. In this context, Ethernet POWERLINK [4] and the FTT-SE protocol [5], both using a master-slave technique, were proposed.

In order to evaluate the performance of different switched Ethernet protocols mathematical analysis and/or experiments can be conducted. However, performing experiments for large scale network topologies, potentially with complex message sets, is not straightforward. Therefore, an approach based on simulation is more effective in particular for the early stages of a protocol.

### *1.1. Goal of the tool*

In this paper we present a modular simulation tool, which is called SEtSim[1] (**S**witched **Et**hernet **Sim**ulator), that can be used to simulate Real-Time Ethernet (RTE) protocols. SEtSim is based on Simulink which makes it modular and allows us to create models of different components, e.g., nodes and switches. Moreover, it can be used for evaluation of real-time control networks, where control nodes exist in the architecture. Using Simulink, as the developing environment, makes the tool possible to use different already implemented control blocks as well as other Matlab toolboxes. Currently, SEtSim is designed and developed to support different architectures of the FTT-SE protocol and the Ethernet AVB protocol. However, once the core is implemented it is easy to extend the simulator with other RTE protocols. Design decisions for developing the tool have been made according to the goals of the simulator. The main goals of SEtSim are listed bellow.

1. Real-time control applications: the ultimate goal for the tool is to evaluate distributed systems consisting of network components and nodes. Moreover, control applications, resource reservation and resource adaptation are the applications we would like to deal with using the tool. Therefore, the environment where the tool is developed should support control functionalities and implementation space for different dunctions in nodes, such as schedulers and control tasks.
2. Ability to connect to the world outside the simulation: it would be helpful for the researchers to evaluate newly developed protocols partially with hardware experiments. For instance, a simulation model can be connected to an Ethernet node via a computer Ethernet port. Therefore, the tool should support such an ability.
3. Modular tool: the functions of the simulator should be developed in a way that they can be reused. Also, the new functions should be easily replaced by new developed ones with a small effort.
4. Response time measurement: it is important that the response time of all messages is measured during run-time. Moreover, the measured response time should be reported after finishing the simulation together with important parameters, such as types of messages and number of deadline misses.

[1]SEtSim is available to download at: https://github.com/m-ashjaei/SEtSim

5. Visualization of messages: one of the goal is to visualize the transmission in each link. The trace of signals is helpful to check the behavior of newly developed protocols.
6. Graphical user interface: it should be easy for the user to maneuver in the tool, thus a graphical user interface is suitable instead of actual coding for modeling an architecture. It significantly reduces the time of evaluation as the user may require to change the architecture frequently.

### *1.2. Organization of the paper*

The rest of the paper is organized in the following manner. The next section discusses some related work on modeling and simulation of real-time network protocols. Section 3 describes three architectures of the FTT-SE protocol. Section 4 presents some backgrounds on the Ethernet AVB protocol. Then, Section 5 presents the SEtSim kernel design, while Section 6 validates the simulator using some experiments. Finally, Section 7 shows the limitations of the tool, and Section 8 concludes the paper.

## 2. Related work

Several techniques have been proposed to model and simulate the Ethernet protocol using different tools and modeling algorithms. In the work presented in [6], models are proposed for nodes, switches and traffic according to the Switched Ethernet protocol. Moreover, an evaluation is performed to validate the performance of the modeling method by comparing the simulation results with the collected data from a specific network application.

In the area of embedded avionics networks, a simulation model considering the AFDX is proposed in [7]. The end systems (nodes), switches, different queue managements in the switch and nodes, and the measurement units to measure the latency of each flow were modeled. Moreover, the validation of the modeling algorithm is performed using a single-switch case-study by applying the worst-case scenario for the flows configuration. However, the simulator was developed only for a particular application and it is not implemented as a general simulator.

Furthermore, a simulation algorithm was proposed in [8] to evaluate the end-to-end upper bound delay in AFDX networks. Finding an upper bound end-to-end delay for each message using simulation, requires to investigate a huge number of possible scenarios. Thus, an approach to reduce the number

of possible scenarios was proposed in [8]. It should be noted that two approaches were presented in the literature to compute the end-to-end delay of traffic in AFDX networks. These approaches include the analysis based on the network calculus [9], and the analysis framework based on the trajectory approach [10].

In addition, different network simulation systems were designed based on available simulation tools. For instance, a network simulator system for AFDX networks was designed and implemented in [11], in which Network Simulation (NS2) as a tool to simulate TCP, routing over wired and wireless networks, was considered for the main platform. However, implementing a particular model on an existing tool is straightforward, whereas extending a tool for a general model is not an easy task. Also, the work presented in [12] proposed a component-based model for AFDX networks in order to verify the behavior of the network model.

As the requirements in automotive industries have changed to support new functions using Ethernet, a simulation environment based on OMNeT++ has been proposed in [13] to evaluate mixed CAN-Ethernet networks. Furthermore, there are many general tools developed for network simulation, such as TrueTime [14], NS2 [15], OMNET++ [16] and OPNET [17].

TrueTime is a toolbox developed based on Simulink. The switched Ethernet protocol as a network block has been supported by the TrueTime toolbox. However, adding new protocols, such as the FTT-SE protocol, needs a lot of modifications and changes to the kernel of the tool. Moreover, the output results that can be generated from the TrueTime blocks are limited and they need to be modified to allow for calculation of response time of messages. Another tool is called OPNET, that is used to evaluate the performance of a network, specially for evaluation of Internet. However, this tool is a commercial tool, and its source code is not freely available to develop new protocols. OMNET++ and NS2 are modular simulators mainly used for sensor networks, internet protocols and performance modeling. NS2 handles a lot of different network protocols, yet it supports limited number of real-time protocols. The core of these simulation tools should be completely changed to cover an RTE protocol, such as the FTT-SE protocol. Basically, some of the RTE protocols require changing in the Ethernet frame (e.g., in both the FTT-SE and the Ethernet AVB) and changing different network layers. Therefore, implementing a new simulation tool, in particular for RTE protocols, is useful in the real-time communication research area.

NS2 uses two programing languages, C and oTCL, where oTCL is an

object oriented language utilized for the graphical interface. Implementing a user-friendly interface in NS2 is very much dependent on oTCL language that is not rich for this task. For instance, developing a block for a component (e.g., a node) with a feature of dragging and dropping is not easy in NS2. Note that, in NS3 oTCL is completely removed because of a big overhead for large simulations [18], however it is not completely settled for extension. Also, in order to build a network example, a descriptive language is required to create nodes and connect them in a network. This increases the chance of making mistake, even in the first stage, i.e., building a network model. In fact, one of the motivations to use Simulink as backbone is providing a user-friendly interface by drag and drop of components in a model. Moreover, NS2 is basically designed for Linux users. Although the installation guide is available for Windows users, there are many bugs reported as some extra packages are required to be installed (e.g., Cygwin package).

As a result of the above survey, neither of the mentioned tools can be used directly to include RTE protocols (e.g., the FTT-SE protocol) along with corresponding response time calculations for messages. Besides, one of the main goals of the tool, except graphical interface and message trace visualization, is using that in control applications. Normally, in control applications different types of signals along with control tasks are used. These control blocks and signals are already designed and tested in Matlab toolboxes. Moreover, an ultimate goal is to use the tool for end-to-end evaluation of distributed embedded systems. Therefore, in this paper we present SEtSim which is developed based on Simulink.

## 3. The FTT-SE protocol

The FTT-SE protocol is a bandwidth-efficient protocol, with respect to the COTS switch [5], which uses the master-slave technique. This protocol was initially presented in [5] for small size networks consisting of one switch and few nodes. In order to extend the protocol to support multi-hop communication, three different approaches were proposed, connecting multiple switches in a tree topology. The first architecture [19] uses a single master connected to the top of the network tree in order to control the traffic transmission in the whole network. In the second architecture [20], each switch has an attached master node, whereas in the third architecture [20] the messages within a group of switches are controlled by one master node. In this section, we briefly sketch these three architectures.

*3.1. Single-master architecture*

The single-master architecture is depicted in Figure 1, in which one master node is attached to the top of the network hierarchy, controlling the traffic in the whole network. The master node is responsible to schedule messages on-line according to any desired scheduling policy (e.g., Fixed Priority Scheduling), on a cyclic basis. The basic cycle has a fixed duration of time and it is called Elementary Cycle (EC). Each EC is partitioned among two types of traffic, i.e., synchronous and asynchronous traffic, resulting in a design where the EC hosts one synchronous and one asynchronous window (Figure 2). The scheduler in the master node checks the synchronous messages, which are activated periodically, whether they can be transmitted during the associated window within EC (i.e., within the synchronous window). The scheduled messages are encoded into a Trigger Message (TM) that is transmitted to all slave nodes at the beginning of the next EC. The TM transmission is based on the master-slave technique where the traffic transmission is controlled by a master node and triggering the transmission within specific cycles.

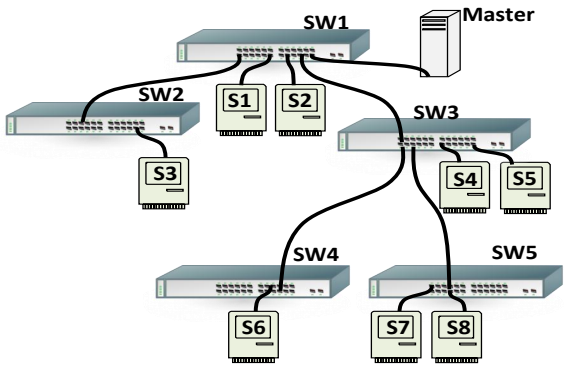



Figure 1: The Single-Master Architecture

The activation of asynchronous messages is unknown in advance and can occur at any time during the EC. Therefore, a signaling mechanism [21] allows the slave nodes to inform the master node about potential pending requests using a Signaling Message (SIG), which is transmitted after reception of the TM (e.g., A, B and C in Figure 2). The master then schedules the asynchronous messages adequately and inserts them into one of the upcoming

TMs. The slave nodes receive the TM, decode it and initiate the transmission of the scheduled messages and in parallel initiate reception of data messages from other slave nodes. Note that, decoding the TM takes an amount of time called Turn Around Time (TRD) (Figure 2). The actual length of the TRD depends on the processing speed of the slave nodes. The response time analysis for the traffic transmitted through the switches in this architecture is developed and presented in [19].

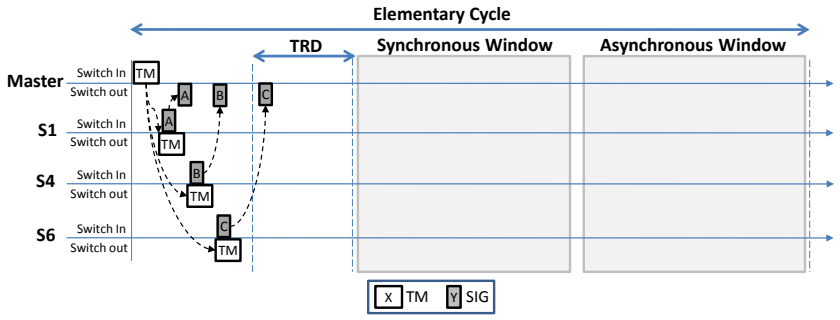



Figure 2: The Elementary Cycle in Single-Master Architecture

### *3.2. Multi-master architecture*

Unlike the previous architecture, the traffic in the multi-master architecture is coordinated using multiple master nodes, each of which is connected to one switch, as illustrated in Figure 3.

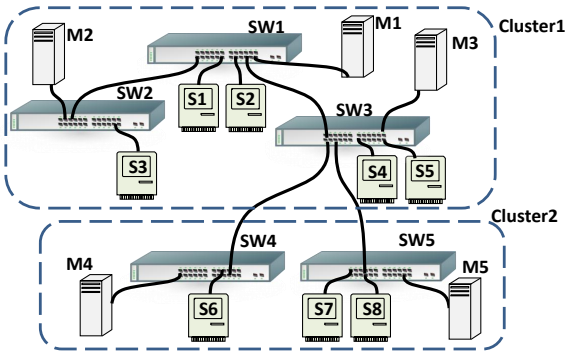



Figure 3: The Multi-Master Architecture

In this architecture, each switch along with all associated nodes that are connected to it is called a sub-network (e.g., SW1, M1, S1 and S2). Moreover, each sub-network is a parent for lower level sub-networks in the tree topology. A group of sub-networks with the same parent sub-network is called a cluster (e.g., Cluster 2 in Figure 3). Note that, the root sub-network is included in its children cluster because it cannot be considered as a separate cluster. Also, the traffic is categorized into two types. A message that is transmitted within a sub-network is called local, whereas a message is called global if it is transmitted beyond a sub-network.

The EC, in this architecture, is divided among different traffic types, i.e., local/global and synchronous/asynchronous. Also, the global asynchronous window is further split among clusters. The EC partitioning is illustrated in Figure 4.

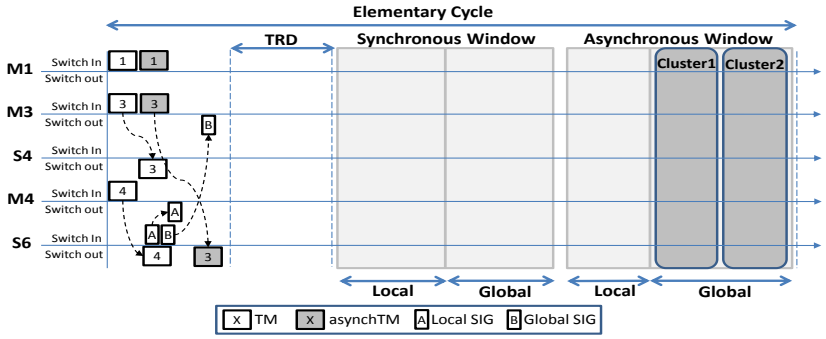



Figure 4: The Elementary Cycle in Multi-Master Architecture

Each master node schedules its associated local synchronous and asynchronous traffic within the dedicated windows. The request for local asynchronous traffic is sent to the sub-network master node in order to be scheduled for the next EC (e.g., message A from S6 to M4 in Figure 4). The global synchronous traffic is scheduled by all master nodes in parallel to have a consistent scheduling in the entire network. The scheduled messages are encoded into the TM and transmitted to the slave nodes in the beginning of the EC. As there is no global view of the activation of global asynchronous traffic, the master nodes cannot schedule them in parallel. Therefore, the master of each cluster is responsible for scheduling them by receiving a separate SIG request from children slave nodes (e.g., message B from S6 to M3 in Figure 4). Then, the global asynchronous traffic is scheduled within a particular dedicated window (e.g., cluster1 sub-window within the asynchronous

window) and the identifiers of the scheduled messages are encoded into a different TM, which is called asynchTM, to be transmitted after the regular TM (e.g., asynchTM3 that is sent from M3 to S6 in Figure 4).

Slave nodes receive both the TM and the asynchTM, they decode them and they initiate the message transmission. Moreover, the SIG requests, both for local and global asynchronous messages, are sent by the slave nodes concurrently. Note that, the partitioning of the windows for message transmission is for scheduling purposes only. The slave nodes start to send their scheduled messages immediately without considering the partitioning of the windows.

In addition, this architecture requires that all master nodes are timely synchronized. Therefore, a clock synchronization mechanism, used among the master nodes, is presented in [22]. The response time analysis of the traffic that is sent through multiple switches is developed and presented in [20].

### *3.3. Cluster-based architecture*

The cluster-based architecture is a hybrid solution of single- and multi-master architectures. The topology is the same as the multi-master architecture, except having a single master for each cluster. The network example is depicted in Figure 3 considering M1 and M3 in the network. The former master schedules the traffic within cluster 1 and the latter master schedules the traffic within cluster 2.

By dividing the network into clusters, the traffic types are categorized as follows. The internal traffic is transmitted within a cluster, whereas the external traffic is sent beyond the cluster. In addition, according to the traffic types, the windows allocation is also split among internal/external and synchronous/asynchronous traffic (Figure 5). The external asynchronous window is still further split into cluster sub-windows.

In this architecture, each master schedules the associated internal messages and all external messages in parallel with other master nodes. SIG messages are used to inform the master of the associated cluster, where the internal/external synchronous messages are requesting to be scheduled (e.g., A, B and C in Figure 5). All the scheduled messages are encoded into a TM to be sent to the slave nodes within the cluster.

The slave nodes commence the message transmission after receiving the TM and decoding it. The request for the asynchronous messages is sent to the master of the cluster in parallel with receiving the TM. This architecture uses less signaling messages compared with the multi-master architecture, while

having the benefits of a lower number of master nodes, i.e., using one TM per each cluster and one SIG per slave node. This topology also requires to have timely synchronized master nodes. Therefore, the same clock synchronization mechanism [22] is used in the cluster-based architecture.

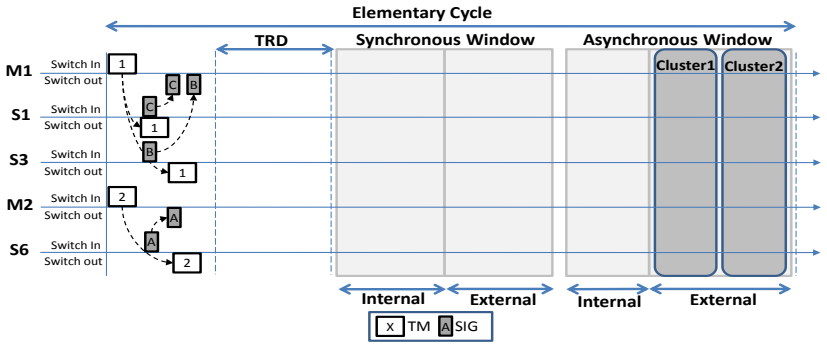



Figure 5: The Elementary Cycle in Cluster-Based Architecture

## 4. The Ethernet AVB

Ethernet AVB protocol is a common name to a set of specifications that includes the following main amendments: (i) IEEE P802.1Qav - Forwarding and Queuing Enhancements for Time-Sensitive Streams, (ii) IEEE P802.1Qat - Stream Reservation Protocol, (iii) and IEEE P802.1AS - Timing and Synchronization.

The IEEE P802.1Qav specifies a set of rules in order to queue and forward the time-critical traffic, i.e., it provides a bounded latency to real-time messages. The standard introduces two transmission algorithms according to a certain application need. The first one is called strict priority transmission algorithm (PQ) and the second one is called credit-based shaping algorithm (CBQ). The non-real-time traffic is transmitted using the former algorithm, whereas the real-time traffic is forwarded according to the latter algorithm. The model of an output port for an AVB switch is depicted in Figure 6. The time-critical traffic is entitled to be Class A or Class B according to its priority. The operation of the CBQ for one queue (e.g., queue for Class A) is depicted in Figure 7.

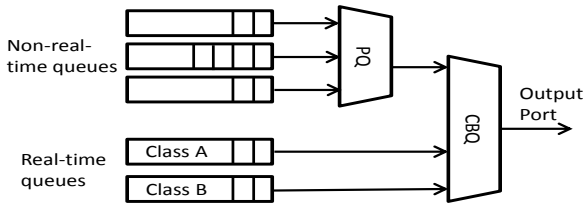



Figure 6: The AVB Output Port Model

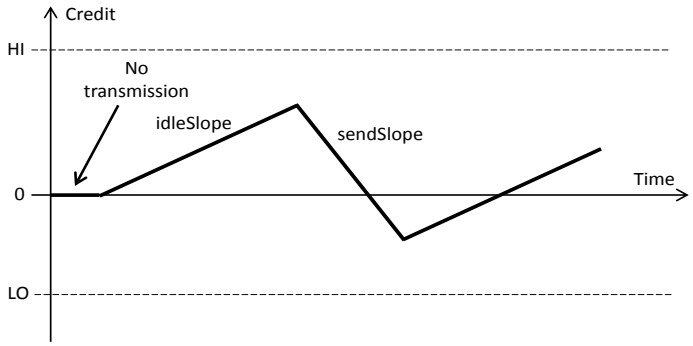



Figure 7: The CBQ Operation

According to the CBQ, a message is queued when it becomes ready. Note that the traffic pattern in AVB network is periodic. The message is selected from the queue for transmission if there is a zero or positive credit. During the message transmission the credit decreases at a constant rate, which is called *sendSlope*. After the transmission, if the credit is still positive, another message in the queue is selected for transmission. However, if there is a high priority message with positive credit is waiting for transmission, it will be selected. After consuming the credit, when there is a message in the queue, the credit increases with a constant rate, which is called *idleSlope*. The increasing and decreasing rates of the credit is set according to the application by defining a bandwidth fraction for the queue, see (1) and (2), where $f$ is the fraction of bandwidth assigned to the queue and $Rate$ is the port transmission rate in *bps*.

$$idleSlope = f \times Rate \tag{1}$$

$$sendSlope = idleSlope - Rate \tag{2}$$

If a given traffic class queue (e.g., Class A) does not have any message available in the queue, the lower priority traffic class queue (Class B) can then be sent. The unused bandwidth, i.e., when the credit is not positive for both Class A and Class B, is available for transmission of non-real-time traffic using the PQ algorithm.

Another draft of the standard is the IEEE P802.1Qat which specifies the admission control in order to manage the resource to be reserved for the specific streams. This is mostly done by the Stream Reservation Protocol (SRP) which introduces a producer-consumer model to negotiate for the resource reservation. The last part of the standard, the IEEE P802.1AS, specifies a clock synchronization protocol to ensure that the synchronization requirements are met.

## 5. Simulator design

Using Simulink/Matlab, we have developed SEtSim to evaluate the timing behavior of messages in switched Ethernet networks. The main ideas of SEtSim are to support RTE protocols and to have a user-friendly graphical interface in order to build a model easily. Simulink can fulfill the latter aim as it allows us to implement nodes and switches as a block to let the user build a network model by drag and drop of the blocks. This is achieved by developing a component model (e.g., the switch) using custom blocks which are available in Simulink. Moreover, there are many predefined blocks exist in Simulink that can be directly used in SEtSim, such as scope blocks and signal generators.

The core of SEtSim is based on a cycle-based approach, where execution of all functions are done within a fixed duration of time, known as time-slots. To simulate the parallel execution of blocks in a model, we have divided the time into a number of small time slots in each of which all the functions are executed. The number of time slots shows the resolution of the simulation which is not fixed but can be changed in the configuration of the simulator. Changing the number of time slots does not affect the accuracy of the message response time. In fact, in any resolution the response time of the traffic in a particular model is constant. However, it affects the accuracy of message passing visualization by a scope block in Simulink, i.e., the granularity of showing on the scope will be different. Note that, the function of blocks in Simulink executes in sequential order which is automatically specified by Matlab in advance. Therefore, the input data of each block is guaranteed

to be available before its execution. In addition, the kernel of SEtSim is designed to be modular. Each block, e.g., the master block, contains several functions which can be replaced with a new function. For instance, the scheduler is a separate function that can be replaced with another scheduling strategy, e.g., EDF Policy, without changing in the rest of the block. This modularity is applied to the main scheduler, reading the inputs, ready queue management, scoping the output, measuring the traffic response time and queue management in AVB switches. In this section we describe the design of the different blocks in SEtSim.

In general, we use S-Function block in Simulink to implement different components, such as Ethernet switch and nodes. S-Function is a customized Simulink block that allows the user to implement a functionality in it. It is possible to define any number of input and output ports for S-Function block. Also, different functions can be developed based on the input values. Each block is executed once in a time-slot.

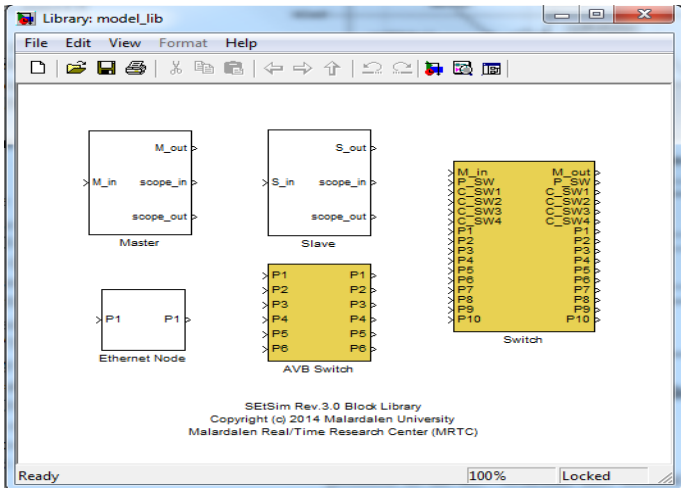



Figure 8: SEtSim Blocks Library

In SEtSim, we have developed five basic models using the S-Function level2 custom block in Simulink to simulate the functionality of the master node, slave node, switch model, Ethernet node and AVB switch. These blocks are stored in a Simulink library file, as illustrated in Figure 8. The

user of SEtSim prepares a network model to evaluate by dragging the blocks from the library and dropping to the new model. Then, the connections between the input and output ports should be done. The further details for the functions are given in this section.

*5.1. Ready queues management*

Each master contains four different ready queues to hold the activated messages of four different message types, including local/global / synchronous / asynchronous in the multi-master architecture and internal/external / synchronous / asynchronous in the cluster-based architecture. The exception is the single-master architecture in which the master uses synchronous and asynchronous queues, only, as it has just two types of traffic. The ready queues are sorted based on the priority of messages in which the highest priority messages are inserted at the head of the queues. We have used the Fixed Priority/Highest Priority First Scheduling Policy based on Rate Monotonic priority assignment algorithm for on-line scheduling in SEtSim. Messages with the same priority are sorted in the queues based on the First Come First Serve (FCFS) policy. For management of the ready queues, we have developed three functions to handle updating the queues before scheduling the messages. These functions, which are implemented in Matlab m-files, are the following:

**Get head message.** This function returns the first message in the ready queue which is always the highest priority message among all messages in the ready queue.

**Remove a message.** If the scheduler checks a message and it selects that message to be transmitted in the current EC, the message should be removed from the ready queue. Therefore, this function removes a message defined by its id together with the ready queue in which the message is residing. The output of this function is the updated ready queue sorted according to the priorities of all messages in the queue.

**Insert a message and sort in ascending format.** Whenever a message becomes ready, it should be inserted in the correct ready queue. This function inserts a message in a queue according to its priority and it re-sorts the queue according to the priorities of messages in which the highest priority message is assigned at the head of the queue.

Note that, the ready queue management is not used in the Ethernet AVB protocol simulation as there is no such ready queues in this protocol.

The output queues are managed with the PQ and CBQ algorithms in FIFO queues.

*5.2. Master block design*

The master block is divided into two sub-blocks dealing with sender and receiver functions. We have defined an array structure for the master node to store its variables and parameters. All master blocks in the model are connected to a single m-file function, which is distinguished with a mask block parameter number, i.e., the mask parameter in Simulink for each block is set with a unique value.

For each master input (receiver) and output (sender) blocks two separate functions are implemented. However, the master function is developed based on a state flow such that each of them should run in order. The master function has three states which are depicted in Figure 9-a. Each state is allowed to run when the previous state has executed, only. The first state is broadcasting the TM (and asynchTM in case of multi-master architecture) to the associated slave nodes. The next state (State 2) is receiving SIG requests from the associated slave nodes during the TM window. The last state (State 3) is performing the scheduling function for all types of messages, and generating a TM (and asynchTM) for the next EC. State 1 and 3 are executed in the master sender function, whereas State 2 is executed in the receiver function. For input and output signals, we have implemented different scope functions (scope-in and scope-out), as shown in Figure 9-b.

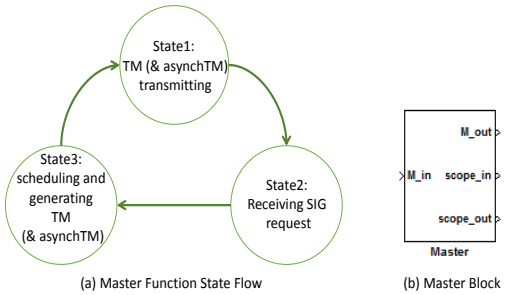



Figure 9: Master Block

Depending on the architecture that has been chosen (i.e., single-/multi-master or cluster-based), the master collects the SIG requests from the associated slave nodes in State 2 of the master function (e.g., collects from all

the slave nodes inside a cluster for the cluster-based architecture). This state finishes and moves to the next state when all SIGs are received. The requests are stored and the master schedules them for the next EC in State 3. In case of the multi-master architecture, the scheduled global asynchronous messages are encoded into a different TM (asynchTM) to be sent to the corresponding children slave nodes.

The algorithm of State 3 for the master block is depicted in Figure 10, which shows the scheduling of one (out of four) ready queue. However, scheduling the messages in the other ready queues is similar to the one presented in Figure 10. The algorithm starts by updating the ready queue and recording the ready time for the ready messages. The ready time is required when calculating the response time of the messages in the simulation. Afterwards, the algorithm checks whether the messages in the ready queue can be fitted in the dedicated window for each type of message according to the route of the message. The remaining messages, which are not fitted in the dedicated window, are kept in the ready queue for the next scheduling round (i.e., the next EC). Finally, the algorithm generates the TM (and asynchTM in case of a multi-master architecture) to be transmitted in State 1 of the master function (Figure 9).

### *5.3. Switch model design*

Similar to the master block, the switch is modeled with a Matlab S-Function using a particular m-file associated to that. Four kinds of connections are defined for the switch model: (i) the master connection identified in port 1, (ii) the parent switch connection is dedicated to port 2, (iii) four children connections for the children sub-networks, and finally (iv) ten connections for the slave nodes. Moreover, for each input and output ports a specific buffer is assigned to store receiving and sending data. The buffers are sorted according to the FIFO policy, similarly to the COTS switches. The general structure of the switch model is to poll the input data and to process the destination address of them, and in turn, insert the data into the related output buffer.

### *5.4. Slave block design*

Similar to the master block, the slave block is divided into two sub-blocks, which are sender and receiver sub-blocks. The slave function is executed

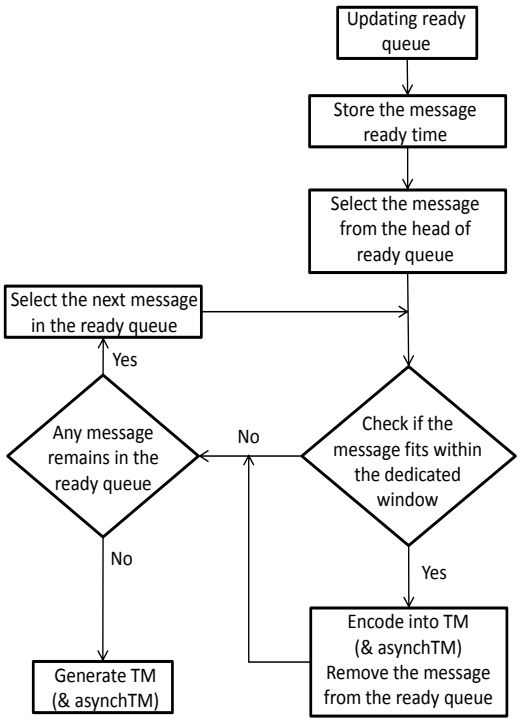
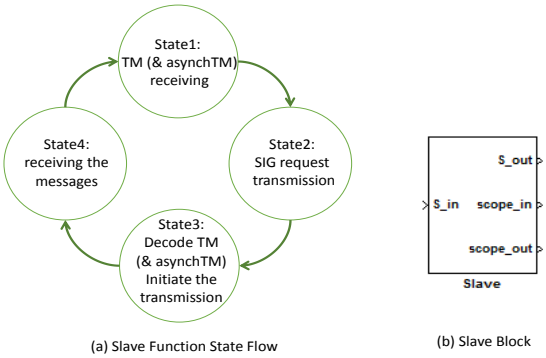



Figure 10: Algorithm of State 3 in The Master Block

based on the state flow which indicates the current state of each slave node. The slave function is composed of four individual states started by the TM reception. The state flow of the slave function is depicted in Figure 11-a.

After receiving the TM (and asynchTM) from the master (and the parent master) node, the slave node checks whether any message is ready to be transmitted. For asynchronous messages, the sporadic model is used to model this type of traffic in which the minimum inter-arrival time is defined for each message. Therefore, in SEtSim activation period of asynchronous messages are randomly set during run-time considering its minimum inter-arrival time. Moreover, the asynchronous messages may become ready at any time during the EC window. In the worst-case scenario the SIG from the slave node is already transmitted when the asynchronous message becomes ready. This means that the asynchronous message is activated slightly after sending the SIG from the slave node so that the SIG cannot include this activation. To consider this behavior in the simulation, the asynchronous messages are always activated after transmission of the SIG. The third state of the slave




(a) Slave Function State Flow

(b) Slave Block

Figure 11: Slave Block

function is decoding the TM (and asynchTM) and transmitting the scheduled messages.

After sending the scheduled messages, the last state (State 4) is to wait for message receiving. The slave nodes read their inputs until the EC time window is finished. When a message is received from the slave node, the receiving time is stored in the related message variable. The time interval between the ready time (stored during State 3 of the master function) and the receiving time of the messages shows the response time of the message. For setting the receiving time, the store-and-forward switch delay and order of messages are considered to simulate as accurately as possible. Since the transmission window was checked in the scheduler, then all scheduled messages should be received in the current EC without any overruns to the next EC. Overrun of a message occurs when the message is not completely received by the end of the EC, hence it interferes with the next EC operation. In case of any overrun, deadline miss or failure in receiving of the message will be reported when simulation is stopped.

For generating the scope output signals to cover both receiving and sending messages, the scope functions store the messages which are sent and received. The messages are scoped according to their transmitting time. Each message is indicated by its identification number which is unique in the entire network.

### *5.5. Ethernet node design*

Ethernet node is designed as a block to be connected to an AVB switch. The node is in an idle mode until a message, to be transmitted from the node, becomes activated. The activation time for all messages is computed

by the node where the messages belong to. Note that the messages in AVB networks are activated periodically. After activation, the node initiates the message transmission. In order to record the response time of a message during run-time, we store the transmission time and the reception time of the message in the function block of the node.

*5.6. AVB switch design*

In order to develop an AVB switch a Matlab S-Function custom block is utilized. The AVB switch contains six ports where the nodes are attached. For each output port, three FIFO queues are considered: (i) traffic class A queue, (ii) traffic class B queue, and (iii) the non-real-time traffic queue. The received messages in the switch is queued according to its defined class. Also, two forwarding algorithms, the PQ and the CBQ, are implemented in order to handle the non-real-time traffic and the time-critical traffic, respectively.

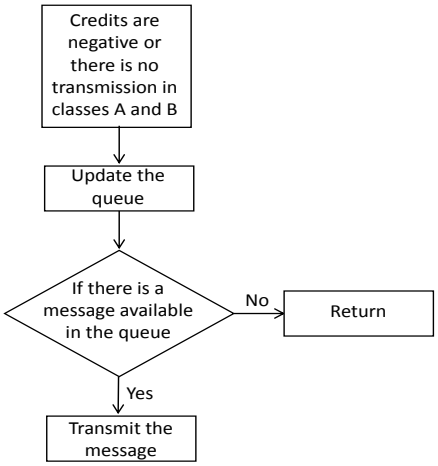



Figure 12: The PQ Algorithm

The PQ algorithm is presented in Figure 12. When the credits of the critical queues are zero or when there is no message in classes A and B, the switch initiates the transmission of the non-real-time traffic using the PQ approach. Therefore, the queue is updated and the first message in the head of the queue is transmitted. In case of no message in the queue, the algorithm returns to the loop where the other queues are serviced.

Furthermore, the algorithm for the CBQ approach is depicted in Figure 13. Note that the algorithm shows one of the classes, however it is the same for all classes of the traffic. The algorithm checks whether any message is available in the queue. If so, the credit for that queue is examined to see if there is a zero or positive credit. In case of zero or positive credit, the message will be transmitted and at the same time the credit is decreased. In contrast, if the credit is negative there will be no transmission and the credit will be increased. The rates of decreasing and increasing for the credit is defined in the configuration m.file, by setting the idleSlope and sendSlope parameters. If there is still a message in the queue, it will start for transmission under two conditions. First, having a remained positive credit and second, no high priority traffic class is ready for transmission. It should be noted that if the queue is empty the credit becomes zero immediately.

*5.7. Settings and configuration*

In order to set the configuration of a network example, such as the EC size, the windows allocation for the FTT-SE protocol and bandwidth set for different classes in the Ethernet AVB protocol, a Matlab m-file is developed. For each network model, different configurations can be determined to assess the performance of the example.

Besides the windows allocation for different message types, other configurations such as number of slave and master nodes in the model, and the message set declaration, should be set before running the simulation. The message set declaration includes the parameters such as transmission time, source node, destination node, period/minimum inter-arrival time, priority and the routing information for the messages.

*5.8. Output and reports*

SEtSim generates two types of output for performance evaluation, (i) the message transmission in the scope block of Simulink, and (ii) a report that shows the response time of the messages. In order to check the message transmission, it is required that a scope block in Simulink is connected to any master or slave node to visualize the input and output messages. Moreover, the output report presents the minimum, average and maximum response time of all messages that are measured during the simulation. This can help to compare the response time of messages to analytical results to figure out the level of pessimism embodied in the analysis, for instance.

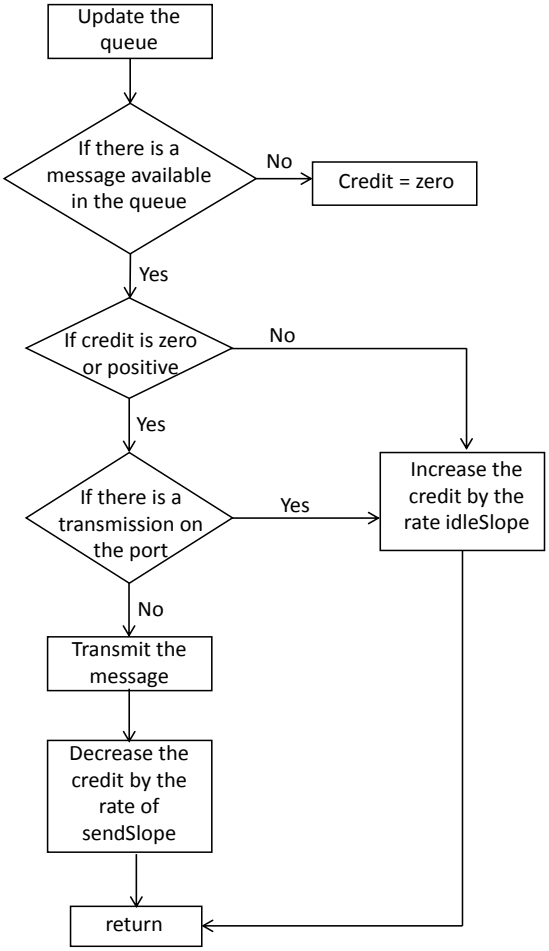



Figure 13: The CBQ Algorithm

## 6. Examples

In this section, we present three different examples. The first one is a network which is formed as a cluster-based FTT-SE architecture consisting of ten switches and 100 slave nodes (ten slave nodes connected to each switch). In the second example we keep the network topology and we change the protocol to multi-master FTT-SE architecture. These two examples show the scalability of SEtSim to large scale networks when using the FTT-SE protocol. Note that SEtSim does not have any limitation in the number of components including switches, nodes and messages. However, our motiva-

tion in this section is to show the applicability of the tool in the area of automotive domains, where the generated examples are relatively large. Finally, the third example is a small network using the Ethernet AVB protocol.

### *6.1. Cluster-based FTT-SE example*

In this example, the network is composed of ten switches along with 100 slave nodes. The parameters for this example are set as follows. The Elementary Cycle (EC) is set to $10ms$, the synchronous window is $4400\mu s$ and the asynchronous window is $5500\mu s$. Within the synchronous window, $2000\mu s$ is dedicated to internal traffic, while $2400\mu s$ is assigned to external traffic. Moreover, the asynchronous window is divided between internal and external messages, $2000\mu s$ for the former and $3500\mu s$ for the latter window. The external asynchronous window is split among the three clusters equally, i.e., $1160\mu s$ for each of them. The transmission speed for this example is considered as $100Mbps$ and 3500 messages are generated randomly. Due to the limited space in this paper, we illustrate a part of the network in Figure 14 and a part of the root sub-network in Figure 15.

The simulation for this example is performed for a time duration of 10000 ECs that took 10 minutes in real time using a computer with Intel Core(TM) i5-2540M CPU at 2.60GHz with 8GB RAM. Note that, using different computer configuration (e.g., different CPU speed) affects the simulation time as the functions could execute faster in higher CPU capacity. However, for this example with a large number of nodes and a large set of messages the simulation time is reasonable. To visualize the message transmission, an ordinary Scope block of Simulink is attached to the respective scope ports of the slave blocks in the model. In this paper, we illustrate three messages among 3500 messages due to the space limitation, which are $m5$, $m9$ and $m12$. The selected messages are synchronous with a period of 2 EC and are assigned the highest priority (priority equals to 1). Moreover, the messages have different transmission times and all of them are transmitted from slave node number 1. Figure 16 shows the output of slave number 1, in which the x-axis is the simulation time and the y-axis shows the message IDs. The selected messages are transmitted sequentially and they are marked in the scope figure.

In addition, Figure 17 depicts the input of slave number 3, which is the slave that $m5$ is transmitted to. As it can be seen, $m5$ is received with a delay of its transmission time and the store-and-forward delay in the switch.

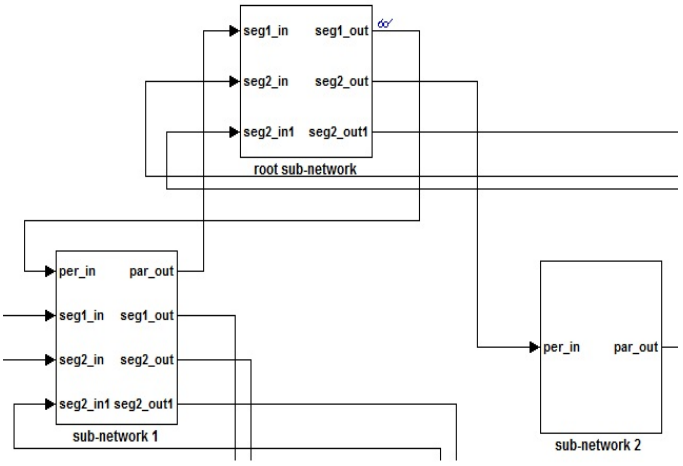



Figure 14: A Part of Network Example Model

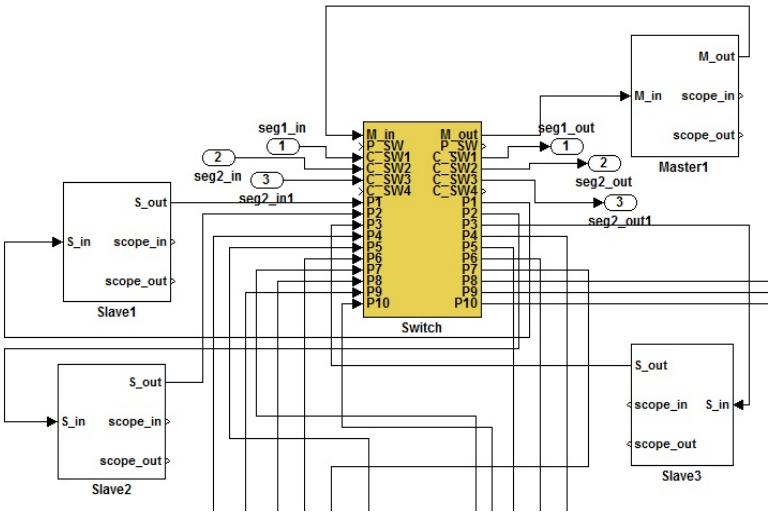



Figure 15: A Part of Root Sub-Network Model

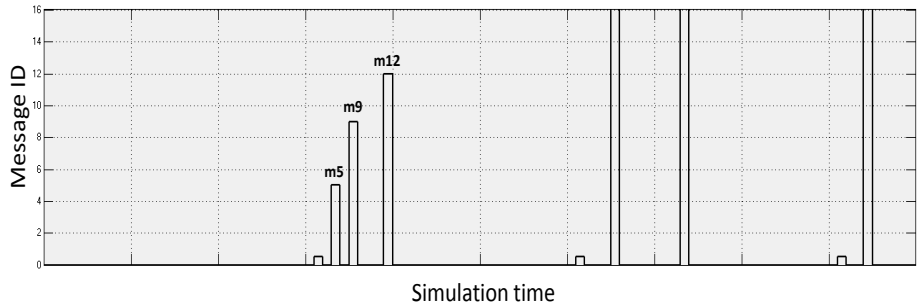



Figure 16: Scope Output of Slave Number 1

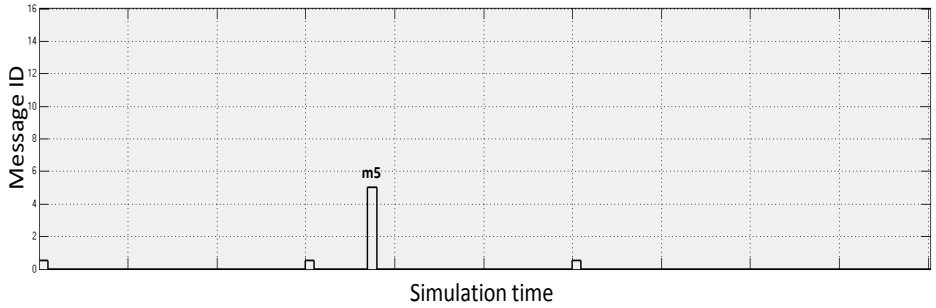



Figure 17: Scope Input of Slave Number 3

The receiving of $m9$ to slave number 11 and $m12$ to slave number 41 is illustrated in Figure 18 and 19, respectively.

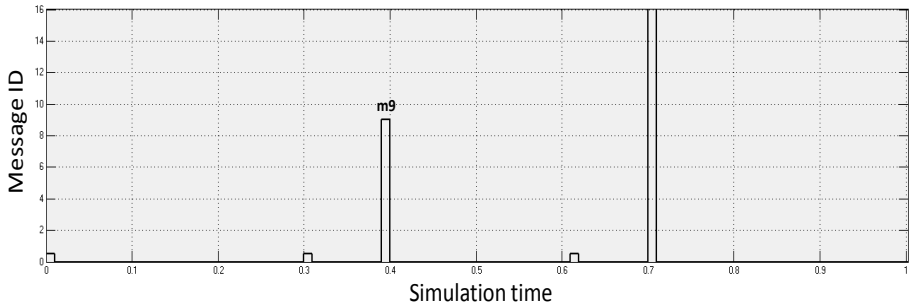



Figure 18: Scope Input of Slave Number 11

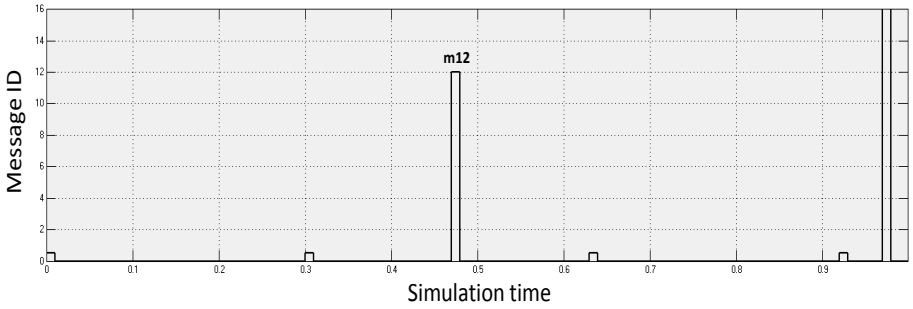



Figure 19: Scope Input of Slave Number 41

Note that, the simulation time in the scopes (Figures 16, 17, 18 and 19) is presented in $10ms$ unit, i.e., each unit, from 1 to 2 in the simulation time of scope, represents $10ms$ of the simulation.

In addition to the message transmission visualization, the response time, which is measured during the simulation, can be reported for all messages in the model. The minimum, average and maximum measured response time for the selected messages is presented in Table 1. Note that, the response time are presented in integer number of ECs.

| id | Min RT (EC) | Avg RT (EC) | Max RT (EC) | Period (EC) | Deadline (EC) |
|---|---|---|---|---|---|
| 5 | 1 | 1 | 1 | 2 | 2 |
| 9 | 1 | 1 | 1 | 2 | 2 |
| 12 | 1 | 2 | 2 | 2 | 2 |

Table 1: Response Time for the Selected Messages

As it can be seen in Table 1, the response time measured for $m12$ during the simulation has minimum 1 EC, while it is 2 EC for maximum and average. The reason to have equal response times for average and maximum is that the average response time is presented as a ceiling of mean among the measured response times. This is due to presenting the response time in number of ECs for the FTT-SE protocol.

### *6.2. Multi-master FTT-SE example*

In order to run the simulation for the same example, but in the multi-master architecture, we need to change the architecture flag variable in the configuration file, only. Changing the model in Simulink is not required as the kernel of the tool handles that easily. This option is helpful when one network example is needed to be evaluated in different protocols. The simulation is

performed for the new architecture (i.e., the multi-master architecture) for 1000EC. We present the response time of the three messages as $m130$, $m232$ and $m1156$ in Table 2.

| id | Min RT (EC) | Avg RT (EC) | Max RT (EC) | Period (EC) | Deadline (EC) |
|---|---|---|---|---|---|
| 130 | 4 | 4 | 4 | 5 | 5 |
| 232 | 7 | 7 | 7 | 7 | 7 |
| 1156 | 1 | 2 | 3 | 6 | 6 |

Table 2: Response Time for the Selected Messages

### *6.3. Ethernet AVB example*

In this example, we model a network composes of two AVB switches along with three nodes, as depicted in Figure 20. The transmission speed is considered as $100Mbps$. From this bandwidth, 40% is dedicated to the traffic class A and 40% is allocated for traffic class B. The rest of the bandwidth is available for non-real-time traffic transmission, i.e., when there is no message in traffic Class A and Class B queues. Moreover, ten messages are randomly generated, where their transmission times are set within $[80, 120]\mu s$ and their priorities are selected as class A (high), class B (low) and non-real-time message, randomly.

The simulation is performed for 1 minute of simulation time. Table 3 shows the minimum, average and maximum response time of three out of ten messages, which is measured during the simulation. These three messages are $m_1$, $m_2$ and $m_{10}$, which are assigned as Class A, Class B and non-real-time types. Note that, the response times, perdiod and deadline of the traffic is shown in milliseconds in Table 3.

| id | Min RT ($ms$) | Avg RT ($ms$) | Max RT ($ms$) | Period ($ms$) | Deadline ($ms$) |
|---|---|---|---|---|---|
| 1 | 0.25 | 0.26 | 0.35 | 20 | 20 |
| 2 | 0.25 | 0.25 | 0.25 | 30 | 30 |
| 10 | 0.30 | 0.45 | 0.55 | 50 | 50 |

Table 3: Response Time for the Selected Messages

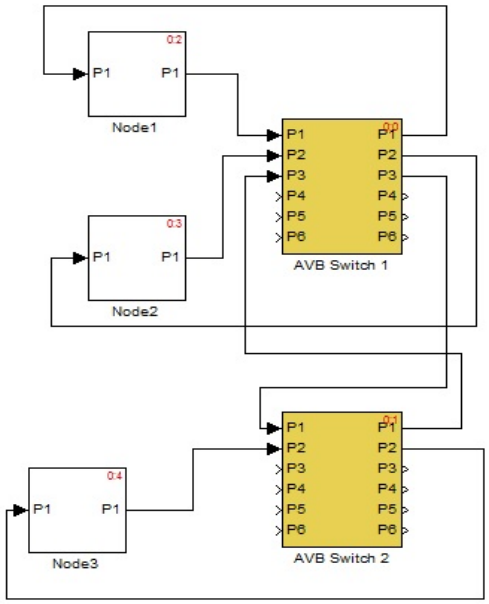



Figure 20: The Example of AVB Switch

## 7. SEtSim limitations

The aim of SEtSim is to cover RTE protocols. In the first version of the tool we have developed different architectures of the FTT-SE protocol as well as the Ethernet AVB protocol. Therefore, it lacks other alternatives such as the HaRTES architecture [23], TT-Ethernet and AFDX protocols.

Moreover, in the current implementation, the number of ports for a switch is limited to ten nodes and it is limited to six nodes for an AVB switch. This limitation can be removed by allowing the user to define the number of ports up to 33 ports, which is supported by the available COTS switches.

In addition, the output generation of the tool is limited to the message passing visualization and the traffic response time measurement. However, adding extra output reports including some statistics such as confidence interval is remained for future work.

## 8. Conclusion and future work

In this paper, we have designed and developed a simulation tool, which is called SEtSim, in order to evaluate the new proposed architectures and protocols based on switched Ethernet. This version of SEtSim supports three

different architectures of the FTT-SE protocol, which are briefly introduced in this paper, and the Ethernet AVB protocol. Moreover, we presented the design of SEtSim, which contains five basic models for the master, the slave, the switch, the Ethernet node and the AVB switch, all implemented as Function blocks in Simulink.

In addition, we demonstrated three network examples, two examples using the FTT-SE protocol and one to show the Ethernet AVB functionalities. The FTT-SE examples consist of ten switches along with 100 slave nodes with a set of 3500 messages to show the scalability of SEtSim. Different output scopes of message transmissions are presented as well as the response time reports. Also, we illustrated an example of Ethernet AVB network comprises one AVB switch with three nodes. We also showed the response time of the traffic transmitted in the Ethernet AVB network. SEtSim allows us to perform a detailed analysis of the protocol before potentially making a hardware implementation of it, with all its associated complexity.

SEtSim is extensible in the sense that we can easily accommodate other switched Ethernet protocols, as the kernel is designed to be modular. The ongoing work is to remove the limitations inside SEtSim such as the availability of a bounded number of supported protocols.

## References


[1] S. Varadarajan, T. Chiueh, EtheReal: a host-transparent real-time fast Ethernet switch, in: 6th International Conference on Network Protocols, 1998.

[2] H. Hoang, M. Jonsson, Switched real-time Ethernet in industrial applications - deadline partitioning, in: 9th Asia-Pacific Conference on Communications, 2003.

[3] I. Land, J. Elliott, Architecting ARNIC 664 (AFDX) Solutions, 2011.

[4] Ethernet POWERLINK, available at http://www.ethernet-powerlink.org.

[5] R. Marau, L. Almeida, P. Pedreiras, Enhancing real-time communication over COTS Ethernet switches, in: 6th IEEE International Workshop on Factory Communication Systems, 2006.

[6] Z. Huang, Y. Zhang, H. Xiong, Modeling and simulation of switched Ethernet, in: 2nd International Conference on Computer Modeling and Simulation, 2010.

[7] H. Charara, C. Fraboul, Modeling and simulation of an avionics full duplex switched Ethernet, in: Advanced industrial conference on telecommunications/service assurance with partial and intermittent resources conference/e-learning on telecommunications workshop, 2005.

[8] J.-L. Scharbarg, C. Fraboul, Simulation for end-to-end delays distribution on a switched Ethernet, in: 12th IEEE International Conference on Emerging Technologies and Factory Automation, 2007.

[9] H. Charara, J.-L. Scharbarg, J. Ermont, C. Fraboul, Methods for bounding end-to-end delays on an AFDX network, in: 18th Euromicro Conference on Real-Time Systems, 2006.

[10] H. Bauer, J.-L. Scharbarg, C. Fraboul, Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network, in: The 14th IEEE Conference on Emerging Technologies Factory Automation, 2009.

[11] S. Dong, Z. Xingxing, D. Lina, H. Qiong, The design and implementation of the AFDX network simulation system, in: International Conference on Multimedia Technology, 2010.

[12] A. Basu, S. Bensalem, M. Bozga, B. Delahaye, A. Legay, E. Sifakis, Verification of an AFDX infrastructure using simulations and probabilities, in: 1st International Conference on Runtime Verification, 2010, pp. 330–344.

[13] J. Matsumura, Y. Matsubara, H. Takada, M. Oi, M. Toyoshima, A. Iwai, A simulation environment based on OMNeT++ for automotive CAN-Ethernet networks, in: 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems, 2013.

[14] D. Henriksson, A. Cervin, K.-E. Årzén, TrueTime: Real-time control system simulation with MATLAB/Simulink, in: Proceedings of the Nordic MATLAB Conference, Copenhagen, Denmark, 2003.

[15] The network simulator NS2, available at http://www.isi.edu/nsnam/ns/.

[16] OMNET++: Component-based C++ simulation library, available at http://www.omnetpp.org.

[17] OPNET: application and network performance, available at http://www.
opnet.com.

[18] Introduction to NS3, available at http://www.nsnam.org.

[19] R. Marau, M. Behnam, Z. Iqbal, P. Silva, L. Almeida, P. Portugal, Controlling multi-switch networks for prompt reconfiguration, in: 9th International Workshop on Factory Communication Systems, 2012.

[20] M. Ashjaei, M. Behnam, L. Almeida, T. Nolte, Performance analysis of master-slave multi-hop switched ethernet networks, in: 8th IEEE International Symposium on Industrial Embedded Systems, 2013.

[21] R. Marau, P. Pedreiras, L. Almeida, Asynchronous traffic signaling over master-slave switched Ethernet protocols, in: 6th International Workshop on Real Time Networks, 2007.

[22] M. Ashjaei, M. Behnam, G. Rodriguez-Navas, T. Nolte, Implementing a clock synchronization protocol on a multi-master switched Ethernet network, in: 18th IEEE International Conference on Emerging Technologies and Factory Automation, 2013.

[23] M. Ashjaei, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, T. Nolte, Reduced buffering solution for multi-hop HaRTES switched Ethernet networks, in: The 20th IEEE International Conference on embedded and Real-Time Computing Systems and Applications, 2014.