Incremental, Distributed and Concurrent Scheduling in Systems-of-Systems with Real-Time Requirements

R. Obermaisser, A. Murshed
University of Siegen

***Abstract*—Systems-of-Systems (SoS) are large-scale networked embedded systems that are characterized by operational and managerial independence of constituent systems, geographical distribution, emergent behavior and evolutionary development processes. This paper introduces a conceptual model and a scheduling algorithm for supporting real-time requirements in SoS. Real-time support is essential in many safety-relevant application areas such medical, military and industrial SoS. The search for a feasible schedule is computed incrementally upon the introduction of new applications in the SoS. The distributed computation of the schedule using the different constituent systems considers the lack of global knowledge and control in the SoS, while also reducing the overall scheduling time. Concurrent scheduling activities are supported to deal with the uncoordinated and possibly simultaneous introduction of multiple applications.**

The paper introduces a high-level scheduling algorithm for the SoS as well as a low-level scheduling problem for individual constituent systems. The incremental scheduling problem for the constituent systems is formulated using IBM CPLEX. An experimental evaluation with automatically generated examples demonstrates the feasibility of the proposed solution.

I. Introduction

The field of embedded systems is faced with the trend of an increasing interconnection of independently developed embedded systems to each other and to the cloud. The resulting Systems-of-Systems (SoSs) are networked together for a period of time to achieve a certain higher goal [1]. Examples of SoSs include smart cities [2], intelligent factories [3] and integrated healthcare systems [4].

SoSs are characterized by operational and managerial independence of constituent systems, geographical distribution, emergent behavior and evolutionary development processes [5]. In addition, many SoSs depend on support for stringent real-time requirements for time-critical application services. Examples are medical monitoring and telemedicine in healthcare systems. Likewise, real-time requirements are imposed by closed-loop control and remote interactions with intelligent factories for industry 4.0.

Real-time support in SoSs is an open research challenge due to the lack of central control as well as the evolving and dynamic nature of the interactions between the constituent systems. In monolithic systems, the dynamic introduction of new applications is performed using schedulability tests in order to ensure that accepted applications meet their real-time requirements and new applications do not affect existing ones. Previous work includes different types of admission control algorithms to determine whether a new application can be accepted based on the quality-of-service requirements [6], [7], [8].

This paper proposes an SoS architecture with support for real-time requirements based on managed traffic and dynamic configuration. Each constituent system is equipped with a Constituent System Manager (CSM), which not only configures the local communication networks within the constituent systems but also interacts with the CSMs of other constituent systems and the backbone infrastructure of the SoS to establish resource reservations.

We formulate an incremental, distributed and concurrent scheduling problem for the CSMs. The computed schedules lead to resource reservations for time-triggered communication and computational activities.

The remainder of the paper is structured as follows. Section II presents the conceptual model of the SoS. The incremental, distributed and concurrent scheduling is discussed in Section III. Section IV presents the Mixed Integer Linear Programming (MILP) scheduling problem. The experimental evaluation is discussed in Section V. The paper finishes with the conclusion in Section VI.

II. Conceptual Model of SoS

This section describes the SoS from logical and physical viewpoints. The introduced structural models are the basis for the subsequent formulation of the dynamic scheduling and allocation problem.

The overall conceptual model of the SoS is depicted in Figure 1. The SoS is comprised of constituent systems, where each constituent system is a distributed embedded systems, which is under the control of a given organization. Each constituent system consists of endsystems that are interconnected by real-time networks. Networks can include communication networks with different protocols and topologies (e.g., multi-star topology as depicted in Figure 1).

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 configures the routers in the network domain, while also coordinating with other network domains and constituent systems.

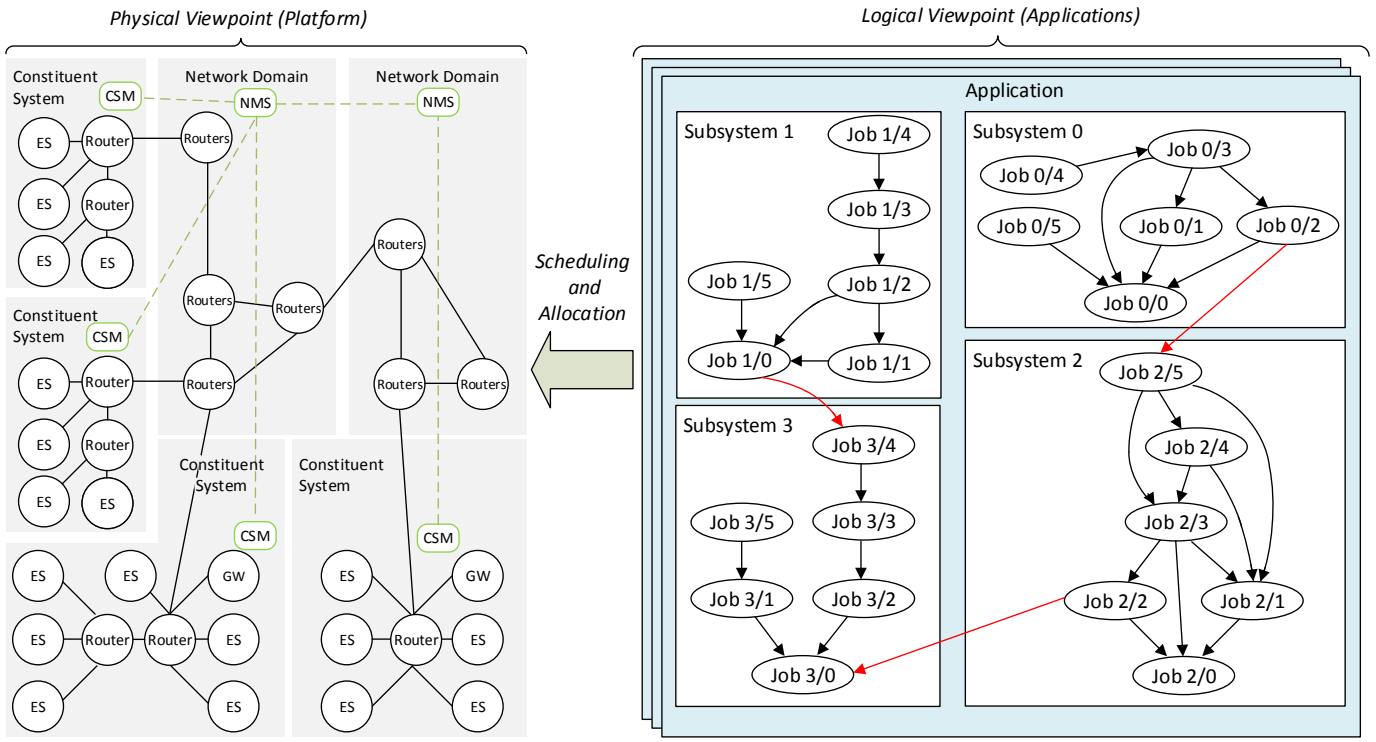

Fig. 1: Physical Viewpoint of the SoS with Constituent Systems, Constituent System Managers (CSM), End-Systems (ES) and Network Management Systems (NMS) and Logical Viewpoint with Applications, Subsystems, Jobs and Messages (Arrows)

Likewise, each constituent systems contains management services named Constituent System Manager (CSM). The CSM performs the local configuration of the endsystems and networks within the constituent system. In addition, the CSM is responsible for the coordination with other constituent systems and network domains.

From a logical point of view, the SoS consists of applications, each of which is a hierarchical Directed Acyclic Graph (DAG) with subsystems and services. The messages between subsystems and services represent the dependencies in the DAG. As an example, consider a medical application for health monitoring and patient care. This application involves different subsystems with respective services. A constituent system 'patient home' hosts a subsystem 'health monitoring' with local services (e.g., sensors, user interfaces). A constituent system 'hospital' can provide a subsystem 'health alarm' including local services for health records, the analysis of sensory data and the issuing of emergency treatment. A constituent system 'caregiver' would offer a subsystem 'emergency response' with services for remote interaction with patients.

From this example, we see the dynamic nature, large-scale, heterogeneity and lack of central control. Numerous of these medical applications will run in parallel for different patients, while sharing the infrastructure (e.g., network domains) and the constituent systems (e.g., hospitals, caregivers). In addition, other types of applications (e.g., energy management) will be active at the same time. The SoS is highly dynamic, e.g., when new patients are integrated into the system. The resource allocation also involves the coordination between different organizations (e.g., providers of network domain, hospitals, patients).

While the discovery and peering of services is addressed in previous work (e.g., service-oriented architectures [9], IoT-A [10], FIWARE IoT Discovery [11]), the end-to-end resource allocation and scheduling for SoS involving real-time, reliability and safety requirements is an open research problem.

In this paper, we provide a solution to this end-to-end resource allocation and scheduling based on the assumption of time-triggered protocols within the constituent systems and the network domains. This assumption is justified given the widespread use of time-triggered protocols in safety-relevant embedded systems (e.g., TTP in railway, TTEthernet in avionics, FlexRay in automotive) and the ongoing standardization activities for IEEE 802.1 [12], which introduces scheduled traffic based on time-triggered communication plans, while also offering run-time configurability and management capabilities. Likewise, Time Division Multiple Access (TDMA) with dynamic configuration capabilities is employed in protocols for the network domain (e.g., MPLS) for the resource allocation and quality-of-service guarantees.

III. Distributed, Incremental and Concurrent Scheduling in SoS

The 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:

- *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 endsystems within each constituent system.
- *Communication within constituent systems:* Messages between jobs of an application subsystem must be scheduled using paths between endsystems along routers.

In many safety-relevant systems, the inherent determinism of the time-triggered paradigm comes at the expenses of significantly reducing flexibility when adaptation to new events is required. For SoS, it is of crucial importance to dynamically adapt to the addition, change and removal of application services and physical building blocks (e.g., constituent systems, network domains). At the same time, we need to retain real-time and safety properties. Overcoming this limitation implies the ability of modifying the time-triggered schedule during runtime rather than precalculating offline schedules.

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, SoS lack central information about the internal structure of all constituent systems.

Therefore, the following three principles are the foundation for the scheduling and allocation in SoSs:

- **Incremental scheduling.** In incremental scheduling, the transmission schedules of specific sending entities in the network are extended or modified whenever additional scheduled messages are required or whenever communication parameters are modified. An incremental transmission schedule thus does not completely replace an existing transmission schedule. However, it may modify some aspects of an existing transmission schedule to facilitate an incremental scheduling step. In order to achieve this, the incremental approach for deterministic networks should not require global knowledge about the overall network topology. The trade-off is that increasing the level of information about the network will result in better schedules at the expense of increased computation resources and network traffic for scheduling.
  In this paper, the incremental scheduling is driven by the dependencies imposed by the DAG of an application. An application subsystem can be scheduled after the relied upon subsystems have been scheduled. The dependencies comprise the messages between the application subsystem, where the transmission times determine the earliest possible start times for the dependant subsystems.
- **Distributed Scheduling.** Distributed scheduling reduces the overall scheduling time by parallelizing the search for a feasible solution using horizontal, vertical and diagonal partitioning schemes. We distribute the scheduling by computing the schedule of each application subsystem at the respective constituent system. The vertical partitioning of the scheduling problem results from the incremental scheduling steps of an application. In addition, the scheduling problem is horizontally partitioned along the different applications.
- **Concurrent Scheduling.** In a SoS many change requests can be requested and processed in parallel. Therefore, a SoS inherently requires concurrent scheduling of change requests while preserving the consistency in the configurations of constituent systems and network domains. For example, several new applications can be introduced at the same time as indicated in the medical monitoring scenario described above.

A. Formal Description of Platform

For the formal description of the physical viewpoint we introduce a set of endsystems ES, a set of constituent systems C, a set of network domains N and a set of routers R. The elementary physical building blocks B (called nodes henceforth) are the routers and endsystems, whereas constituent systems and network domains are composite structures.

$$B = ES \cup R \tag{1}$$

The platform is described by the following graph:

$$G_P =< V_P, E_P >, V_P = B, E_P = B \times B \tag{2}$$

Vertices are endsystems and routers, while edges represent the communication links between routers and constituent systems.

Each node either belongs a constituent system or it is part of a network domain of the SoS backbone infrastructure. This mapping is described by the following function f:

$$f_P : B \mapsto C \cup N \tag{3}$$

For a given constituent system or network domain, the nodes and the edges between these nodes must form a connected subgraph of G_P.

Based on the constituent systems and network domains, we can define a high-level physical graph G_{HP} of the SoS.

$$G_{\text{HP}} =< V_{\text{HP}}, E_{\text{HP}} >, V_{\text{HP}} = C \cup N$$
$$E_{\text{HP}} = \{(e_1, e_2) | \exists \alpha, \beta \in E_P : f_P(\alpha) = e_1 \wedge f_P(\beta) = e_2\}$$

B. Formal Description of Applications

From a logical point of view, the SoS consists of applications, where each application consists of jobs J that interact via the exchange of messages. An application A is described by the following DAG:

$$G_A =< V_A, E_A >, V_A = J, E_A \subseteq J \times J \tag{4}$$

The edges between the jobs are messages, which are exchanged between jobs.

Each application consists of application subsystems AS, which are connected subgraphs of G_A. The mapping of jobs to

application subsystems is described by the following function f_A:

$$f_A : J \mapsto AS \tag{5}$$

Based on the application subsystems, we can define a high-level logical graph G_{HA} of an application. This graph does not include jobs, but only application subsystems and the messages (i.e., edges) between application subsystems.

$$G_{\text{HA}} =< V_{\text{HA}}, E_{\text{HA}} >, V_{\text{HA}} = AS, E_{\text{HA}} \subseteq AS \times AS$$
$$E_{\text{HA}} = \{(e_1, e_2) | \exists \alpha, \beta \in E_A : f_A(\alpha) = e_1 \wedge f_A(\beta) = e_2\}$$

C. Formal Description of Scheduling and Allocation in SoS

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 to constituent systems:

$$\text{ALLOC}_{\text{AS}} : AS \mapsto C \tag{6}$$

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.

$$\begin{aligned}\text{ALLOC}_m : E_{\text{HA}} \mapsto p, E_{\text{HA}} =< \alpha, \beta >, p = (p_1, p_2, \ldots, p_n)\\ p_1 = \text{ALLOC}_{\text{AS}}(\alpha)\\ p_n = \text{ALLOC}_{\text{AS}}(\beta)\\ \forall i \in \{1, 2, \ldots, n-1\} :< p_i, p_{i+1} >\in E_{\text{HP}}\end{aligned}$$

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 endsystems $\overline{ES}$ of c:

$$\begin{aligned}\text{ALLOC}_{\text{job,c}} : \bar{J} \mapsto \overline{ES}\\ \bar{J} = \{j \in J | f_A(j) = as \wedge \text{ALLOC}_{\text{AS}}(as) = c\}\\ \overline{ES} = \{es \in ES | f_P(es) = c\}\end{aligned} \tag{7}$$

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

$$\begin{aligned}\text{SCHEDULE}_{m,c} : M \mapsto p,\\ M = \{< \alpha, \beta >\in E_A | f_A(\alpha) = f_A(\beta) = as \wedge \text{ALLOC}_{\text{AS}}(as) = c\}\\ p = (p_1, p_2, \ldots, p_n)\\ p_1 = \text{ALLOC}_{\text{job,c}}(\alpha)\\ p_n = \text{ALLOC}_{\text{job,c}}(\beta)\\ \forall i \in \{1, 2, \ldots, n-1\} :< p_i, p_{i+1} >\in E_P, f_P(p_i) = f_P(p_k) = c\end{aligned}$$

A message is an edge $< \alpha, \beta >$ in the 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.

trigger: new application A with $G_{\text{HA}} =< V_{HA}, E_{HA} >$
determine ALLOC_{AS}
determine ALLOC_m
M_u = E_{HA} // set of unscheduled messages
M_a = V_{HA} // set of unscheduled application subsystems
while $M_u \neq \varnothing$ **do**
  determine enabled messages $M_e \subseteq M_u$
  pick a message $m =< as_1, as_2 >\in M_e$
  // retrieve path
  $p \leftarrow \text{ALLOC}_m(m)$
  // schedule sending application subsystem as_1
  // at constituent system $c = p_1$ (if unscheduled)
  if $as_1 \in M_a$ **then**
    incremental update of $ALLOC_{\text{job,c}}$ for jobs in as_1
    incremental update of $\text{SCHED}_{\text{m,c}}$ for msgs. in as_1
    $M_a \leftarrow M_a \setminus as_1$
  end
  // schedule network domains
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    incremental update of $\text{SCHED}_{\text{m,n}}$ for $<p_i, p_{i+1}>$
  end
  // schedule receiving application subsystem as_2
  // at constituent system $c = p_n$ (if unscheduled)
  if $as_2 \in M_a$ **then**
    incremental update of $ALLOC_{\text{job,c}}$ for jobs in as_2
    incremental update of $\text{SCHED}_{\text{m,c}}$ for msgs. in as_2
    $M_a \leftarrow M_a \setminus as_2$
  end
  $M_u \leftarrow M_u \setminus m$
end

Algorithm 1: Scheduling algorithm for new application A

D. Scheduling and Allocation Algorithm

The scheduling and allocation algorithm is summarized in Algorithm 1. The scheduling process is triggered by the arrival of a new application A. Initially, the allocation of subsystems to constituent systems $ALLOC_{\text{AS}}$ and the paths between constituent systems $ALLOC_m$ are determined. Thereafter, an enabled message is retrieved from the high-level application graph. A message is enabled if the relied upon application subsystems were already scheduled or if there are no relied upon application subsystems. In this case, the allocation and scheduling of the jobs and messages within the sending application subsystem as_1 is performed (i.e., $\text{ALLOC}_{\text{job,c}}$ and $\text{SCHEDULE}_{m,c}$). After the messages are scheduled on the network domains, the jobs and messages within the receiving application subsystem as_2 are scheduled.

IV. Scheduling Model

This section presents the scheduling model for the incremental scheduling steps as introduced in the previous section.

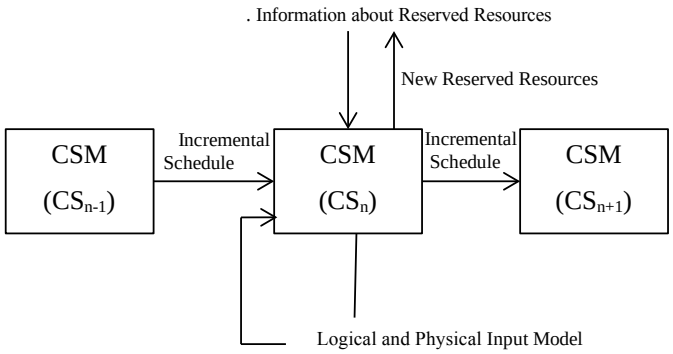

Fig. 2: Logical Viewpoint of Local Scheduler

The model serves for the scheduling of an application subsystem in a constituent system according to Algorithm 1. Hence, the presented scheduling model serves for the local scheduling problem that needs to be solved by a CSM.

The CSM needs to interact with other CSM as part of the distributed and incremental scheduling. In general, a subsystem will depend on messages from other subsystems and provide relied-upon messages to other subsystems. We denote these messages as border messages (red arrows in Figure 1) and we distinguish between incoming and outgoing border messages.

A. Input

1) Input Model: Table I depicts a summary of the constants with their associated domains. A constituent system consists of a number of routers R that can be interconnected in different topologies. Each of these routers has a number of endsystems that are connected in a star topology to the router. The total number of endsystems is ES and the number of nodes of a constituent system is $B = ES + R$. These nodes are interconnected using bi-directional physical communication links which can be described by a two-dimensional boolean array C, in which the B^2 values of the matrix are either 0 (not connected) or 1 (connected). In this work, the connectivity matrix is sorted where all endsystems come first and then the routers. This helps to reduce the computation time for scheduling.

To simulate the transmission and reception of border messages between applications in the proposed model, border routers BR are introduced in each constituent system. These routers are the access-points of constituent systems to the respective network domain. Conceptually, these routers allocate the jobs that either send or receive border messages. For better understanding and simplicity of the model, one border router is introduced in each constituent system.

The connection of routers to endsystems is listed in a vector D_{Router} that is determined by the connectivity matrix C. Each message requires a certain time, depending on the size of the message, to be transmitted on a link. Thus, every time a message is sent from one link to another one, its time is advanced by a hop transmission time U.

The application subsystem consists of a number of jobs J that communicate with each other by the exchange of M messages. These uni-directional messages are sent by the sending jobs, which are denoted by the vector S, where one job can send more than one message. These messages are received by jobs which can be specified in a two-dimensional boolean array D, where rows represent messages and columns represent receiving jobs. For example, $d_{2,4} = 1$ denotes that message 2 is sent to job 4.

When a message is transmitted inside a subsystem, the sender of this message is an endsystem. On the other hand, when the message originates from outside the subsystem, then the sender of the message is modelled as a border router in the scheduling problem. A boolean vector SN is used to specify whether a message is locally injected ($sn_m = 1$) or from another constituent system ($sn_m = 0$). Similarly, a boolean vector DN is used to differentiate between locally received messages ($dn_m = 1$) or outgoing border messages ($dn_m = 0$).

To keep track of the number of incoming border messages and outgoing border messages in each subsystem the constants INC and OUT are used respectively. Every message in the SoS has a unique identifier GID called the global message ID.

The computation time of jobs E is the execution time needed by the receiving job before sending a subsequent message.

2) Resource Information of CS: The introduction of a new application subsystem in a constituent system triggers dynamic reconfiguration by requiring a schedule for the additional jobs and messages. The CSM needs to calculate a new schedule for these jobs taking into account the reserved resources of previous schedules. Therefore, a multi-dimensional array Res is used to keep track of these reserved resources. The first and the second dimensions refer to the indices of the two nodes connecting the link (i.e., range $1 \dots B$). Finally, the third dimension denotes the index of the reservation of this link. Each link can have more than one reservation.

3) Scheduler State: Incremental scheduling in an SoS is the scheduling of messages that are transmitted between different subsystems. This requires information about transmission times of border messages in each subsystem in order to schedule these messages in the next subsystem.

A set of tuples BM is used that records the finish times of all border messages sent between subsystems. Each tuple contains two non-negative numbers, namely a global message ID and a finish time (ft). The finish time denotes the time by which a incoming border message is received at the border router towards the other constituent systems.

$$BM = \{(gid_1, ft_1), (gid_2, ft_2), \dots\}$$
$$\text{with } gid_i \in \{1, 2, \dots, M\}, ft_i \in \mathbb{N}$$

B. Decision Variables

The local scheduler of the CSM generates two types of output information. A new schedule state for the new jobs and updated information about reserved resources. The latter

Domain	Constant name	Type	Description
Constituent System	ES	$\mathbb{N}$	Number of endsystems
	R	$\mathbb{N}$	Number of routers
	B	$ES + R \in \mathbb{N}$	Numbers of nodes (ES+R)
	BR	$\mathbb{N}$	Number of border routers
	C	$\begin{bmatrix} c_{1,1} & \dots & c_{1,B} \\ \vdots & \ddots & \vdots \\ c_{B,1} & \dots & c_{B,B} \end{bmatrix} \in \{0,1\}^{N \times N}$	Node connectivity
	D_{Router}	$[dr_1 \dots dr_{ES}]^T \in \{ES+1, ..., B\}^{ES}$	Router connected to an endsystem
	U	$[u_1 \dots u_M]^T \in \mathbb{N}^M$	Hop transmission time
Application Subsystem	J	$\mathbb{N}$	Number of jobs
	M	$\mathbb{N}$	Number of messages
	S	$[s_1 \dots s_M]^T \in \{1, ..., J\}^M$	Sender jobs
	D	$\begin{bmatrix} d_{1,1} & \dots & d_{1,J} \\ \vdots & \ddots & \vdots \\ d_{M,1} & \dots & d_{M,J} \end{bmatrix} \in \{0,1\}^{M \times J}$	Destination jobs
	SN	$[sn_1, \dots, sn_M]^T \in \{0,1\}^M$	Vector denoting for each msg. whether of local origin
	DN	$[dn_1, \dots, dn_M]^T \in \{0,1\}^M$	Vector denoting for each msg. whether with local destination
	INC	$\mathbb{N}$	Number of incoming border messages
	OUT	$\mathbb{N}$	Number of outgoing border messages
	E	$[e_1 \dots e_J]^T \in \mathbb{N}^J$	Job execution time
	GID	$[gid_1, \dots, gid_M]^T$	Global message ID

TABLE I: Overview Input Table

is used to update the reserved resource database for subsequent scheduling steps.

1) New Schedule: This output information contains the schedule of the new jobs and messages. It consists of a schedule for time-triggered messages (i.e., mapping of jobs to endsystems, message paths) taking into account the dependencies with other local messages and border messages.

a) Job Allocation: These variables denote the allocation of jobs to the nodes of the physical platform model. Jobs that send and receive local messages can only be allocated to endsystems while jobs that either send or receive border messages are allocated to the border router. Since nodes are sorted with endsystems and border routers coming first, the maximum value a_i of an allocation variable is the sum of the numbers of endsystems and border routers.

$$A = [a_1 \dots a_J]^T \in \{1, ..., ES + BR\}^J$$

To ensure that each job is allocated to exactly one endsystem, a boolean matrix $ALLOCM$ is used where the rows relate to jobs and columns to endsystems. For example, $mat_{3,1} = 1$ means that job 3 is allocated to endsystem 1.

$$ALLOCM = \begin{bmatrix} mat_{1,1} & \dots & mat_{1,ES} \\ \vdots & \ddots & \vdots \\ mat_{J,1} & \dots & mat_{J,ES} \end{bmatrix} \in \{0,1\}^{J \times ES}$$

To keep track of routers via which a job can transmit a message we use a vector R. The vector R denotes for each job an access-point router that is directly accessible from the endsystem where the job is located. All other routers can only be reached by more than one hop. For example, $sr_2 = 14$ denotes that the router with ID 14 is the access-point router for the endsystem hosting the job 2.

$$SR = [sr_1 \dots sr_J]^T \in \{Z, ..., B\}^J$$

where $Z = ES + BS + 1$.

b) Hop Count: A message is injected at the source endsystem, where the sender job was allocated. It is then transported along one or more routers before being received by the endsystem of the destination job. In order to express the number of visited routers for each message after the access-point router the vector hop count H is used and the maximum value of its elements denotes the critical path length. In the absence of cyclic paths, the maximum path length is $\max_H = R - 1$.

$$H = [h_1 \dots h_M]^T \in \{1, ..., \max_H\}^M$$

c) Injection Time: This one-dimensional array represents the times by which the messages are injected in the network of thec onstituent system.

$$I = [i_1 \dots i_M]^T \in \{0, ..., \mathbb{N}\}^M$$

d) Path and Visited Routers: To record the path between the message's source and destination endsystem, the path array P is used. Since the sending and the receiving jobs are known beforehand, each row represents the path of a message starting from the router connected to the endsystem which allocates a source job to the router connected to the endsystem in which the destination job is allocated. For example, $p_{1,2} = 14$ means

that the second router that message number 1 visits is the node with ID 14. The maximum number of nodes in a path equals the maximum number of hops.

$$P = \begin{bmatrix} p_{1,1} & \cdots & p_{1,R} \\ \vdots & \ddots & \vdots \\ p_{M,1} & \cdots & p_{M,R} \end{bmatrix} \in \{Z, ..., B\}^{M \times R}$$

where $Z = E + 1$.

For the purpose of calculating the end-to-end latency, a boolean matrix O is used to denote the routers that are passed by a message. For example, $o_{2,3} = 1$ means that message 2 travels through a router with ID 3.

$$O = \begin{bmatrix} o_{1,1} & \cdots & o_{1,R} \\ \vdots & \ddots & \vdots \\ o_{M,1} & \cdots & o_{M \times R} \end{bmatrix} \in \{0, 1\}^{M \times R}$$

2) Reserved Resources: After a schedule is generated, the transmission links for paths of all messages are used to update a reserved resources database Res. Each entry in this database consists of the IDs of the endsystems and/or routers connecting the reserved link in addition to the start time of a message at the specified link. For example, $Res_{3,5,2} = 10$ denotes that the link connecting nodes 3 and 5 has two reservations. The second reservation starts at 10 ms and has a duration of the transmission time of the message u_m.

C. Scheduling Constraints

This part describes the constraints that are used in the scheduling of time-triggered messages in a constituent system.

1) Distributed Scheduling Constraints: As a prerequisite for the distributed scheduling, the CSM requires information about the transmission times of border messages that are exchanged between different application subsystems.

The injection times of local messages ($sn_m = 1$) as well as incoming border messages ($sn_m = 0$) in a subsystem can be evaluated as follows:

$$\begin{aligned} &\forall m_i \ \in \{1, ..., M\} : \\ &\quad sn_{m_i} = 1 \rightarrow i_{m_i} \geq u_{m_i} \\ &\quad sn_{m_i} = 0 \rightarrow i_{m_i} \geq i_{m_i} \text{ where } (gid_{m_i}, i_{m_i}) \in BM \end{aligned} \tag{8}$$

2) Incremental Scheduling Constraints: New applications introduce additional jobs where the new schedule must take into account the reserved resources of the previous schedule as denoted by Res. These corresponding constraints can be divided into three groups:

- Constraints for links between sending endsystems and their access-point routers
- Constraints for links among routers
- Constraints for links between receiving endsystems and their access-point routers

a) Reserved resources between sending endsystems and their access-point routers: Endsystems are connected to routers in a star topology. Hence, if the sender is an endsystem, it means that there is only one link where the first node is an endsystem and the second node is its access-point router.

$$\begin{aligned} &\forall m_1 \in \{1, ..., M\}, \forall r_1, r_2 \ \in \{1, ..., B\} : \\ &\quad (Res_{r_1, r_2, 0} \geq 0) \wedge (r_1 \leq ES) \wedge (sn_{m_1} = 1) \\ &\quad \rightarrow \Bigg((a_{s_{m_1}} \neq r_1 \vee p_{m_1,0} \neq r_2) \\ &\qquad \vee \Big(\bigwedge_{z=1}^{M} (i_{m_1} \leq Res_{r_1, r_2, z}) \\ &\qquad \vee (i_{m_1} - u_{m_1} \geq Res_{r_1, r_2, z} + i_{m_1}) \Big) \Bigg) \end{aligned} \tag{9}$$

b) Reserved resources among routers: Since the connections of the routers can have different topologies, all possible paths need to checked regarding the reserved resources.

$$\begin{aligned} &\forall m_1 \in \{1, ..., M\}, \forall r_1, r_2 \ \in \{1, ..., B\} : \\ &\quad (Res_{r_1, r_2, 0} \geq 0) \wedge (r_1 \leq ES) \wedge (dn_{m_1} = 1) \\ &\quad \rightarrow \Bigg(\bigvee_{r_3=2}^{R} \Big((p_{m_1, r_3 - 1} \neq r_1 \vee p_{m_1, r_3} \neq r_2) \\ &\qquad \wedge (p_{m_1, r_3 - 1} \neq r_2 \vee p_{m_1, r_3} \neq r_1) \Big) \\ &\qquad \vee (h_{m_1 < r_3}) \\ &\qquad \vee \Big(\bigwedge_{z=1}^{M} (i_{m_1} + r_3 \cdot u_{m_1} \leq Res_{r_1, r_2, z}) \\ &\qquad \vee (i_{m_1} + (r_3 - 1) \cdot u_{m_1} \geq Res_{r_1, r_2, z} + i_{m_1}) \Big) \Bigg) \end{aligned} \tag{10}$$

c) Reserved resources between receiving endsystems and their access-point routers: Again, if the receiving node is an endsystem, it means that there is only one link where the first node is an endsystem and the second node is its access-point router.

$$\begin{aligned} &\forall m_1 \in \{1, ..., M\}, \forall r_1, r_2 \ \in \{1, ..., B\} : \\ &\quad (Res_{r_1, r_2, 0} \geq 0) \wedge (r_1 > ES) \wedge (r_2 > ES) \\ &\quad \rightarrow \forall j_1 \in \{1, ..., J\} : \\ &\qquad d_{m_1, j_1} = 1 \\ &\quad \rightarrow \Bigg((a_{j_1} \neq r_1 \vee p_{m_1,0} \neq r_2) \\ &\qquad \vee \Big(\bigwedge_{z=1}^{M} (i_{m_1} + r_3 \cdot u_{m_1} \leq Res_{r_1, r_2, z}) \\ &\qquad \vee (i_{m_1} + (r_3 - 1) \cdot u_{m_1} \geq Res_{r_1, r_2, z} + i_{m_1}) \Big) \Bigg) \end{aligned} \tag{11}$$

3) Connectivity Constraint: The first constraint considers the path topology of the network based on the node connectivity C. Since an endsystem is connected to only one router,

the connectivity constraints can be reduced by considering only the routers. If there is no direct connection between two routers a and b, then the path of a message must not include a hop from a to b.

$$\begin{aligned}&\forall m_1 \in \{1,...,M\}, \forall r \in \{1,...,Max_H\}: \\ &h_{m_1} \geq r+1 \\ &\rightarrow \left(\bigvee_{a,b=ES+1}^{B} c_{a,b} = 1 \rightarrow \textit{Connected(a,b)} \right)\end{aligned} \tag{12}$$

where the function *Connected()* states that a message's path is allowed to pass through the link between the two routers a and b.

$$\textit{Connected}(a,b) = (p_{m_1,r} = a \wedge p_{m_1,r+1} = b)$$

4) Collision Avoidance Constraint: These constraints are divided into three groups:

- Constraints to avoid collisions between a sending node and its access-point router
- Constraints to avoid collisions between routers
- Constraints to avoid collisions between a receiving node and its access-point router

The first constraints apply when a job sends more than one message. Since, there is only one link between any endsystem and its access-point router, the constraints ensure that transmission times following the injection times I do not overlap.

$$\begin{aligned}&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\}, \\ &s_{m_1} = s_{m_2} \\ &\rightarrow (i_{m_1} \geq i_{m_2} + u_{m_2} \\ &\quad \vee\, i_{m_2} \geq i_{m_1} + u_{m_1})\end{aligned} \tag{13}$$

To prevent collisions of transmissions between routers, the scheduling of time-triggered messages ensures that no two messages are transmitted on one link at the same time. Thus, the messages should be transmitted on different paths or one needs to be scheduled before or after the transmission of the other message.

$$\begin{aligned}&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\}, \\ &\forall r_1, r_2 \in \{1,...,Max_H\}: \\ &(p_{m_1,r_1} \neq p_{m_2,r_2} \vee p_{m_1,r_1+1} \neq p_{m_2,r_2+1} \\ &\vee\, r_1+1 > h_{m_1} \vee r_2+1 > h_{m_2} \\ &\vee\, i_{m_1} + (r_1+1)\cdot u_{m_1} \leq i_{m_2} + r_2 \cdot u_{m_2} \\ &\vee\, i_{m_2} + (r_2+1)\cdot u_{m_2} \leq i_{m_1} + r_1 \cdot u_{m_1})\end{aligned} \tag{14}$$

The third type of constraints is used when a job receives more than one message. Since there is only one link between an endsystem and its access-point router, these constraints ensure that the transmission times of the messages from the access-point router to the endsystem do not overlap.

$$\begin{aligned}&\forall m_1 \in \{1,...,M\}, m_2 \in \{m_1+1,...,M\}, \\ &j_1 \in \{1,...,J\}: \\ &d_{m_1,j_1} = 1 \wedge d_{m_2,j_1} = 1 \\ &\rightarrow (i_{m_1} + (h_{m_1}+1)\cdot u_{m_1} \leq i_{m_2} + h_{m_2} * u_{m_2} \\ &\quad \vee\, i_{m_2} + (h_{m_2}+1)\cdot u_{m_2} \leq i_{m_1} + h_{m_1} * u_{m_1})\end{aligned} \tag{15}$$

5) Job Dependency Constraint: Depending on the precedence constraints between the jobs, jobs may need to wait for the output of the transmission of other jobs before they begin the transmission. This constraint ensures that if a job sends a message m_1 to another job that needs the output of m_1 in order to send m_2, the start time of m_2 must be after the end of the transmission and execution of m_1.

$$\begin{aligned}&\forall m_1, m_2 \in \{1,...,m\}, \forall j_1 \in \{1,...,j\}: \\ &d_{m_1,j_1} = 1 \ \wedge \ s_{m_2} = j_1 \\ &\rightarrow i_{m_1} + (h_{m_1}+1)\cdot u_{m_1} + e_{j_1} < i_{m_2}\end{aligned} \tag{16}$$

Each message must reach the destination node within its path and the selected number of hops.

$$\begin{aligned}&\forall m_1 \in \{1,...,M\}, \forall j_1 \in \{1,...,J\} \\ &d_{m_1,j_1} = 1 \\ &\rightarrow \left(\bigvee_{r_1=1}^{W} (p_{m_1,r_1} = sr_{j_1} \ \wedge \ r_1 = h_{m_1}) \right)\end{aligned} \tag{17}$$

6) Job Assignment Constraints: These constraints ensure that a job can be assigned to only one endsystem. This is done by having the sum of each row in ALLOCM (i.e., for each job) equal to 1. Then, the allocated endsystems are stored in the allocation array A and the access-point routers of the endsystems are stored in SR.

$$\begin{aligned}&\forall j_1 \in \{1,...,j\}: \\ &\left(\sum_{r_1=1}^{E} mat_{j_1,r_1} \right) = 1 \\ &\left(\bigvee_{r_1=1}^{E} mat_{j_1,r_1} = 1 \rightarrow (a_{j_1} = r_1 \wedge sr_{j_1} = dr_{r_1}) \right)\end{aligned} \tag{18}$$

To allow only one job to be allocated to an endsystem, the sum for each endsystem must be less than or equal to one.

$$\begin{aligned}&\forall r_1 \in \{1,...,E\}: \\ &\left(\sum_{j_1=1}^{j} mat_{j_1,r_1} \right) \leq 1\end{aligned} \tag{19}$$

In order to start the path of each message with the access-point router of the endsystem that hosts the job, the first node for each message path $p_{1,1} \ldots p_{m,1}$ is required to be the access-point router.

$$\begin{aligned}&\forall m_1 \in \{1,...,M\}, \forall j_1 \in \{1,...,J\}: \\ &s_{m_1} = j_1 \\ &\rightarrow \left(\bigvee_{r_1=1}^{ES} (a_{j_1} = r_1 \wedge p_{m_1,1} = dr_{r_1}) \right)\end{aligned} \tag{20}$$

D. Objective Function

The objective is to minimize the maximum transmission time of the time-triggered messages (i.e., minimization of critical path). This is done by first finding the transmission time of each time-triggered message, expressed as the sum of

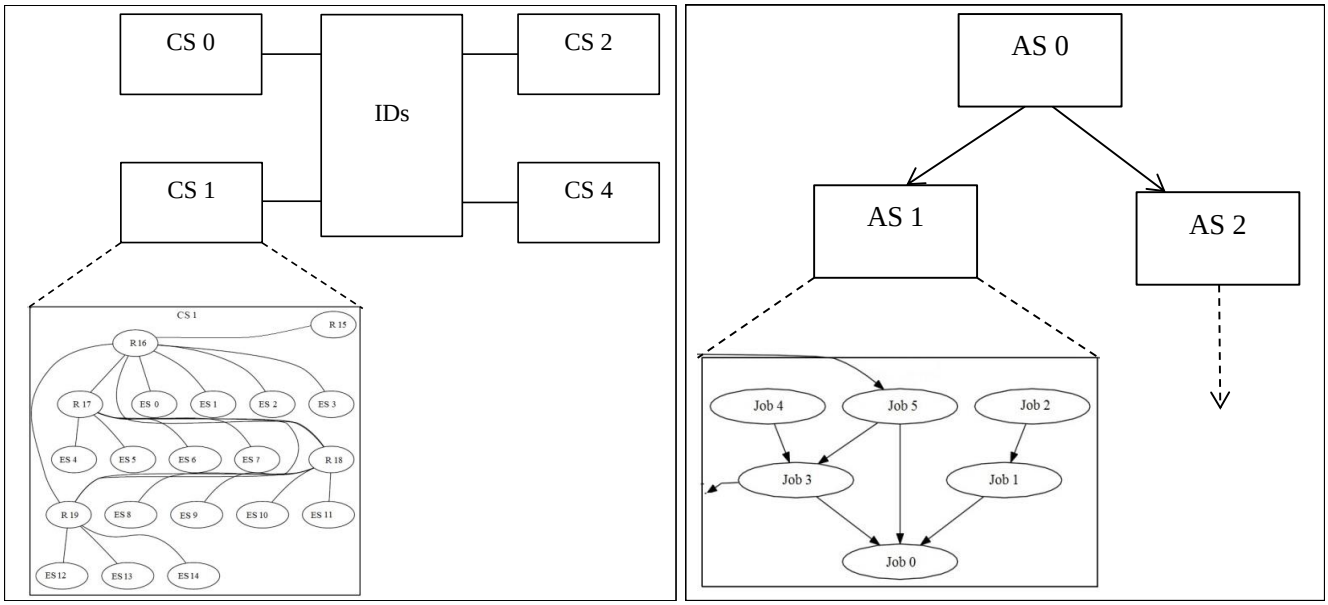

Fig. 3: Generated Platform and Application

the injection time i_m and the number of hops h_m multiplied by the transmission duration of a message u_m. Then, the objective function minimizes the highest value among all these messages.

$$\begin{aligned} &\forall m_1 \in \{1, ..., M\} : \\ &\quad CP[m_1] = (i_{m_1} + h_{m_1} \cdot u_{m_1}) \\ &minimize \quad max(CP) \end{aligned} \tag{21}$$

V. Experimental Evaluation

This section discusses the experimental evaluation. A generator for example scenarios and a high-level scheduler were implemented to validate the scheduling problem.

A. Generator for Example Scenarios

A generic SoS generator was realized to build example scenarios for the evaluation of the proposed scheduling models. Based on input parameters, the generator creates random platforms and applications according to the conceptual model introduced in Section II. 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 routers. 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.

The Stanford Network Analyzer Platform (SNAP) [13] library was used for the generation of DAGs and undirected graphs in the generations. A DAG is required for the graph of jobs in each application subsystem as well as for the interconnection of application subsystems. The undirected graphs describe the connectivity of the routers in constituent systems as well as the interconnection of network domains and constituent systems. The outputs are visualized using the GraphViz library.

An example of a generated scenario is shown in Figure 3. The figure depicts in detail one of the constituent systems with 20 endsystems and routers where node ID 15 is a border router; it has also one of the application subsystems with 6 jobs. The jobs of the application subsystem 1 send 6 local messages and one border message. In addition, one incoming border message is received from another constituent system.

B. High-Level Scheduler

A high-Level scheduler was implemented to evaluate the conceptual model and the scheduling problem. This high-level scheduler implements Algorithm 1 by performing a random allocation of application subsystems to constituent systems. The paths between application subsystems are determined by computing the shortest paths. An extension of the high-level scheduler with support for an optimized allocation of application subsystems and path determination at the SoS-level is planned as future work.

The output of the high-level scheduler are CPLEX scheduling models with the constants, constraints and decision variables as introduced in Section IV.

C. Results

Table II depicts the scheduling time for three different SoSs. Every SoS consists of seven applications each containing four application subsystems. The computation times were obtained with CPLEX 12.6.1 running on a 12 processor Intel(R) Xeon(R), 2.2 GHz server with the operating system Linux Ubuntu 14.04.1. CPLEX was used for the local scheduling in each constituent system, either stopping after a feasible solution is found or computing an optimal local schedule.

Scenario	Logical Viewpoint			Physical Viewpoint			Finish Time		Execution Time	
	Applications	Jobs	Messages	CS	Endsystems	Routers	Feasible	Optimal	Feasible	Optimal
SoS 1	1	30	30	4	75	21	39	36	3.08	3.48
	2	30	38				51	48	11.96	27.76
	3	30	30				69	42	14.42	13.55
	4	30	26				63	42	19.47	95.31
	5	30	30				87	84	112.5	331.23
	6	30	30				114	102	136.95	522.36
	7	30	30				96	90	236.21	737.3
SoS 2	1	30	30	4	85	23	39	33	3.72	6.41
	2	30	30				48	60	10.69	11.86
	3	30	30				45	36	10.34	17.91
	4	30	30				123	117	49.61	50.41
	5	30	26				60	54	81.75	98.74
	6	30	26				90	69	39.18	41.39
	7	30	26				75	81	298.76	545
SoS 3	1	30	26	4	74	21	54	51	2.59	2.93
	2	30	30				72	72	4.52	6.45
	3	30	38				78	60	33.56	105.5
	4	30	30				66	66	32.34	520.3
	5	30	26				90	84	21.25	111.33
	6	30	26				72	81	108.37	452.19
	7	30	26				81	93	351.8	692

TABLE II: Results of Different SoS Scenarios

The scheduling time is measured in seconds to find a feasible solution as well as an optimal solution. The finish times in the table denote the makespans of the respective applications in ms. As can be seen in the table, in some cases the optimal local schedule leads to increased makespans of later applications. The reason is the unavailbility of early time slots for messages of subsequent applications.

VI. Conclusion

The increasing importance of SoS with real-time requirements demands techniques for scheduled end-to-end communication with resource reservations and temporal guarantees. The end-to-end communication channels need to span constituent systems with operational and managerial independence, while also supporting an evolutionary development and the dynamic introduction of new applications. The proposed scheduling algorithm solves this challenge by performing an incremental, distributed and concurrent scheduling of applications. Each constituent system and each network domain is equipped with management services for incrementally computing local schedules for the respective application subsystems. The resulting timing information of relied upon messages is exchanged between constituent systems to satisfy the temporal dependencies between application subsystems.

The introduced distributed and incremental scheduling reduces the overall computation time for feasible schedules, while at the same time coping with the inherent limitations of SoSs such as the managerial independence and the lack of global knowledge about the internals of constituent systems. The evaluation demonstrates the achievable real-time guarantees and the computation time for scheduling based on example scenarios.

Acknowledgments

This work has been supported by the European project DREAMS under the Grant Agreement No. 610640.

References

[1] M. Jamshidi, *Systems of Systems Engineering - Principles and Applications*. CRC Press, 2009.

[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *Internet of Things Journal, IEEE*, vol. 1, no. 1, pp. 22–32, Feb 2014.

[3] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *Automation, Quality and Testing, Robotics, 2014 IEEE International Conference on*, May 2014, pp. 1–4.

[4] N. Wickramasinghe, S. Chalasani, R. Boppana, and A. Madni, "Healthcare system of systems," in *System of Systems Engineering, 2007. SoSE '07. IEEE International Conference on*, April 2007, pp. 1–6.

[5] M. Maier, *Architecting Principles for Systems-of-Systems – Systems Engineering*, 1998.

[6] O. Yang and J. Lu, "Call admission control and scheduling schemes with qos support for real-time video applications in ieee 802.16 networks," *Journal of Multimedia*, vol. 1, no. 2, pp. 21–29, 2006.

[7] C. J. Hamann, M. Roitzsch, L. Reuther, J. Wolter, and H. Hartig, "Probabilistic admission control to govern real-time systems under overload," in *Real-Time Systems, 2007. ECRTS'07. 19th Euromicro Conference on*. IEEE, 2007, pp. 211–222.

[8] S. Agrawal, P. Chaporkar, and R. Udwani, "Call admission control for real-time applications in wireless network," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 330–334.

[9] D. Mora, M. Taisch, A. Colombo, and J. Mendes, "Service-oriented architecture approach for industrial system of systems: State-of-the-art for energy management," in *10th IEEE International Conference on Industrial Informatics (INDIN)*, July 2012, pp. 1246–1251.

[10] D. Suparna, G. Cassar, B. Christophe, S. Fredj, M. Bauer, N. Santos, T. Jacobs, R. de las Heras, G. Martin, G. Völksen, and A. Ziller, "Internet of things architecture. concepts and solutions for entity-based discovery of iot resoures and managing their dynamic associations," Tech. Rep., 2012.

[11] FIWARE, *IoT Discovery – User and Programmers Guide*, 2015.

[12] IEEE, *IEEE 802.1Qbv – Enhancements for Scheduled Traffic, Draft 2.4*, 2015.

[13] J. Leskovec, " Stanford Network Analysis Package(SNAP)." http://snap.stanford.edu/, [Online; accessed 26-February-2015].