A Model-Based Approach to Automatic Generation of TSN Network Simulations

Maxime Samson
Thales Research & Technology
Palaiseau, France
LORIA – Université de Lorraine
Nancy, France
maxime.samson@thalesgroup.com

Thomas Vergnaud
Thales Research & Technology
Palaiseau, France
thomas.vergnaud@thalesgroup.com

Éric Dujardin
Thales Research & Technology
Palaiseau, France
eric.dujardin@thalesgroup.com

Laurent Ciarletta
LORIA – Université de Lorraine
Nancy, France
laurent.ciarletta@loria.fr

Ye-Qiong Song
LORIA – Université de Lorraine
Nancy, France
ye-qiong.song@loria.fr

***Abstract*—The IEEE 802.1 TSN working group published a set of standards which adds new functionalities to switched Ethernet networks. These new functionalities notably aim at making the design of deterministic Ethernet networks possible, enabling their use for real-time applications.**

This determinism comes however at the cost of a greatly increased complexity in configuration effort, especially for large-scale networks. In addition, this new complexity also affects the network simulation tools that are commonly used when designing such networks. As most of the existing ones only partially support new TSN functionalities, one has often to combine several simulators for achieving a reliable TSN network design.

In this paper, we propose a model-based approach which aims at assisting the design of TSN networks. Modeling allows the creation of a formal representation of the network which can then be used to automatically generate configurations. This approach has been successfully implemented as a TSN configuration software called MoBACT, that enables the use of multiple simulation/emulation tools during the design phase by generating configurations for different targets for easing cross-check of simulation results. Since each generated configuration is derived from the same representation of the network, our approach guarantees the consistency of the different configurations generated for each tool.

***Index Terms*—TSN, Configuration generation, Simulation, Model-Based software, MARTE specification.**

I. Introduction

Time Sensitive Networking (TSN) is a set of standards published by the IEEE 802 TSN working group. This set of standards fits into the IEEE 802.1 family of standards, mainly IEEE 802.1Q [1], adding numerous new functionalities, such as time synchronization, traffic shaping and resource reservation. These new functionalities, when used together, allow the design of deterministic Ethernet networks, which can offer real-time guarantees, like bounded latencies for example.

TSN allows the transmission of mixed-criticality traffic. Critical and non-critical data streams can share the network while maintaining the guarantee that non-critical traffic will not hinder critical traffic to make it miss its deadline.

The newly introduced functionalities, however, come at a cost: an increased complexity in design of communication networks in terms of their configuration, simulation and verification/validation.

The first reason for this increased complexity is the high amount of new functionalities and the diversity of different aspects of a network they affect (routing, scheduling, redundancy, security, etc).

A good illustration of this complexity can be shown by the traffic shaping mechanisms introduced in IEEE 802.1Qav and IEEE 802.1Qbv. These standards respectively define the Credit-Based Shaper (CBS) and the Time Aware Shaper (TAS). Both are scheduling mechanisms and they each require their own configuration on each egress port used by the data streams they manage. This already requires a considerable configuration effort because obtaining a valid schedule, in the case of the TAS, is a NP-complete problem [2].

An even higher level of complexity can be reached when using both of these traffic shaping mechanisms together, because the TAS has an impact on the CBS, making the task of configuring these functionalities even harder [3].

The second reason is the impact that TSN has on the tools used during the design phase of a TSN network. To the best of our knowledge, open-source tools support varying limited subsets of TSN functionalities. Some commercial tools seem to cover TSN much better, but their helpers or viewers differ in use and capabilities. This diversity can make the use of several tools a necessity. Additional reasons for using multiple tools are that access to a given tool may be limited; and the complexity of TSN making simulation fidelity a real challenge, being able to compare measurements on two or more simulators is at least reassuring through cross-check.

However, each of these tools has a different way of defining and configuring the network, which adds to the complexity of the configuration itself and is error prone when done manually.

The automatic configuration generation approach we present in this paper is based on models and mainly contributes to solving the two above-mentioned issues: 1) how to formally model TSN networks and automatically generate configurations to avoid error-prone manual design, 2) how the user can benefit from multiple simulation tools without going through the high learning curve of each.

Automatically generating the configuration eliminates the risk of human error when using a design tool, and makes it so knowing exactly how to configure it is no longer required, only knowing how to extract and interpret the results is. Not having to manually configure each tool represents a considerable time gain and has the added benefit of ensuring consistency across the different generated configuration.

In order to illustrate our approach, we implemented a configuration tool, named MoBACT (Model-Based Automatic Configuration for TSN). In its current state, MoBACT has the ability to generate configuration files for 3 different design tools: Mininet[1] [4], NeSTiNg[2] [5] and RTaW-Pegase[3]. Each of these tool has its own specificities and use cases.

In the remainder of this paper, we present in Section II some important TSN functionalities, in Section III the related works on which our contribution is based, and in Section IV the simulation/emulation tools we selected to illustrate our approach. In Section V, we describe our solution, how we implemented it in MoBACT, and some metrics which give an example of the time gain brought by our approach. Finally we give in Section VI conclusions and point out some futures works.

II. TSN

TSN standards define multiple traffic shaping mechanisms, some of which are based on time division, whose goal is to guarantee that data streams will respect their real-time requirements. Time synchronization, used to synchronize the clocks of the different network nodes, is thus required and is a cornerstone of TSN mechanisms. The synchronization mechanism is provided by the IEEE 802.1AS standard [6], which defines the Generalized Precision Time Protocol (gPTP).

To ensure that the network will be able to respect the real-time requirements of data streams, resources can be reserved at the egress ports along their paths. For example, the sum of the bandwidths required by the different streams going through a port must not exceed its capacity. The IEEE 802.1Qcc standard defines the Stream Reservation Protocol (SRP) which is able to make such reservation.

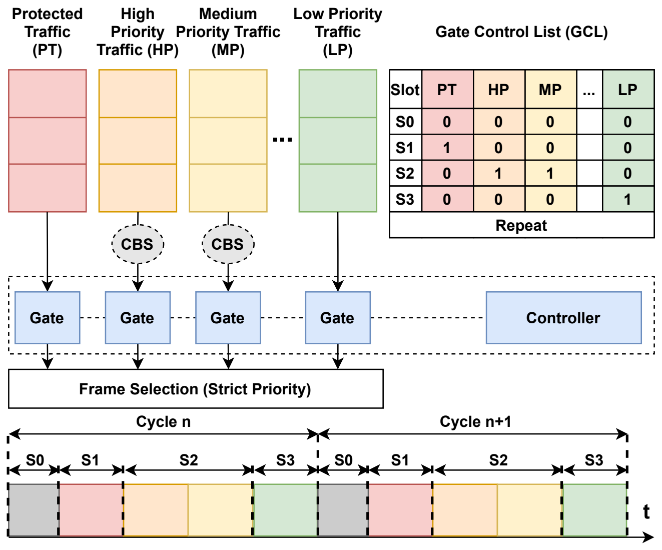
Having enough bandwidth does not guarantee the respect of real-time requirements; using this bandwidth in a way that makes it possible to bound the latency of critical streams, and that spares non-critical streams from suffering of unfair scheduling, is also essential: this is the role of traffic shaping mechanisms.

The Credit-Based Shaper (CBS) is one of these traffic shaping mechanisms. Its goal is to prevent large and bursty streams

[1]http://mininet.org

[2]https://gitlab.com/ipvs/nesting

[3]https://www.realtimeatwork.com/rtaw-pegase/



Slot	PT	HP	MP	...	LP
S0	0	0	0		0
S1	1	0	0		0
S2	0	1	1		0
S3	0	0	0		1
Repeat					

Fig. 1. Representation of the different scheduling mechanisms used on a TSN egress port

from momentarily saturating parts of the network, which could lead to increased delays or even packet loss if buffers are not large enough. The CBS also prevents high priority traffic from blocking lower priority traffic for indefinite amounts of time.

In order to achieve this, traffic classes are defined (Scheduled traffic, Audio, Video, Best effort, etc.) and, for some of these classes, a rate at which they gain credit can be assigned by the user, called the *idleSlope*. Frames belonging to a specific traffic class can only be transmitted if the amount of credit associated to this traffic class is non-negative. When frames are transmitted, the amount of credit associated to their traffic class decreases at a rate equal to the bandwidth of the port minus *idleSlope*. When frames are blocked by a negative credit or the transmission of frames belonging to another traffic class, the amount of credit associated to their traffic class increases at a rate equal to *idleSlope*.

When the traffic associated with a traffic class is large enough, the use of the CBS necessarily creates intervals of time during which frames cannot be transmitted and have to wait for the credit to reach a non-negative value again. This behaviour is undesirable for scheduling the most critical data streams because they require the lowest latency and jitter.

To match these requirements, the IEEE 802.1Qbv standard defines the Time Aware Shaper (TAS). This shaping mechanism defines a cycle, which is represented by a fixed amount of time, and splits this cycle into different time slots. During each time slot, only the specified traffic classes are allowed to transmit. This can be used to isolate the transmission of the most critical data streams by creating time slots that are exclusively allocated to a traffic class, guaranteeing that critical frames always have a unique time window during which they can be transmitted.

This time division is illustrated in Fig. 1, in the form of a Gate Control List (GCL). The GCL defines the time slots and which traffic classes are allowed to transmit during each of them. Multiple traffic classes can share the same time slot, in this case the final selection for which frame to transmit is managed based on the priority of the frames. This priority is defined by a Priority Code Point (PCP), ranging from 0, the lowest priority, to 7, the highest priority. The PCP is included in the VLAN tag of the Ethernet header.

Fig. 1 presents these mechanisms, used by the egress ports of bridges in a TSN network. Each egress port has its own scheduling policy so they all have to be configured individually in order to respect real-time requirements. This leads to a large amount of configuration parameters and creates high complexity in the configuration of TSN networks.

III. Related Works

In this section, we present research on the use of tools in TSN networks design. There exist multiple tools to assist the design of TSN networks by simulating them before beginning to use actual equipment.

NeSTiNg [5] is a TSN simulation framework which supports multiple TSN standards: 802.1Qav, 802.1Qbv and 802.1Qbu. CoRE4INET is another simulation framework which was originally developed for simulating other real-time Ethernet technologies such as TTEthernet [7]. It can now simulate TSN networks and supports the following TSN standards: 802.1Qav, 802.1Qbv, 802.1Qci. TSimNet [8] also is a simulation framework for TSN networks and supports the 802.1Qbu, 802.1Qci and 802.1CB standards, which are non time-based TSN functionalities such as TAS.

These three simulation frameworks all rely on OMNeT++ [4], completed by the INET [5] framework, which makes their association a widely used platform for the simulation of TSN networks.

Finally, other tools exist that do not rely on this platform, such as RTaW-Pegase, a commercial simulation and analysis tool for TSN networks. RTaW-Pegase supports more standards then the previous simulation frameworks and can perform worst-case analysis on end-to-end latency.

Network simulators are not the only kind of tools used to assist in the design of TSN networks. The works in [9] and [10] make use of UPPAAL [11], a model checker using timed automata.

In [10], UPPAAL is used for modeling and analyzing different scheduling mechanisms available in TSN standards, such as the Credit-Based Shaper. In [9], UPPAAL is used to define a formal model for other scheduling mechanisms such as the Time Aware Shaper. The impact of frame preemption alongside these scheduling mechanisms is also analyzed.

There are other works that do not intend to help the user verify the behaviour of an already defined network, as network simulators do, but that directly assist the configuration process. These tools include schedule synthesizers such as TSNSched [12]. Making use of a SMT solver, TSNSched is able to perform the synthesis of configuration for the Time Aware Shaper. The synthesized configurations will, if possible, respect the latency and jitter requirements specified by the user. In [2], other methods of computing Time Aware Shaper configurations are presented. Another approach is used in [13] which provides a genetic algorithm instead of SMT solvers. In this work, the genetic approach is used to solve both the scheduling and the routing problems. The computed configuration can then be used for the Time Aware Shaper.

In [14] finally, an automatic configuration generation tool is presented. This tool allows the user to automatically configure a simulated network for NeSTiNg. The tool is integrated into OMNeT++ as a plugin and can therefore only be used to generate files for the NeSTiNg simulation framework. In our approach, we also define an XML syntax which is used as the input format for MoBACT.

In summary, the state of the art solutions provide separately formal behavior verification [9], [10] and performance evaluation [5], [8], the approach that we developed combines the formal modeling and the performance simulation. Moreover, our approach also supports multiple simulators, allowing thus larger TSN functionality coverage leading to higher design reliability.

IV. Selected design tools

Our approach aims at assisting the design of TSN networks by making the use of design tools easier. There are multiple design tools available to design networks and some are specific to the design of TSN networks. Each of these tools have their own specificities and they do not necessarily support the same TSN functionalities.

To illustrate our approach, we selected 3 different tools: Mininet, NeSTiNg and RTaW-Pegase. These tools are very different in multiple ways: the TSN functionalities they support, their analysis capabilities and their ease of access (i.e. some are freely available and some are commercial products).

The first tool we selected is Mininet, a network emulator which allows the creation of a network made of virtual bridges, links and end-points. This kind of virtual network is useful for prototyping and testing networks on a single computer before deploying it, and to deploy the actual applications in the virtual end-points. Network topology can be generated either by our MoBACT tool, or by a Python API to specify it or even change it dynamically. Its emulated switches also support OpenFlow, which makes Mininet popular for Software-Defined Networking (SDN). The ability to run true applications makes it attractive for early integration tests and for measuring the real network payloads that they produce. Although the virtual bridges available in Mininet do not support TSN functionalities, it is still useful to generate a Mininet configuration file from a TSN network model, in order to test the application, especially if it includes dynamic network changes. We also foresee TSN switches to support SDN [15], and Mininet may then turn into a common platform for TSN engineering.

[4]https://omnetpp.org/

[5]https://inet.omnetpp.org/

TABLE I
DIFFERENCES IN THE SELECTED TOOLS

	TSN functionalities	Analysis capabilities	Ease of access
Mininet	ø	ø	++
NeSTiNg	+	+	++
RTaW-Pegase	++	++	- -

Lastly, Mininet is an open-source, freely available tool. It does not provide any analysis capabilities.

We also selected NeSTiNg, a simulation model for TSN networks which relies on OMNeT++ and its INET framework.

OMNeT++ is a discrete event based simulation platform, mainly used for network simulation. INET is a network simulation framework for OMNeT++, it contains the basic elements necessary to simulate wired, wireless and mobile networks. The NeSTiNg simulation model adds TSN functionalities to the bridges and end-points made available in INET and a way to configure them, making the simulation of TSN possible in OMNeT++.

NeSTiNg supports multiple important TSN functionalities: Time Aware Shaper (IEEE Std 802.1Qbv), Credit-Based Shaper (IEEE Std 802.1Qav) and frame preemption (IEEE Std 802.1Qbu and IEEE Std 802.3br). NeSTiNg is a freely available tool but, unlike Mininet, it also offers some analysis capabilities. Once the simulation is concluded and data has been collected, the tool allows the user to display graphs of different parameters, such as end-to-end delays of different data streams. There is also a way for the user to extract simulation data and produce their own analysis.

Finally, we selected RTaW-Pegase as our third supported tool. RTaW-Pegase is a network simulator which supports most TSN functionalities and offers a lot of analysis capabilities to the user.

After running a simulation in RTaW-Pegase, end-to-end delays can be displayed as well as Gantt charts presenting the flow of frames in the network and the time of their arrival in the different elements of the networks. These charts can also display the worst case (in terms of end-to-end delay) that was encountered during the simulation.

The tool also provides worst-case end-to-end delay computation. This analysis is separated from the simulation and is performed through network calculus techniques. The analysis provides the user with an upper-bound on the end-to-end delays of each data stream. This is an important feature because this upper-bound might not be encountered during simulations and it can be extremely important in critical systems.

RTaW-Pegase, however, is a commercial product and requires a license to be used, which makes it harder to access than the previous tools.

Table I gathers the differences of the 3 tools we selected.

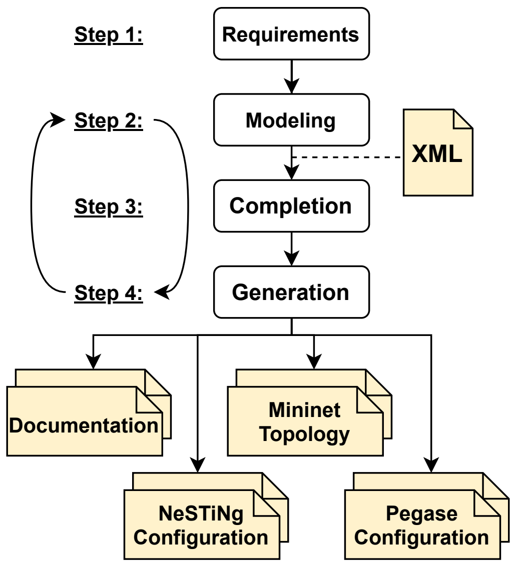


Fig. 2. Approach

V. SOLUTION

A. Approach

In order to assist the process of designing TSN networks and dealing with the complexity of configuring them, we propose a model-based approach containing multiple steps, presented in Fig. 2.

The first step of our approach is expressing the various requirements that the network must be able to satisfy. At this step, requirements do not have to follow a specific format and can affect different aspects of the network.

The most important requirements to express are the ones about the different data streams which will be transmitted on the network. The requirements must provide information that will be used to characterize the data streams. For example, in the case of a periodic data stream, the information must contain the source (talker) and the destinations (listeners) nodes, the size of the payload and the transmission period. Another requirement that can be specified is the deadline the data stream must never miss. Network topology can also be part of the requirements. Indeed, the network can be under space and weights constraints in embedded systems, such as in a car or a drone, which has a direct impact on network topology. The network equipment to use can also have an impact on network topology if the amount of bridges or the amount of ports per bridge is limited.

The second step of our approach is modeling. The goal of this step is to create a representation of each element of the network in accordance with the requirements expressed at the previous step and that contains the information required for automatic configuration generation. Once complete, this model will guarantee consistency across the different generated configurations.

The third step is completion. The goal of this step is to complete the model with information that are hard for humans to compute. This step produces the TAS configuration based on the information contained in the model. GCL are thus generated for every port to guarantee the respect of deadline requirements for each critical data streams.

Finally, once the model is complete, configuration can be automatically generated for the specified targets. To assist the design phase of a TSN network, this approach allows the user to easily iterate through steps 2 to 4 by evaluating the behaviour of the network then modifying the model, generating the new configuration and running the simulations again without having to re-write any of the configuration.

B. Implementation

1) Modeling: Our modeling approach is inspired by the UML profile for MARTE specification [16] and more specifically by its Generic Resource Modeling (GRM) chapter. We based our modeling approach on two concepts defined in this chapter: ComputingResource and CommunicationMedia.

We use the concept of ComputingResource to represent a node in the network. This concept is used to model TSN bridges and end-points. We use the concept of CommunicationMedia to represent elements of the network that connect or are connected to network nodes. This concept is used to model Ethernet links and data streams. We do not use the entirety of the MARTE specification but we selected the parts suited to our needs.

The MARTE specification is well known in the real-time embedded systems domain which makes it a good starting point for modeling deterministic Ethernet networks.

Our modeling approach uses the notion of *resource* for terminology: ComputationResource (compRsc) and CommunicationResource (commRsc).

Following the object oriented principle, we differentiate between definition and instantiation of network elements: definitions are named compRscDef and commRscDef, instantiations are named compRsc and commRsc. For example, the definition of a data stream is a CommRscDef that contains all the different parameters required for its characterization. The instantiation of a data stream is a commRsc that makes use of the definition as a reference and specifies the values of the different parameters. There can be multiple different instances of each definition.

Separating definition and instantiation allows our solution to better integrate with Sigil-UCM [17], a code generator developed in our laboratory, which has no use for the actual values but only for the definition of parameters.

In the remainder of this paper, we will use the same example. The network topology presented in fig. 3 is that of a real-time embedded system. Table II contains the list of data streams transmitted over this network.

This real-time embedded system is supposed to be used on a drone. The main path used by data streams goes through MainSwitch1 and MainSwitch2. BackupSwitch is used as a redundant path for critical systems in the drone and it is only used when the main path is no longer functional.

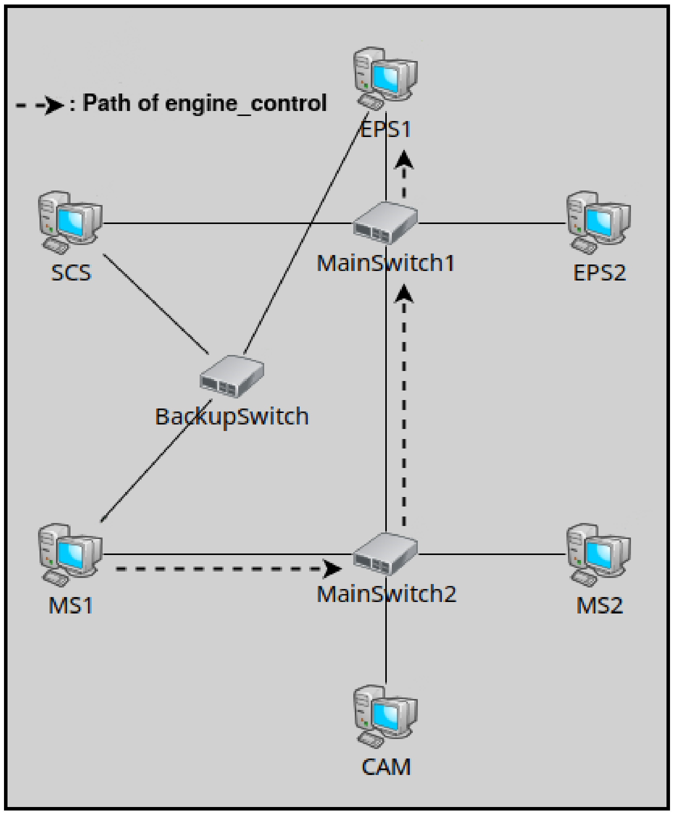


Fig. 3. Network topology of the real-time embedded system, the path of the engine_control data stream is shown with dotted arrows

TABLE II
LIST OF STREAMS

Name	TX	RX	Path	Critical
engine_control	MS1	EPS1	mainSwitch2_eth1 mainSwitch1_eth3	yes
telemetry	MS1	SCS	mainSwitch2_eth1 mainSwitch1_eth3	no
remote_command	SCS	MS1	mainSwitch1_eth0 mainSwitch2_eth0	yes
video1	CAM	MS1	mainSwitch2_eth3	no
video2	CAM	SCS	mainSwitch2_eth3 mainSwitch1_eth3	no
video3	MS1	SCS	mainSwitch2_eth1 mainSwitch1_eth3	no

We use an XML syntax, because of its flexibility, to create models which will then be used as input for MoBACT.

Listing 1 presents the definition of the engine_control data stream in our modeling approach.

Listing 2 presents the instantiation of the engine_control data stream making use of the definition presented in listing 1.

This data stream belongs to the traffic class of the higher priority and it must satisfy its deadline requirement of 2ms. It is a periodic data stream with a 100 µs transmission period and its payload contains 128 bytes. The talker and-point is MS1, its only listener is SCS and its path goes through mainSwitch2 then mainSwitch1.

One of our contributions is to propose the necessary definitions to represent the network elements. The user can then use these definitions to create a model of a network by instantiating all the elements.

```
<commRscDef name="Periodic_Stream">
  <configParam max="1" min="0" name="deadline"
               type="time_t"/>
  <configParam max="1" min="1" name="payload"
               type="payload_t"/>
  <configParam max="1" min="1" name="pcp"
               type="pcp_t"/>
  <configParam max="1" min="1" name="vlan_id"
               type="vlan_id_t"/>
  <configParam max="1" min="1" name="period"
               type="time_t"/>
  <rscParam max="1" min="1" name="talker">
    <allowedRscDef ref="Ethernet_Interface"/>
  </rscParam>
  <structParam max="-1" min="1"
               name="listeners">
    <rscParam max="1" min="1"
              name="listener_port">
      <allowedRscDef
              ref="Ethernet_Interface"/>
    </rscParam>
    <structParam max="-1" min="1"
                 name="paths">
      <rscParam max="-1" min="1"
                name="path_member">
        <allowedRscDef
                ref="Ethernet_Interface"/>
      </rscParam>
    </structParam>
  </structParam>
</commRscDef>
```

Listing 1. Definition the engine_control data stream

```
<commRsc def="Periodic_Stream"
         name="engine_control">
  <config def="deadline" value="{2, ms}"/>
  <config def="payload" value="{128, B}"/>
  <config def="pcp" value="7"/>
  <config def="period" value="{100, us}"/>
  <config def="vlan_id" value="1"/>
  <rscConfig def="talker"
             value="MS1_eth0"/>
  <structConfig def="listeners">
    <rscConfig def="listener_port"
               value="SCS_eth0"/>
    <structConfig def="paths">
      <rscConfig def="path_member"
                 value="mainSwitch2_eth0"/>
      <rscConfig def="path_member"
                 value="mainSwitch1_eth2"/>
    </structConfig>
  </structConfig>
</commRsc>
```

Listing 2. Instantiation of the engine_control data stream

2) Model completion: The model completion step is used to insert additional information into the network model that are either tedious to input for the user or hard for humans to compute.

This step can, for example, automatically generate MAC addresses for end-points and use default values for the bandwidth of network nodes. This greatly eases the design task.

The principal use of the completion step is the generation of parts of the configuration that are hard for humans to compute, such as the configuration of GCL for the Time Aware Shaper.

To automatically compute GCL configuration, we integrated an external tool, TSNSched, into our configuration generation tool. TSNSched is a tool that can generate GCL configuration from information contained in our network representation. It uses a set of constraints and a SMT solver, z3 [18], to compute a schedule, if it is possible, that respects the requirements of critical data streams in terms of both end-to-end delay and jitter.

TSNSched computes the start time and the duration of time slots used by critical streams while ensuring that enough time is left available before these time slots for a guard band to fit. Once the computation of TSNSched is done, the generated configuration is extracted and inserted into the network model.

Our work is then to generate the start time and duration of the remaining time slots: guard bands and time slots allocated to non critical traffic. Guard bands are time slots placed right before slots allocated to critical traffic. Their role is to guarantee that no other traffic can disturb the transmission of critical traffic. This is achieved by preventing any new transmission of frames for the duration of the transmission of the largest frame in the network. By using guard bands, even if a large frame begins its transmission right before the end of its time slot, it will not have any impact on critical traffic because the transmission will finish before the end of the guard band rather than during the time slot allocated to the critical traffic.

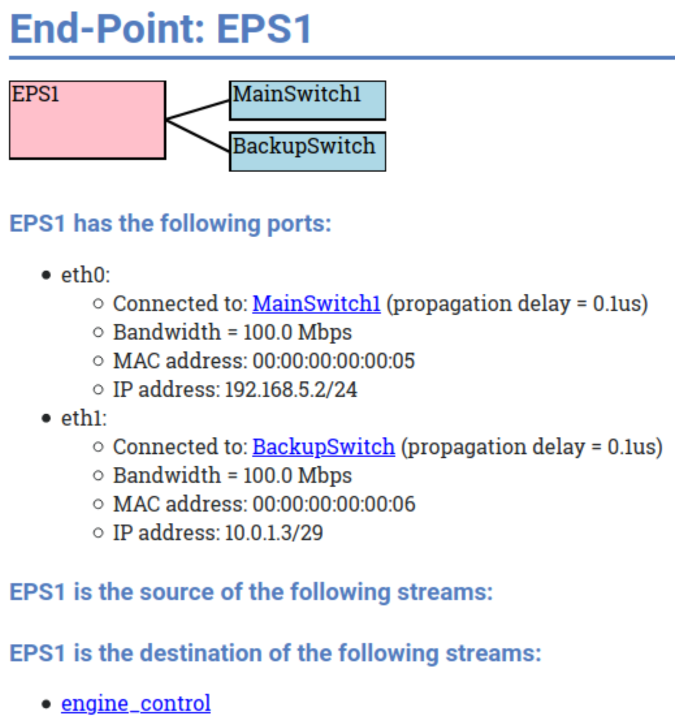
TSNSched also has the advantage of trying to optimise critical traffic transmission. It tries to align time slots allocated to critical traffic so that the expected arrival time of a critical frame in a switch occurs right before the beginning of its allocated transmission time slot.

3) Configuration generation: Our configuration generation tool takes the model created during the modeling step as input in an XML format. The user specifies the model to use, the targets to generate configuration for and whether or not the model should go through the completion step.

After extracting data from the model, MoBACT can generate files containing documentation about the network and configuration files. Targets for which configuration can be generated are Mininet, NeSTiNg, and RTaW-Pegase.

Documentation is generated as a set of HTML files. There is a file for each network element which contains all the information about this element and links to other elements it is connected to. This kind of documentation is useful to easily share information in a format which is easier to read than a model file and only requires a web browser. Fig. 4 gives one example of the node MS1.

For Mininet, the network topology can be specified through a Python API. From the information contained in the model, MoBACT can generate a file which uses this API. The file



End-Point: EPS1

EPS1 has the following ports:

- eth0:
  - Connected to: MainSwitch1 (propagation delay = 0.1us)
  - Bandwidth = 100.0 Mbps
  - MAC address: 00:00:00:00:00:05
  - IP address: 192.168.5.2/24
- eth1:
  - Connected to: BackupSwitch (propagation delay = 0.1us)
  - Bandwidth = 100.0 Mbps
  - MAC address: 00:00:00:00:00:06
  - IP address: 10.0.1.3/29

EPS1 is the source of the following streams:

EPS1 is the destination of the following streams:

- engine_control

Fig. 4. Example of HTML documentation

contains the instantiation of the entire network topology: end-points, switches and links.

For NeSTiNg, multiple files are necessary to describe the network topology, the network configuration and the simulation parameters. Network topology is described in a NED file. Network configuration is split across multiple XML files for routing, scheduling and the instantiation of data streams. Simulation parameters are specified in a INI file.

Listing 3 presents the initialization of the engine_control data stream in the INI file and listing 4 presents its instantiation in an XML file. These two listings use the data stream defined and instantiated in listings 1 and 2.

```
MS1.numApps = 1
MS1.app[0].typename = "UdpScheduledTrafficApp"
MS1.app[0].trafficGenerator.localPort = 1000
MS1.app[0].scheduleManager.
    ↪ initialAdminSchedule = xmldoc("xml/flows
    ↪ .xml", "/schedules/datagramSchedule[@id
    ↪ ='0']")

SCS.numApps = 1
SCS.app[0].typename = "UdpSink"
SCS.app[0].localPort = 1000
```

Listing 3. Initialization of the engine_control data stream in NeSTiNg

```
<datagramSchedule id="0" cycleTime="100us">
  <event payloadSize="128B"
         destAddress="SCS"
         destPort="1000" pcp="7" vid="1"/>
</datagramSchedule>
```

Listing 4. Instantiation of the engine_control data stream in NeSTiNg

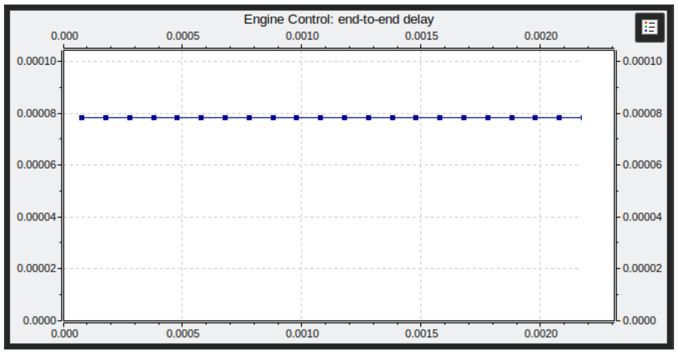


Fig. 5. End-to-end delay of the engine_control stream when using the Time Aware Shaper in NeSTiNg

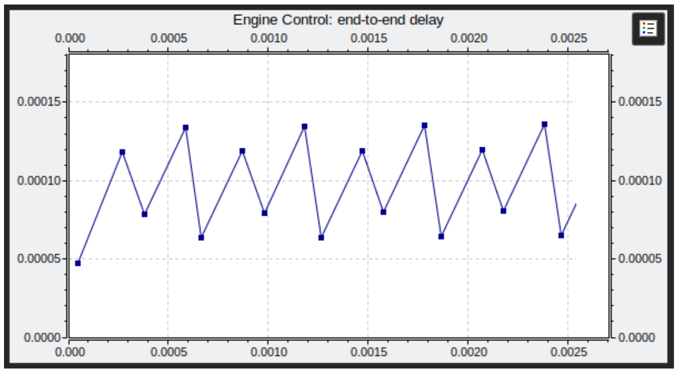


Fig. 6. End-to-end delay of the engine_control stream without using the Time Aware Shaper in NeSTiNg

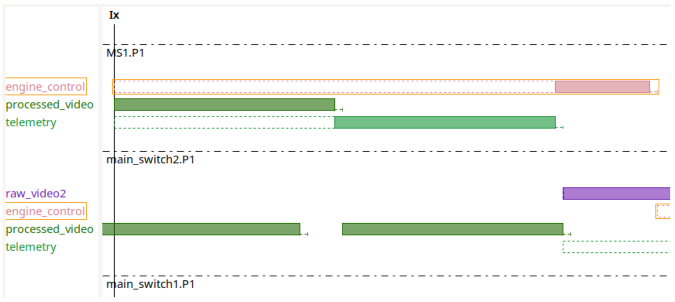


Fig. 7. Gantt charts for the engine_control data stream in RTaW-Pegase

After running the simulation, end-to-end delay graphs can be visualized in OMNeT++. Fig. 5 and 6 present the end-to-end delay of the engine_control data stream respectively with and without the use of the TAS. As expected, the use of the TAS reduces the latency and the jitter.

For RTaW-Pegase, everything required to run the simulation is contained in a single XML file. After running the simulation, Gantt charts can be visualized to analyze the worst case (in terms of latency) that was encountered, as shown in fig. 7. As expected, the worst case happens when frames belonging to both the data streams that share the same path as engine_control are transmitted before the engine_control frame in mainSwitch2.

VI. Metrics

The time our approach can save to the user can be evaluated in 3 different ways: the time it takes to develop a generator back-end for a new target tool, the time it takes to create a model and the execution time of MoBACT.

The development time to implement a generator back-end for a new target tool depends on the complexity of the format used by the tool. This can typically take between a few days and a few weeks. As the work is done once and for all, this saves a lot of efforts over time, because users of MoBACT are then not obliged to master the syntax of the new target tool.

The time it takes to create a new model of a TSN network depends on the size of the network. This typically takes between a few minutes and a few hours. Because we use this representation as a central model, this time only has to be spent once instead of once for every tool the user wants to use. It also saves time when modifying the representation of the network for the same reason. In addition, editing only the MoBACT model and then generating inputs for each tool ensures consistency between data.

The execution time of our configuration generation tool greatly depends on the amount of data streams and nodes in the network. In the example we used throughout this paper, the execution time of MoBACT for completing the model and generating configuration files is approximately 1 second. The machine we used runs openSUSE Leap 15.2 on a Core i7 6820HQ and has 8GB of RAM. For a larger network, this can take much longer due to the scheduling problem which is a NP-complete problem. The authors of [12] have run experiments on their tool for networks of different sizes.

VII. Conclusion

In this paper, we presented a model-based approach to the automatic generation of configurations for simulating TSN networks by using multiple simulation tools. Our modeling approach is inspired from the concepts of MARTE and allows the user to create a formal representation of the network. We developed MoBACT tool that can generate the configuration of the Time Aware Shaper and configuration files for multiple tools. These tools are Mininet (for only open virtual switched networks, not yet with TSN support), NeSTiNg and RTaW-Pegase.

This approach offers various advantages to the user. It saves time by using a single, central representation of the network and generating files for multiple simulation tools from it. Generating files for multiple simulators makes MoBACT unique in its category. Using this approach, the TSN designer no longer has to spend as much time learning how to use each of these different tools. It also prevents human error which the process of writing configuration is very prone to.

As future work, we aim at continuing to work on improving this approach. As shown in this paper, multiple generation targets can be supported by MoBACT, so adding new ones is a possibility. We also want to enrich our approach by linking it with another model-based approach for including the application description: model-based software engineering (MBSE). Having a model of the applications that will use the network would allow us to automatically create the representation of data streams together with the constraints in our models. This would enrich our approach and save additional time when modeling both real-time applications and their underlying networks.

We also plan on adding the generation of configuration in the YANG format [19], used to configure switches, which would make MoBACT able to generate configuration files for actual TSN equipment.

Because MoBACT helps save time in the design process while enforcing the consistency between inputs of TSN-related tools, it can be used to make the analysis of different combinations of TSN functionalities simpler.

References

[1] *IEEE 802.1Q 2018 – Bridges and Bridged Networks*, IEEE Std., 2018.

[2] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in ieee 802.1qbv time sensitive networks," in *24th Int. Conf. on Real-Time Networks and Systems (RTNS)*, 2016.

[3] D. Maxim and Y.-Q. Song, "Delay analysis of AVB traffic in time-sensitive networks (TSN)," in *25th Int. Conf. on Real-Time Networks and Systems (RTNS)*, 2017.

[4] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets)*, 2010.

[5] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Durr, S. Kehrer, and K. Rothermel, "NeSTiNg: Simulating IEEE time-sensitive networking (TSN) in OMNeT++," in *Int. Conf. on Networked Systems (NetSys)*, 2019.

[6] *IEEE 802.1AS 2020 - Timing and Synchronization for Time-Sensitive Applications*, IEEE Std., 2020.

[7] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "Ttethernet dataflow concept," in *IEEE 8th Int. Symposium on Network Computing and Applications*, 2009.

[8] P. Heise, F. Geyer, and R. Obermaisser, "TSimNet: An Industrial Time Sensitive Networking Simulation Framework Based on OMNeT++," in *IFIP 8th Int. Conf. on New Technologies, Mobility and Security (NTMS)*, 2016.

[9] W. Guo, Y. Huang, J. Shi, Z. Hou, and Y. Yang, "A formal method for evaluating the performance of tsn traffic shapers using uppaal," in *IEEE 46th Conf. on Local Computer Networks (LCN)*, 2021.

[10] J. Lv, Y. Zhao, X. Wu, Y. Li, and Q. Wang, "Formal analysis of tsn scheduler for real-time communications," *IEEE Transactions on Reliability*, 2021.

[11] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal – a tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, 1996.

[12] A. C. T. d. Santos, B. Schneider, and V. Nigam, "Tsnsched: Automated schedule generation for time sensitive networking," in *Formal Methods in Computer Aided Design (FMCAD)*, 2019.

[13] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *IEEE 23rd Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2018.

[14] B. Houtan, A. Bergström, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "An automated configuration framework for tsn networks," in *IEEE 22nd Int. Conf. on Industrial Technology (ICIT)*, 2021.

[15] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *24th Int. Conf. on Real-Time Networks and Systems (RTNS)*, 2016.

[16] *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, http://www.omg.org/spec/MARTE, OMG Std., 2019.

[17] Thales, "Sigil-UCM, an open source implementation of UCM," https://www.thalesforge.thalesgroup.com/projects/sigil-ucm/, 2020.

[18] L. M. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*, 2008.

[19] *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*, IETF Std., 2010.