A Communication Synthesis Infrastructure for Heterogeneous Networked Control Systems and Its Application to Building Automation and Control

Alessandro Pinto
University of California, Berkeley
545P Cory Hall, Berkeley, CA 94720-1770
apinto@eecs.berkeley.edu

Luca P. Carloni
Columbia University
466 Computer Science Building
1214 Amsterdam Avenue, New York, NY 10027-7003
luca@cs.columbia.edu

Alberto L. Sangiovanni-Vincentelli
University of California, Berkeley
515 Cory Hall, Berkeley, CA 94720-1770
alberto@eecs.berkeley.edu

ABSTRACT

In networked control systems the controller of a physically-distributed plant is implemented as a collection of tightly-interacting, concurrent processes running on a distributed execution platform. The execution platform consists of a set of heterogeneous components (sensors, actuators, and controllers) that interact through a hierarchical communication network. We propose a methodology and a framework for design exploration and automatic synthesis of the communication network. We present how our approach can be applied to the design of control systems for intelligent buildings. The input specification of the control system includes (i) the constraints on the location of its components, which are imposed by the plant, (ii) the communication requirements among the components, and (iii) an estimation of the real-time constraints for the correct behavior of the algorithms implementing the control law. The output produces an implementation of the control networks that is obtained by combining elements from a pre-defined library of communication links, protocols, interfaces, and switches. The implementation is optimal in the sense that it satisfies the given specification while minimizing an objective function that captures the overall cost of the network implementation.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-aided design

General Terms

Algorithms, Design, Theory

Keywords

Communication Synthesis, Networked Embedded Systems, Building Automation System

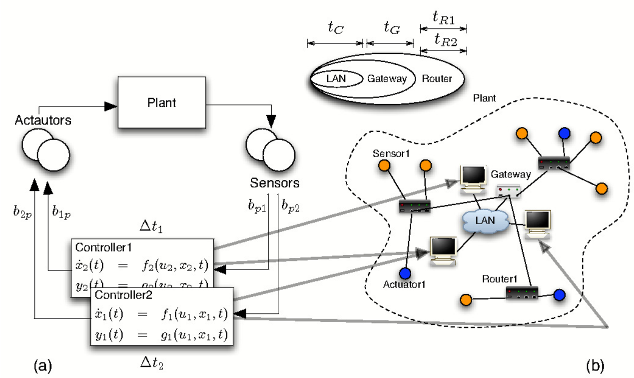

Figure 1: A distributed embedded control system: (a) controller specification and (b) networked execution platform.

1. INTRODUCTION

Electronics controllers for a large number of applications such as public infrastructure management, industrial plant control, automotive networks, avionics, and building automation are *networked* because of the distributed nature of the plant that they have to control. Figure 1 illustrates the design process of mapping an embedded control specification onto a networked execution platform. At the specification level, an abstract model of the plant is used to derive the desired property of the feedback controllers such as stability and robustness [11]. Each controller C_i, which is derived assuming a continuous-time model, is then discretized with the choice of a suitable sampling period Δt_i that preserves its properties [8]. Complex plants with multiple physical quantities to be controlled typically require multiple controllers that may be discretized with different sampling periods. At each sampling period, a certain amount of data is transferred from the sensors to the input of the controllers and from their outputs to the actuators. Therefore, each logical connection between the controllers and the plant implicitly defines a *message frequency.* For instance, controller C_1 in Figure 1(a) receives b_{p1} messages per second from the sensors and sends b_{1p} messages per second to the actuators.

Obviously, a distributed computation requires multiple computers [1] that need to exchange data via an interconnect network.

The network shown in Figure 1(b)) is heterogeneous and hierarchical: a high-performance local area network (LAN), also known as the *backbone network*, connects the various computers and is attached via a *gateway* to a *control network island*, or *zone*. In general the plant is partitioned in multiple zones according to its physical characteristics. The various zones are also connected via gateways. Each zone contains a subset of the sensors and the actuators that are linked to its gateway by a network of links and multiple *routers*. For simplicity, Figure 1(b)) shows only a single zone: here the control network is made of six sensors, three actuators, and three routers that are linked via the gateway to the backbone network.

This two-tier architecture is not the only possible network organization, but it is becoming increasingly popular for many important applications including *heating, ventilation and air-conditioning (HVAC)* control systems [9]. Generally, the goal of feedback control in a HVAC system is to regulate physical quantities such as temperature, humidity, and pressure to optimize an indoor environment for human comfort (comfort HVAC) or for machine operations (industrial HVAC) while minimizing operation, installation, and maintenance costs. The design of the communication network plays an increasingly important role in reaching these goals.

Once distributed on a set of computers interconnected by a backbone LAN, the overall control system requires that the worst-case computation time t_C be bounded. The messages from/to the sensors/actuators need to cross the gateway boundary that accounts for a worst-case communication delay equal to t_G. Since a controller C_i can tolerate a loop delay not greater than its sampling period Δt_i, the design of the control networks must satisfy a set of real-time constraints like ($t_{Ri} \leq \Delta t_i - t_C - t_G$) while guaranteeing that all required messages are gathered from the sensors and delivered to the actuators. In addition, the network cost (given by the sum of the costs of its components and of the installation costs) should be minimized.

Today it is standard practice to deploy the networked embedded system first on a predefined distributed architecture chosen on the basis of experience and heuristic considerations and then tweak the software implementation of the control algorithm to meet latency, bandwidth, and reliability requirements. To relax the dependency of the correctness of the algorithm from the communication performance, the network is often over-designed. This is far from ideal, since many systems are highly cost sensitive and using a network that is not tailored to the application and not optimized is clearly expensive. Moreover, the complexity of large networked embedded systems continues to increase, thus making heuristic and experience-based design practices inadequate. For instance, the scale of control networks for the automation of large buildings is of the order of thousands of sensors distributed on a surface of hundred thousands square meters, while a rich variety of alternative protocols and technologies are available to build such networks [9]. In summary, new design tools are needed to assist engineers in the design process so that the final implementation satisfies specifications taking into consideration the overall cost of deployment including development cost and time. We advocate a design process that starts from the formal specification of the network design problem, goes through high-level design exploration process, and ends with the automatic synthesis of the low-level details of the optimal control networks.

In this paper, we propose a methodology inspired by Platform-Based Design [7, 13, 15, 12, 14] for optimized *communication synthesis*. The basic tenets of the methodology are:

- formal capture of the communication requirements of the control application,
- the mathematical description of constraints on communication infrastructure and implementation possibly including the feasible physical positions of sensors, actuator, and gateways,
- mathematical description of objectives,
- a set of available network components (together with their performances and costs) that limit the search space of possible solutions and provide a communication *platform*, and
- mapping algorithms of the requirements onto the platform to move from one level of abstraction to the next until the final implementation is obtained.

To support this methodology, we built COSI (Communication Synthesis Infrastructure), a general and flexible software infrastructure that can be used as the basis for developing various specialized design flows to solve the communication synthesis in different application domains. In particular, we present a design flow for the synthesis of control networks in building automation systems and we discuss its application to the specific case where such networks are realized using daisy-chain busses.

The paper is organized as follows. We first give a general presentation of our communication synthesis approach and the COSI framework (Section 2). Then, in the following sections we illustrate the most important concepts with a case study: the synthesis of wired control networks for a simplified version of a HVAC system. In particular, we provide details on the process of specifying the communication constraints (Section 3), we show how to model an execution platform and its components (Section 4), and we present a communication synthesis algorithm that is tailored to the chosen case study (Section 5).

2. COMMUNICATION SYNTHESIS INFRASTRUCTURE

COSI is the software framework developed to support the PBD approach for communication synthesis. In Figure 2, we show the organization of the software and the PBD design flow, and the UML [4] class diagrams of the most important data structures that are at the core of our software technology.

2.1 PBD for Communication Networks

A platform in PBD is defined by the collection of available architectural components, also called library, that can be used to implement a functional specification. A *platform*

[1] These computers are given different names in different application areas, like *direct digital controls* in building automation, *programmable logic controllers* in industrial automation, and *electronic control units* (ECU) in automotive electronics.

instance is a particular "legal" composition of a set of library elements. In the context of networked embedded systems, a component is a network (with single nodes and single links as special cases).

In COSI, a *network* is defined as a directed graph $G(V, E)$ together with labeling functions associated to the vertices in V and the edges in E. The vertices represent network nodes such as communication sources/destinations, routers, and repeaters. Edges represent the communication links connecting the nodes in a network. A labeling function of the nodes is a map $V \rightarrow D$ where D is the range of values of the labels. Similarly, labeling functions can be defined for the links. For instance, the initial specification of a communication problem is defined as a point-to-point network and is represented by graph $G_C(V_C, E_C)$ with associated position and types of the nodes and bandwidth and latency of the links (Section 3). In the network specification, each node represents only a source or a target of an end-to-end communication, while each link is associated to a single end-to-end communication.

The *network library* $\mathcal{L}$ is a collection of networks. The labeling functions of a library component are used to capture its performance and cost figures. For instance, a network $G_P(V_P, E_P) \in \mathcal{L}$ can be annotated by the maximum bandwidth (also called capacity) that the links E_P can support. Usually, many labeling functions can characterize the performance of the same component. For instance, the position of a node can be assigned to many points inside a building. The set of labeling functions for each library element characterizes the performance space of each component.

The instantiation of a component is done by renaming its vertices and selecting one labeling function for the nodes and one for the links (i.e. by configuring the component). The composition of two networks is an important operation in our framework. Such operation must be commutative and associative. Furthermore, the composition defines how to obtain the labeling functions of a network starting from the labeling functions of its components. For example, in Section 4 we discuss how to model a library of daisy-chain buses and we define an operation to compose a bus with a new node (there called extension) that specifies also how to compute the total bandwidth and latency of the bus after composition (e.g., the bandwidth is simply the sum of all the transmission bandwidths of the nodes connected to the bus).

The possible network implementations depend on the definition of the composition operation, which is denoted by the symbol $\|$, and on the available library $\mathcal{L}$. The set of all possible network implementations is the *network platform* generated by $\mathcal{L}$. It is defined as

$$\langle\mathcal{L}\rangle = \mathcal{L} \cup \{G = G' \| G'_L : \; G'_L \; instance \; of \; G_L \in \mathcal{L}, \; G' \in \langle\mathcal{L}\rangle\}$$

An element $G \in \langle\mathcal{L}\rangle$ is called a *network platform instance.*

A synthesis algorithm takes the specification G_C and the platform $\langle\mathcal{L}\rangle$ and generates a network implementation G_I that minimizes a cost function while satisfying the specification. Different synthesis algorithms can be developed to leverage the particular structure of the communication synthesis problem in a given domain, thus exploring the design space more efficiently.

The final implementation $G_I(V_I, E_I)$ is also represented as a directed graph, but this is not necessarily a point-to-point network. Instead, parts of G_I, or the entire G_I, can represent multi-hop networks with links shared across multiple end-to-end communications: in this case, a node can represent also a router or a repeater, while a link may implement a hop between two routers carrying simultaneously multiple segments of different end-to-end communications.

The implementation G_I becomes the specification of a similar design problem at a the lower level of abstraction.

2.2 Data Structures

In COSI a directed graph is implemented with a data structure `IdGraph` where each node is uniquely identified by an integer and each link by a pair of integers. This data structure includes operations to add/remove nodes and link, to sweep over nodes, to access the adjacency list of nodes etc. A label is a particular instance of a variable defined by a data structure that extends the basic class `Variable`. For instance, a real number is defined by class `CosiDouble` containing a floating point number; a point in space (that is used to defined the position of a node) is defined by three real numbers. Another example is the message frequency B that is represented by a real number as well. Networks are defined by extending the graph data structure and attaching labels to nodes and links. Labels can be associated to nodes and links incrementally (during the refinement process) by extending networks. For instance, to define a network with bandwidth and delay labels associated to the links, we just need to extend the `P_B_Network` data structure.

We implemented the platform data structure starting from three orthogonal concepts: (1) the set of library components, (2) the performance and cost model, and (3) the physical properties of the environment that hosts the network. The library data structure contains a set of components. A component can be an entire network (e.g., a daisy-chain bus) that contains nodes and links. Depending on the network configuration, which is given by the value of the variables associated with nodes and links, it is possible to compute the performance and cost of a component using the `PerformanceCostModel`. This is a data structure that declares the API used by any library to estimate the performance and cost of a component. For instance, the cost of a network is the sum of the costs of nodes and links that is provided by the model. We keep the models and the components separate because the same component can be annotated by different models depending on its actual implementation. For instance, the same communication medium (a twisted-pair) can be used by different protocols and each protocol can have its own model.

The `Platform` data structure is the most complex in our framework. It contains the `Library` data structure, the composition rules and a characterization of the constraints imposed by the environment. For instance, as explained in Section 3, for the building automation application we capture floors, walls, surfaces on which wires can be laid out and locations where gateways and routers can be installed.

The `Platform` data structure provides an interface that allows the correct instantiation and configuration of library components. For instance, an algorithm that wishes to instantiate a router in a certain location p and connect the router to a gateway, should refer to the platform to determine if the router can be located at p, and if the connection can be established. In particular, in the building automation application the platform would carry the information on how many meters of wire are required for the connec-

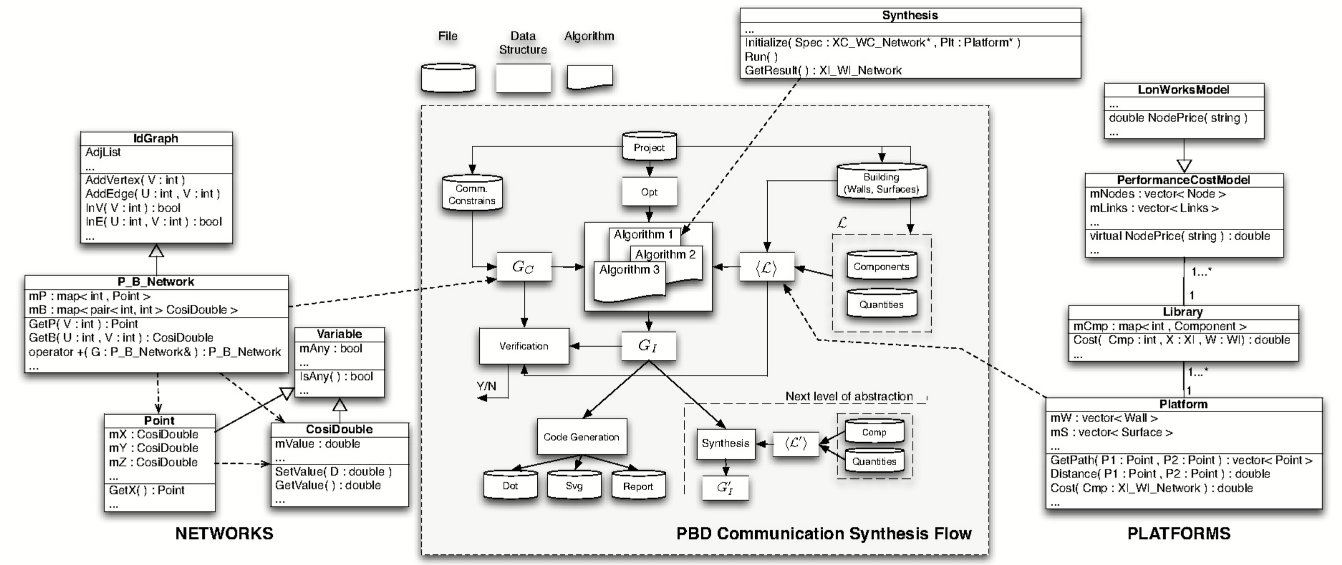

Figure 2: Software organization of the Communication Synthesis Infrastructure (COSI).

tion, and if the library contains a link that can span that distance. This orthogonalization allows us to use the same components with different models or the same library with different physical constraints.

In the following sections we discuss the application of the COSI infrastructure to the design of control networks in building automation systems for the particular case where the final network implementation is obtained with a network library made of daisy chain, token ring buses.

3. CONTROL NETWORK SYNTHESIS FOR BUILDING AUTOMATION SYSTEMS

As discussed in the introduction, a building automation system (BAS) is partitioned into multiple gateway zones according to the physical characteristics of the building. Typically a gateway zone coincides with a floor and the gateways can only be installed in specific closets [9]. The computers processing the control algorithms are also typically installed in pre-determined locations. The high-speed backbone LAN that connects the gateways and the computers is not the subject of this paper. Instead, we focus on the problem of synthesizing an optimal control network for each gateway zone. The control network for the entire building is then obtained as the composition of the control network of each gateway zone and the high-speed backbone LAN.

A gateway zone contains a gateway g, a set S of sensors and a set A of actuators. Sensors and actuators are connected to the gateway through routers. The number of routers within the control network may vary as well as their positions. The number of possible routers positions, however, is typically limited since they must be easy to access and kept away from possible hazards. In fact, the choice of how many routers to install and where to install them is part of the design of the control network and does affect its cost. While the link between the routers and the gateway offer relatively high-bandwidth and low-latency, the links between the sensors/actuators and the routers are implemented with

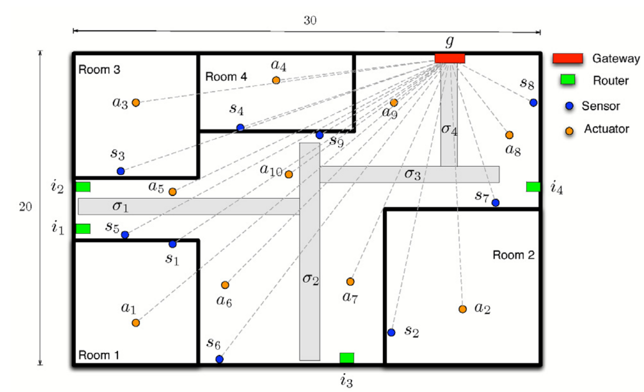

Figure 3: Example of gateway zone associated to a building floor.

twisted-pair wire technology. [2] Typically for each router a bus connects a subset of the sensors and the actuators in the zone. The choice of a bus standard and the length of the wires implementing the link affects directly the cost of the control network. Various protocol standards at different layers of the OSI model are available to control these busses like BacNet[10, 5], LonWorks [6], and ARCNET [1]. In many industrial cases, independently of the protocol of choice, the suggested topology for the physical implementation of the network is the *daisy-chain bus*. The main reason behind this choice is the impedance matching that can be performed by installing simple devices at the end of the chain. Due to

[2] Wireless links can be considered an alternative option for future implementations based on wireless sensor network (WSN) technology. This could potentially reduce the installation costs of a control network as long as it will be possible to have guarantees on the minimum latency communication in the wireless links similar to those provided by current wired technologies.

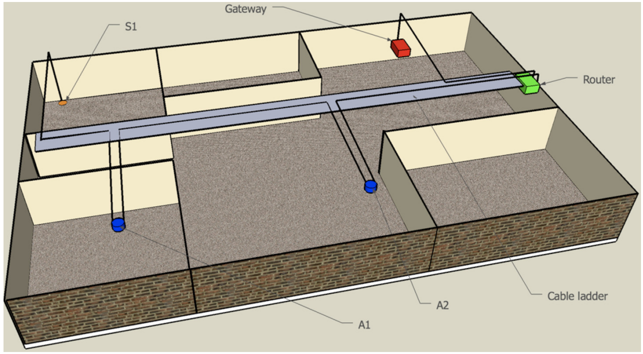

Figure 4: Example of how wires are laid out in a building.

their ubiquity, *we assume that the interconnect topology is based on daisy-chain bus topologies*.

Figure 3 illustrates a gateway zone for a simplified version of a HVAC building automation system, which we use as a case study in this paper. The floor of the building measures 30×20 m^2 and the ceiling height is $3m$. In Figure 3, $A = \{a_1, ..., a_{10}\}$ is the set of actuators that are placed at the ceiling level, and $S = \{s_1, ..., s_9\}$ is the set of sensors that are placed at $1.3m$ from the floor. The gateway is placed on the north wall and up to four routers may be installed on the other walls. Each potential router in the set $R = \{i_1, \ldots, i_4\}$ has an associated fixed position $p(i_j)$.

For each gateway zone, the end-to-end communication constraints between the nodes and the gateway are captured as a *constraint graph* $G_C(V_C, E_C)$ where $V_C = \{g\} \cup S \cup A$ and $E_C = (S \times g) \cup (g \times A)$. Each vertex $v \in V_C$ has an associated position $p(v) = (x, y, z)$ in the Euclidean space. In the sequel, we often use the term *node* to refer to either a sensor or an actuator, i.e. to the elements of $S \cup A$. Each edge $e \in E_C$ represents a point-to-point communication link between a node and a router (i.e. from a sensor to a router or from a router to an actuator). Each edge has associated a minimum message frequency $B(e)$ and a maximum delay $T(e)$. As discussed in the introduction, these constraints are derived from the control application requirements and its deployment across the backbone network connecting the computers and the gateways. In our simplified HVAC example, we assume that for each edge $B(e) = 10$ messages per second and $T(e) = 80$ ms.

The constraint graph G_C captures only part of the specification of the communication synthesis problem. The links in the control network are ultimately made of twisted-pair wires whose layout depends on many factors including the network topology, the building structure, ease of installation/operation and certification. Figure 4 shows an example of wire layout for a daisy-chain bus that connects a sensor and two actuators to a router. The layout is constrained by the network topology and the building structure. The standard way of laying out wires relies on raceways or cable ladders that are installed along the building aisles. Special conduits are used to bring wires from the nodes to the cable ladders. We capture these constraints with a set Σ of rectangular surfaces in the Euclidean space. We constraint wires to travel on these surfaces only. Wires from nodes are first laid out to the closest raceway and then towards their destination. For the example of Figure 3, the set of surfaces is $\Sigma = \{\sigma_1, .., \sigma_4\}$. They are about one meter wide and disposed at the ceiling level.

Both performance and cost of the network depend directly on the length of the wires. Hence, it is important to have a precise metric that accounts for the building constraints. Given two points in the Euclidean space $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$, the length of a link connecting them can be defined at different abstraction levels. For instance, for a given ceiling level h, we could define the distance as $d(p_1, p_2) = |z_1 - h| + |z_2 - h| + ||(x_1, y_1) - (x_2, y_2)||_1$ if $p_1 \neq p_2$, and zero otherwise. This definition captures the fact that each vertex must be wired to the ceiling first. The use of the L^1-norm captures the fact that wires follow straight lines, but it does not capture the real layout of the wires. In fact, the effective distance between any pair of elements of the control network is neither the L^1- norm, i.e. the Manhattan distance, nor the L^2-norm, i.e. the Euclidean distance. We adopt a more refined model. We first compute the distance from p_1 to the closest point q_1 in space that belongs to a raceway. Then, we compute the distance from p_2 to the closest point q_2 that belongs to a raceway. Finally, we derive the actual layout of the wire between q_1 and q_2 and we compute its length. The final distance between p_1 and p_2 is obtained by adding up these three contributions.

Given a model of the performance and cost of the components of the communication platform, which in our case is bused on daisy-chain busses as discussed in Section 4, and given the distance between all its nodes and the router, we can compute its performance as well as its contribution to the cost of the overall control network.

In summary, the problem of synthesizing the control network in building automation systems can be defined as follows: *given a constraint graph G_c synthesize a control network as a composition of busses by installing a number of routers and laying out a bus from each router such that: (a) each node in G_C is connected to one bus; (b) for each edge in G_C its constraints as minimum message frequency $B(e)$ and maximum delay $T(e)$ are satisfied; (c) and the sum of the costs of all the busses is minimized.*

4. MODELING THE NETWORK PLATFORM

The implementation of the control network for our case study is based on the LonWorks platform [6]. The LonWorks protocol defines the necessary services to exchange messages among the nodes of a network. LonkWorks can use different media to communicate as well as different protocols to implement the physical and data link layers of the OSI model. We selected ARCNET [1] as a local area network that interconnects LonWorks devices. ARCNET is a *token passing bus* with deterministic performance that can operate at different speeds ranging from $19Kbps$ up to $10Mbps$ (but optimized for $2.5Mbps$). A token passing bus (Figure 5(a)) is a centralized communication system where nodes are logically organized in a ring. A node can send messages only when it holds the token. The token is passed from one node to its logical neighbor that is the one with the next highest address.

Figure 5(b) shows the physical instantiation of a daisy chain bus. In order to connect a node to a bus, a twisted-pair wire has to be laid out on a path from the node to

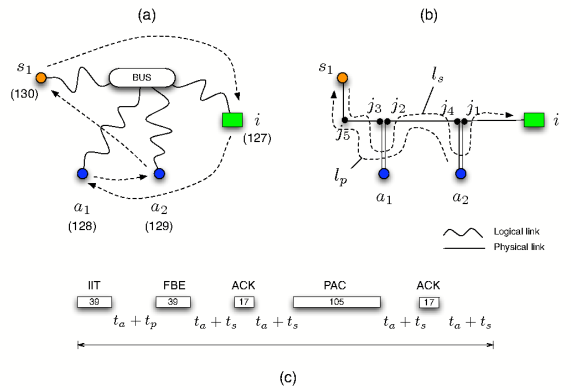

Figure 5: Graphical representation of a daisy-chain bus: (a) logical network, (b) physical network, (c) sequences of messages generated by the token passing protocol for a short packet transmission.

another node of the daisy chain. A *wiring path* is a sequence of points in space that defines the exact layout of a wire. For instance, a wiring path from actuator a_1 to actuator a_2 in Figure 5(b) is simply the sequence $\pi(a_1, a_2) = \langle p(a_1), j_2, j_4, p(a_2)\rangle$. Given a wiring path π, its length, denoted by $l(\pi)$ can be easily computed as the sum of the Manhattan distances between each point and its successor in the path.

To model the performance of a set of LonWorks components connected in a daisy-chain on an ARCNET bus, we need to analyze the token passing bus protocol. Consider the case where sensor s_1 sends a short packet to router i. The physical distance between them is $l_s = l(\pi(s_1, a_1)) + l(\pi(a_1, a_2)) + l(\pi(a_2, i))$. Also, assume that actuator a_2 holds the token and that the distance from a_2 to s_1 is $l_p = l(\pi(a_2, a_1)) + l(\pi(a_1, s_1))$. A successful transmission of a message from s_1 to i requires a sequence of protocol messages that includes: a token pass (Invitation to Transmit IIT), a Free Buffer Enquiry (FBE), an Acknowledge (ACK), a Packet (PAC) and a final ACK. Figure 5(c) shows each message in the sequence annotated with its length in number of bits (in this example we assumed that a payload of the message contains one byte only).

Between one protocol message and the next one, two other delays contribute to the total communication delay: the response time t_a of the chip that implements the protocol interface, and the propagation delay t_p and t_s relative to the signal traveling distances l_p and l_s respectively. Table 1 shows a realistic characterization of the components that we use to build the network. The delay of $12.6\mu s$ refers to t_a.

To compute the worst case communication delay between a node and a router connected on the same bus, we proceed as follows. The *short packet delay* from a node to the router can be computed as follows. The number of bits required for each message is 217, therefore if we use ARCNET at $2.5Mbps$, the time required to sent the bits of of the message is $217/2.5 = 86.8\mu s$ plus five times t_a, plus the propagation delays. Consider a set of nodes V connected on the bus such that one of them is the router. The worst case communication delay between one node $v \in V$ and the router occurs when the token is held by the logical neighbor of v. In order

Component	Performance	Cost
BUS (twisted-pair)	Degree : 8 Length: $120m$ Delay: $5.5ns/m$ Bandwidth: $2.5Mpbs$	Price: $0.6/m$ Inst.: $7/m$
Router	Delay: $320ns$	Price: $500 Inst.: $240
Sensor	Delay: $12.6\mu s$	Price: $110 Inst.: $50
Actuator	Delay: $12.6\mu s$	Price: $200 Inst.: $50

Table 1: Characterization of the intrinsic performance and cost of a realistic library of components for building automation systems.

to send a message, v must wait until the token comes back to it (token loop time). Assume that each node on the bus has a message to send to the router, then the worst case communication delay is the sum of the short packet delays from each node to the router.

To compute the maximum number of messages per second that a bus can support, we proceed as follows. Consider an ARCNET bus configured at $2.5Mbps$. The number of bits necessary to send a message of one byte is 217, therefore we obtain that at most $2.5/217 = 11520$ packets per second can be sent on the bus.

Other limitations apply to the maximum number of nodes that can be connected on a bus (also called degree) and the maximum length of the daisy chain. These two parameters change depending on the speed of the bus.

The cost of a daisy chain bus can be computed by adding together the cost of each node plus the wiring cost. Notice that, for each component, also the installation cost must be taken into account.

5. SOLVING THE SYNTHESIS PROBLEM

In this section we present our approach to solve the communication synthesis problem discussed in the previous section. Given the constraint graph G_C relative to a gateway zone, the synthesis algorithm deploys a sufficient number of busses to interconnect all sensors $S \subseteq V_C$ and actuators $A \subseteq V_C$ to the gateway $g \in V_C$. In deploying a bus, the algorithm takes into account the following constraints. The deployment of a daisy-chain bus is valid if it satisfies degree and length constraints. Further, point-to-point communication bandwidth and latency constraints as specified in G_C must be satisfied. For a node v connected on a bus, the worst-case communication delay must be less than or equal to the required latency. The sum of all message frequencies relative to the nodes on the bus must be less than or equal to the maximum number of messages per second that the bus can support. We say that a daisy-chain, or simply a chain, is valid if it satisfies all the aforementioned constraints. Given a specification G_C, a valid network implementation is *a set of valid daisy chains, each containing exactly one router, such that each sensor and each actuator in G_C is contained in exactly one chain.*

We solve the communication synthesis problem with a two-step approach: (1) chain generation and (2) chain selection. A chain c is a list of vertices whose extreme elements are $left(c)$ and $right(c)$. For any chain c we also define its

Algorithm 1: Find all minimum-length valid chains

Input: Available routers $I = \{i_1, ..., i_m\}$; Specification G_C
Output: Set of valid chains C
forall $i \in R$ **do**
  $A[v] \leftarrow false, \forall v \in V_C$
1   $c \leftarrow i$; $Extended \leftarrow true$
2   **while** $Extended$ **do**
    $Extended \leftarrow false$
3     $v_l \leftarrow \arg\min_{v \in V_C : A[v]=false} d(v, left(c))$
4     $v_r \leftarrow \arg\min_{v \in V_C : A[v]=false} d(v, right(c))$
    if $d(v_l, left(c) < d(v_r, right(c))$ **then**
      $v \leftarrow v_l$; $u \leftarrow left(c)$; $Left \leftarrow true$
      $c \leftarrow v \smile c$
    else
      $v \leftarrow v_r$; $u \leftarrow right(c)$; $Left \leftarrow false$
      $c \leftarrow c \smile v$
    $C' \leftarrow \emptyset$
5   **forall** $c' \in C(i)$ **do**
6   **if** $Left \wedge left(c') = u$ **then**
7   **if** `Extend`(c', v) **then**
          $C' \leftarrow C' \cup \{v \smile c'\}$
8   **else if** $\neg Left \wedge right(c') = u$ **then**
9   **if** `Extend`(c', v) **then**
          $C' \leftarrow C' \cup \{c' \smile v\}$
      $Extended \leftarrow true$
      $A[v] \leftarrow true$
    $C(i) \leftarrow C(i) \cup C'$
return C

cost $f(c)$, bandwidth $b(c)$ and worst case communication delay $t(c)$. Finally, the chain degree, i.e. the number of vertices in the chain is denoted by $|c|$.

Given R and G_C, let $C = \{c_1, ..., c_n\}$ be the set of all valid chains and f_i be the cost of chain c_i. Let $z_j \in \{0, 1\}$ be a binary variable that evaluates to one if chain c_j is taken in the final implementation. Also, let x_{ij} and y_{jk} be two binary variables such that $x_{ij} = 1$ if chain j contains router i and $y_{jk} = 1$ if node (either a sensor or an actuator) k belongs to chain j. The optimization problem that we want to solve can be stated as follows:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} f_j z_j \\ s.t. \quad & \sum_{j=1}^{n} x_{ij} z_j = 1, \forall i \quad \sum_{j=1}^{n} y_{jk} z_j = 1, \forall k \\ & z_j, x_{ij}, y_{jk} \in \{0, 1\} \end{aligned}$$

This represents an instance of the Binate Covering Problem (BCP), which is NP-complete. Since various algorithms for the exact or heuristic solution of BCP are known [18], in the sequel we focus on discussing our algorithm for the generation of valid chains.

Algorithm 1 is a greedy algorithm that generates minimum-length valid chains. We use the example in Figure 6 to explain how the algorithm proceeds. Given a chain c and a vertex v, the right extension of c, denoted by $c \smile v$, is a new chain where vertex v has been added at the end of the list of vertices of c. Similarly, the left extension of c, denoted by $v \smile c$ is a new chain where vertex v has been added at the front of the list.

For each possible router position i, the algorithm starts by creating a chain c that initially contains only a router vertex (line 1). Then, it tries to expand this chain by attaching other vertices, each representing either a sensor or an actuator that may end up being "covered" by the router in position i. If this attempt fails then the chain is discarded as an invalid chain. Otherwise, it is included in the set of valid chains that are passed to the binate covering algorithm. An array A is used to track those vertices that have been already considered to extend chains. At each iteration of the main loop, the algorithm attempts to select two vertices, v_l and v_r, among the sensors/actuators that have not been considered yet (lines 3 and 4).

For instance, consider the router position i_1 in Figure 6. During the first iteration actuator a_1 is selected as the the closest vertex to extend the chain of i_1. Since the left and right extreme coincide with the router at the beginning, a left extension is performed first by the algorithm (notice that left and right are simply convention and they don't relate to the physical position of the nodes). Chain c_1 is generated that covers vertex a_1 only.

On line 5, the algorithm starts analyzing all the chains that should be extended. If the node v to be added to the chain is closest to vertex u and a left/right extension is required, only the chains containing u as left/right extreme are considered for extension (lines 6, 8).

On line 7 and 9, the algorithm checks if the chain can be extended with the new vertex v. Function `Extend` checks that bandwidth, length, degree and delay constraints are met. If this is not the case it returns false and the chain is not extended. All newly generated chains are saved in the set C' and eventually added to $C(i)$.

In the example of Figure 6, chain c is now $a_1 \smile i$. The closest vertex to the left is the closest vertex to a_1 that is s_2 while the closest to the right is the closest to i_1 that is s_1. Since s_2 is closer to a_1 than s_1 to i_1, the algorithm extends the chain to the left generating $c_2 = s_2 \smile a_1 \smile i_1$. Chain c is now equal to $s_2 \smile a_1 \smile i_1$. The last vertex is s_1 that is closer to i_1. Therefore, the algorithm extends all chains in $C(i_1)$ with their right extreme equal to i_1. The newly-generated chains are c_3, c_4 and c_5.

If an extension violates the constraints, than the extended chain is not generated and, therefore, not added to the set of chains of a router i. Algorithm 1 returns a set of valid chains each covering a subset of the sensors and actuators

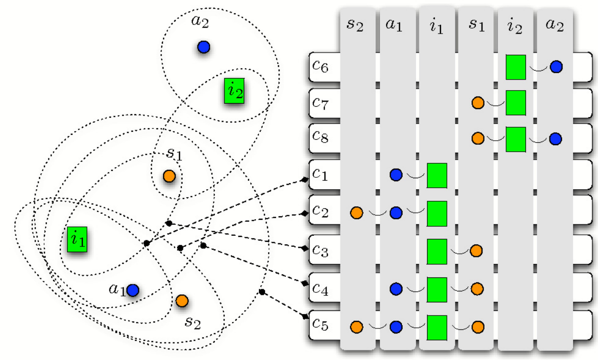

Figure 6: The chains generated by Algorithm 1 and the resulting covering matrix.

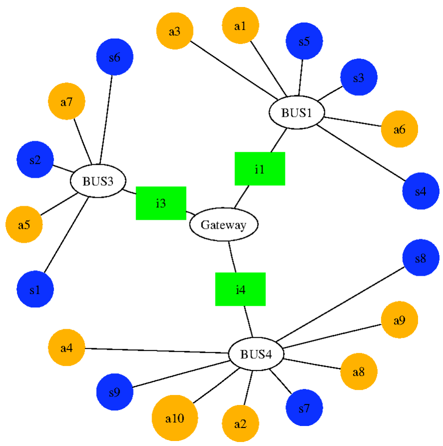

Figure 7: Logical components of the synthesized network for the example of Figure 3.

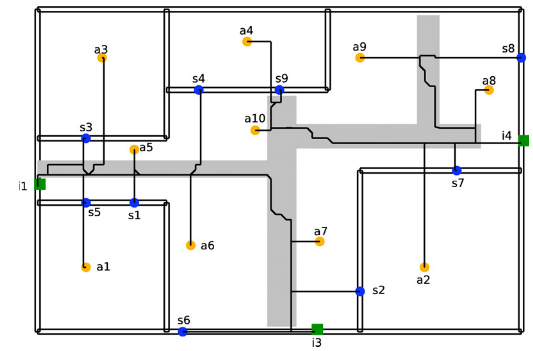

Figure 8: Physical deployment of the synthesized network implementation for the example of Figure 3.

Router	Deg	Length $[m]$	Delay $[\mu s]$	Bandwidth $[Kbps]$	Cost $[dollars]$
LonWorks on ARCNET @ $2.5Mbps$					
i_2	8	53	1367	17.3	$(2380, 404)$
i_3	3	20	607	6.5	$(1400, 152)$
i_4	8	58	1377	17.3	$(2380, 442)$
LonWorks on ARCNET @ $78kbps$					
i_4	19	142	54000	41.1	$(4680, 1080)$

Table 2: Performance and cost of the synthesis result.

and having a cost associated with them. This can be directly translated into a covering matrix for the binate covering problem.

In the example of Figure 6, chains c_6, c_7 and c_8 are generated for router i_2. Vertexes s_2 and a_1 are covered by router i_1 only, while vertex a_2 is covered by router i_2 only. Hence, in this case both routers are essential and must be installed. Which of the two routers will cover sensor s_1 depends on the cost of the chains. For this example, the only two possible solutions are $C' = \{c_2, c_8\}$ and $C'' = \{c_5, c_6\}$. The cost of C' is $f_2 + f_8$ while the cost of C'' is $f_5 + f_6$. The covering algorithm will select the least cost solution.

The complexity of the chain generation algorithm depends on the maximum degree of the chains. Let D denote the maximum degree. The main loop starting at line 2 is executed at most $2 \cdot D$ times corresponding to D left and D right extensions. Each loop iteration removes one vertex for the set of sensors and actuators to be covered. Also, at most $|C(i)|$ new chains are generated at each iteration. Therefore, the number of basic operations in the main loop is at most

$$\sum_{i=1}^{2 \cdot D} [(|S| + |A| - i) + (i + 1)]$$

where S and A are the sets of sensors and actuators, respectively. For $|S| + |A| \gg D$ the complexity is $O(D(|S| + |A|))$. The maximum number of chains that are generated by the algorithm is $(D + 1)(D + 2)/2 - 1$.

We run our synthesis flow on the example of Figure 3 for two different ARCNET configurations: $2.5Mbps$ and $78Kbps$. We generate three different outputs to analyze the synthesis result: a textual report of the performances and cost of the network, a `dot` [2] file that contains the logical structure, and an `svg` [3] file that contains the physical structure of the network.

Figure 7 shows the logical structure of the LonWorks network on ARCNET @$2.5Mbps$. The solution has three daisy-chain busses. The daisy-chains are limited both by the maximum number of nodes (8) and by the maximum wire length ($120m$). Given the high speed of the bus, there is a large bandwidth and delay slack. Figure 8 shows the physical implementation of the network.

Table 2 shows the estimation of cost and performance for each sub-network. The cost is represented by a pair of values: the first value is the cost of the components (sensors, actuators, and routers) and the second value is the cost of the wires. Observe that the delay is considerably smaller than the required delay and that the bandwidth utilization is fairly low. This suggests that for this network we could consider a different implementation with lower speed and lower cost. For instance, with a slower signaling, wires can be longer and, moreover, the degree can be higher.

For instance, at $78Kbps$ ARCNET allows to connect up to 64 nodes on a bus segment that can be as long as $1200m$. Using this kind of protocol, we obtain a considerably cheaper solution (\$5760 compared to \$7160) in exchange for a longer delay. The delay is longer not only because the number of devices connected on the bus is higher but also because its signaling speed is much lower. The bandwidth utilization is close to 50%.

On the other hand, while the cheaper solution is sufficient to support the application under design, once it is deployed it may prevent the future extension of the building automation system to support other applications. Since the deployment of a wired network in a building has considerable installation costs, this is another trade-off that must be considered carefully. In this regard, our tool can be useful to quickly analyze alternative solution hypothesis during the design-exploration phase.

6. CONCLUSIONS AND FUTURE WORK

We presented a methodology and a framework for design exploration and automatic synthesis of the communication network in distributed embedded systems. We applied this methodology to the special case of synthesizing control networks in building automation systems. The input specification of the control system includes (i) the constraints on the location of its components, which are imposed by the plant, (ii) the communication requirements among the components, and (iii) an estimation of the real-time constraints for the correct behavior of the algorithms implementing the control law. The output produces an implementation of the control networks that is obtained by combining elements from a pre-defined library of communication links, protocols, interfaces and switches. The implementation is optimal in the sense that it satisfies the given specification while minimizing an objective function that captures the overall cost of the network implementation.

Previous contributions in the literature focused on the analysis of distributed control systems under the assumptions that the network structure is given and its delay can be statistically characterized [16, 17]. The novelty of our approach consists of offering a solution for the *automatic synthesis of the control network* for these systems.

We are investigating different interconnect topologies, protocols, and interconnection links (e.g., we will consider as option in the synthesis problem the choice of a wireless link with its appropriate protocols). We are actively collaborating with United Technologies Research Center in bringing this approach to reality in the case of the building automation industry.

Acknowledgments

The authors thank Clas Jacobson of UTRC for the continuous support of our work. This research is sponsored in part by the Artist and HYCON network of excellence, the National Science Foundation (under Award #: 0644202), and the GSRC Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

7. REFERENCES

[1] Ata. arcnet (http://www.arcnet.com/lit.htm).

[2] Graphviz (http://www.graphviz.org/).

[3] Svg (http://www.w3.org/graphics/svg/).

[4] Uml (http://www.uml.org/).

[5] S. T. Bushby. Bacnet^{TM} - a standard communication infrastructure for intelligent buildings. *Automation in Construction*, 6(5–6):529–540, 1997.

[6] Echelon. Lonworks core technology (http://www.echelon.com/developers/lonworks/default.htm).

[7] A. Ferrari and A. L. Sangiovanni-Vincentelli. System design: Traditional concepts and new paradigms. In *Proceedings of the International Conference on Computer Design*, pages 1–12, Oct. 1999.

[8] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, 4th edition, 2002.

[9] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman. Communication systems for building automation and control. *Proceedings of the IEEE*, 93(6):1178–1203, June 2005.

[10] H. M. Newman. *Direct Digital Control of Building Systems: Theory and Practice*. Wiley, 1994.

[11] K. Ogata. *Modern Control Engineering*. Prentice Hall, 4th edition, 2001.

[12] A. Pinto, A. Bonivento, A. L. Sangiovanni-Vincentelli, R. Passerone, and M. Sgroi. System level design paradigms: Platform-based design and communication synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, 11(3):537–563, 2006.

[13] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[14] A. L. Sangiovanni-Vincentelli. Quo vadis sld: Reasoning about trends and challenges of system-level design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[15] A. L. Sangiovanni-Vincentelli, L. P. Carloni, F. D. Bernardinis, and M. Sgroi. Benefits and challenges of platform-based design. In *Proceedings of the Design Automation Conference*, pages 409–414, San Diego, CA, June 2004. IEEE.

[16] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry. Foundations of control and estimation over lossy networks. *Proceedings of the IEEE*, 95(1):163–187, Jan 2007.

[17] S. Tatikonda and S. Mitter. Control under communication constraints. *IEEE Transactions on Automatic Control*, 49(7):1056–1068, July 2004.

[18] T. Villa, T. Kam, R. Brayton, and A. L. Sangiovanni-Vincentelli. Explicit and implicit algorithms for binate covering problems. *IEEE Transactions on Computer-Aided Design*, 16(7):677–691, July 1997.