Simulation and Validation Framework for Safety-Critical Applications in System-of-Systems

Ayman Murshed*,Mohammed Abuteir†,Roman Obermaisser*
*Chair for Embedded systems, Siegen University, Germany. † TTTech Computertechnik AG, Vienna, Austria,
Email: *murshedayman@gmail.com,†mohammed.abuteir@tttech.com, *roman.obermaisser@uni-siegen.de

***Abstract*—System-of-Systems (SoS) consist of the complex interconnection of large numbers of networked embedded systems that are characterized by operational and managerial independence of constituent systems, geographical separation, and emergent behavior in a constantly changing environment. The support for real-time communication is of crucial importance for many application areas such as medical, business, and military systems. The scheduling for such complex systems requires corresponding algorithms with additional requirements compared to classic monolithic systems. In this paper, two-level scheduling is assumed to deal with the SoS complexity where the first level is mapping application subsystems to constituent systems and the second level is mapping the application subsystem jobs in each constituent system to the local computational and communication resources to deal with the managerial independence of the SoS. This paper introduces a simulation framework for SoS that supports high-level scheduling as well as low-level scheduling for each constituent system. The proposed framework is built using the OPNET simulator with Time-Triggered Ethernet (TTEthernet) simulation models while the scheduling problem for the constituent systems is solved using IBM CPLEX.**

I. Introduction

The increasing demand of interconnecting independently developed embedded systems is driven by the growing importance for emerging services that necessitate the cooperation of different organizational systems. These heterogeneous systems are temporarily connected together to form a System-of-Systems (SoS) and to accomplish a certain goal [1]. Such SoS are found in healthcare services [2], military systems [3], [4], and industrial manufacturing.

These systems require a modular construction and component layout for the rapid, seamless, and inexpensive composition of services. Depending on the application domain, an SoS must also support extra-functional requirements such as real time, safety and security. Moreover, the increasing importance of SoS with real-time requirements demands techniques for scheduled end-to-end communication with resource reservations to ensure temporal guarantees.

Compared to monolithic systems, SoSs are based on a number of operationally and administratively independent, evolutionary developed, and geographically distributed constituent systems [5]. Hence, a two-level scheduling approach is used in scheduling SoS real-time applications; resource reservations within each constituent system and resource reservations at the SoS-level. Firstly, application subsystems must be mapped to constituent systems. Secondly, the detailed scheduling and allocation of the jobs within each application subsystem can be performed.

A naive approach relies on centrally computing new time-triggered schedules upon requests. However, the computation time needed to generate such a global schedule makes this approach unfeasible for fast-changing systems in addition to lack of central management and global knowledge of the SoS architecture. Thus, the scheduling needs to be incremental, distributed, and concurrent. The [6] has investigated an approach towards dynamic scheduling of real-time messages for the industrial Internet of Things (IoT), which provides the basis for the ad-hoc reconfiguration of the network, i.e. adding new or removing old bandwidth reservations from a pre-configured schedule. The [7] introduces a high-level approach for the automatic configuration of existing real-time Ethernet solutions. Ongoing EU projects such as Productive 4.0, CITADEL and AUTOWARE are focusing on fully dynamic systems based on the Ethernet.

Previous work [8] implemented a scheduling algorithm for independent constituent systems with real-time requirements. The algorithm supports the dynamic addition of applications based on incremental, distributed, and concurrent scheduling. The incremental scheduling allows the scheduler to reserve the resources from previous applications in order to avoid contention with the newly generated application schedules. The distributed cooperation among the different Constituent System Managers (CSMs) is a prerequisite for supporting the autonomy and lack of central control in SoS on one hand and simplifying the scheduling problem on the other hand by executing each application subsystem in its allocated constituent system which reduces the overall execution time and memory usage.

In this paper we introduce an architecture and a simulation framework based on Time-Triggered Ethernet (TTEthernet) communication to enable not only the dynamic composition of services but also provide the necessary preservation of the essential system characteristics such as guarantees for determinism, real-time behavior, fault containment, and availability.

The presented simulation environment integrates:

- Simulation models for SoS-level and CS-level communication using generic building blocks, namely end-systems and switches, realized with the OPNET tool.
- CSM building blocks of the constituent systems for scheduling SoS applications witch are implemented with IBM CPLEX in the simulation framework.

A generic SoS scenario generator was realized to build input traces for the evaluation of the proposed scheduling models. The traces are generated to trigger service establishment with real-time characteristics. The simulation environment provides tools to evaluate and validate the schedule results. It keeps track of packet collisions and drops during the simulation runtime, records end-to-end latency for time-triggered traffic, and analyzes the received messages at the application level.

The remainder of the paper is structured as follows. Section II introduces a conceptual model of the SoS with a description for incremental, distributed, and concurrent scheduling. After that, a brief description of the SoS simulation framework building blocks is given, namely TTEthernet switches and end-systems. Section IV presents a detailed description of the coordination protocol used to schedule SoS applications between the constituent systems. Section V presents an example scenario and simulation results using OPNET. The paper finishes with a conclusion in Section VI.

II. SoS Architecture with Incremental, Distributed, and Concurrent Scheduling

A. Concept of SoS Architecture

Figure 1 depicts the conceptual model of the SoS. The SoS comprises of constituent systems, where each constituent system is a distributed embedded system, which is under the control of a given organization. Each constituent system consists of end-systems that are interconnected by real-time networks of different protocols and topologies.

The interconnection of constituent systems occurs using a backbone communication infrastructures consisting of network domains. In analogy to the constituent systems, each network domain is within the responsibility of an organization that controls the resource allocations and their use by application subsystems. Technically, this control is realized by management services named Network Management System (NMS) of the network domains. The NMS identifies, configures and track the performance of the routers in the network domain, while also coordinating with other network domains and constituent systems.

Likewise, the management services for each constituent system are done by a CSM. The CSM performs the local configuration of the end-systems and networks within the constituent system. In addition, the CSM is responsible for the coordination with other constituent systems and network domains.

From the viewpoint of the logical model, an SoS consists of a number of applications where each application is comprised of a number of connected subsystems. The messages between jobs and subsystems represent the dependencies. Thus, there are two types of messages in the SoS, local messages (black arrows) and border messages (red arrows) as depicted in the logical viewpoint of Figure 1. Local messages are transmitted by the jobs of a subsystem while border messages are communicated between the subsystems of an application. We distinguish two types of traffic transmitted in the SoS, namely time-triggered and rate-constrained. Time-triggered messages are sent using a prioridetermined time slots with respect to a global time base. This type of messages are based on the Time Division Multiple Access (TDMA) medium access strategy to achieve a bounded delay and low jitter where messages are not queued in the switches. The global time concept is realized by synchronizing the local clocks of the communicating end-systems and switches as a basis for avoiding conflicts in resource allocations as well as global time-stamp assignment of messages [9]. The second type of message is rate-constrained, which is also used for real-time applications that send event-triggered messages. Rate-constrained messages are sent in time intervals that are not used by time-triggered messages. Therefore, rate-constrained messages are used by applications with less stringent timing requirements compared to time-triggered ones. For example, rate-constrained messages are used in the Avionics Full DupleX switched ethernet (AFDX) protocol with bounded latencies, but higher jitter compared to time-triggered messages [10].

B. Incremental, Distributed and Concurrent Scheduling

An SoS is characterized by its dynamic nature, where applications are introduced at runtime. Therefore, communication resources and computational resources of the platform have to be dynamically allocated to the application. More precisely, the following decisions need to be taken for a new application [8]:

- *SoS-level allocation:* Each application subsystem must be allocated to a constituent system.
- *SoS-level communication:* Messages between application subsystems must be mapped to paths between constituent systems along network domains.
- *Allocation within constituent systems:* Jobs must be allocated to end-systems within each constituent system.
- *Communication within constituent systems:* Messages between jobs of an application subsystem must be scheduled using paths between end-systems along routers.

Two levels of scheduling and allocation can be distinguished in SoSs. Firstly, application subsystems must be mapped to constituent systems. Secondly, the detailed scheduling and allocation of the jobs within each application subsystem can be performed.

a) High-Level Allocation of an Application: The first step of the allocation is the mapping of application subsystems AS to constituent systems C:

$$\mathrm{ALLOC}_{\mathrm{AS}} : AS \mapsto C \tag{1}$$

Thereafter, each edge $< \alpha, \beta >$ (i.e., message) of the high-level application graph G_{HA} must be allocated to a path p in the high-level physical graph. Such a path in the high-level physical graph consists of a sequence of network domains from the constituent system of the sender α to the constituent system of the receiver β.

b) Low-Level Allocation and Scheduling in Constituent Systems and Network Domains: For each application subsystem that is allocated to a constituent system C, the jobs $\bar{J}$ need to be allocated to the end-systems $\overline{ES}$ of C:

$$\mathrm{ALLOC}_{\mathrm{job,c}} : \bar{J} \mapsto \overline{ES} \tag{2}$$

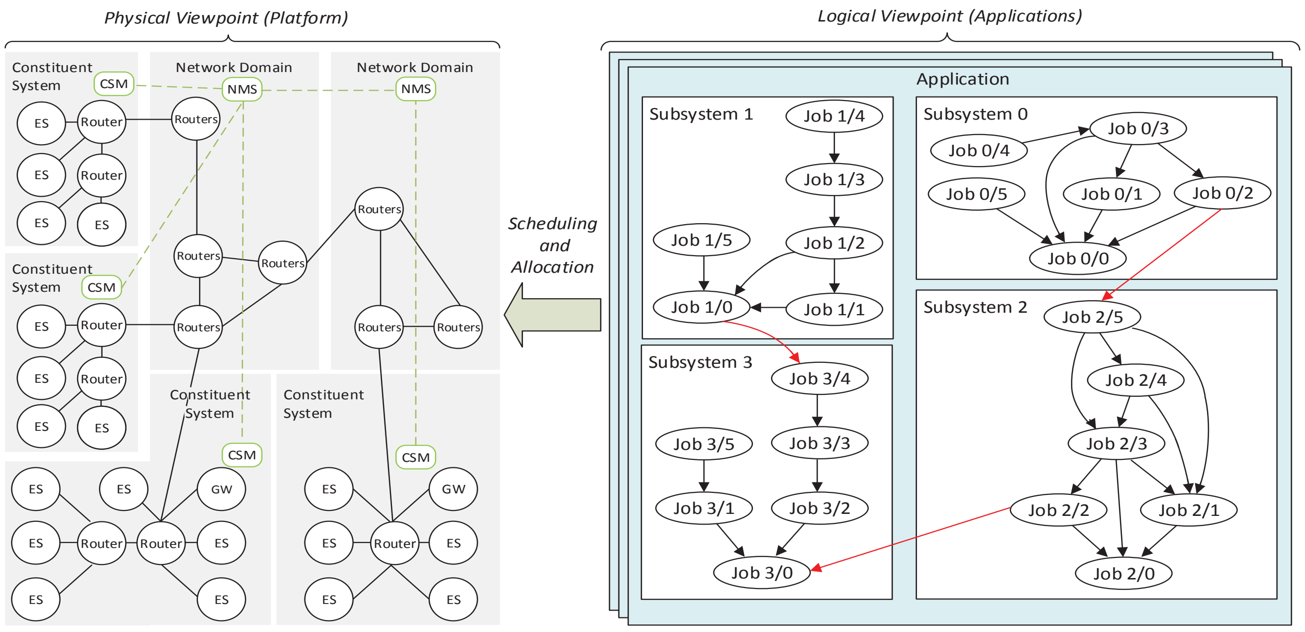


Fig. 1. Physical and logical Viewpoint of the SoS

Likewise, for each application subsystem that is allocated to a constituent system C the respective messages M must be mapped to paths and schedules:

$$\begin{aligned}\text{SCHEDULE}_{m,c} : M \mapsto p,\\ p = (p_1, p_2, \ldots, p_n)\\ p_1 = \text{ALLOC}_{\text{job,c}}(\alpha)\\ p_n = \text{ALLOC}_{\text{job,c}}(\beta)\end{aligned}$$

A message is an edge $<\alpha, \beta>$ in the Directed Acyclic Graph (DAG) of the logical viewpoint. The respective jobs α and β must belong to the same application subsystem AS that is allocated to a constituent system C. The links along the path $p_1, p_2, \ldots, p_n$ must be connected according to the graph G_P of the physical viewpoint.

III. OPNET-based Simulation Environment

The simulation environment realizes the SoS architecture from Section II. The simulation model of each constituent system implements a CSM with a scheduler process for local scheduling and communication with other CSMs. Each CSM produces schedule for the configuration of the nodes of its constituent system to be simulated. The OPNET simulation takes the application schedules from the incremental scheduler which is realized using IBM CPLEX [8]. The scheduler formulates a Mixed Integer Linear Programming (MILP) problem which consists of a number of constants, decision variables, constraints and an objective function.

Each CSM realizes a coordination protocol for the interaction with the CSMs in other constituent systems. The high-level allocation of an application is performed by the CSM in the initiating constituent system where the establishment of an SoS application is being requested. Afterwards, the CSM in the initiating constituent system contacts the CSMs at the constituent systems of the other application subsystems and triggers local scheduling at the respective CSMs.

Therefore, the CSM serves two purposes:

- high-level scheduling for initiation of SoS applications.
- local scheduling for service requests from other constituent systems.

The introduced communication methods and the dynamic scheduling algorithm of the SoS architecture model have been realized using TTEthernet model building blocks (e.g., TTEthernet switches and TTEthernet End-systems (ESs)) [11]. This simulation environment uses the OPNET tool suite for discrete event simulations of TTEthernet communication networks [12]. Simulation models in OPNET are organized hierarchically consisting of four main levels: the SoS network, constituent systems, node models and process models.

The top level refers to the SoS network which contains a number of constituent system models and a network domain, that connects these constituent systems with each other, using building blocks from the standard library and user-defined components. TTEthernet is used for both SoS communication as well as for the communication inside the constituent systems. At this level, statistics about the network are collected, the simulation is executed, and results are viewed. The second level is the constituent system that is implemented using the time-triggered nodes (i.e. ESs and switches) in addition to the CSM node. The node models are at the third level in the hierarchy and have a modular structure. The node is defined by connecting various modules with packet streams. The connections between modules allow packets and status information to be exchanged between modules. The modules in the nodes are implemented by using process models, the lowest level in the hierarchy.

The main simulation building blocks are generic infrastructure elements of a TTEthernet-system, which can be configured and extended to create an application-specific simulation model:

- Generic model of a TTEthernet switch. TTEthernet switches are central building blocks of a TTEthernet-based system. A generic simulation model of a TTEthernet switch supporting time-triggered, rate-constraint, and best effort communication is developed. In order to construct the overall simulation model, the user can perform multiple instantiations of the generic switch, establish connections to ESs and other switches, and assign to each switch instantiation a corresponding configuration.
- Generic model of a TTEthernet ES. TTEthernet ES are the communication end points within the TTEthernet system. The details description of the simulation modules is explained in [11]. In this work, we extend the ES for implementing a new model called SoS scheduler responsible for applying the coordination protocol between the constituent systems and network domains as explained in section IV.

IV. Coordination Protocol between Constituent Systems and Network Domains

An SoS application can be scheduled where the CSMs in the constituent systems communicate with each other while generating their distributed schedules and then deploying the resulting schedule. This section presents the rules and mechanisms that are used between the constituent systems of the SoS. It defines a list of steps required whenever a new application is introduced. Moreover, it includes the control and the operational interfaces between the building blocks (end-systems, switches, and CSMs).

1) Message-based Interfaces between Building Blocks: Figure 2 illustrates the integration of an SoS scheduler layer into the block diagram of a TTEthernet end-system [11]. Besides the generic source that generates all traffic types (i.e., time-triggered, rate-constrained and best-effort), a scheduler is added to accept new application requests, communicate with other CSMs to generate the incremental schedules, and configures the nodes inside its constituent system with the generated schedule. The configuration messages sent and received by the scheduler layer are of best-effort type.

The job of the scheduler layer is summarized in Algorithm 1. The interested node sends a new application request to the CSM, which performs a high-level allocation to the request of this application. After that, the CSM of the initiating constituent system (henceforth called root CSM) sends this information to all CSMs related to the participating application subsystems which are allocated to constituent systems and waits for their acknowledgement messages. Moreover, the root CSM starts with the low-level allocation and scheduling of the jobs and messages within its application subsystem. After these CSMs received the application dependency information and send back the acknowledgement to the root CSM, the

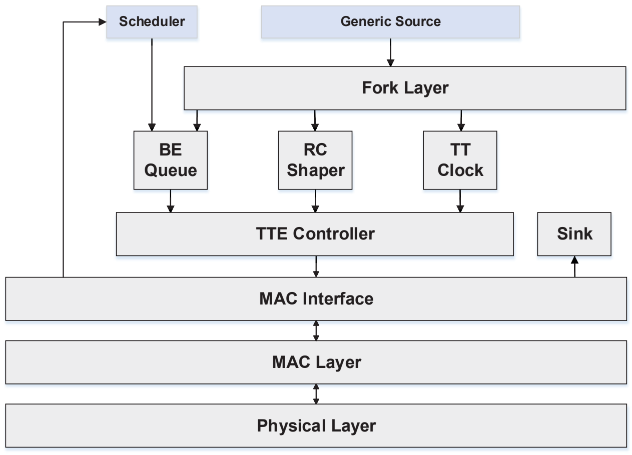


Fig. 2. Block diagram of CSM in TTEthernet End system

latter starts sending the global messages to those CSMs. A CSM will perform low-level allocation and scheduling of its jobs and messages if and only if the times of all dependent global messages have arrived. When all CSMs received acknowledgments about the completion of schedule generations from all participating CSMs, they will configure the interested end-systems and switches within the constituent system.

2) Sequence of Activities to Introduce a new Application: The interactions between the CSMs in the SoS, as well as, interactions between the requested application in a constituent system with its CSM are classified into four main operations. The first operation consists of finding out the high-level application subsystems of the application requests. The next operation is to perform a mapping of the high-level application subsystems to high-level physical models. This is followed by the scheduling of the low-level logical model in each mapped constituent system. Finally, the generated schedules are deployed in the selected physical models that necessitate the configuration of the end-systems in each constituent system by its CSM. The aforementioned operations can be explained in details using twelve phases as depicted in figure 3.

Phase 0: A user, connected from an end-system, logs into the system and requests a service from the SoS. For example, in case of a tele-health monitoring and patient service where a number of sensors are used to keep track of the status of a patient and send emergency requests in case of abnormal readings. The request is delivered to the operating system of the end-system.

Phase 1: An application in an end-system sends a request for a particular service to its CSM. It is assumed that any request made by an application includes information about the specific application subsystems, i.e., high-level logical models, which serve the intended request.

Phase 2: In this phase the CSM then consults the broker in the constituent system in order to procure the suitable constituent systems, i.e., high-level physical models, and perform a high-level allocation for the specific application subsystems

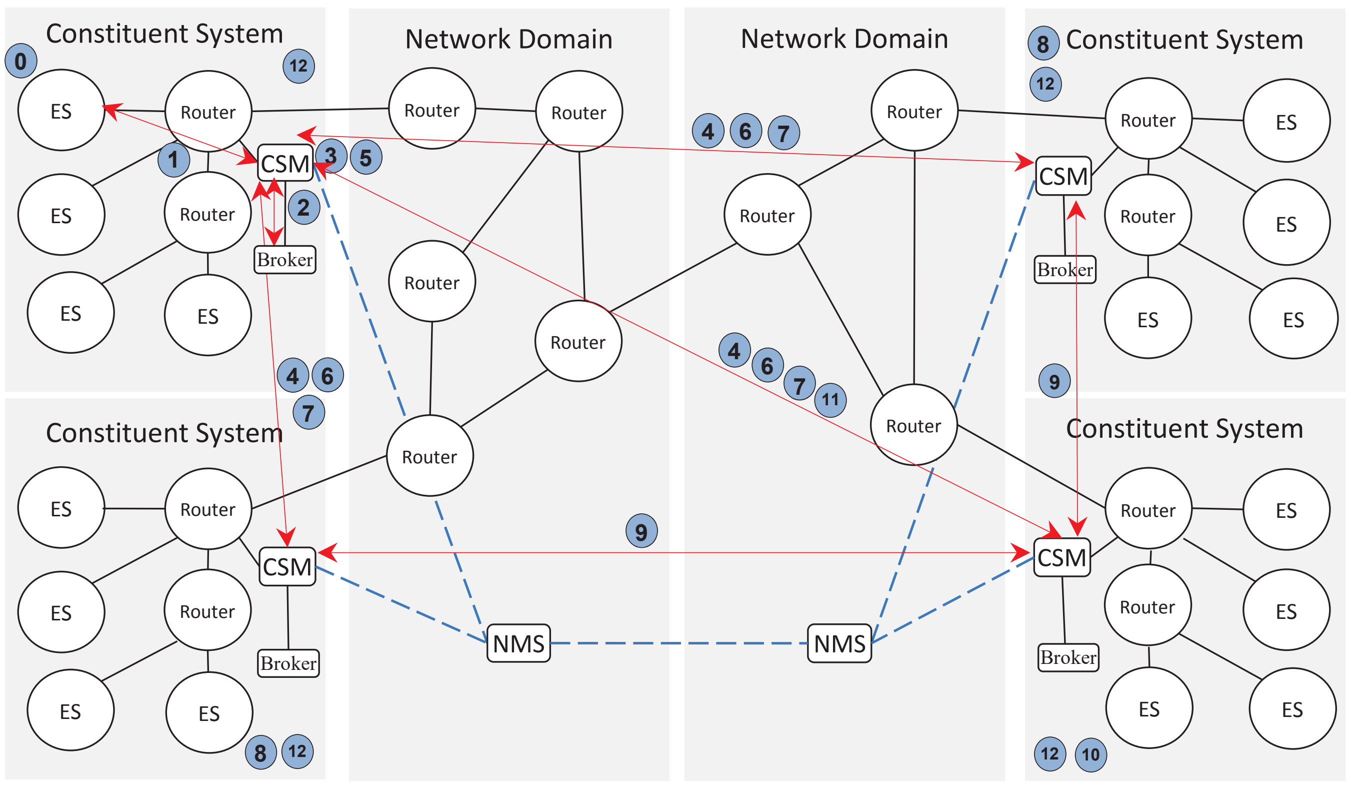


Fig. 3. Different phases of requests managements in SoS

defined in the request.

Phase 3: The CSM performs preliminary calculations to obtain the predecessor- and successor constituent systems for each of the involved constituent systems, i.e., high-level application dependencies.

Phase 4: In phase 4 the root CSM sends messages to all CSMs of the involved constituent systems about the high-level-application dependencies. Thus, every CSM is aware when to start its low-level scheduling.

Phase 5: The CSM performs a low-level scheduling algorithm. This includes the allocation of local jobs to end-systems as well as computing the optimal paths for all local and global messages of the first application subsystem.

Phase 6: When all CSMs receive a high-level-application dependency information message, they send back to the root CSM an acknowledgement message. This message confirms that all CSMs of successor constituent systems are ready to receive messages about finish time of high-level dependent messages.

Phase 7: In phase 6 the finish times of the global messages are sent to the CSMs of successor constituent systems. This information is used as an input, in addition to its low-level logical model, for the received CSMs in order to start their low-level scheduling algorithm.

Phase 8: When the CSMs of the successor constituent systems receive the predecessor message, they start computing their low-level schedule.

Phase 9: After the CSMs of the successor constituent systems generate the schedule, they hand in the results of their global messages to their successors' constituent system.

Phase 10: When the CSMs of the last successors receive the predecessor message, they compute their low-level schedule.

Phase 11: When the CSMs of the last successor constituent systems have finished with the generation of their low-level schedules, they send their global messages to the first CSM. This message indicates the end of the incremental scheduling for all involved constituent systems of the request.

Phase 12: The last phase is concerned with the deployment procedure of the request with the actual reconfiguration of the interested constituent systems based on the generated schedules.

Figure 4 illustrates the timing diagram of the application request management messages that are sent to schedule the application subsystems and to reconfigure the related constituent systems' nodes, namely end-systems and switches. The deployment of a new application schedule to the interested nodes is executed in the next global period in order to ensure its correct execution and to avoid negative interference with the running applications.

Figure 5 depicts the description of the messages transferred when a request is made by an end-system.

The *Service Request* message is an Ethernet message initiated by the requester end-system. It consists of the ID of the request and the high-level logical models that will serve the

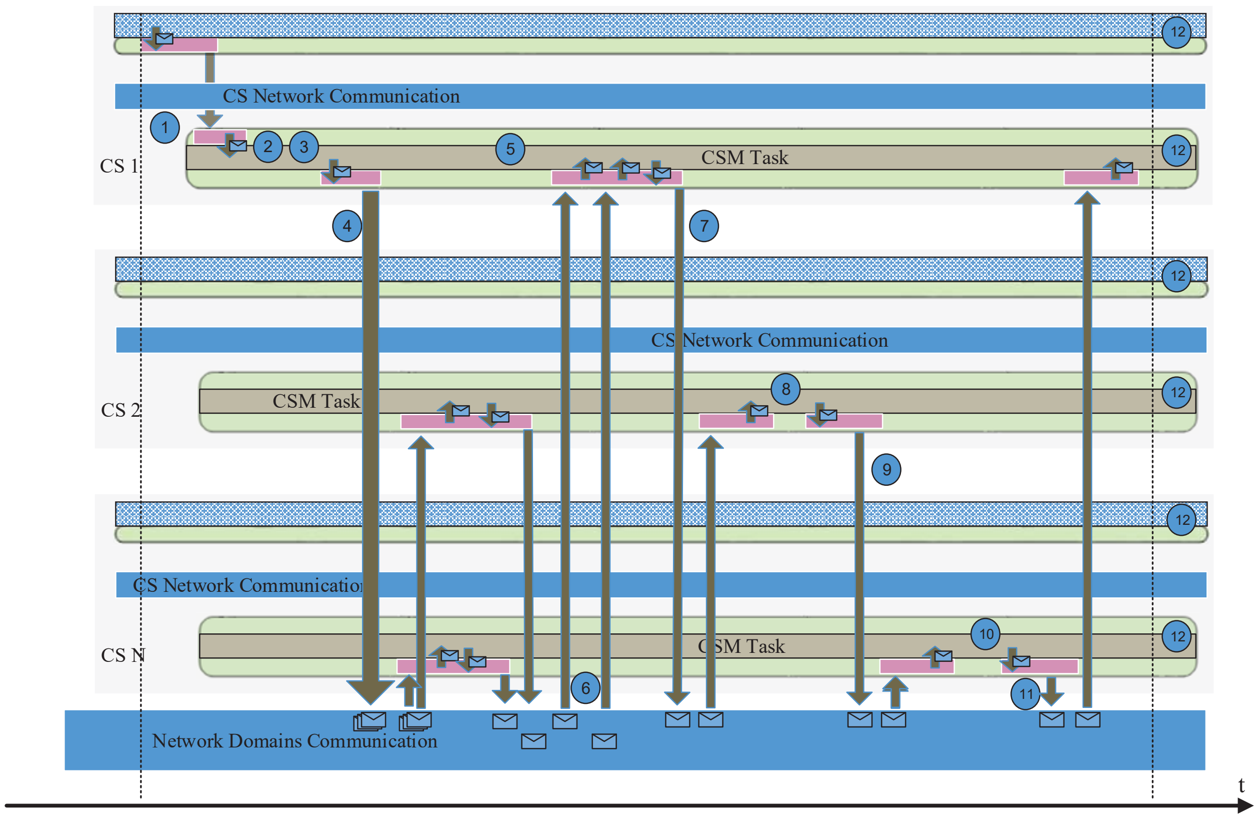


Fig. 4. Message sequence diagram

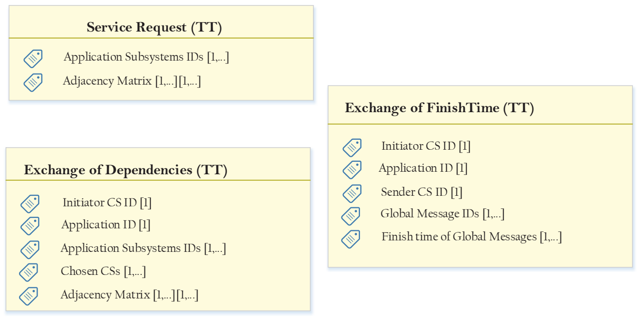
intended request. The logical models contains a vector of the required application subsystems and the adjacency matrix for the dependencies between the application subsystems.

The *Exchange of Dependencies* message is an Ethernet message that is sent by the base CSM to the CSMs of the selected constituent systems. It consists of the ID of the request and a unique application number which is a combination of the ID of the requester end-system and a sequence number generated by the CSM. Moreover, it contains the logical models and the selected constituent systems that will allocate the application subsystems.

The *Exchange of FinishTime* message is an Ethernet message that is sent by the CSMs of the predecessor constituent systems. In addition to the request ID and the application ID, this message consists of the sender constituent system's ID and the finish time of the global messages that are sent by this constituent system.

V. Experiments and Results

The scenarios generated for the analysis of this work are based on the Stanford Network Analysis Platform (SNAP) library which is widely used in numerous academic researches [13]. The SNAP is extended to enable the creation of physical and logical graphs in terms of undirected and directed graphs, respectively. Based on input parameters, the



Service Request (TT)
Application Subsystems IDs [1,...]
Adjacency Matrix [1,...][1,...]
Exchange of FinishTime (TT)
Initiator CS ID [1]
Application ID [1]
Sender CS ID [1]
Global Message IDs [1,...]
Finish time of Global Messages [1,...]
Exchange of Dependencies (TT)
Initiator CS ID [1]
Application ID [1]
Application Subsystems IDs [1,...]
Chosen CSs [1,...]
Adjacency Matrix [1,...][1,...]

Fig. 5. Description of CSM messages

generator creates random platforms and applications according to the SoS model. The input parameters for the physical viewpoint include the desired number of constituent systems and network domains, the average number of end systems and routers per constituent system, and the average node degree of the switches. In the logical viewpoint, input parameters are the desired number of applications, the average number of subsystems per application, the number of jobs per application subsystem and the average node degree of jobs.

Figure 6 depicts the physical model to be used in the

```
switch Type do
    case Request to Root CSM do
        Add new APP
        Send APP Dependency to all participating CSMs
        // perform low-level allocation and scheduling to $AS$ allocated to the root $CS$
        incremental update of $ALLOC_{\text{job,c}}$ for jobs in $ApplicationSubsystem$
        incremental update of $\text{SCHED}_{\text{m,c}}$ for msgs. in $ApplicationSubsystem$
        break
    case Exchange of APP Dependency do
        Send APP Dependency ACK to Root CS
        break
    case APP Dependency ACK do
        Send Times of Dependent Msgs. to requester CSM
        break
    case Exchange Global Messages do
        if ALL Dependent Msgs Received then
            // perform low-level allocation and scheduling to $ApplicationSubsystem$ allocated to this $ConstituentSystem$
            incremental update of $ALLOC_{\text{job,c}}$ for jobs in $ApplicationSystem$
            incremental update of $\text{SCHED}_{\text{m,c}}$ for msgs. in $ApplicationSubsystem$
        end
        break
    case Schedule Generated ACK do
        if ALL other Schedules of Application Subsystems Generated then
            Configure the end-systems and switches inside $ConstituentSystem$
        end
        break
    otherwise do
        Do Nothing
    end
end
```

Algorithm 1: Interaction of the scheduler layer to configuration messages

evaluation of time-triggered SoS applications. The physical model scenario of the SoS consists of 7 constituent systems where each one is a distributed network consisting of a number of switches connected to a number of end-systems. Each constituent system has a CSM to manage and coordinate between other CSMs in order to schedule SoS applications. Table I lists the size of each constituent system and their assigned CSMs. The clock synchronization algorithm of the TTEthernet was not simulated in the evaluation. The schedules which are generated using CPLEX are the optimal solutions for the MILP problem. Table II describes the logical model attributes of the scenario which contain timing information of the requested applications. Each application has 4 application subsystems, where each one consists of a number of jobs that send communication messages.

A high-level scheduler performs a random allocation of application subsystems to constituent systems. The paths between application subsystems are determined by computing the shortest paths. The output of the high-level scheduler are CPLEX scheduling models with the constants, constraints and decision variables. The CSMs in all allocated constituent systems coordinate together to schedule the requested application as explained in section IV.

Table III contains information of the high-level and low-level scheduling. The application subsystems are mapped to constituent systems based on the shortest path. After that, a low level scheduling is performed to each application in a distributed manner. This is obviously seen in the applications

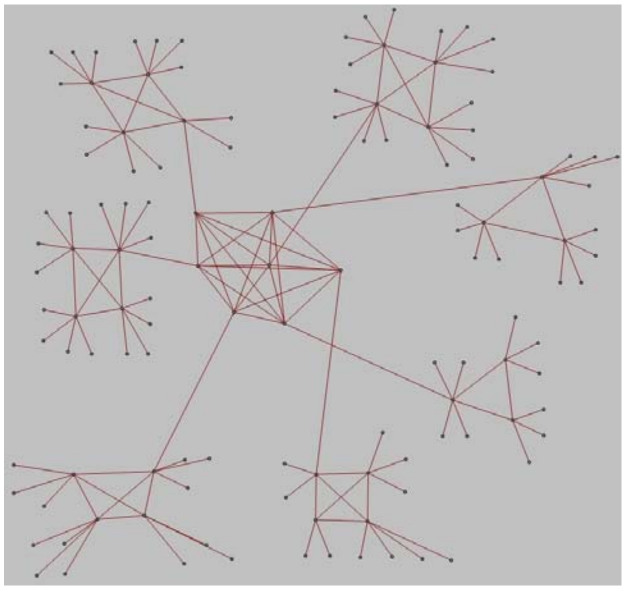


Fig. 6. Physical scenario use-case with 7 CSs

CS	End-systems	Swiches	CSM ID
1	12	3	117
2	16	4	77
3	14	4	12
4	16	4	32
5	13	4	144
6	10	3	153
7	11	4	175

TABLE I
PHYSICAL PARAMETERS IN SOS SCENARIO

Application	AS No.	Jobs	Messages	Requester CS	Time
1	4	30	36	1	10.3
2	4	30	36	4	15
3	4	30	36	2	15

TABLE II
APPLICATION REQUESTS IN SOS SCENARIO

App.	High-Level Allocation		Request Time	Finish Time
	AS	CS		
1	0	1	10.3	11.202
	1	3		11.96
	2	2		12.072
	3	6		12.2825
2	0	4	15	15.425
	1	2		16.643
	2	3		16.992
	3	5		16.339
3	0	2	15	17.6365
	1	5		18.8995
	2	7		19.6233
	3	1		18.6985

TABLE III
RESULTS OF SOS USE-CASE

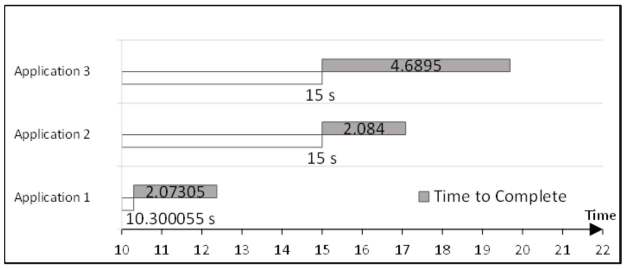


Fig. 7. Real-time application invocations

2 and 3, where the scheduling of their application subsystems are performed in parallel. Moreover, incremental scheduling is performed in constituent systems 1,2, and 5 where more than one application subsystems are mapped and scheduled in each constituent system using previous reserved resources information. Figure 7 depicts the timing diagram of the three real-time applications. Applications 2 and 3 are scheduled at the same time of their request. However, the scheduling of application 3 takes more time because of the incremental scheduling, where more reserved resources constraints are added to the scheduling problem.

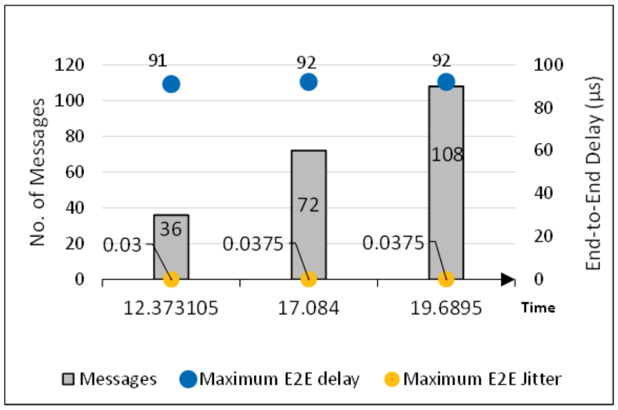


Fig. 8. Detailed timing diagram of generated applications

Figure 8 illustrates a more detailed timing information regarding the invocations for these three SoS applications with the total number of time-triggered messages that are being transmitted throughout the different constituent systems in the SoS. Each time an application request is mapped and deployed to the high-level physical model, the total number of transmitted messages is increased. Moreover, information regarding the maximum end-to-end latency as well as the maximum jitter for the messages being transmitted are recorded.

VI. CONCLUSION

The simulation framework for safety critical applications in an SoS is an excellent tool to evaluate and validate schedule results of SoS applications. The SoS simulation framework provides feedback on the temporal behavior of the SoS nodes, namely TTEthernet end-systems and switches. It keeps track of packet collisions and drops during the simulation run-time, records end-to-end latency and jitter for time-triggered as well as rate-constrained messages, and analyzes the received messages in the application level.

ACKNOWLEDGMENT

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 730830.

REFERENCES

[1] M. Jamshidi, *Systems of Systems Engineering - Principles and Applications*. CRC Press, 2009.
[2] E. Grigoroudis, V. S. Kouikoglou, and Y. A. Phillis, “A system-of-systems approach for improving healthcare systems,” in *World Automation Congress (WAC), 2012*. IEEE, 2012, pp. 1–6.
[3] P. Boily and N. Harrison, “A simulation system of systems to assess military aircraft protection,” in *Systems Conference (SysCon), 2012 IEEE International*. IEEE, 2012, pp. 1–6.
[4] K. M. Welch and C. R. Lawton, “Applying system of systems and systems engineering to the military modernization challenge,” in *System of Systems Engineering (SoSE), 2011 6th International Conference on*. IEEE, 2011, pp. 245–250.
[5] M. Maier, *Architecting Principles for Systems-of-Systems – Systems Engineering*, 1998.
[6] W. Steiner, F. Bonomi, and H. Kopetz, “Towards synchronous deterministic channels for the internet of things,” in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, March 2014, pp. 433–436.
[7] L. DÃijrkop, J. Jasperneite, and A. Fay, “An analysis of real-time ethernets with regard to their automatic configuration,” in *2015 IEEE World Conference on Factory Communication Systems (WFCS)*, May 2015, pp. 1–8.
[8] R. Obermaisser and A. Murshed, “Incremental, distributed, and concurrent scheduling in systems-of-systems with real-time requirements,” in *Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1918–1927.
[9] H. Kopetz, “Why a global time is needed in a dependable sos,” *arXiv preprint arXiv:1404.6772*, 2014.
[10] A. D. Network, “Part 7: Avionics full duplex switched ethernet (afdx) network,” *ARINC Report 664P7-1*, 2009.
[11] M. Abuteir and R. Obermaisser, “Simulation environment for time-triggered ethernet,” in *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*. IEEE, 2013, pp. 642–648.
[12] *OPNET Modeler 17.1 Documentation*, OPNET Technologies.
[13] J. Leskovec, “ Stanford Network Analysis Package(SNAP).” http://snap.stanford.edu/, [Online; accessed 26-February-2015].