This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

Network Synthesis for Distributed Embedded Systems

Enrico Fraccaroli, *Student Member, IEEE,* Francesco Stefanni, Romeo Rizzi, Davide Quaglia, *Member, IEEE,* and Franco Fummi, *Member, IEEE*

Abstract—The amazing proliferation of communication technologies for embedded systems opens the way for completely new applications but forces designers to adopt new methodologies to meet time-to-market constraints. CAD has been traditionally applied to computers and embedded systems *in isolation* without considering them as a global inter-connected system. The paper contributes to fill this gap by proposing *1)* a communication-aware design flow for network-interconnected embedded systems and *2)* a formal framework to efficiently synthesize their network aspects by formulating and solving an optimization problem. Presented case studies show the potentiality of the proposed approach to address heterogeneous scenarios, e.g., related to smart spaces up to the ever-more-mentioned Internet-of-Things.

Index Terms—Networked embedded systems, network topology, network protocols, design-space exploration, design tools and techniques, optimization, mixed-integer linear programming, MILP, model-driven design.

1 Introduction

The recent advances in communications for embedded systems open to completely new distributed applications in which hundreds or thousands of smart devices interact together through different types of channels and protocols [1], [2], [3]. We can define them as Distributed Embedded Systems (DESs) since these applications present three important features that distinguish them from traditional network-connected applications, *i.e.*, *1)* the communication aspects affect the design flow of the embedded systems, *2)* system-of-systems nature, and *3)* strict relation with the surrounding physical environment.

To clarify the discussion, we introduce a common example related to the temperature control of a building as depicted in Figure 1. Several sensors (in figure denoted by *S*) detect local temperature. Collected data are sent to controllers (denoted by *C*), which send commands to actuators (denoted by *A*), *e.g.*, coolers. Controllers decide the activation of actuators according to various policies both centralized and distributed; for example, a controller may be present in each room to adjust the local temperature but a centralized controller is also present to ensure that room settings comply with the total energy budget. A controller application can also be executed by personal mobile devices so that each user can control the temperature of the currently occupied space according to a personal profile.

In DESs, *the communication aspects affect the design flow*. Considering the example, physical channels among nodes can be either wireless or wired according to deployment constraints (*e.g.*, cabling costs and feasibility in historical buildings) and mobility requirements. Communication protocols depend on the type of these physical channels, on required reliability and quality of service (*e.g.*, maximum latency). Assuming that highly optimized

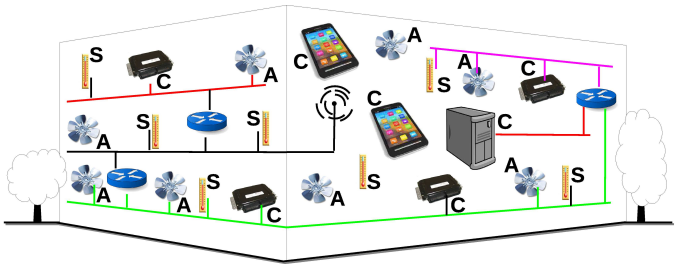


Fig. 1. Example of distributed embedded system for building automation.

nodes are desirable, the choice of physical channels affects the definition of hardware network interfaces while the choice of communication protocols affects the memory and computational requirements of the hardware platform.

Up to now, the embedded systems design flow has jointly addressed hardware and software aspects, while communication aspects have been faced separately by a different research community. This lack of coordination may lead to non-optimal solutions in the system design; in fact, past work demonstrated that HW/SW design and network design are correlated [4]. To further push performance, energy saving and reliability, the network among nodes should be jointly designed with hardware and software components [5]. In particular, CAD should be fruitfully applied not only to each node, as currently done in the context of electronic systems design, but also to the network among them. For this reason, *a communication-aware design flow* is required.

A DES can be seen as a *System-of-Systems* since even if the various nodes can independently operate, they interact together to achieve the *good behavior of the global application* [6]. In the mentioned example, the final objective is to achieve a good control of the temperature and it does not matter the set of nodes that provides such functionality, as long as the global application behavior satisfies design objectives. Thus, DESs pose new questions to designers, traditionally mainly interested in the

- *E. Fraccaroli, R. Rizzi, D. Quaglia and F. Fummi are with the Department of Computer Science, University of Verona, Verona, Italy. E-mail: name.surname@univr.it*
- *F. Stefanni is with EDALab s.r.l., Verona, Italy. E-mail: francesco.stefanni@edalab.it*

Manuscript received July 12, 2017; revised December 12, 2017.

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

specification of each single network node as done for Internet servers and clients. Most relevant issues are:

- finding the optimal number of nodes to achieve the common mission;
- finding the best assignment (according to given metrics) between software tasks and hosting nodes by taking into account tasks' requirements and nodes' capabilities;
- finding the best set (according to given metrics) of network protocols by taking into account communication requirements and the presence of a legacy network infrastructure.

The last distinguishing feature is the *strict relationship with the environment*. In fact, networked sensors and actuators should be placed where is required by the application. Furthermore, environment affects communications in such systems; for instance, walls and distance may affect wireless communications whereas area size affects the deployment cost of wired infrastructure. Finally, the number and position of nodes affect the communications among them and application performance.

Solving these issues leads to the so-called *Network Synthesis*, *i.e.*, the allocation of functionality onto nodes and the complete definition of the communication infrastructure among them. In this context the **contributions of the paper** are:

- a *communication-aware design flow* centered on Network Synthesis for DESs;
- a *communication-aware formal specification of the whole distributed system* to formulate Network Synthesis as an optimization problem;
- the formulation of Network Synthesis as a Mixed Integer Linear Programming (MILP) problem.

The paper can be considered as part of the research roadmap started by [7] and continued by [8]. In [7] a preliminary version of the flow and the formal specification was described but many issues were present and the optimization strategy was missing. In [8] the focus was on model-driven design and simulation by means of a UML-based representation. However, such representation does not allow, per se, the formulation of an optimization problem which was left in background.

The paper is organized as follows. Related work is presented in Section 2. The building blocks of a formal representation for distributed embedded applications and the corresponding design flow are described in Section 3. The formulation of network synthesis as a MILP problem is provided in Section 4. The analysis of its complexity and scalability is reported in Section 5. Experimental results are provided in Section 6. Finally, conclusions and future work are reported in Section 7.

2 Related Work

The design of distributed systems relies on methodologies for their functional specification and techniques for network design.

2.1 System Functional Specification

The information driving network synthesis can be extracted from a platform-independent description of an application created using well-known languages as those reported in this section.

MARTE [9] is an UML [10] profile designed to allow an easy specification of real-time and embedded systems. It provides some sub-profiles, like Non-Functional-Properties (NFPs), which allow to describe the "fitness" of the system behavior (*e.g.* performance, memory usage, energy consumption, *etc.*). The Software Resource Modeling (SRM) and the Hardware Resource Modeling (HRM) profiles are derived from NFP, and they address the modeling of resources. The System Modeling Language (SysML) [11] is an UML extension which provides a general-purpose modeling language for systems engineering applications.

Mathworks has developed Simulink [12] and Stateflow [13] to model and simulate dynamic and embedded systems. The former allows to represent an application as the inter-connection of analog or digital blocks while the latter allows to describe applications as finite-state machines. They can also be combined to represent hybrid automata.

The Ptolemy Project [14] was born to model concurrent real-time and embedded systems. One of its main advantages is the support for heterogeneous mixtures of computation models. Ptolemy supports simulation by using the actor-oriented design; actors are software components executed concurrently and able to communicate by sending messages through interconnected ports. Ptolemy also supports communication modeling through Khan Process Networks (KPN), *i.e.*, groups of deterministic sequential processes which are communicating through unbounded FIFO channels [15].

They are distributed Models of Computation (MoCs) based on tokens which focus on the flow of computation, thus it seems well suited to check properties on the communication schema.

SystemC [16], initially born as an hardware description language, has been extended with the Transaction-Level Modeling (TLM) [17], to describe HW/SW systems. SystemC and TLM allow to describe tasks as nested components with event-driven or clock-driven processes. Communications between tasks can be described by using standard protocols and payloads which simplify the specification of their behavior. TLM was born to represent local communications, such as bus interconnections or accesses to devices.

SpecC [18] is an extension of the C language to be used as system-level design language, like SystemC/TLM. The SpecC methodology is a top-down design flow, with four well-defined levels of abstraction. It allows different ways to describe the target control (sequential, FSM, parallel and behavioral). One key concept of SpecC is the clear separation of the communication and computation model which can be useful to specify computation and communication aspects of tasks.

Metropolis [19] is a framework based on the idea of meta-model to support various communication and computation semantics in a uniform way. This approach implements the abstract semantics of process networks and uses the concepts advocated by the Platform-Based Design (PBD) methodology, *i.e.*, functionality and architecture across models of computation and abstraction levels, and the mapping relationships between them.

2.2 Network Design

Network design has been addressed by many research works, in different fields, such as Wireless Sensor Networkss (WSNs). In [20] a virtual architecture has been proposed in order to simplify the synthesis of WSNs algorithms. Network topology and high-level functionality are used to configure the virtual architecture. This work is mainly focused on the application part of the system rather than on communication aspects.

In [21], PBD has been adopted to design WSNs for industrial control. In PBD, the application is usually designed at a high level and then mapped onto a set of possible actual candidates for the

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

nodes. However, no guideline is provided for selecting an appropriate network architecture and communication protocol. Scope-based techniques have been proposed in macro-programming to specify complex interactions between heterogeneous nodes of a WSN [22]. However, the nodes number and network topology are an input required by the technique and not a result, as in the proposed approach.

A tool for the optimal design of WSNs for building automation has been proposed in [23]. It suggests to integrate the network design flow with the knowledge about the routing algorithm used after the deployment of the network. Since the routing algorithm is known a priori, it further proposes to systematically introduce redundancy in order to maximize the performance of the chosen algorithm. Then, it proposes a sub-optimal polynomial-time heuristic for the synthesis problem and compares it with a custom formalization of the MILP proposed in [24].

A communication synthesis methodology and HW/SW integration for embedded system design has been addressed in [25]. The method is based on task graphs and used after HW/SW partitioning and task scheduling. On one hand, this inversion has the advantage that the scheduling problem is simplified, since communication components will be designed later. On the other hand, it does not consider that scheduling could be optimized if communication aspects were considered earlier.

The design of Network on Chips (NoCs) offers an example of computation-communication integrated approach which is close to the purpose of this paper. NoCs are embedded systems which are designed with the traditional specification-refinement-synthesis flow; nevertheless, they have also a communication infrastructure which is a simplified version of a packet-switched network [26]. The design of the internal NoC communication infrastructure presents problems similar to the one of traditional packet-based networks [27]. For example, the design of NoC to meet hard latency constraints is addressed in [28]. The problem of the optimal mapping of tasks onto NoC's cores is known to be NP-hard. In some works, heuristics based on graph-decomposition techniques have been used [29], [30]. A MILP formulation of the problem has been proposed [31]. It assumes a regular 2D mesh topology and shortest-path static routing. This methodology allows two different optimization criteria, *i.e.*, minimization of the average hop distance (which is proportional to energy consumption and communication delay), as well as minimization of the bandwidth (which consists in minimizing the most-congested link-queuing time and maximizing the throughput). Network synthesis in NoCs is based on strong assumptions on network's features (*e.g.*, the topology); thus, such approaches are not general enough to be applied also to normal networks as proposed in this work.

The efficient routing of communication paths gives another opportunity of optimization, also at different levels of the protocol stack. At the lowest level, a synthesis process for routing of physical wires inside an automotive system is proposed in [32]. It aims at meeting requirements about delay, quality of signal, power and temperature. First, a Steiner tree is generated using a customized Kou-Markowsky-Berman (KMB) algorithm minimizing connections length. Then, a Linear Programming (LP) problem is formulated and its solution is used to modify the Steiner tree such that the overall delay is minimized and signal quality maximized. At a higher level Xu et al. [33] propose a MILP formulation applied to ZigBee wireless networks. It comprises four specific groups of constraints: devices placement, link activation for routing, connections scheduling and communication quality of service. Their formulation is limited to ZigBee architectures.

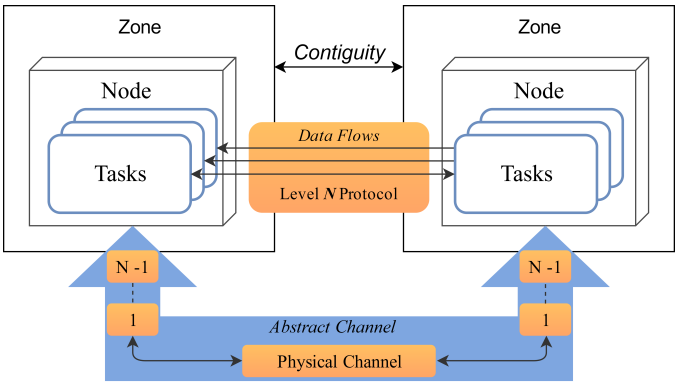


Fig. 2. Entities for the communication-aware specification.

Synthesis of communication protocols is another research topic related to this work. Automatic tools have been adopted to derive the actual implementation of protocols specified through finite state machines [34], [35], Petri Nets [36], trace models [37], and languages like LOTOS [38]. All these approaches focus on the behavioral aspect of communication without taking into account the design of the nodes. A general modeling framework for a global design flow could be useful to allow the joint exploration of HW, SW and Network design space dimensions as addressed in the proposed approach.

3 Communication-aware Design Flow

The creation of a specific design flow for distributed embedded systems requires the definition of new entities to formulate a design problem that accounts for communications; then, the traditional flow for embedded systems can be extended to solve the problem. Both aspects are described in the following text.

3.1 Network Specification

This section introduces the entities and relationships representing the communication aspects to be designed in distributed embedded systems. The proposed formal model is network-centric, *i.e.* it describes the characteristics which are related to communications while all the other details are omitted or highly abstracted. In fact, the objective is the description of communication requirements as an optimization problem whose solution leads to the network synthesis. Therefore, this formalization is neither a distributed model of computation, as Kahn Process Networks, nor a language for executable specification as SystemC.

Figure 2 shows a general picture of such entities and their relationships. It consists of tasks (Section 3.1.1), implementing the behavior of the distributed system, which are hosted inside network nodes (Section 3.1.3). The stream of data between tasks is represented by data-flows (Section 3.1.2). Tasks and corresponding nodes are deployed inside specific partitions of the environment named zones (Section 3.1.5). Zones are related together by contiguity which models the influence of the environment on communications, *i.e.*, obstacles, walls, distances (Section 3.1.6).

An abstract channel (Section 3.1.4) is established between nodes to convey the data-flows of the hosted tasks. The intention is to generalize the concept of physical channel with an abstraction which takes into account also the presence of higher protocol layers (we refer to the ISO/OSI representation). The highest

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

layer encompassed in the abstract channel depends on the type of protocols implemented in the conveyed data-flows. To understand the underlying idea of this generalization, let us consider two examples. In the first example, tasks implement a datalink protocol, *e.g.* IEEE 802.11ac, and therefore the abstract channel just represents the physical channel. In the second example, tasks implement a temperature control application through messages exchanged over a channel which is assumed to be reliable and byte-oriented. In this case, the abstract channel encompasses all the layers from physical channel up to TCP/IP.

In the following text, the network entities will be described in detail together with the relationships between them. Regarding notation, $\mathbb{R}_{\geq}$ is used to denote the non-negative real numbers, $\mathbb{R}_{[x,y]}$ identifies the real numbers between x and y, and and $\mathbb{B} := \{true, false\}$ for boolean values. Furthermore, the term *Time Unit* (TU) refers to a timing value of one second and *Space Unit* (SU) to a distance of one meter.

3.1.1 Tasks

A task represents a basic functionality of the whole application; it takes some data as input and provides some output. From the point of view of network synthesis the focus is not on the description of the functionality itself and its HW/SW implementation but rather on its computational and mobility requirements to decide its assignment to a given network node. A task $t = [s, m, z] \in \mathcal{T}$ is a triple defined as follows

- $s \in \mathbb{R}_{\geq}$ represents the task size, *i.e.*, the computational resources required to perform its activity;
- $m \in \mathbb{B}$ specifies whether the task should be placed on a mobile node;
- $z \in \mathcal{Z}$ specifies to which zone the task belongs.

Defining the appropriate task size is a designer's responsibility. It would be easy to generalize the description to the case where $t.s \in \mathbb{R}^k_{\geq}$, allowing to consider an array of k different types of resources.

3.1.2 Data-Flows

A data-flow (DF) represents the flow of messages between two tasks; output from the source task is delivered as input to the destination task. A data-flow $d = [st, dt, s, d, e] \in \mathcal{D}$ is characterized by the attributes

- $st, dt \in \mathcal{T}$ are the source and destination tasks;
- $s \in \mathbb{R}_{\geq}$ represents the data-flow size, *i.e.* bit-rate;
- $d \in \mathbb{R}_{\geq}$ indicates the maximum acceptable delay;
- $e \in \mathbb{R}_{\geq}$ specifies the maximum acceptable error rate.

Network synthesis is mainly driven by the communication requirements of the data-flows which affect the choice of channels and protocols between the nodes hosting the involved tasks.

3.1.3 Nodes

A node can be seen as a container of tasks. At the end of the whole design flow, nodes will be instances of HW platforms with CPUs and network interfaces and tasks will be implemented as either custom HW components or SW processes. From the point of view of network synthesis, the focus is on the resources made available by the node to host a number of tasks. A node $n = [s, k, e, te, ek, m] \in \mathcal{N}$ is a tuple whose attributes are as follows

- $s \in \mathbb{R}_{\geq}$ represents the node size, *i.e.*, the available computational resources;
- $k \in \mathbb{R}_{\geq}$ denotes the node economic cost;
- $e \in \mathbb{R}_{\geq}$ is the intrinsic energy consumption of the node without considering the executed tasks;
- $te \in \mathbb{R}_{\geq}$ determines the energy consumption of the tasks assigned to the node over a *TU* (each task t mapped into the node n consumes an amount of energy equal to $t.s$ times $n.te$);
- $ek \in \mathbb{R}_{\geq}$ relates the consumed energy with a specific cost based on the energy source (*e.g.*, batteries, solar panels, energy service company *etc.*);
- $m \in \mathbb{B}$ identifies if the node is mobile or static.

The network synthesis process assigns tasks to nodes. Tasks with the mobile attribute set to true must be placed on mobile nodes.

Regarding energy consumption, there are two contributions. The first one, denoted by e, is constant and independent of task operations while the second, denoted by te, accounts for the energy consumed to execute each task operation. Regarding economic cost, there is a constant contribution, denoted by k, to acquire the use of the node (*e.g.*, because of purchase or rent) and a variable contribution due to energy consumption. To compute this contribution, we introduced ek which describes the cost of each energy unit. This cost depends on energy source, *e.g.*, cost of batteries and their replacement or energy service company bill. This unit cost can be zero in case of energy harvesting (e.g., solar panels).

3.1.4 Abstract Channels

An Abstract Channel (AC) can be seen as a container of data-flows. It is an ideal medium connecting two or more nodes. Referring to the ISO/OSI model, it is defined as follows

Definition *Assuming that there is a data-flow implementing a level-N protocol, it is hosted by an AC which represents the physical channel and all the protocol entities up to level $N-1$.*

An abstract channel $ac = [e, de, k, ek, w, pp, s, dl, er] \in \mathcal{A}$ is a tuple characterized as follows

- $e \in \mathbb{R}_{\geq}$ is the intrinsic energy consumption of the channel without considering hosted data-flows;
- $de \in \mathbb{R}_{\geq}$ is the energy required to send a bit through the channel over a *TU* (each data-flow d deployed inside the channel c consumes an amount of energy equal to $d.s$ times $c.de$);
- $k \in \mathbb{R}_{\geq}$ specifies the economic cost of this communication architecture;
- $ek \in \mathbb{R}_{\geq}$ relates the consumed energy with a specific cost based on the energy source;
- $w \in \mathbb{B}$ specifies if the channel is wireless or wired;
- $pp \in \mathbb{B}$ specifies if the channel is point-to-point;
- $s \in \mathbb{R}_{\geq}$ specifies the channel size, *i.e.* its capacity;
- $dl \in \mathbb{R}_{\geq}$ specifies the maximum transmission delay of the channel;
- $er \in \mathbb{R}_{\geq}$ specifies the maximum error rate of the channel.

Data-flows between mobile tasks (hosted by mobile nodes) can be assigned only to wireless abstract channels. The last three attributes of the AC represent the *Quality of Service (QoS)* resulting from the presence of a given physical channel and all encompassed protocols. Similar attributes are present in the data-flow description; they represent the QoS *required* by the data-flow which should be *provided* by the hosting abstract channel. This is one of the driving rules of the network synthesis. Attribute $ac.pp$ distinguishes between point-to-point and multi-point

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

channels. It is worth noting that this information is orthogonal to wireless/wired attribute. In fact there are multi-point wired channels (*e.g.*, CAN bus) and point-to-point wireless channels (*e.g.*, Bluetooth connections and wireless bridges).

For economic cost and energy consumption, similar reasoning as for nodes applies since the proposed definition of Abstract Channel accounts for both the physical channel and possible intermediate systems (*e.g.*, switches and routers). Regarding energy consumption, there are two contributions. The first one, denoted by e, is constant and independent of communications while the second, denoted by de, accounts for the energy consumed to transmit data-flow bits. Regarding economic cost, there is a constant contribution, denoted by k, to acquire the use of the channel (*e.g.*, because of purchase or rent of the line and intermediate systems) and a variable contribution due to energy consumption. To compute this contribution, we introduced ek which describes the cost of each energy unit. This cost depends on energy source, *e.g.*, cost of batteries and their replacement or energy service company bill. This unit cost can be zero in case of energy harvesting (e.g., solar panels).

3.1.5 Zones

In distributed embedded applications tasks should be active in specific positions of the 3D space. In the mentioned example, temperature sensors and actuators are distributed in the various rooms of the building. Position of tasks is important for their assignment to nodes. Then nodes position is also important to determine the effect of obstacles and distance on communications between them. In general, we want to address properties like "between nodes n_i and n_j there is an obstacle". Information about precise 3D positioning may be not available and even not useful for a given application (for instance, a temperature sensor is required in each room but its position may be not so relevant). Therefore we propose to describe the position of tasks and nodes in the 3D space by partitioning it according to application needs (*e.g.*, rooms) and the presence of communication-relevant properties such as obstacles and distances. We denote by $\mathcal{Z}$ the set of Zones generated by this partition.

3.1.6 Contiguity Relationship

Contiguity relationship describes the relationship between zones from the communication perspective. We assume that nodes placed in the same zone are always able to communicate with the default quality of service of the involved abstract channel (see Section 3.1.4). If nodes are deployed into different zones, the quality of service might drop because of distance or obstacles. The level of degradation also depends on the type of abstract channel. Furthermore, in case of wired channels, the relationship between zones can be also used to capture the wiring cost. A contiguity element $cnt = [z_1, z_2, ac, c, dc] \in \mathcal{C}$ is a tuple whose attributes are characterized as follows

- $z_1, z_2 \in \mathcal{Z}$ are the involved zones;
- $ac \in \mathcal{A}$ is the abstract channel to which the contiguity applies;
- $c \in \mathbb{R}_{[0,1]}$ is the attenuation coefficient to compute the remaining level of QoS of the given abstract channel ac after crossing the border between the given zones;
- $dc \in \mathbb{R}_{\geq}$ represents the wiring cost to deploy the given channel between the given pair of zones; this attribute is relevant only for wired channels and takes into account both medium type and length.

3.2 Design Flow

White boxes in the right part of Figure 3 represent the traditional design flow of embedded systems. The starting point is the set of *Application Requirements* both functional and non-functional. A platform-independent *Functional Specification* is created starting from application requirements. Interacting components are expressed through languages like UML and C/C++ or through the use of tools like Matlab/Simulink/Stateflow (see Section 2.1). With reference to the entities defined in Section 3.1, a functional specification can be given as a set of Tasks exchanging information through Data-Flows.

This specification, together with a description of the target platform, is the subject of a Design Space Exploration (DSE) which maps Tasks onto HW and SW components of the target platform. The result is a platform-dependent description of the system, in which the HW blocks correspond to actual devices while the software is implemented and compiled for the target processors. Such flow is well suited for isolated embedded systems. However, in case of distributed applications made of many embedded systems it lacks a specific path devoted to the design of the communication infrastructure among them.

For this reason, this work proposes to extend the flow with new steps shown in light green on the left side. The new design path is quite symmetric *w.r.t.* the traditional one since it applies the same concepts to the communication aspects of the whole system. A *Communication-aware Problem Formulation* for the whole application is created by using information taken from the *Application Requirements* and the *Functional Specification*. Such information can be described with reference to the entities defined in Section 3.1.

The *Application Requirements* block provides:

- a description of the environment as a set of Zones and Contiguity relationships among them;
- a definition of the application as a set of Tasks and Data-Flows with Task-Zone assignments;
- an optimization objective function (e.g., energy minimization).

The *Functional Specification* allows to obtain the attributes of Tasks and Data-Flows. Data-Flow attributes represent communication constraints of the various data-flows of the distributed application, *e.g.*, their bit-rate as well as maximum acceptable delay and error rate.

The Communication-aware Problem Formulation describes a constrained optimization problem which links metrics to be optimized with constraints to be satisfied. This problem description, together with a description of available abstract channels and nodes (defined in Section 3.1), is the subject of DSE aiming at searching the optimal solutions. Similarly to how components are defined in electronic system design, this process is named *Network Synthesis*.

Definition *Network synthesis is a design process which starts from an optimization problem and finds a feasible solution which defines its communication infrastructure in terms of mapping of application Tasks onto network Nodes, their spatial displacement onto Zones, the type of channels and protocols among them, and the network topology.*

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

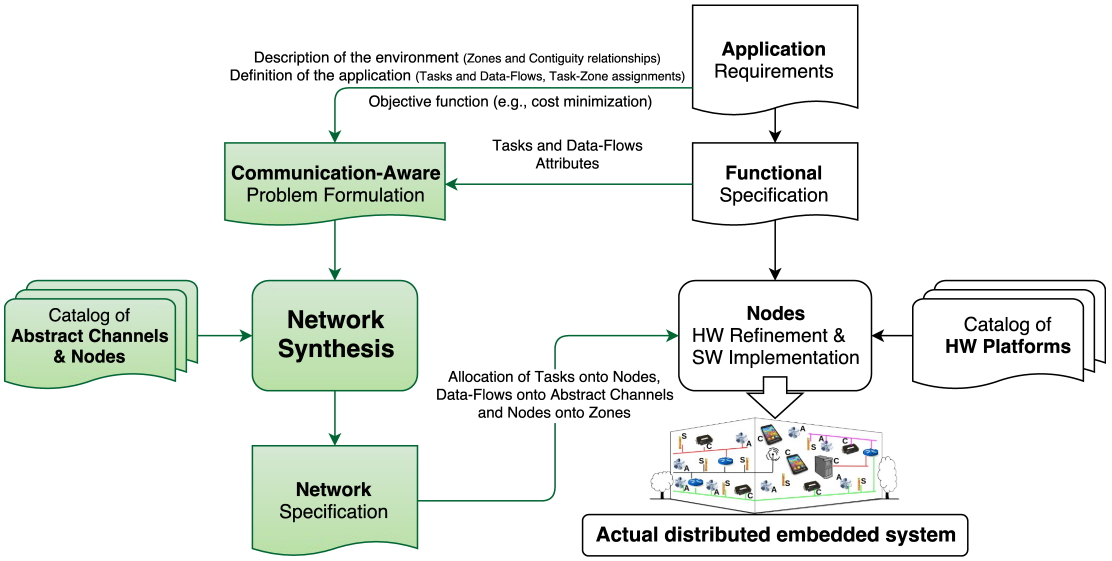


Fig. 3. Proposed design flow for distributed embedded systems: the new steps for network design (in light green on the left) are added symmetrically to the state-of-art design flow (in white on the right).

The final result is the *Network Specification* which contains important information for the design of each node of the network, *i.e.*, the list of functions assigned to it and the presence of new computation tasks to handle network protocols. For this reason, this description is used as input in the traditional DSE of each node as reported in the right part of Figure 3. The proposed flow has the following advantages which match with the properties of distributed embedded systems:

- Network features are decided before the design of HW and SW components; in this way the impact of communications can be taken into account in the early phase of the design process.
- The environment is taken into account during network design and therefore its impact is considered in the following design of HW and SW components.
- The proposed top-down approach for the design of the distributed embedded system matches with its nature of system-of-systems. In the context of network deployment, the traditional approach is bottom-up by making some implementation-specific assumptions based on designer's experience. For instance, the designer starts assuming to build an Ethernet network and then connects Ethernet nodes and switches without considering if other technologies may be more suitable. To the best of our knowledge, this is the first proposal that considers all available network architectures at the beginning of the flow.
- The decomposition of the application functionality into tasks and their allocation to nodes allows to distribute a single heavy function over multiple nodes and the process is driven by the optimization objective, *i.e.*, cost, reliability, and so on.

4 Network Synthesis

The network synthesis problem is the core of the previous design flow. It can be formulated as an optimization problem by using the entities defined in Section 3.1. Among several optimization techniques that can be used to solve this problem, a mixed-integer linear problem (MILP) is presented and solved.

4.1 Problem Formulation

Referring to Figure 3 and using the entities defined in Section 3.1 it is now possible to formulate the *network synthesis* problem.

The Application Requirements allow to obtain the set of tasks (denoted by $\mathcal{T}$), data-flows ($\mathcal{D}$), zones ($\mathcal{Z}$) and contiguity elements ($\mathcal{C}$). The Functional Specification allows to obtain the attributes of tasks and data-flows. All these information elements allow to build the Communication-aware Problem Formulation.

Network synthesis process is also fed by the set of nodes (denoted by $\mathcal{N}$) and abstract channels ($\mathcal{A}$) which represent the technological libraries for this design space exploration. For each type of node and abstract channel its name and attributes are specified.

Network synthesis can be formulated as an optimization problem in which the allocation of tasks onto nodes and data-flows onto abstract channels is driven by a set of constraints and metrics to be optimized. A possible way to formulate and solve such problem consists in describing it as a MILP problem. Independently of the formulation technique, there are some strong constraints that should be always considered:

- a non-mobile node cannot host a mobile task;
- a task with a given computational requirement cannot be hosted by a node which does not provide at least such resources;
- a data-flow with a given QoS requirement cannot be hosted by an abstract channel which does not provide at least such QoS;
- abstract channel types cannot be used between zone pairs whose contiguity brings to zero their QoS.

To model these general constraints, the following functions are defined and populated during a preprocessing phase:

- $\alpha_n(t)$, $t \in \mathcal{T}$ returns the set of *allowed nodes* to which the task t can be mapped;
- $\alpha_t(n)$, $n \in \mathcal{N}$ returns the set of *allowed tasks* which can be mapped into a node of type n;
- $\alpha_c(d)$, $d \in \mathcal{D}$ returns the set of *allowed channels* in which the data-flow d can be mapped. It can be further subdivided

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

into $\alpha_{wc}(d)$ and $\alpha_{cc}(d)$ that only considers respectively the allowed *wireless* and *wired* channels;
- $\alpha_c(z_1, z_2)$, $z_1, z_2 \in \mathcal{Z}$, $z_1 \neq z_2$ returns the set of *allowed channels* which can be used to connect two nodes deployed respectively in z_1 and z_2. Two sub-sets $\alpha_{wc}(z_1, z_2)$ and $\alpha_{cc}(z_1, z_2)$ are defined, respectively identifying the allowed *wireless* and *wired* channels;
- $\alpha_d(c)$, $c \in \mathcal{A}$ returns the set of *allowed data-flows* which can be mapped into a channel of type c;
- $cont(z_1, z_2, ac)$, $z_1, z_2 \in \mathcal{Z}$, $ac \in \mathcal{A}$, is a hash function that allows to efficiently retrieve the contiguity relationship and thus the corresponding conductivity c and the wiring cost dc.

The tasks connected to a data-flow and the zones in which they are placed is well-known from Application Requirements. Therefore, given an abstract channel c, the set $\alpha_d(c)$ does not contain data-flows whose tasks are placed in zones across which c has zero conductivity.

4.2 MILP Variables

In the following, all the variables used during the MILP formalization are presented and explained in detail. The first two sets of variables play a distinguished structural role, in that they imply the space of all the other variables.

- $N_{n,z}$, $n \in \mathcal{N}$, $z \in \mathcal{Z}$ for each node-type $n \in \mathcal{N}$ and zone $z \in \mathcal{Z}$, $N_{n,z}$ denotes how many nodes of node-type n are deployed in zone z.
- C_c, $c \in \mathcal{A}$ for each channel type $c \in \mathcal{A}$, C_c states how many channels of type c are activated by the solution.

Since we can not write MILPs with an infinite number of variables, we need to rely on the following two parameters which can be conveniently computed in a preprocessing phase.

- $\overline{N}_{n,z}$, $n \in \mathcal{N}$, $z \in \mathcal{Z}$ for each node-type $n \in \mathcal{N}$ and zone $z \in \mathcal{Z}$, parameter $\overline{N}_{n,z}$ provides an upper bound on the value of $N_{n,z}$. Before running our model we need to fix parameter $N_{n,z}$ to a natural value. We want this value to be as small as possible, since the number of variables allocated by our MILP grows polynomially in $N_{n,z}$. However, we should make sure that there exist optimal solutions in which $N_{n,z} \leq \overline{N}_{n,z}$, *i.e.*, $\overline{N}_{n,z}$ should be a valid upper bound.
- $\overline{C}_c$, $c \in \mathcal{A}$ for each channel type $c \in \mathcal{A}$, parameter $\overline{C}_c$ provides an upper bound on the value of C_c. This means that, as above, we should make sure that there exist optimal solutions in which $C_c \leq \overline{C}_c$, or that we are ready to anyhow limit our search for good solutions below this parameter. Again, we want $\overline{C}_c$ to be as small as possible since the number of variables allocated by our MILP grows polynomially in this parameter.

This work proposes to consider

$$\overline{N}_{n,z} := |\{t \in \alpha_t(n) | t.z = z\}| \tag{1}$$

$$\overline{C}_c := |\{d \in \alpha_d(c)\}| \tag{2}$$

Bound 1 indeed provides an upper bound to the number of type-node n in zone z based on the number of allowed tasks for node n in zone z. Bound 2 is also a valid upper bound since the upper-bound of a given channel c is equal to the number of allowed data-flow inside that channel.

The purpose of our first set of boolean variable x is to activate, in a well structured way, single instances of nodes of any given type. For each node $n \in \mathcal{N}$, $z \in \mathcal{Z}$ and $p \leq \overline{N}_{n,z}$, their intended value is as follows

$$x_{n,z,p} = \begin{cases} 1 & \text{if there are at least } p \text{ nodes of type } n \text{ allocated in zone } z, \\ 0 & \text{otherwise.} \end{cases} \quad \forall n \in \mathcal{N}, \quad \forall z \in \mathcal{Z}, \quad \forall p \leq \overline{N}_{n,z} \tag{3}$$

Analogously, the second set of variables y determines the number of allocated channels of any given type

$$y_{c,p} = \begin{cases} 1 & \text{if at least } p \text{ channels of type } c \text{ are allocated,} \\ 0 & \text{otherwise.} \end{cases} \quad \forall c \in \mathcal{A}, \quad \forall p \leq \overline{C}_c \tag{4}$$

Another issue is represented by the presence of point-to-point channels, which can only be deployed between two nodes. Such aspect has been formalized with the support of two variables. First, the tasks of a data-flow and a third task are related by a variable γ, whose formalization follows

$$\gamma_{d,t} = \begin{cases} 1 & \text{if tasks } d.st, \, d.dt, \text{ and } t \text{ are mapped into 3 different nodes,} \\ 0 & \text{otherwise.} \end{cases} \quad \forall d \in \mathcal{D}, \quad \forall t \in \mathcal{T}, \tag{5}$$

Variable ρ instead, is formalized as

$$\rho_{t_1,t_2} = \begin{cases} 1 & \text{if tasks } t_1 \text{ and } t_2 \text{ are mapped into different nodes,} \\ 0 & \text{otherwise.} \end{cases} \quad \forall t_1, t_2 \in \mathcal{T}, \quad t_1 \neq t_2 \tag{6}$$

The key aspects of the proposed formulation are the assignment of tasks to nodes and the deployment of data-flows into channels. Concerning the positioning of tasks inside nodes, a new boolean variable $w_{t,n,p}$ defined as

$$w_{t,n,p} = \begin{cases} 1 & \text{if task } t \text{ is associated with the } p\text{-th node of type } n \text{ in zone } t.z, \\ 0 & \text{otherwise.} \end{cases} \quad \forall t \in \mathcal{T}, \quad \forall n \in \alpha_n(t), \quad \forall p \leq \overline{N}_{n,z} \tag{7}$$

Deployment of data-flows inside channels is identified by a variable $h_{d,c,p}$ defined as

$$h_{d,c,p} = \begin{cases} 1 & \text{if the data-flow } d \text{ is placed in the } p\text{-th channel of type } c, \\ 0 & \text{otherwise.} \end{cases} \quad \forall d \in \mathcal{D}, \quad \forall c \in \alpha_c(d), \quad \forall p \leq \overline{C}_c \tag{8}$$

Variable q_{c,z_1,z_2} is statically solved before executing the optimization. It is initialized by checking if the conductance of the channel between the two zones is greater than zero. It is defined as follows

$$q_{c,z_1,z_2} = \begin{cases} 1 & \text{if a channel of type } c \text{ can connect nodes inside } z_1 \text{ and } z_2, \\ 0 & \text{otherwise.} \end{cases} \quad \forall z_1, z_2 \in \mathcal{Z}, \forall c \in \mathcal{A} \tag{9}$$

Finally, variable $j_{c,p}$ is introduced in order to keep track of the deployment cost for each instance of deployed channel. It is defined as

$$j_{c,p} \in \mathbb{R}_{\geq} \quad \forall c \in \mathcal{A}, \forall p \leq \overline{C}_c \tag{10}$$

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

4.3 MILP Objectives

Four metrics were considered to be of major importance within a distributed embedded system, and then subject to optimization. These metrics are: *Economic cost*, *Energy consumption*, *Transmission delay*, and *Error rate*. For all the metrics described above the optimization can be determined by a minimization function.

4.3.1 Economic Cost Minimization

Its objective is to minimize the total economic cost of the distributed embedded system, and it is defined as follows

$$\min \left[\begin{array}{l} \sum_{n}^{\mathcal{N}} \sum_{z}^{\mathcal{Z}} \sum_{p=1}^{\overline{N}_{n,z}} (x_{n,z,p} * (n.k + n.e * n.ek)) + \\ \sum_{c}^{\mathcal{A}} \sum_{p=1}^{\overline{C}_c} (j_{c,p} + y_{c,p} * (c.k + c.e * c.ek)) + \\ \sum_{t}^{\mathcal{T}} \sum_{n}^{\alpha_n(t)} \sum_{p=1}^{\overline{N}_{n,t.z}} (w_{t,n,p} * n.te * t.s * n.ek) + \\ \sum_{d}^{\mathcal{D}} \sum_{c}^{\alpha_c(d)} \sum_{p=1}^{\overline{C}_c} \frac{h_{d,c,p} * c.de * c.ek * d.s}{cont(d.st.z, d.dt.z, c).c} \end{array} \right] \tag{11}$$

The first two sums of the metric considers the base cost of deployed nodes and channels plus their energetic cost. Then, the third sum considers the supplementary cost for wired channels whenever they are placed between different zones. The last two sums concerns the economic cost deriving from the consumed energy. For tasks, this is done by multiplying the total energy consumed by a task deployed inside a node for the specific energy cost for that node. Similarly, the last sum considers the energetic cost of data-flows by multiplying the total amount of energy consumed by a deployed channel for the price of the energy for that particular channel.

4.3.2 Energy Consumption Minimization

The second optimization objective is to minimize the total energy consumption of the distributed embedded system, and is defined as follows

$$\min \left[\begin{array}{l} \sum_{n}^{\mathcal{N}} \sum_{z}^{\mathcal{Z}} \sum_{p=1}^{\overline{N}_{n,z}} (x_{n,z,p} * n.e) + \\ \sum_{c}^{\mathcal{A}} \sum_{p=1}^{\overline{C}_c} (y_{c,p} * c.e) + \\ \sum_{t}^{\mathcal{T}} \sum_{n}^{\alpha_n(t)} \sum_{p=1}^{\overline{N}_{n,t.z}} (w_{t,n,p} * n.te * t.s) + \\ \sum_{d}^{\mathcal{D}} \sum_{c}^{\alpha_c(d)} \sum_{p=1}^{\overline{C}_c} \frac{h_{d,c,p} * c.de * d.s}{cont(d.st.z, d.dt.z, c).c} \end{array} \right] \tag{12}$$

The first two sums of the metric consider the energy consumed by deployed nodes and channels. The third sum takes into account the task's resource requirements and multiplies it for the coefficient used to calculate contribution of each task to the node energy consumption (*i.e.*, attribute $n.te$). The last sum multiplies the size of dataflows for the contribution to the energy consumption of channels where they are deployed (*i.e.*, attribute $n.de$).

4.3.3 Transmission Delay Minimization

Its purpose is to minimize the total transmission delay of the distributed embedded system. Follows its definition

$$\min \left[\sum_{d}^{\mathcal{D}} \sum_{c}^{\alpha_c(d)} \sum_{p=1}^{\overline{C}_c} \frac{h_{d,c,p} * c.dl}{cont(d.st.z, d.dt.z, c).c} \right] \tag{13}$$

The above metric sums the transmission delay of channels where dataflows are deployed, enhanced by the effects of the border between the involved zones on the communication quality. This metric considers the delay for each dataflow and not only once for each deployed channel.

4.3.4 Error Rate Minimization

The optimization objective is to minimize the total error rate of the distributed embedded system. The function has the same structure as for the transmission delay minimization but, instead of summing the channel delay, its error rate value is used. Follows its definition

$$\min \left[\sum_{d}^{\mathcal{D}} \sum_{c}^{\alpha_c(d)} \sum_{p=1}^{\overline{C}_c} \frac{h_{d,c,p} * c.er}{cont(d.st.z, d.dt.z, c).c} \right] \tag{14}$$

4.4 MILP Constraints

Constraints on the Number of Instantiated Components

The first group of constraints activates in accordance to the number of nodes and channels as well as defining the values of the upper-bounds of such components. More in details constraints C.1 and C.2 concerns the nodes. For all $n \in \mathcal{N}$ and $z \in \mathcal{Z}$

$$N_{n,z} = \sum_{p=1}^{\overline{N}_{n,z}} x_{n,z,p} \quad \forall n \in \mathcal{N},\ \forall z \in \mathcal{Z} \tag{C.1}$$

$$N_{n,z} \geq p * x_{n,z,p} \quad \forall n \in \mathcal{N},\ \forall z \in \mathcal{Z},\ \forall p \leq \overline{N}_{n,z} \tag{C.2}$$

Moreover, the second set of constraints C.3 and C.4, concerns the channels.

$$C_c = \sum_{p=1}^{\overline{C}_c} y_{c,p} \quad \forall c \in \mathcal{A} \tag{C.3}$$

$$C_c \geq p * y_{c,p} \quad \forall c \in \mathcal{A},\ \forall p \leq \overline{C}_c \tag{C.4}$$

Constraints on the Existence of Used Components

Constraints C.5 and C.6 ensure that nodes and channels are instantiated whenever tasks and data-flows use them.

$$w_{t,n,p} \leq x_{n,t.z,p} \quad \forall t \in \mathcal{T},\ \forall n \in \alpha_n(t),\ \forall p \leq \overline{N}_{n,t.z} \tag{C.5}$$

$$h_{d,c,p} \leq y_{c,p} \quad \forall d \in \mathcal{D},\ \forall c \in \alpha_c(d),\ \forall p \leq \overline{C}_c \tag{C.6}$$

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

Constraints C.7 and C.8 instead ensure that only the nodes and channels which are necessary are activated.

$$x_{n,z,p} \leq \sum_{t}^{(\alpha_t(n) \wedge t.z = z)} w_{t,n,p} \quad \forall n \in \mathcal{N},\ \forall z \in \mathcal{Z},\ \forall p \leq \overline{N}_{n,z} \tag{C.7}$$

$$y_{c,p} \leq \sum_{d}^{\alpha_d(c)} h_{d,c,p} \quad \forall c \in \mathcal{A},\ \forall p \leq \overline{C}_c \tag{C.8}$$

Constraints on Components Capacity

The assignment of tasks to nodes has to be compliant with the size (*i.e.*, resources) of each involved node. Constraint C.9, ensures that the total amount of resources required by tasks inside a given node is at most the size of the node.

$$\sum_{t}^{(\alpha_t(n) \wedge t.z = z)} t.s * w_{t,n,p} \leq n.s \quad \forall n \in \mathcal{N},\ \forall z \in \mathcal{Z},\ \forall p \leq \overline{N}_{n,z} \tag{C.9}$$

Constraint C.10, ensures that the total amount of bit-rate used by data-flows mapped into a given channel is at most the size (*i.e.*, capacity) of the channel. Furthermore, the effect of the environment has to be taken into consideration.

$$\sum_{d}^{\alpha_d(c)} \frac{d.s * h_{d,c,p}}{cont(d.st.z, d.dt.z, c).c} \leq c.s \quad \forall c \in \mathcal{A},\ \forall p \leq \overline{C}_c \tag{C.10}$$

Constraints on Tasks and Data-Flows Assignment

Tasks and data-flows are unique entities, specific of the application functionality, thus they must be assigned only once to nodes and channels, respectively. For what concerns tasks, Constraint C.11, ensures that they are assigned to a node only once.

$$\sum_{n}^{\alpha_n(t)} \sum_{p=1}^{\overline{N}_{n,t.z}} w_{t,n,p} = 1 \quad \forall t \in \mathcal{T} \tag{C.11}$$

However, for what concerns data-flows, their placement depends on whether the tasks which they connect reside in the same node or not. In the former case, formalized in Constraint C.12, the data-flow is not necessarily assigned to a channel and in fact its placement depends on variable ρ. For data-flows which instead have tasks which reside in different zones their placement inside a channel is necessary and ensured by Constraint C.13.

$$\sum_{c}^{\alpha_c(d)} \sum_{p=1}^{\overline{C}_c} h_{d,c,p} = \rho_{d.st,d.dt} \quad \forall d \in \mathcal{D},\ d.st.z = d.dt.z \tag{C.12}$$

$$\sum_{c}^{\alpha_c(d)} \sum_{p=1}^{\overline{C}_c} h_{d,c,p} = 1 \quad \forall d \in \mathcal{D},\ d.st.z \neq d.dt.z \tag{C.13}$$

Constraints on Point-to-Point Channels

The next sets of constraints concern point-to-point channels, which have to abide a more tightening rule. Each of them can connect no more than a pair of nodes. Constraint C.14 ensures that variable ρ is correctly set whenever two tasks are mapped into different nodes. Constraint C.15 instead, sets ρ to constant 1 when the pair of tasks resides in different zones.

$$\rho_{t,t'} \geq (w_{t,n,p} + w_{t',n',p'} - 1) \quad \begin{array}{lll} \forall t, t' \in \mathcal{T}, & t.z = t'.z, & t \neq t', \\ \forall n \in \alpha_n(t), & \forall p \in \overline{N}_{n,t.z}, & \\ \forall n' \in \alpha_n(t'), & \forall p' \in \overline{N}_{n',t'.z}, & \\ (n \neq n') \vee (p \neq p') & & \end{array} \tag{C.14}$$

$$\rho_{t,t'} = 1 \quad \forall t, t' \in \mathcal{T},\ t.z \neq t'.z, t \neq t' \tag{C.15}$$

Constraint C.16 has to keep track of all the data-flows which have a node in common. This is necessary since whenever a data-flow mapped into a point-to-point channel share a node with another data-flow, the source and destination task of the latter have to be mapped into the same node *w.r.t.* the tasks of the former.

$$\begin{array}{c} \gamma_{d,d'.st} \leq 2 - h_{d,c,p} + h_{d',c,p} \\ \text{with } (d.st \neq d'.st) \wedge (d.dt \neq d'.st) \\ \gamma_{d,d'.dt} \leq 2 - h_{d,c,p} + h_{d',c,p} \\ \text{with } (d.st \neq d'.dt) \wedge (d.dt \neq d'.dt) \\ \forall c \in \mathcal{A},\ \forall p \leq \overline{C}_c,\ \forall d, d' \in \alpha_d(c), \\ c.pp = true, d \neq d' \end{array} \tag{C.16}$$

Constraint C.17 ensures that, if the tasks of a data-flow d and a third task t are placed all in different nodes (C.14, C.15 and C.17), then data-flow d and the one connected to task t must be mapped into different channels (C.16).

$$\begin{array}{c} \gamma_{d,t} = 0 \\ \text{with } (d.st = t) \vee (d.dt = t) \\ \gamma_{d,t} = 1 \\ \text{with } (d.st.z \neq t.z) \wedge (d.dt.z \neq t.z) \wedge (d.st.z \neq d.st.z) \\ \gamma_{d,t} \geq \rho_{t,d.st} + \rho_{t,d.dt} + \rho_{d.st,d.dt} - 2 \\ \text{with } (d.st.z = t.z) \vee (d.dt.z = t.z) \vee (d.st.z = d.st.z) \\ \forall d \in \mathcal{D},\ \forall t \in \mathcal{T} \end{array} \tag{C.17}$$

Constraints on Wireless Channels

Constraint C.18 ensures that whenever two data-flows are placed inside the same wireless channel, all tasks of the data-flows are able to communicate with each other. Thus, the conductivity with the given channel between the four combinations of the zones in which the tasks reside is greater than zero.

$$\begin{array}{r} h_{d,c,p} + h_{d',c,p} \leq 1 + (q_{c,d.st.z,d'.st.z} * \\ q_{c,d.st.z,d'.dt.z} * \\ q_{c,d.dt.z,d'.st.z} * \\ q_{c,d.dt.z,d'.dt.z}) \\ \forall c \in \mathcal{A},\ c.w = true,\ \forall p \leq \overline{C}_c,\ \forall d, d' \in \alpha_d(c),\ d \neq d' \end{array} \tag{C.18}$$

Constraints on deployment cost of wired channels

Finally, Constraint C.19 poses an lower-bound on variable $j \in \mathbb{R}_{\geq}$. Such lower-bound is equal to the *highest* deployment cost for those channels which are placed between two zones. It can be appreciated that such constraint is defined only for those pairs

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

of zone interested by data-flows. The upper-bound on variable j is intrinsically ensured by the objective function, which aims at minimizing the variables.

$$j_{c,p} \geq h_{d,c,p} * cont(d.st.z, d.dt.z, c).dc$$
$$\forall c \in \mathcal{A},\ c.w = false,\ \forall p \leq \overline{C}_c, \quad \forall d \in \alpha_d(c),\ d.st.z \neq d.dt.z \tag{C.19}$$

5 Complexity and Scalability

One should be aware that the network synthesis model proposed and solved in this paper is strongly NP-hard even in the following two very extreme special cases:

- $|\mathcal{N}| = 1$, $|\mathcal{Z}| = |\mathcal{A}| = |\mathcal{D}| = 0$, $|\mathcal{T}| \in \mathbb{N}$. Without loss of generality, both size and cost of each node instance are 1. Every node instance can be regarded as a *bin* that we can open or not in order to accommodate a set of tasks, each one representing an *item* of the same size.
- $|\mathcal{A}| = |\mathcal{N}| = 1$, $|\mathcal{T}| = 2$, $|\mathcal{Z}| = 1$, $|\mathcal{D}| \in \mathbb{N}$. Without loss of generality, both size and cost of each channel instance are 1. Every channel instance can be regarded as a *bin* that we can open or not in order to accommodate a set of data-flows, each one representing an *item* of the same size.

As a consequence, unless P=NP, every general algorithm solving our model will exhibit a running time which is exponential *both* in $|\mathcal{T}|$ *and* in $|\mathcal{D}|$. Nonetheless, the MILP solution here offered works remarkably well in practice. In our tests, we never gave up the ambition of obtaining a proof of optimality. By necessity, there are surely limits to the scalability of this approach, but these should be regarded more as limits to the ambition of solving the general network design model to optimality rather than limits in the tested solution. The model and solution we have designed has successfully modeled and solved to optimality the situations form the applications we had in mind. In fact, its efficiency allows to solve instances whose share size was beyond our original commitment.

To get a rough idea on the size of the instances that can be addressed, consider first the asymptotic growth of the number of variables that get allocated by our MILP formulation. Table 1 reports on the growth for each category of variables introduced.

TABLE 1
Number of allocated variables.

x	$O(\|\mathcal{N}\|\|\mathcal{T}\|\|\mathcal{Z}\|)$
y	$O(\|\mathcal{A}\|\|\mathcal{D}\|)$
γ	$O(\|\mathcal{D}\|\|\mathcal{T}\|)$
ρ	$O(\|\mathcal{T}\|^2)$
w	$O(\|\mathcal{N}\|\|\mathcal{T}\|^2)$
h	$O(\|\mathcal{A}\|\|\mathcal{D}\|^2)$
q	$O(\|\mathcal{A}\|\|\mathcal{Z}\|^2)$
j	$O(\|\mathcal{A}\|\|\mathcal{D}\|)$

Those reported in the table are only worst case upper bounds as quite fewer variables get actually allocated in many instances; it is however easy to propose natural instance families meeting these bounds. Since allocating a variable takes $O(1)$ time and space, then the phase where the variables get introduced in the model one by one, through calls to the competent functions of the Gurobi dynamic library interface, takes

$$O(\max\{|\mathcal{N}||\mathcal{T}||\mathcal{Z}|, |\mathcal{D}||\mathcal{T}|, |\mathcal{N}||\mathcal{T}|^2|\mathcal{A}||\mathcal{D}|^2|\mathcal{A}||\mathcal{Z}|^2\})$$

time and space. Since inserting a constraint takes $O(1)$ time and memory for each one of its non-zero coefficients, the upper bound on the constraints specification phase can be similarly drawn from Table 2.

TABLE 2
Number of defined constraints.

C.1	$O(\|\mathcal{N}\|\|\mathcal{Z}\|)$
C.2	$O(\|\mathcal{N}\|\|\mathcal{Z}\|\|\mathcal{T}\|)$
C.3	$O(\|\mathcal{A}\|)$
C.4	$O(\|\mathcal{A}\|\|\mathcal{D}\|)$
C.5	$O(\|\mathcal{N}\|\|\mathcal{T}\|^2)$
C.6	$O(\|\mathcal{A}\|\|\mathcal{D}\|^2)$
C.7	$O(\|\mathcal{N}\|\|\mathcal{Z}\|\|\mathcal{T}\|)$
C.8	$O(\|\mathcal{A}\|\|\mathcal{D}\|)$
C.9	$O(\|\mathcal{N}\|\|\mathcal{Z}\|\|\mathcal{T}\|)$
C.10	$O(\|\mathcal{A}\|\|\mathcal{D}\|)$
C.11	$O(\|\mathcal{T}\|)$
C.12	$O(\|\mathcal{A}\|\|\mathcal{D}\|^2)$
C.13	$O(\|\mathcal{A}\|\|\mathcal{D}\|^2)$
C.14	$O(\|\mathcal{N}\|^2\|\mathcal{T}\|^3)$
C.15	$O(\|\mathcal{T}\|^2)$
C.16	$O(\|\mathcal{A}\|\|\mathcal{D}\|^3)$
C.17	$O(\|\mathcal{D}\|\|\mathcal{T}\|)$
C.18	$O(\|\mathcal{A}\|\|\mathcal{D}\|^3)$
C.19	$O(\|\mathcal{A}\|\|\mathcal{D}\|^2)$

The max of these bounds works as an upper bound only for the model set up phase, whereas the true optimization phase managed by Gurobi requires further memory and may easily take exponential time. However, based on experiments and talking with reference to a desktop architecture, we are confident that these bounds may offer a rather good prediction on the ultimate performance of our code when considering to apply our implementation of the model, as it is, to other settings. More precisely, we are confident that these bounds may offer you a rather good prediction on the ultimate performance of our code over the limited range where the predicted memory consumption for the only allocation phase is not prohibitive.

6 Experimental Results

In this section two case studies are presented with the aim of showing the expressiveness of the proposed design flow and the computational demand of the optimization process. Network synthesis has been performed by using Gurobi 7.5.1 tool with Python 2.7.12 front-end on a 64-bit machine running Ubuntu 16.04 LTS; the machine features an Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz with 16 GB memory.

6.1 Case Study 1

The first case study concerns the implementation of a distributed building automation application spanning over two adjacent buildings. The scenario is depicted in Figure 4. It consists of different kinds of tasks (*i.e.*, controllers, routers, sensors and actuators), deployed inside rooms delimited by thick walls, and exchanging a series of data-flows shown as red lines. Each building hosts a total of twenty-four tasks: a central controller, two routers and twenty-one sensors/actuators for room monitoring and regulation. Tasks are distributed over ten zones which comprise a control room, two technical closets for routers and seven offices. Each office contains two sensors and one actuator which have access to the central controller through an adjacent router. For this scenario, the technological libraries of available nodes and channels are

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

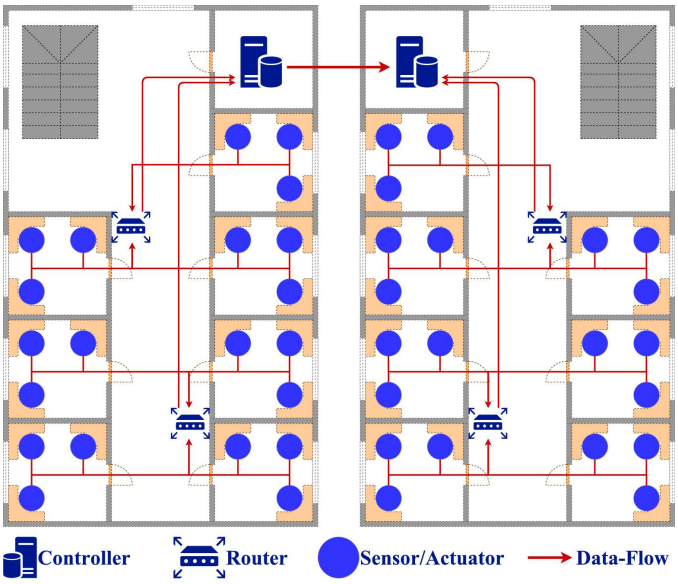


Fig. 4. Network topology with various network architectures connected through routers.

TABLE 3
Catalog of available nodes for Case Study 1.

Label	s	k	e	te	ek	m
Development Board Type 1	32	5	5	1	0.10	*false*
Development Board Type 2	84	18	7	2	0.15	*false*
Development Board Type 3	64	22	8	2	0.30	*true*
Development Board Type 4	128	98	12	5	0.41	*true*
Development Board Type 5	256	128	15	6	0.33	*true*
Development Board Type 6	512	512	30	12	1.20	*false*

TABLE 4
Catalog of available abstract channels.

Label	s	k	e	de	ek	dl	er	w	pp
Bluetooth 4.0	24	9	1	1	0.16	12	10	*true*	*true*
IEEE 802.11ac	7000	34	3	2	0.30	8	7	*true*	*false*
IEEE 802.11ad	7400	79	7	4	0.28	3	4	*true*	*false*
Ethernet	200000	320	18	2	0.21	3	1	*false*	*true*
Fiber	273000	367	14	1	0.12	1	3	*false*	*true*

reported in Table 3 and Table 4, respectively. The tables use the attribute symbols defined in Section 3.1.

As explained in Section 3.1.4, an Abstract Channel encompasses the physical layer and all the required upper layers according to application scenario: in our case, the abstract channels in Table 4 include also TCP/IP. The presence of various network architectures allows to select the most appropriate one for each interconnection between nodes. Considering the environment depicted in Figure 4, the data-flow between the controllers has to cross a pair of thick walls which may hinder the use of wireless LAN. Such effect is represented as a lower value of conductance between the corresponding zones (*i.e.*, control rooms) for the wireless channels as reported in boldface in Table 5. Therefore, the optimization process will select a wired architecture even if it leads to higher cost for cabling which is computed by summing up dc attribute values for the given zone pairs and the chosen wired channel type (reported in boldface in Table 5).

TABLE 5
Example of contiguity values for Case Study 1.

z_1	z_2	ac	c	dc
Office 1	Technical closet 1	Bluetooth 4.0	0.45	0
		IEEE 802.11ac	0.58	0
		IEEE 802.11ad	0.97	0
		Ethernet	0.00	0
		Fiber	0.00	0
Technical closet 1	Control room 1	Bluetooth 4.0	0.12	0
		IEEE 802.11ac	0.46	0
		IEEE 802.11ad	0.84	0
		Ethernet	0.69	645
		Fiber	0.95	1,252
Control room 1	Control room 2	Bluetooth 4.0	**0.07**	0
		IEEE 802.11ac	**0.12**	0
		IEEE 802.11ad	**0.21**	0
		Ethernet	0.78	**824**
		Fiber	1.00	**1,486**

TABLE 6
Case Study 1: performance and results of the network synthesis as a function of the optimization objective.

Minimization Objective	CPU Time (s)	Economic Cost ($)	Energy Consumption (J)	Delay (s)	Error Rate (%)
Econ. Cost	11.38	**38,903**	56,306	543.96	47.0
Energy	5.07	41,762	**48,330**	514.00	45.1
Delay	3.89	68,312	86,604	**283.35**	19.8
Error	3.97	67,760	93,809	287.84	**19.6**

In this example, the values in Tables 3, 4, and 5 have been obtained by performing a relative comparison between different technologies with the unique purpose of testing the optimization engine. More accurate values can be found in HW datasheets for nodes and actual benchmarks for network architectures [39]. Contiguity relationship should be evaluated by the designer for each specific scenario, *e.g.*, by using well-known tools for WiFi deployment [40].

Vice versa, for other zones of the same scenario wireless communications will be preferred for their lower cost. For instance, with energy minimization, we use 2 Bluetooth links, 7 IEEE 802.11ac links, 12 IEEE 802.11ad links, 10 Ethernet links, and 5 fiber links. To the best of our knowledge, this is the first network synthesis approach that allows mixing different network architectures.

Table 6 shows the statistics of the synthesized networks with the four different optimization objectives presented in Section 4.3. The table reports the CPU time spent to find the solution; it depends on the optimization target but anyway it is acceptable.

The composition of the synthesized network depends on the optimization metric. For instance, for cost minimization, the optimizer chooses 12 nodes of Type 2, 17 nodes of Type 3, and 9 nodes of Type 4, whereas for energy minimization, it chooses 19 nodes of Type 2, 17 nodes of Type 3, and 9 nodes of Type 4. Furthermore, application tasks can be grouped in different ways according to task-node assignment which is determined by the optimization process. For instance, let us consider three sensor/actuator tasks, denoted as $T1$, $T2$, and $T3$, belonging to the same zone (*i.e.*,

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

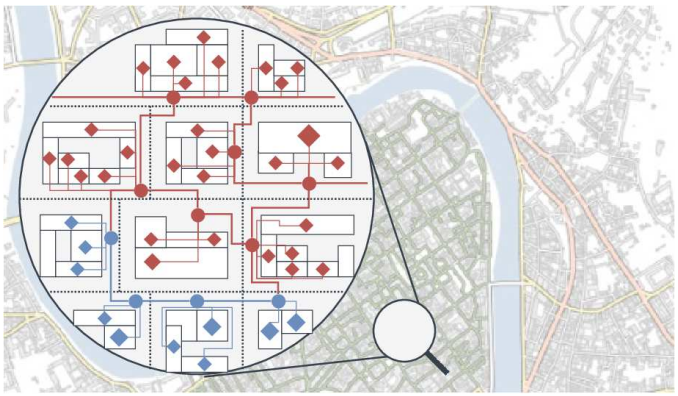


Fig. 5. Wide urban area test case: the tasks and data-flows colored in blue represent a preexisting network while red elements have been added during network synthesis.

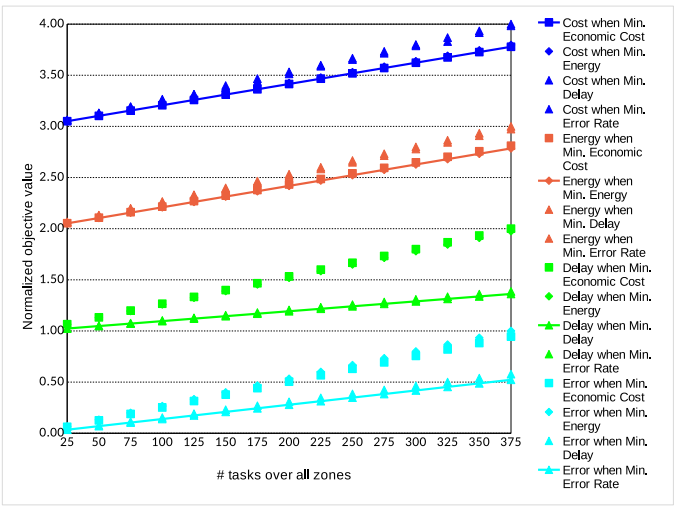


Fig. 6. Value of the objective function as a function of number of tasks over all zones and the optimization objective.

room). With economic cost minimization, all the tasks are placed inside the same Type 4 node whereas with delay minimization, two of them are placed inside inside distinct nodes of Type 3 while the third is placed inside a node of Type 4.

In summary, this Case Study shows that:

- different network architectures can be mixed in the synthesis process;
- a single heavy application can be distributed over multiple nodes of a distributed embedded system;
- the optimization process can distribute tasks in different ways over the network according to the optimization objective.

6.2 Case Study 2

The second case study concerns the implementation of a smart city application. For example, energy efficiency is a well-known design problem in this context [41]. The description of the environment for the proposed case study is given by the cartography in Figure 5. The area is subdivided into zones delineated by dotted lines. Each zone contains a task named *distributor* (represented by a circle) and a variable number of *user* tasks, all connected through data-flows. In such a huge public context, it is quite common to exploit a pre-existing network and add new pieces of infrastructure. This fact gives us the opportunity to show how the proposed synthesis flow is able to handle this kind of constraint. By referring to Figure 5, the tasks and data-flows colored in blue represent a pre-existing network, *i.e.*, they have already been placed inside nodes and channels, respectively. Vice versa, the red tasks and data-flows are assigned to nodes and channels by the network synthesis process. The catalogs of nodes and channels are shown in Table 7 and Table 4, respectively. The contiguity values between zones are mainly dependent on their distance.

TABLE 7
Catalog of available nodes for Case Study 2.

Label	*s*	*k*	*e*	*te*	*ek*	*m*
Development Board Type 1	64	10	2	1	0.05	*false*
Development Board Type 2	98	24	4	2	0.15	*true*
Development Board Type 3	128	64	8	4	0.40	*true*
Development Board Type 4	256	128	14	7	0.32	*true*
Development Board Type 5	512	378	20	10	0.60	*false*

This case study aims at evaluating the scalability of the approach as a function of scenario size, *i.e.*, number of zones, tasks, and data-flows. For this purpose, we automatically generated instances with increased size by using two different approaches.

6.2.1 Scalability over zones

The first scalability test regards the creation of large scenarios by increasing the number of zones, while the number of tasks per zone remains quite small. The generation of instances is based on the following rules:

- zones are arranged as a chain and a new instance is automatically generated by increasing the chain;
- each zone has a set of contiguity values defined only for the precedent zone and subsequent zone unless it is the first or the last zone of the chain;
- each zone contains four user tasks and one distributor;
- in each zone there are four data-flows connecting each user task with the distributor of the zone;
- the distributor of a zone is connected to the distributor of the precedent zone and the one of the subsequent zone unless it belongs to the first or last zone of the chain;
- attributes of tasks and data-flows are constant and their value depends on the role of the task, *i.e.*, distributor or user, and on the type of connection, *i.e.*, between distributors or between user and distributor.

Figure 6 shows the value of the objective function as a function of the size of the input instance (number of tasks). Even if values have been normalized to fit in the same plot, each of them is the minimum when the corresponding objective function is used to drive the optimization. Therefore, we can conclude that the behavior of the synthesizer is consistent over a large set of problem instances.

Figure 7 shows the total optimization time for the synthesizer as a function of the size of the input instance (number of tasks over all zones) for all optimization targets. Time values have been computed by using Python `time.clock()` function which takes into account the effort spent by the CPU in each thread of the process [1] thus avoiding artifacts due to the current load of

1. https://www.pythoncentral.io/measure-time-in-python-time-time-vs-time-clock

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

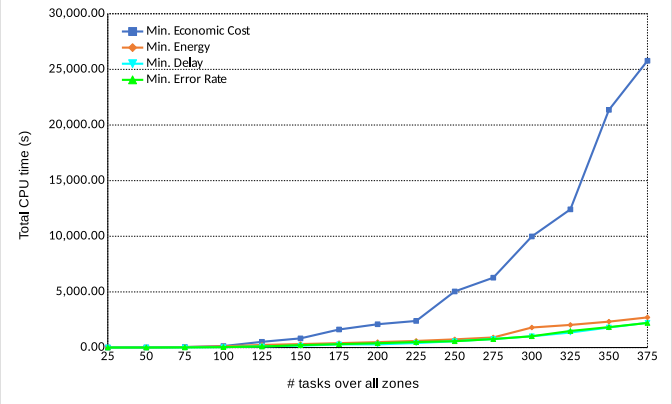


Fig. 7. Total CPU time for Case Study 2 as a function of number of tasks over all zones.

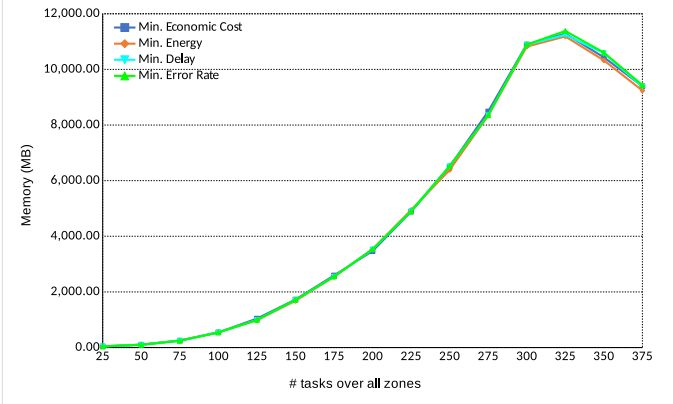


Fig. 9. Memory usage for Case Study 2 as a function of number of tasks over all zones.

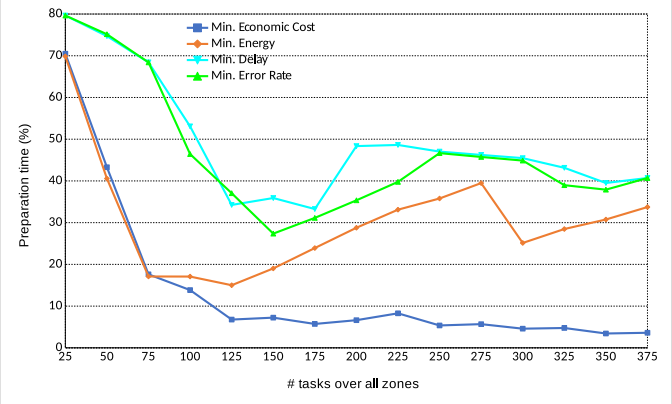


Fig. 8. Percentage of CPU time devoted to pre-optimization activities for Case Study 2 as a function of number of tasks over all zones.

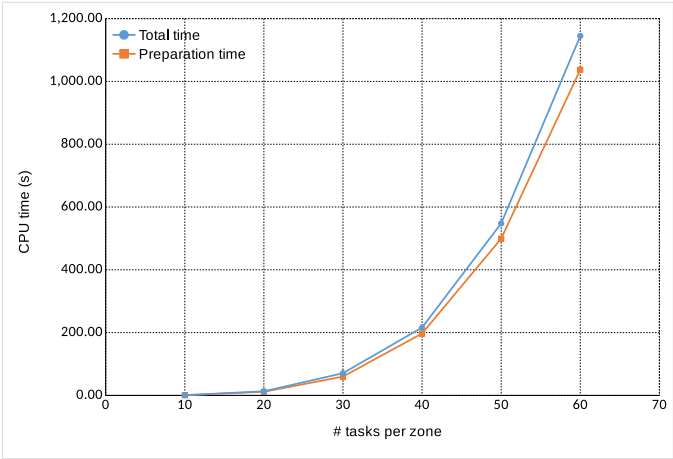


Fig. 10. CPU time for Case Study 2 as a function of the number of tasks per zone.

the workstation and hyper-threading techniques. The graph trend of the objective functions in Figure 7 can be directly related to the algorithmic complexity and the number of involved variables of the objective functions reported in Section 4.3. Even if the economic cost and energy consumption minimization functions have the same number of loops, the former has a greater number of involved variables. As such, the economic cost minimization requires more effort than the other functions as shown in the graph.

As described in Section 5, the synthesis process consists of some steps which prepare the proper optimization phase, *i.e.*, parsing of the instance description file, generation of variables, and generation of constraints. It is worth analyzing how CPU time is spent in this optimization flow. Figure 8 shows the percentage of the total optimization time (reported in Figure 7) devoted to pre-optimization activities. For economic cost minimization, the real optimization phase is mainly predominant over preparation but for simpler objective functions and small instances preparation time can be larger than time spent to search the optimal solution. To complete the scalability analysis, Figure 9 shows the corresponding memory usage.

6.2.2 *Scalability over tasks*

The second scalability test regards the creation of large scenarios by increasing the number of tasks in the same zone. The generation of instances is based on the following rules:

- only one large zone is considered;
- no contiguity relationships are set;
- each zone contains an increasing number user tasks and one distributor;
- in each zone there are as many data-flows as user tasks since they connect each user task with the distributor of the zone;
- attributes of tasks and data-flows are constant and their value depends on the role of the task, *i.e.*, distributor or user, and on the type of connection, *i.e.*, between distributors or between user and distributor.

Figure 10 shows the total optimization time for the synthesizer as a function of the number of tasks per zone. We only show economic cost minimization since the previous analysis proved it to be the most computationally intensive target. Figure 11 shows the corresponding memory usage.

7 Conclusions

This work focused on the peculiarities of distributed embedded systems and proposed an extended design flow to address them. Assuming that highly optimized nodes are desirable, the network infrastructure should be decided before designing HW and implementing SW thus leading to the concept of network synthesis. A communication-aware formalization was proposed to specify constraints and optimization metrics. Network synthesis was formalized as an optimization problem using mixed-integer

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

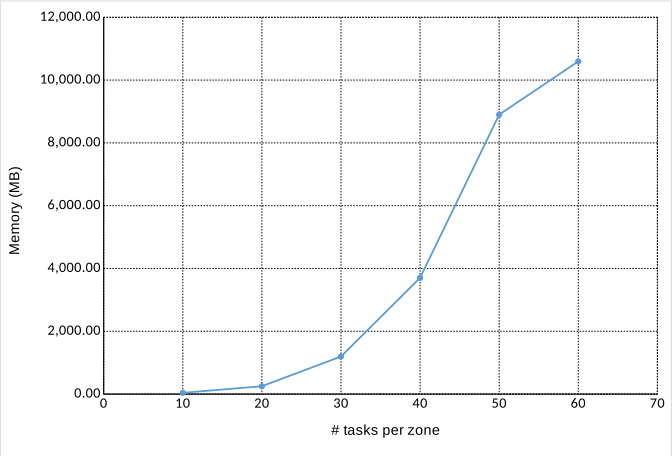


Fig. 11. Memory usage for Case Study 2 as a function of the number of tasks per zone.

linear programming. We defined the following formal entities: tasks, data-flows, nodes, abstract channels, and zones. The last two entities are particularly innovative. The Abstract Channel generalizes the concept of network architecture (*i.e.*, physical channels and protocols) so that the final solution can combine different types of network architectures. The Zone generalizes the concept of physical location adapting location accuracy to the requirements of the application and focusing on the impact of node placement on communications and cost. The framework was applied to real case studies with the aim to show the advancement with respect to state of the art. The first case study shows the possibility to create network infrastructures containing different network architectures according to users' needs and environmental constraints. The second one highlights the possibility to synthesize a network by adding components to an existing infrastructure which is a common problem in real life scenarios. Future work aims at investigating the scalability issues of the MILP approach and proposing communication-aware heuristics to address very large problems.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. of the Workshop on Mobile Cloud Computing*, 2012, pp. 13–16.

[2] T. Savolainen, J. Soininen, and B. Silverajan, "IPv6 addressing strategies for IoT," *IEEE Sensors Journal*, vol. 13, no. 10, pp. 3511–3519, 2013.

[3] M. Li, Z. Yang, and Y. Liu, "Sea depth measurement with restricted floating sensors," *ACM Trans. on Embedded Computing Systems*, vol. 13, no. 1, pp. 1–21, aug 2013.

[4] N. Bombieri, F. Fummi, and D. Quaglia, "System/network design-space exploration based on TLM for networked embedded systems," *ACM Trans. on Embedded Computing Systems*, vol. 9, no. 4, pp. 1–32, mar 2010.

[5] P. Sayyah, M. T. Lazarescu, S. Bocchio, E. Ebeid, G. Palermo, D. Quaglia, A. Rosti, and L. Lavagno, "Virtual platform-based design space exploration of power-efficient distributed embedded applications," *ACM Trans. Embedded Computing Systems*, vol. 14, no. 3, pp. 49:1–49:25, Apr. 2015.

[6] K. Tsilipanos, I. Neokosmidis, and D. Varoutas, "A system of systems framework for the reliability assessment of telecommunications networks," *IEEE Systems Journal*, vol. 7, no. 1, pp. 114–124, 2013.

[7] F. Fummi, G. Lovato, D. Quaglia, and F. Stefanni, "Modeling of communication infrastructure for design-space exploration," in *Proc. of Forum on Specification & Design Languages*, Sep. 2010, pp. 1–6.

[8] E. Ebeid, F. Fummi, and D. Quaglia, "Model-driven design of network aspects of distributed embedded systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 4, pp. 603–614, Apr. 2015.

[9] Object Management Group, "MARTE." [Online]. Available: http://www.omgmarte.org

[10] ——, "Unified Modeling Language." [Online]. Available: http://www.uml.org

[11] ——, "SysML." [Online]. Available: http://www.sysml.org

[12] The MathWorks, Inc., "Simulink." [Online]. Available: http://www.mathworks.com/products/simulink/

[13] ——, "Stateflow." [Online]. Available: http://www.mathworks.com/products/stateflow/

[14] Center for Hybrid and Embedded Software System, "Ptolemy." [Online]. Available: http://ptolemy.eecs.berkeley.edu/index.htm

[15] G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing*, pp. 471–475, 1974.

[16] "IEEE standard for standard SystemC language reference manual," *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, Jan 2012.

[17] Transaction Level Modeling Working Group, "OSCI TLM 2.0," 2006. [Online]. Available: http://www.systemc.org

[18] Center for Embedded and Computer Systems, "SpecC." [Online]. Available: http://cecs.uci.edu/~specc/

[19] Center for Electronic Systems Design, "Metropolis." [Online]. Available: http://embedded.eecs.berkeley.edu/metropolis/index.html

[20] A. Bakshi and V. Prasanna, "Algorithm design and synthesis for wireless sensor networks," in *Proc. of Int. Conf. on Parallel Processing*, 2004, pp. 423–430 vol.1.

[21] A. Bonivento, L. P. Carloni, and A. Sangiovanni-Vincentelli, "Platform-based design of wireless sensor networks for industrial applications," in *Proc. of the Design Automation & Test in Europe Conference*, 2006, pp. 1103–1107.

[22] L. Mottola, A. Pathak, A. Bakshi, V. K. Prasanna, and G. P. Picco, "Enabling scope-based interactions in sensor network macroprogramming," in *Proc. of IEEE Int. Conf. on Mobile Adhoc and Sensor Systems*, 2007, pp. 1–9.

[23] A. Puggelli, M. M. R. Mozumdar, L. Lavagno, and A. L. Sangiovanni-Vincentelli, "Routing-aware design of indoor wireless sensor networks using an interactive tool," *IEEE Systems Journal*, vol. 9, no. 3, pp. 717–727, Sep. 2015.

[24] A. Pinto, M. D'Angelo, C. Fischione, E. Scholte, and A. Sangiovanni-Vincentelli, "Synthesis of embedded networks for building automation and control," in *Proc. of the American Control Conference*, Jun. 2008, pp. 920–925.

[25] G. Gogniat, M. Auguin, L. Bianco, and A. Pegatoquet, "Communication synthesis and HW/SW integration for embedded system design," in *Proc. of the 6th Int. Workshop on Hardware/Software Codesign*, 1998, pp. 49–53.

[26] L. Benini and G. De Micheli, *Networks on chips: Technology and Tools*. Morgan Kaufmann, 2006.

[27] E. Zahavi, I. Cidon, and A. Kolodny, "Gana: A novel low-cost conflict-free NoC architecture," *ACM Trans. on Embedded Computing Systems*, vol. 12, no. 4, pp. 109:1–109:20, Jun. 2013.

[28] C. Seiculescu, D. Rahmati, S. Murali, H. Sarbazi-Azad, L. Benini, and G. De Micheli, "Designing best effort networks-on-chip to meet hard latency constraints," *ACM Trans. on Embedded Computing Systems*, vol. 12, no. 4, p. 1, Jun. 2013.

[29] A. Agarwal, B. Raton, C. Iskander, H.-t. Multisystems, and R. Shankar, "Survey of network on chip (NoC) architectures & contributions," in *Networks*, vol. 3, no. 1, 2009.

[30] U. Y. Ogras and R. Marculescu, "Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach," in *Proc. of the Design Automation & Test in Europe Conference*, 2005, pp. 352–357.

[31] C. E. Rhee, H. Y. Jeong, and S. Ha, "Many-to-many core-switch mapping in 2-D mesh NoC architectures," in *Proc. of IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, 2004, pp. 438–443.

[32] C. W. Lin, L. Rao, P. Giusto, J. D'Ambrosio, and A. L. Sangiovanni-Vincentelli, "Efficient wire routing and wire sizing for weight minimization of automotive systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1730–1741, Nov. 2015.

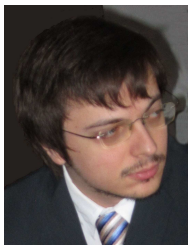
[33] S. Xu, R. Kumar, and A. Pinto, "Correct-by-construction and optimal synthesis of beacon-enabled ZigBee network," *IEEE Trans. on Automation Science and Engineering*, vol. 10, no. 1, pp. 137–144, Jan. 2013.

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2812797, IEEE Transactions on Computers

[34] Y.-X. Zhang, K. Takahashi, N. Shiratori, and S. Noguchi, "An interactive protocol synthesis algorithm using a global state transition graph," *IEEE Trans. on Software Engineering*, vol. 14, no. 3, pp. 394–404, Mar. 1988.
[35] A. Khoumsi, R. Dssouli, and G. V. Bochmann, "Protocol synthesis for real-time applications," in *Proc. of Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, 1999, pp. 417–433.
[36] H. Yamaguchi, K. Okano, T. Higashino, and K. Taniguchi, "Protocol synthesis from time Petri net based service specifications," in *Proc. Int. Conf. on Parallel and Distributed Systems*, Dec. 1997, pp. 236–243.
[37] R. L. Probert and K. Saleh, "Synthesis of communication protocols: Survey and assessment," *IEEE Trans. on Computers*, vol. 40, no. 4, pp. 468–476, Apr. 1991.
[38] P. V. Eijk and J. Schot, "An exercise in protocol synthesis," in *Formal Description Techniques IV. North-Holland*, 1991, pp. 117–131.
[39] T. A. Gonsalves and F. A. Tobagi, "Comparative performance of voice/data local area networks," *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 5, pp. 657–669, Jun. 1989.
[40] A. Luntovskyy and A. Schill, "Functionality of wireless network design tools," in *Proc. of 19th Int. Crimean Conference Microwave Telecommunication Technology*, Sep. 2009, pp. 343–345.
[41] Z. Kaleem, T. M. Yoon, and C. Lee, "Energy efficient outdoor light monitoring and control architecture using embedded system," *IEEE Embedded Systems Letters*, vol. 8, no. 1, pp. 18–21, Mar. 2016.

Enrico Fraccaroli (S'16) received his master degree in computer science and engineering from the University of Verona, Italy, in 2015. He is currently a Ph.D. Student at the Department of Computer Science, University of Verona, Italy. His research interests are the development of new methodologies for the efficient simulation and functional safety evaluation of embedded platforms composed of analog, digital and network components.



Francesco Stefanni received his Ph.D. degree in Computer Science in 2011, from University of Verona. Since January 2014 he is project manager and SW architect in EDALab srl where he leads the development of HIFSuite and SCNSL and provides internal training. His research interest encompasses networked embedded system modeling and verification, formal modeling, semantics mapping and efficient simulation. In 2011, he won the best paper award at Forum on specification and Design Languages (FDL).

Franco Fummi (M'92) received the Ph.D. degree in electronic engineering from Politecnico di Milano, Italy, in 1995. He is currently the Head of the Department of Computer Science, University of Verona, Italy, where he is a Full Professor since 2000, and where he became an Associate Professor in computer architecture in 1998. Since 1995, he has been with the Department of Electronics and Information, Politecnico di Milano, as an Assistant Professor. He is a co-founder of EDALab, an EDA company developing tools for the design of networked embedded systems. His current research interests include electronic design automation methodologies for modeling, verification, testing, and optimization of embedded systems.

Romeo Rizzi received his Ph.D. degree from the Department of Mathematics of Padova University, Italy, in 1997. He held researcher positions at centers like CWI (Amsterdam, Netherlands), BRICS (Aarhus, Denmark) and IRST (Trento, Italy), University of Trento and University of Udine, Italy. Since 2011, he has been an associate professor at the University of Verona, Italy. He has a background in Operations Research and his main interests are in Combinatorial Optimization and Algorithms. He is an Area Editor of 4OR. He published a hundred research papers in a broad range of scientific journals in the areas of Discrete Mathematics, Combinatorics, and Algorithms. He also authored several papers in conference proceedings, and invited chapters. Since 2004, he has intensively acted as a trainer of the Italian team for the iOi.

Davide Quaglia (M'03) received his Ph.D. degree in Computer Engineering from Politecnico di Torino (Italy) in 2003. Currently he is Assistant Professor at the Computer Science Department of University of Verona (Italy). He is member of IEEE and DATE Program Committee. He is co-author of about 70 papers on Networked Embedded Systems, Networked Control Systems, Cyber-Physical Systems. He was co-founder of EDALab, a spin-off of University of Verona.

0018-9340 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.