4-2010

Embedded Virtual Machines for Robust Wireless Control and Actuation

Miroslav Pajic
University of Pennsylvania, pajic@seas.upenn.edu

Rahul Mangharam
University of Pennsylvania, rahulm@seas.upenn.edu

Embedded Virtual Machines for Robust Wireless Control and Actuation

Abstract
Embedded wireless networks have largely focused on open-loop sensing and monitoring. To address actuation in closed-loop wireless control systems there is a strong need to re-think the communication architectures and protocols for reliability, coordination and control. As the links, nodes and topology of wireless systems are inherently unreliable, such time-critical and safety-critical applications require programming abstractions and runtime systems where the tasks are assigned to the sensors, actuators and controllers as a single component rather than statically mapping a set of tasks to a specific physical node at design time. To this end, we introduce the Embedded Virtual Machine (EVM), a powerful and flexible programming abstraction where virtual components and their properties are maintained across node boundaries. In the context of process and discrete control, an EVM is the distributed runtime system that dynamically selects primary-backup sets of controllers to guarantee QoS given spatial and temporal constraints of the underlying wireless network. The EVM architecture defines explicit mechanisms for control, data and fault communication within the virtual component. EVM-based algorithms introduce new capabilities such as predictable outcomes and provably minimal graceful degradation during sensor/actuator failure, adaptation to mode changes and runtime optimization of resource consumption. Through case studies in process control we demonstrate the preliminary capabilities of EVM-based wireless networks.

Keywords
Real-time systems, control systems, wireless networks, fault-tolerant systems

Comments
Suggested Citation:
Pajic,M. and R. Mangharam. “Embedded Virtual Machines for Robust Wireless Control and Actuation”. 16th IEEE Real-Time and Embedded Technology and Applications Symposium (IEEE RTAS). April 2010.

Embedded Virtual Machines for Robust Wireless Control and Actuation

Miroslav Pajic and Rahul Mangharam
Department of Electrical and System Engineering
University of Pennsylvania
{pajic, rahulm}@seas.upenn.edu

***Abstract*—Embedded wireless networks have largely focused on open-loop sensing and monitoring. To address actuation in closed-loop wireless control systems there is a strong need to re-think the communication architectures and protocols for reliability, coordination and control. As the links, nodes and topology of wireless systems are inherently unreliable, such time-critical and safety-critical applications require programming abstractions and runtime systems where the tasks are assigned to the sensors, actuators and controllers as a single component rather than statically mapping a set of tasks to a specific physical node at design time. To this end, we introduce the Embedded Virtual Machine (EVM), a powerful and flexible programming abstraction where virtual components and their properties are maintained across node boundaries. In the context of process and discrete control, an EVM is the distributed runtime system that dynamically selects primary-backup sets of controllers to guarantee QoS given spatial and temporal constraints of the underlying wireless network. The EVM architecture defines explicit mechanisms for control, data and fault communication within the virtual component. EVM-based algorithms introduce new capabilities such as predictable outcomes and provably minimal graceful degradation during sensor/actuator failure, adaptation to mode changes and runtime optimization of resource consumption. Through case studies in process control we demonstrate the preliminary capabilities of EVM-based wireless networks.**[1]

I. Introduction

Automation control systems form the basis for significant pieces of our nation's critical infrastructure. Time-critical and safety-critical automation systems are at the heart of essential infrastructures such as oil refineries, automated factories, logistics and power generation systems. Discrete and process control represent an important domain for real-time embedded systems with over a trillion dollars in installed systems and $90 billion in projected revenues for 2008 [1].

In order to meet the reliability requirements, automation systems are traditionally severely constrained along three dimensions, namely, operating resources, scalability of interconnected systems and flexibility to mode changes. Oil refineries, for example, are built to operate without interruption for over 25 years and can never be shutdown for preventive maintenance or upgrades. They are built with rigid ranges of operating throughput and require a significant re-haul to adapt to changing market conditions. This rigidity has resulted in proprietary systems with limited scope for re-appropriation of resources during faults and retooling to match design changes on-demand. For example, automotive assembly lines lose an average of $22,000 per minute of downtime [2] during system faults. This has created a culture where the operating engineer is forced to patch a faulty unit in an ad hoc manner which often necessitates masking certain sensor inputs to let the operation proceed. This process of unsystematic alteration to the system exacerbates the problem and makes the assembly line difficult and expensive to operate, maintain and modify.

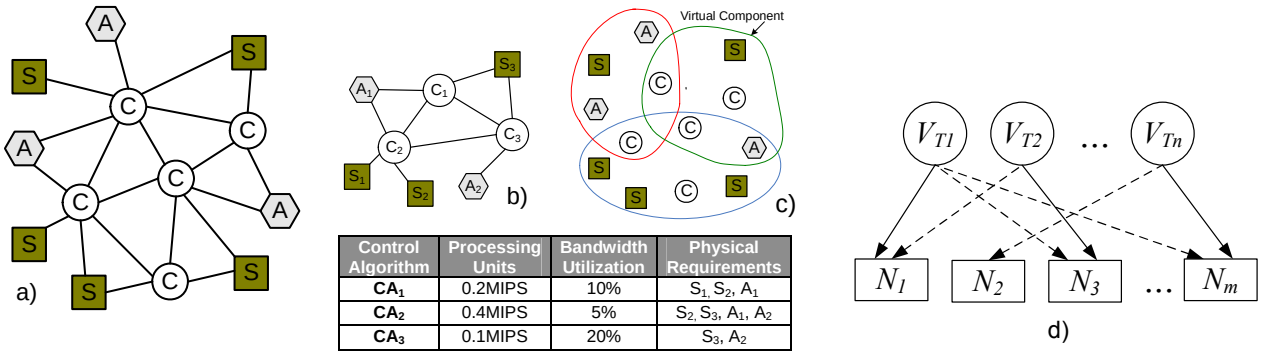
Embedded Wireless Sensor-Actuator-Controller (WSAC) networks are emerging as a practical means to monitor and operate automation systems with lower setup/maintenance costs. While the physical benefits of wireless, in terms of cable replacement, are apparent, plant owners have increasing interest in the logical benefits.

With multi-hop WSAC networks, it is possible to build modular systems which can be swapped out for off-line maintenance during faults. Modular systems can be dynamically assigned to be primary or backup on the basis of available resources or availability of the desired calibration. Modularity allows for incremental expansion of the plant and is a major consideration in emerging economies. WSAC networks allow for runtime configuration where resources can be re-appropriated on-demand, for example when throughput targets change due to lower price electricity during off-peak hours or due to seasonal changes in end-to-end demand.

A. Embedded Virtual Machines

The current generation of embedded wireless systems has largely focused on open-loop sensing and monitoring applications. To address actuation in closed-loop wireless control systems there is a strong need to re-think the communication architectures and protocols for reliability, coordination and control. As the links, nodes and topology of wireless systems are inherently unreliable, such time-critical and safety-critical applications require programming abstractions where the tasks are assigned to the sensors, actuators and controllers as a *single component* rather than statically mapping a set of tasks to a specific physical node at design time (see Fig. 1(a)). Such wireless controller grids are composed of many wireless nodes, each of which share a common sense of the control application but without regard to physical node boundaries. Our approach, as shown in Fig. 1, is to *decouple* the functionality (i.e. tasks) from the inherently unreliable physical substrate (i.e. nodes) and allow tasks to migrate/adapt to changes in the underlying topology.

[1]This research work was supported in part by the NSF CPS-0931239, CSR-0834517 and MRI-0923518 grants.



Control Algorithm	Processing Units	Bandwidth Utilization	Physical Requirements
CA_1	0.2MIPS	10%	S_1, S_2, A_1
CA_2	0.4MIPS	5%	S_2, S_3, A_1, A_2
CA_3	0.1MIPS	20%	S_3, A_2

Figure 1. (a) A wireless sensor, actuator and controller network. (b) Algorithm assignment to a set of controllers, each mapped to the respective nodes. (c) Three Virtual Components, each composed of several network elements. (d) Decoupled virtual tasks and physical nodes with runtime task mapping.

To this end, we introduce the Embedded Virtual Machine (EVM), a powerful and flexible programming abstraction where a Virtual Component (VC) and its properties are maintained across node boundaries, as shown in Fig. 1(c). EVMs differ from classical virtual machines. In the enterprise or on PCs, one (powerful) physical machine may be partitioned to host multiple virtual machines for higher resource utilization. On the other hand, in the embedded domain, an EVM is composed across multiple physical nodes with the goal to maintain correct and high-fidelity operation even under changes in the physical composition of the network. The goal of the EVM is to maintain a set of *functional invariants*, such as a control law and *para-functional invariants* such as timeliness constraints, fault tolerance and safety standards across a set of controllers given the spatio-temporal changes in the physical network. By incorporating EVMs in existing and future wireless automation systems, our aim is to realize:

1. *Predictable outcomes in the presence of controller failure.* During node or link faults, EVM algorithms determine if and when tasks should be reassigned and provide the mechanisms for timely state migration.

2. *Provably minimal QoS degradation without violating safety.* In the case of (unplanned) topology changes of the wireless control network, potential safety violations are routine occurrences and hence the EVM must reorganize resources and task assignments to suit the current resource availability (i.e. link bandwidth, available processing capacity, memory usage, sensor inputs, etc.).

3. *Composable and reconfigurable runtime system through synthesis.* In the EVM approach, a collection of sensors, actuators and controllers make a *Virtual Component* (VC) as shown in Fig. 1(c). A VC is a composition of interconnected communicating physical components defined by object transfer relationships. At runtime, nodes determine the task-set and operating points of different controllers in the VC, as shown in Fig. 1(b). This machine-to-machine coordination requires task-set generation, task migration and remote algorithm activation which are executed via synthesis at runtime, as shown in Fig. 2.

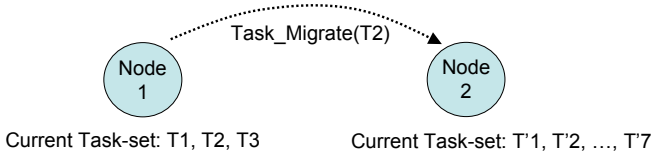


Figure 2. Task migration for real-time operation (instructions, stack, data & timing/fault tolerance meta-data) on one physical node to another.

4. *Runtime Resource Re-appropriation and Optimization for dynamic changes in service.* For planned system changes such as a factory shift, increase in output or retooling for a different chassis, nodes are required to be re-scheduled in a timely and work conserving manner.

B. Challenges with Wireless Control

While there has been considerable research in the general area of wireless sensor networks, a majority of the work has been on open-loop and non-real time monitoring application. As we extend the existing programming paradigm to closed-loop control applications with tight timeliness and safety requirements, we identify four primary challenges with the design, analysis and deployment of extending such networks:

1. Programming motes in the event-triggered paradigm is tedious for control networks.

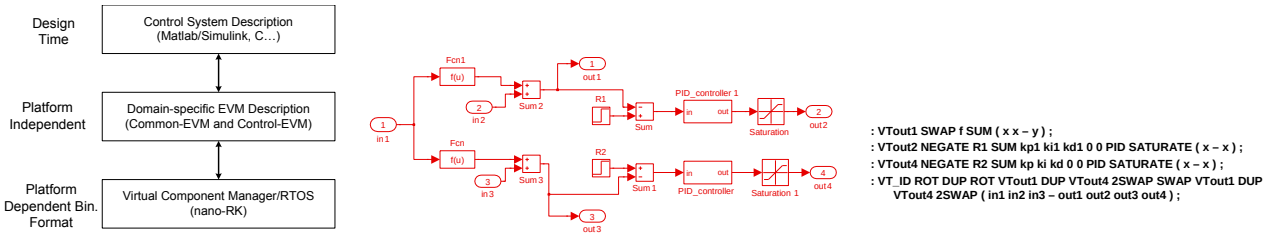
2. Programming of sensor networks is at the physical node-level.

3. Design of systems with flexible topologies is hard with physical node-level programming as the set of tasks (or responsibility) is associated with the physical node.

4. Fault diagnostics, repair and recovery are manual and template-driven for a majority of networked control systems.

C. Overview of the EVM Approach

While wireless system engineers optimize the physical, link and network layers to provide an expected packet error rate, this does not translate accurately to the stability of the control problem at the application layer. For example, planned and unplanned changes in the network topology with node/link failures are currently not easily captured or specifiable in the metrics and requirements for the control engineer. For a given plant connected to its set of controllers via wireless links, see Fig. 1(a-b), it is necessary that the controller process the sensor inputs and perform actuation within a bounded sampling interval. While one approach



(a) EVM Design flow (b) A control algorithm (VT) modeled in Simulink (c) Generated code

Figure 3. Generation of EVM functional description from Simulink model

is to design specialized wireless control algorithms that are robust to specified range packet errors [3], [4], it is non-trivial to design the same for frequent topological changes. Furthermore, it is difficult to extend the current network infrastructure to add/remove nodes and to redistribute control algorithms to suit environmental changes such as battery drain for battery-operated nodes, increased production during off-peak electricity pricing, seasonal production throughput targets and operation mode changes.

The EVM approach is to allow the control engineer to utilize the same wired-network control algorithms on the wireless network without knowledge of the underlying network protocols, node-specific operating systems or hardware platforms. The virtual machine executing on each node (within the virtual component) instruments the VC to adapt and reconfigure to changes while ensuring the control algorithm is within its stability constraints. This approach is complementary to the body of network control algorithms [3], [4] as it provides a logical abstraction of the underlying physical node topology.

The paper is organized as follows: Section II presents the automated design flow from a control problem specification to binding controller tasks to nodes within a VC. Section III describes the architecture of the EVM and mechanisms for parametric and programmatic control. Given these mechanisms, Section IV presents the key task assignment and scheduling algorithm to optimize operation during network changes. Finally, we describe the implementation on real hardware in Section VI and a case study in Section VII.

II. EVM Design Flow

Our focus is on the design and implementation of wireless controllers and in providing such controllers with runtime mechanisms for robust operation in the face of spatio-temporal topological changes. We now describe the design flow from control problem formulation, automatic translation of control models to platform-independent EVM interpreter-based code and finally to platform-dependent binaries (see Fig. 3(a)). These binaries are assigned to nodes within a VC using assignment and scheduling algorithms presented in Section IV.

At design time, control systems are usually designed using software tools, such as Matlab/Simulink, that incorporate both modeling and simulating capabilities. Therefore, in order to automatize the whole design flow, the EVM is able to automatically generate functional models from the Simulink control system description to define the processes by which input sampled data is manipulated into output data for feedback and actuation. These functional models are represented by generated code and meta data that are *platform and node independent system descriptions*, thus allowing the system designer to exclusively focus on the control problem design.

From the functional description in the platform-independent and domain-specific language, the EVM design flow automatically extracts additional para-functional properties, like *timing, inter-task dependencies, etc* from the Simulink model. All these properties, along with functional description are used to define a *platform optimized binary* for each virtual task (VT). VTs have the option of static or dynamic binding depending on the capabilities and timing requirements for the given control problem specification.

A. Control Problem Synthesis

A functional description of a VT is automatically extracted from the Simulink design, using the fact that each block can be represented as a composition of other Simulink blocks. Thus, each model can be presented as a hierarchical composition of basic Simulink blocks. This organization of Simulink models allows for natural extraction of a functional description using predefined words from the *EVM programming language* dictionary (described in the next sub-section). Similarly, when a new block in Simulink is defined as a composition of previously defined blocks, for the EVM functional description, a new word is described using previously defined words, until a level is reached where all words belong to the EVM dictionary. Therefore, a VT description is obtained using a parser that processes the Simulink model file by searching for a new block definitions along with the interconnections between blocks. An example of creation of a VT's functional description from a Simulink design model in Fig. 3(b) is presented in Fig. 3(c). As the platform-independent language is stack-based, the notation is in reverse.

B. Platform Independent Domain Specific Language

In order to generate functional description of the designed system, the EVM programming language is based on FORTH, a structured, stack-based, extensible programming language[5]. On the other hand, since the goal of EVM design is to allow flexibility and designing utilities independent of chosen programming language, the intermediate programming language is not constrained to FORTH.

- Dictionary consists of predefined blocks/functions that are usually used for control systems description
- PID (K_P K_I K_D *addr_in addr_out* –)
- PI (K_P K_I *addr_in addr_out* –)
- TF (*m n addr_alpha addr_beta addr_in addr_out* –)

$$G(z) = \frac{N(z)}{D(z)} \qquad N(z) = \alpha_0 + \alpha_1 z^{-1} + \alpha_2 z^{-2} + \ldots + \alpha_m z^{-m},$$
$$D(z) = \beta_0 + \beta_1 z^{-1} + \beta_2 z^{-2} + \ldots + \beta_n z^{-n}.$$

- LTI (*p m n addr_A addr_B addr_C a_in a_x a_out* -)

$$x[k+1] = Ax[k] + Bu[k],$$
$$y[k] = Cx[k], A \in R^{n \times n}, B \in R^{n \times m}, C \in R^{p \times n}$$

Figure 4. Control-EVM platform-independent and domain-specific language for expressing functional and timing description of Simulink models

The use of the EVM intermediate programming language enables **domain-specific constructs**, where basic programming libraries are related to the type of application that is being developed. For example, for use in embedded wireless networks for industrial control, we developed two predefined libraries, Common-EVM, based on the standard FORTH library [5], and Control-EVM that contains functionality widely used for developing control applications as shown in Fig. 4. In addition, the extensibility of EVM allows us to define the Automotive-EVM, Aviation-EVM or Medical-EVM libraries that will contain functionalities specific to each of these application fields. Using these libraries, the code generator creates a system description from a predefined component, thus creating a task description file for each of the Virtual Tasks (VTs).

Timing parameters (period and worst-case execution time) are also extracted from the model. Since we consider only discrete controllers as potential VTs, Simulink design rules force the designer to define a sampling rate for each discrete block, allowing us to a extract period that is the least common divider for all task timings. These values are also extracted from the model file and added to the VT's description file, along with the functional description.

III. EVM Architecture and Implementation

We now describe the node-specific architecture which implements the mechanisms for the virtual machine on each node. The Common-EVM and Control-EVM description for each set of controllers are scoped within Virtual Tasks that are mapped at runtime by the the task Assignment Algorithm described in Section IV. This description is interpreted by the interpreter running on each node. The EVM runtime system is built on top of the nano-RK real-time operating system [6] as a *supertask*, allowing node-specific tasks to execute natively and virtual tasks (VTs), i.e. those that are dynamically coupled with a node, to run within the EVM. The EVM services are shown in Fig. 5 and the EVM block-level reference architecture is presented in Fig. 6. This allows the EVM to maintain node-specific functionalities and be extensible to runtime task evocation of existing or new VCs.

The interface between nano-RK and all VTs is realized using the Virtual Component Manager (VCM) supertask. The

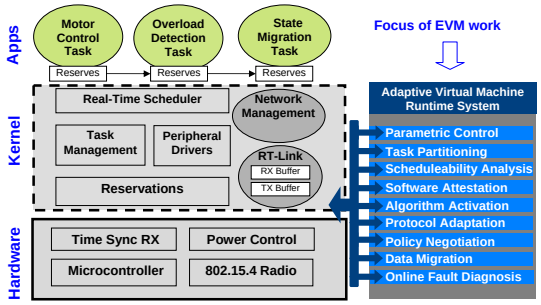


Figure 5. nano-RK RTOS architecture with EVM extensions

VCM maintains local resource reservations (CPU, network slots, memory, etc.) within nano-RK, the local state of the VTs and global mapping of VTs within the VC. The VCM is responsible for memory and network management for all VTs-to-physical nodes and presents a mapping between local and remote ports which is transparent to all local VTs. The VCM includes a FORTH-like interpreter for generic and domain-specific runtime operations and a Fault/Failure Manager (FFM) for runtime fault-tolerant operation. The VCM is implemented in a modular form so the FORTH interpreter, FFM and other specialized modules may be swapped with extensions and improvements over time and for domain-specific applications.

A. nano-RK Real-Time OS

To address the need for timing precision, priority scheduling and fine-grained resource management the nano-RK resource kernel [6] has been previously developed with timeliness as a first-class concern. nano-RK is a fully preemptive RTOS with multi-hop networking support. All networking is conducted over the RT-Link [7], a real-time link protocol that is native to nano-RK.

nano-RK has been design as fully static OS, configured at the design time. Therefore to allow parametric and programmatic run-time changes in the code nano-RK was redesigned and extended with several new features (see Fig. 5):

• *Runtime Parametric Control:* Support for dynamic change of sampling rates, runtime task and peripheral activation/de-activation and runtime modification to the task utilization has been added. These facilities are exposed and executed via the Common-EVM programmer interface.

• *Runtime Programmatic Control:* As a part of EVM design dynamic task migration has been implemented. This requires runtime schedulability analysis, capability checks to migrate a subset of the task data, instructions, required libraries and task control block.

• *Dynamic Memory Management:* Both *Best-fit* and *First-fit* memory allocation methods are supported. The Garbage Collector is scheduled only when its execution does not influence execution of other tasks.

B. Virtual Component Interpreter

The Virtual Component Interpreter provides the EVM programmer with an interface to define and execute all VTs. Every VT is defined as a word within the VCM library.

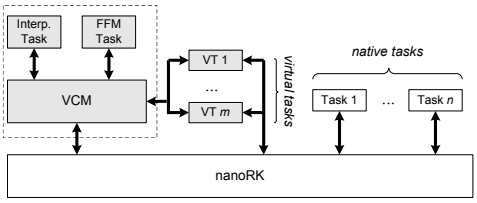


Figure 6. EVM architecture with Virtual Component Manager running as a supertask along side native nano-RK tasks

When a new VT description is received over the network, the VCM calls the interpreter which defines a new word using the description file of the task and the existing VC libraries. When a VT is activated, each execution of VT is implemented as a scheduled activation of the interpreter with the VT's word as an input. To allow preemptivity of the tasks, each call of the interpreter uses a VT-specific stack and dedicated memory segments. In addition, since each VT is capable of dynamically allocating memory during its execution in cases when there is no space in already dedicated memory, the EVM's memory manager allocates a new block of fixed size (currently 128B). Therefore, the interpreter is designed to use logical addresses in form $(block\_index, local\_address\_inside\_block)$. In order to be able to use the node's physical memory, the VCM supports address translation and garbage collection for the interpreter.

Each node has a local copy of standard Common-EVM dictionary and Control-EVM dictionaries. If a new word needs to be included in the existing library, upon reception of a message that contains the word, the VCM initiates execution of the interpreter. The interpreter first checks the global word identifier and revision number in order to discard obsolete versions. If a newer version is at hand, the interpreter (re)defines the word in the dictionary. Since all VTs executions are performed using the interpreter, there is no need for any additional kind of provider-subscriber relation as described in [8]. In addition, potential failure modes that may result from inconsistent versions on each node are avoided using this simple approach. Each local word update, also updates the revision number of the VT word, therefore keeping the VT fully updated without requiring consensus across the VC.

C. Virtual Tasks

Each VT is described using the Virtual Task's Description Table (VTDT), comprised of a global and local description of a VT (for details see [9]). Copies of the Descriptor Table are stored on all members of the VC. While this requirement for consistency currently results in an issue of scalability, a majority of the higher-speed control in a SCADA system require networks with less than 20 nodes and is hence within the practical limits of the current approach. Each task specifies its *Deployment constraints* as to whether it is *Fixed* or *Flexible* in terms of binding to specific physical nodes. Each VT's global description has information about memory requirements, stack size and number of fixed size memory blocks (e.g.128B) that are used. In addition to the above meta data, network requirements in terms of number of active RT-Link transmit and receive slots, are specified at design time. The above descriptors are specified within the VCM's Task Control Block (TCB) for each VT.

D. Virtual Component Manager

The fundamental difference between native nano-RK and the VCM is that the scope of nano-RK's activities is local, node-specific and defined completely at design time, while the scope of the VCM is the VC that may span multiple physical nodes. The current set of functionalities supported is:

1. Virtual Task handling:
1.1 VC state is maintained, keeping every node with a consistent mapping of VTs. The VCM in each controller node within the VC periodically broadcasts its information about VT mapping in order to keep consistency between all members of the VC. Currently a centralized consensus protocol is used and a distributed consensus protocol is needed to scale operations.
1.2 Task migration that can be triggered as a result of a fault/failure procedure or by request of either the VT or VCM. As a part of a task migration, the task's VTDT is sent along with all memory blocks used by the task. If the VT is already defined on a node (checked by exchange of hash values), only task parameters are exchanged. In addition, due to physical limitations (network, differences between controller/actuator nodes, etc) before migrating a VT to a particular node, network and CPU schedulability analysis are performed for nodes that are potential candidates. If analysis show that no node can execute the task correctly, an error message is returned.
1.3 VT activation: After a VT is defined, the VCM performs local CPU and network schedulability analysis prior to task activation. This is done in order to ensure that task will not adversely affect correct execution of previously defined VTs.
1.4 Control of tasks executed on other nodes: For all VTs in *Backup* mode, the VCM shadows the execution of the VT in *Primary* mode. If a departure from the desired operation is observed (i.e. low battery level, decreased received packet signal strength), *Backup* nodes may be assigned to *Primary* mode based on policy.

2. Network Management:
2.1 Transparent radio interface: Using the message header which contains information about message type, the VCM determines which task should be informed about message arrival. Messages containing tasks and their parameter definitions are first processed by the VCM, before the VCM activates the Interpreter (for remote ports). Local task messages are transferred to their respective local ports.
2.2 Logical-to-physical address mapping: All communication between VTs is via the VCM. Since a VT does not have information on which nodes other tasks are deployed, the VCM performs logical-to-physical address mapping.

IV. Virtual Task Assignment

We now discuss the EVM algorithm for determining the best set of physical controller nodes to execute virtual tasks, given a snapshot of current network conditions. The goal of the assignment procedure is to determine: (1) Assignment of the control algorithms to the set of nodes V, where each VT is assigned to one node as a *Primary* and to R nodes as a *Backup*. (2) Communication schedule that determines active links at each time slot. Communication between nodes is scheduled using TDMA based protocol with frame size F_S. (3) Computational schedule that determines in which time slot each control algorithm is executed. We first sketch the general formulation followed by a more intuitive relaxed formulation.

To define the problem as an optimization problem, the following assumptions have been made:
1. For each process j, all *Primary* and *Backup* nodes assigned for its execution are scheduled in the same time slot(s).
2. Virtual tasks are mutually independent.
3. A process i will remain stable if its sampling rate is less than $T_i, \forall i \Rightarrow F_S \leq min(T_1, T_2, ..., T_p)$.

The first assumption, simplifies the assignment problem formulation and allows for an easier schedulability analysis scheme. A significant class of process controllers execute a large number of simple and independent control loops. Hence, the second assumption is reasonable. As future work, this assumption will be relaxed to consider dependencies between tasks. Finally, the last assumption can be justified as shown in [4].

For example, consider a Networked Control System (NCS) used to control a plant modeled with continuous Linear-Time Invariant (LTI) dynamics $\dot{x}(t) = Ax(t) + Bu(t),\ y(t) = Cx(t)$, controlled by a discrete, state feedback controller $u(kT) = -Kx(kT)$. If a network induced delay τ_k is less than one sampling period a control feedback has a form $u(t^+) = -Kx(kT),\ t \in [kT + \tau_k, (k+1)T + \tau_{k+1})$. Thus $u(t)$ is a piecewise continuous function that changes values only at time $kT + \tau_k$. Since the EVM uses fully synchronous networks, after a message is delivered to the actuators they are scheduled to actuate at the same time. This maintains the same delay for all inputs of a plant at each period ($\tau_k = \tau, \forall k$). Using the simulation approach as in [4], a stability region with respect to T and τ can be determined. The region is used to determine maximal allowed sampling period for which, if a network delay is less than a period ($\frac{\tau}{T} \leq 1$), the system maintains stable.

Since standard link protocols for wireless factory automation, such as WirelessHART[10], recommend that only one physical node may transmit in each time slot, we were able to obtain an efficient reformulation of the relaxed assignment problem.

A. General Formulation

To develop an assignment algorithm we considered a multi-hop control network, consisting of p processes and m nodes (sensors, actuators and controllers). Due to space limitations, we provide a sketch of the assignment algorithm. For the complete solution the reader is encouraged to consult [9].

In order to ensure correct behavior of the system, the following set of constraints have to be met:
1. Primary node assignment: Every control algorithm should be executed on exactly one, *Primary* node
2. Redundancy constraint: Every control algorithm should be assigned to R additional, *Backup*, nodes (different from the one executing the control algorithm).
3. Link reliability constraint: Only links with link quality above a given threshold should be considered.
4. Routing constraints: Each *Primary* node uses 2 different paths to deliver information to all actuators related to the control of the process. All *Backup* nodes must be connected with all sensors used for the control algorithm execution.
5. Monitoring constraint: All R backup nodes have to be 1-hop neighbors of the primary node.
6. Computation schedule constraint: Each control algorithm has to be scheduled **exactly once** during a frame. Future extensions will allow processes to have different sampling intervals.
7. Communication schedulability constraint: All communication should be interference-free (two interfering nodes cannot transmit in the same time slot).
8. Stability constraints: All control loops have to be stable. Using the third assumption, all control loops will be stable if the end-to-end delay from sensors to controller to actuators along with the time for the controller's computation is less than F_S

The problem considers the following set of objectives:
1. Minimize aggregate number of links used, f_{LN}.
2. Maximize aggregate link quality, f_{LQ}.
3. Maximize use of disjoint routing for the monitoring of the primary controller: All of the R backup nodes should receive sensor values using maximum disjoint paths. Also, two paths from the *Primary* node to actuators should be maximum disjoint.

The assignment problem was formulated as a binary integer programming optimization problem and solved using branch and bound[11]. In addition, as the problem formulation has a large number of decision variables, it can be computationally expensive to solve the problem even for a small network. Thus we have translated the problem into the satisfiability problem, by transforming each constraint into conjunctive normal form (CNF) (more details in [9]) which is then solved using a very efficient satisfiability solver zChaff [12]. This allows us to solve the previous problem in real-time even for larger networks.

B. Problem Relaxation

When only one node in the VC can transmit in each time slot, the number of slots needed to send a message from node v_1 to node v_2 is equal to the distance (number of hops) between the two nodes. In the relaxed formulation, the fact

that only one node can transmit per time slot eliminates the need to include communication and computation schedule decision variables (used in the general form) and therefore significantly reduces the complexity of the optimization problem. In addition, the communication constraints and the stability constraint are one and the same.

As a first step for the problem formulation, two maximum node-disjoint paths (routes) r^1_{i,a_c} r^2_{i,a_c} are determined for each node v_i and each actuator a_c. The existence of two node disjoint paths from a node to all sensors and actuators can be checked using Menger's theorem[13]. For details see [9]. When a node has two node-disjoint paths, using a polynomial time algorithm (MIN-SUM 2-paths) we can determine paths r^1_{i,a_c} r^2_{i,a_c} with minimal total length [14]. Otherwise, the path r^1_{i,a_c} is determined again in a polynomial time as a shortest path to the actuator. Path r^2_{i,a_c} is calculated as the shortest path to the actuator after removing nodes from path r^1_{i,a_c}, while preserving connectivity. Using a similar approach, for each $(R+1)$-tuple $v_i, v_{i_1}, ..., v_{i_R}$ where $v_{i_1}, ..., v_{i_R}$ are neighbors of v_i, a set of $R+1$ paths is created between each sensor s and the nodes $v_i, v_{i_1}, ..., v_{i_R}$, with distances $(d_{i,s}, d_{i_1,s}, ..., d_{i_R,s})$.

To formulate the assignment problem we used $2mp$ binary assignment variables $x^{st}_{i,j} (\in \{0,1\})$, $i \in \{1,...,m\}, j \in \{1,...,p\}, st \in \{a,b\}$, where:
- $x^a_{i,j} = 1$ if and only if node v_i executes j^{th} control algorithm as the *Primary* node.
- $x^b_{i,j} = 1$ if and only if node v_i is used as a *Backup* for the j^{th} control algorithm.

Now the problem can be reformulated as follows:

$$minimize\ w_1 \cdot f_{LN}(x) + w_2 \cdot f_{LQ}(x),$$

with the respect to
$x = [x^a_{1,1}, x^b_{1,1}, ..., x^a_{1,j}, x^b_{1,j}, ..., x^a_{m,p}, x^b_{m,p}] \in \{0,1\}^{2mp}$, where weights w_1 and w_2 are used to emphasize impacts of a cost function. The problem is constrained with:

$$\sum_{i=1}^{m} x^a_{i,j} = 1, j = 1, ..., p$$

$$\sum_{i=1}^{m} x^b_{i,j} = R, j = 1, ..., p$$

Monitoring Constraints:

$$\sum_{k,N(i,k)=1} x^b_{k,j} \geq R \cdot x^a_{i,j},\ \forall j,\ N(i,k) = \begin{cases} 1, & (i,k) \in E_T \\ 0, & (i,k) \notin E_T \end{cases}$$

Stability Constraint:

$$\sum_{\substack{i \in V \\ j \in \{1,...,p\}}} \{ \sum_{s \in S_j} (x^a_{i,j} \cdot d_{i,s} + \sum_{k,(i,k) \in E_T} \frac{1}{deg(v_k)} x^b_{k,j} \cdot d_{k,s}) + \sum_{a \in A_j} x^a_{i,j} \cdot (d(r^1_{i,a}) + d(r^2_{i,a})\} + d_{max} \leq F_s$$

where d_{max} is maximal deadline (in number of slots) for all control algorithms.

The last constraint requires that all communication is done in one frame, and therefore, meets the timing requirements necessary for the stability of the system. The stability constraint is the only one that depends on the numbers of VTs and used data routing. Thus, each control loop, operating across the same physical set of controllers, can be considered separately which significantly simplifies system analysis. This allows for **compositional analysis** that can be used for scheduling extraction. Since the EVM is focused on networks with less than 20 nodes, we are able to run optimization algorithm on every node in a VC.

V. Virtual Task Execution

When each VT is assigned with *Primary* and *Backup* nodes along with appropriate routing, the VC starts executing its VTs. The existence of multiple maximum node-disjoint paths allows more robust information flow with respect to a ***single*** failure. We are focused on system faults due to failures and not due to malicious behavior of nodes in the VC.

A. Planned and Unplanned adjustments

Planned adjustments occur in situations when a node estimates future changes in VC state, for example when a node detects that its battery level is below some threshold. To chose a node to migrate its task, the primary node has to execute computation schedulability analysis and the communication schedulability analysis and pick a node that maximizes communication slack value (details in next section).

For unplanned changes caused by potential failures, we have considered the following cases:
• The *Primary* node dies: This initiates execution of computation and communication schedulability analysis in the $k = 1$-hop neighborhood. Since data state of a *Primary* node is maintained at *Backup* nodes, a new *Primary* node continues VT execution.
• *Backup* node dies: The *Primary* node detects that a *Backup* has died and selects a new *Backup* from one of its neighbors.
• Forwarding node dies or a link's quality goes below some threshold value: The detection of a forwarding node failure is performed by its predecessor/successor on the routing path. Again a communication schedulability analysis is performed (for only the affected sensor and actuator) to determine a new routing scheme.

B. Schedulability Analysis

Communication Schedulability: When a VT is to be migrated from a node v_i to a node v_j, we define a set S_{VT} of all sensor inputs that are necessary for VT execution and a set A_{VT} of all actuators that need to be informed about VT outputs. Also, we define as $v^k_{i,s}$ for all $s \in S_{VT}$, a node that is k-hops away from node v_i on a route from sensor s to node v_i and similarly $v^k_{i,a}$ for all $a \in A_{VT}$ a node that is k-hops away from node v_i on a route from the node to the actuator a. In addition, we define N^i_u as number of unused time slots in time interval between the first slot in which **all** nodes $v^k_{i,s}$ were suppose to receive values from sensors in S_{VT} and a first slot in the frame in which **at least one** node $v^k_{i,a}$ is scheduled to receive information from the node v_i.

The idea of the communication schedulability is to determine whether we can reassign (with the respect to the current communication schedule) available slots and slots used to send data in k-hop neighborhood of a node v_i. The re-assignment should re-route all necessary data from these nodes to node v_j. A new feasible communication schedule can be generated if $\Delta \geq 0$, where Δ is a communication slack value defined as: $\Delta = \sum_{s \in S_{VT}} d(v_i, v_{i,s}^k) + \sum_{a \in A_{VT}} d(v_i, v_{i,a}^k) + N_u^i - \sum_{s \in S_{VT}} d(v_j, v_{j,s}^k) - \sum_{a \in A_{VT}} d(v_j, v_{j,a}^k)$, where $d(v_p, v_q)$ is a distance between nodes v_p and v_q.

If more than one task is migrated from a node, similar analysis is performed where the previous equation is adjusted to contain sums for all sensors and actuators for all tasks. If more than one task should be migrated from node v_i to separate nodes, a schedulability analysis is performed on a pairwise basis, separately between nodes v_i and v_j where j goes from lower to higher node IDs.

In order to decrease response time, each node uses its idle computation time to calculate in advance the optimal reaction to potential failures. This approach enables triggering the recalculation of the global assignment procedure if it is determined that in some case there is no adjustment that can meet all constraints. Since the global assignment procedure is computationally much more expensive, this allows enough time for its computation. In addition, if the assignment procedure can not come up with the feasible assignment, an alarm is raised to add more nodes in the network.

Computation Schedulability: For computation schedulability analysis we used standard real-time response analysis [15] and the mode-change protocol presented in [16] and [17], adapted for the EVM. Consider a node v_i that executes a task set $\mathbb{T} = \{T_{i_1}, ..., T_{i_m}, VT_{i_1}, ..., VT_{i_n}\}$, where tasks T_{i_j} are local, node specific tasks, while tasks VT_{i_j} are VTs assigned to the node (in descending order of priority). We define a set $HP\_VT(T)$ as a set of all VTs with higher priority than local task T and, similarly, a set $HP\_T(VT)$ as a set of all node-specific tasks, with higher priority than task VT.

To allow an assignment of a new VT, a schedulability analysis is performed where both active and inactive tasks are considered as active. Although this approach is conservative, it eliminates the need for repeated schedulability analysis prior to tasks activation. Each node-specific task is denoted as $T_j = (p_j, e_j)$ and each VT as $VT_j = (p_{VT_j}, e_{VT_j}, \phi_{VT_j}, d_{VT_j})$ (period, execution time, offset and deadline respectively). Schedulability of a new task set is performed by checking the schedulability of all tasks with a lower priority than the new VT.

As mentioned in the previous section, we currently consider the case where all VT have same period of execution. Since a controller's processing is triggered by a reception of sensed signals and must be executed before its scheduled communication to actuators, its deadline must be much lower than its period. Thus, from VT's activation till its deadline, all other VTs can be active at most once, so for a task $VT_i, i \geq k$ we can write

$$w_{VT_i}(t) = e_{VT_i} + \sum_{j \in HP\_T(VT_i)} \left\lceil \frac{t}{t_{T_j}} \right\rceil \cdot e_{T_j} + \sum_{j=1}^{i-1} e_{VT_j}$$

The equation is too conservative since it assumes that all VTs can be activated at the same time. However, VTs are activated when a last radio message containing necessary data is received. In addition, since all VT's periods (T_{VT}) are multiples of TDMA slot duration, when a communication schedule is known, all possible offset combinations of a task activation can be easily calculated.

Therefore, for a task VT_i, released at time t_i, for all possible combinations of release times t_j of VTs with higher priority the time-demand function for $t \geq t_i$ is defined as:

$$w_{VT_i}^{(t_0,t_1,...,t_{i-1})}(t) = e_{VT_i} + \sum_{k \in HP\_T(VT_i)} \left\lceil \frac{t}{t_{T_k}} \right\rceil \cdot e_{T_k} + \sum_{\substack{j=1, \\ t_i + d_i \geq t_j \geq t}}^{i-1} min(e_{VT_j}, t - t_j) + \sum_{\substack{j=1, \\ t_i \in [t_j, t_j + d_j]}}^{i-1} min(e_{VT_j}, t_j + d_j - t_i)$$

A task VT_i is schedulable, if for all combinations of activation times, the schedulability condition is satisfied.

Although the previous equation seems complicated, for most control systems, all loops usually have the same sampling periods or all sampling periods are integer multiples of one of the periods. Therefore, in these cases, a number of possible combinations of $(t_0, t_1, ..., t_{i-1})$ is very small.

A similar approach is used for schedulability analysis of a node-specific task T_i.

VI. Implementation

To evaluate the EVMs performance in a real setting with multiple coordinated controller operations, we used a factory simulation module (FischerTechnik model factory). The factory consists of 22 sensors and actuators that are to be controlled in a coordinated and timely manner. A block of wood is passed through a conveyor, pushed by a rammer on to a turn table and operated upon by up to three milling/cutting/pneumatic machines. The factory module was initially controlled by wired programmable logic controllers (PLCs). We converted it to use wireless control with FireFly nodes controlling all sensors and actuators via a set of electical relays. FireFly [18] is a low-cost, low-power, platform based on the Atmel ATmega1281 8-bit micro-controller with 8KB of RAM and 128KB of ROM along with a Chipcon CC2420 IEEE 802.15.4 standard-compliant radio transceiver. FireFly nodes support tight global hardware-based time synchronization for real-time TDMA-based communication with the RT-Link protocol [7]. The EVM also works on TI MSP-430F5xxx and MSP430FG4xxx architectures for more efficient execution of the FORTH-like stack-based language on a Von Neumann architecture. We have demonstrated:
1. On-line capacity expansion when a node joins the VC

2. Redistribution of VTs when adding/removing nodes
3. Planned VT migration triggered by the user
4. Unplanned VT migration due to node/link failure
5. Multiple coordinated work-flows

In a second experiment setup we considered multiple concurrent workflows, where each workflow uses a subset of the actuators. Each new block type triggers the assignment of the associated VT and migrates from one controller to the next as the block physically moves through the factory. Therefore, with every new type of block a reconfiguration of the VC is performed. We have tested the setup with a batch of 9 input blocks consisting of 3 different types. This is an example of the logical benefits of the EVM as it enables a more agile form of manufacturing. Details on both experiments with videos can be seen in [19].

VII. Case study

We have simulated the performance of the EVM for the case when a wireless network is used to control the Shell Problem, a well-known problem from the process control theory [20], [21]. Fig. 7(a) presents a Simulink framework used for the simulation, where the *Controller* (shown in Fig. 3(b)) and the *Plant* are similar to models from [20]. The major difference is that *Plant*'s dynamics have been sped up in order to be able to test system's performance. The functional description of the VT, shown in Fig. 3(c), is extracted using the technique described earlier. Since all continuous outputs of the *Plant* have to be sampled to be eligible for processing using a discrete controller, the sampling rate defined in the SampleAndHold blocks in Simulink is used to extract the period of each VT. In this case, this extraction is simplified by the fact that all conversions to the discrete domain are performed at the same sampling rate.

Fig. 7(b) presents the initial topology of the VC along with the *Primary* and the *Backup* nodes. To be able to address the effects of message drops, we have assigned each link in the network with a Packet Delivery Ratio (PDR) that is less than 1. For communication between nodes, a TDMA protocol with 32 slots per frame is used, where 24 were used for transfer data related to the control problem, while 8 remaining slots per frame were used to exchange messages about VC's status. The system response to a series of different step inputs (a new one arrives every 60s) for the initial topology is presented in Fig. 7(d). If the initial topology changes after some of the links fail (as shown in Fig. 7(c) but without changing the position of the *Primary* and the *Backup* node), the system without the EVM, where only re-routing algorithms are used, will have a response presented in Fig. 7(e). This results in a system response that rapidly deteriorates. The system becomes unstable due to the increase in end-to-end communication time from all sensors to the *Primary* node to all actuators.

Fig. 7(f) shows how the EVM's adaptation to unplanned changes in link quality allow us to keep the system's response similar to that in the initial topology. For the case presented in Fig. 7(c), we simulated the system at time $t = 60s$. The network topology changes to that presented in Fig. 7(f). Due to the task re-assignment, one execution step of the control algorithm is omitted, but as it can be seen, without significant influence to the overall system dynamics.

VIII. Related work

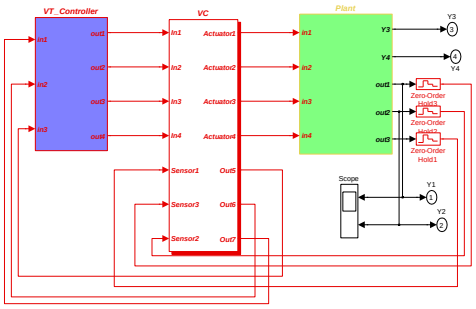
There have been several variants of virtual machines, such as Maté [22], Syclla [23] and SwissQM[24], and flexible operating systems, such as SOS[8], Contiki[25], Mantis[26], Pixie[27] and LiteOS[28], for wireless sensor networks. The primary differences that set the EVM apart from prior work is that it is centered on real-time operation of controllers and actuators. Within the design of EVM's operating system, link protocol, programming abstractions and operation, timeliness is a first-class citizen and all operations are synchronized. The EVM does not have a single node-perspective of mapping operations to one virtualized processor on a particular node but rather maintains coordinated operation across a set of controllers within a VC.

Maté implements a simple, communication-centric VM built on top of the TinyOS [29]. It is designed as a high level interface where code is written using limited instruction set, defined at design-time, and executed with a FORTH-like interpreter. EVM utilizes a similar FORTH-like interpreter but is extensible at runtime and allows for fully preemptive tasks. Syclla is a more conventional system VM that allows code mobility by providing a virtualized processor abstraction on each node. SOS uses dynamically-loaded modules while kernel implements messaging, dynamic memory, and module loading and unloading. EVM allows for dynamic task evocation but in addition has mechanisms for direct interaction with other nodes, which SOS lacks.

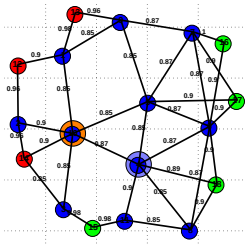
In last few years, several different systems for macro-programming in WSN have been developed[30], [31], [32]. Welsh *et al.* [30] have defined a set of abstractions representing local communication between nodes in order to expose control over resource consumption along with providing feedback on its performance. EVM is not a generic macro-programming system as it focuses on closed-loop control and actuation problems with native support for task migration and on-line task assignment.

IX. Discussion

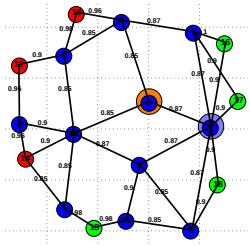
We have investigated several fundamental challenges with the use of wireless networks for time-critical closed-loop control problems. Our approach was to build the networking infrastructure to maintain state across physical node boundaries so tasks are decoupled from the underlying unreliable physical substrate. We present an IP formulation of the runtime task assignment problem and show that it is possible to compute task assignment efficiently and in a composable manner across concurrent control problems. We implemented an initial version of the EVM infrastructure on commodity embedded nodes and demonstrated the capability in an all-wireless factory across 22 sensors/actuators. This paper presents an initial stab at a problem that unravels a series of



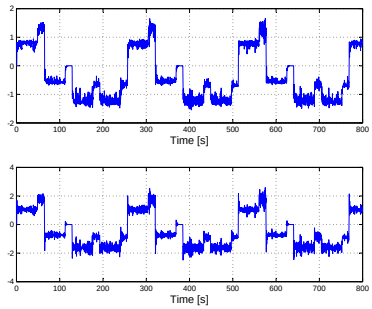
(a) Simulink model for Shell problem



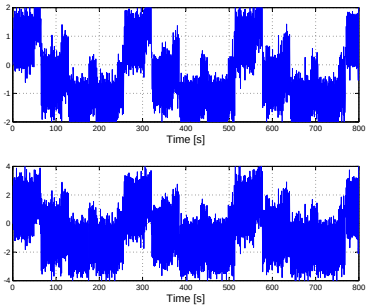
(b) Initial net topology



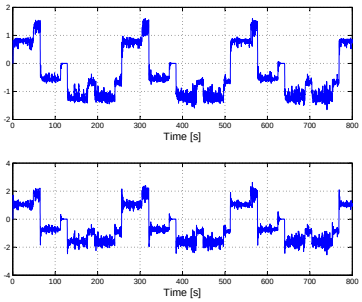
(c) Topo after link failures



(d) System response for initial configuration, showing outputs Y1 and Y2.



(e) System response for when EVM is not used (when only re-routing is used)



(f) System response when EVM adapts to changes in network conditions

Figure 7. Simulation of EVM behavior when used for 'Shell problem' control; Nodes: green nodes - actuators, red nodes - sensors, blue circle - highlights *Primary* node, blue circle - highlights *Backup* node

problems at the heart of network Cyber-Physical Systems. The complexity of reaching consensus limits our current implementation a centralized architecture. Future work will focus on distributed and scalable wireless control networks.

REFERENCES

[1] Frost and Sullivan, North American Sensor Markets, Technical Report A-761-32, 2004.

[2] Nielsen Research, Downtime Costs Auto Industry, 2006.

[3] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu. A survey of recent results in networked control systems. *Proc. of the IEEE*, 2007.

[4] W. Zhang, M.S. Branicky, and S.M. Phillips. Stability of networked control systems. *IEEE Control Sys. Mag.*, 2001.

[5] L. Brodie. Starting FORTH. *2nd Ed., Prentice-Hall.*

[6] nano-RK Sensor RTOS. http://nanork.org.

[7] A. Rowe, R. Mangharam, and R. Rajkumar. RT-Link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hop Wireless Networks. *IEEE SECON*, 2006.

[8] S. Han et. al. SOS : A Dynamic Operating System for Sensor Nodes. *ACM Mobisys*, 2005.

[9] M. Pajic and R. Mangahram. Embedded Virtual Machines: Technical Report ESE-2009-01, 2009.

[10] HART Field Communication Protocol Spec., Rev 7, 2007.

[11] A. Schrijver. Theory of Linear and Integer Programming. *John Wiley & sons, 1998, ISBN 0-471-98232-6*, 1998.

[12] Z. Fu, Y. Mahajan, and S. Malik. New Features of the SAT'04 versions of zChaff . *SAT Competition*, 2004.

[13] T. Bhme, F. Gring, and J. Harant. Menger's Theorem. *Journal of Graph Theory, vol. 37. no. 1*, 2001.

[14] B. Yang, S.Q. Zheng, and E. Lu. Finding Two Disjoint Paths in a Network with α^{+}-MIN-SUM Obj. Function. *PDCS*, 2008.

[15] J.W.S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[16] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode Change Protocols for Priority-driven Preemptive Scheduling. *J. Real-Time Sys.*, 1989.

[17] Jorge Real and Alfons Crespo. Mode Change Protocols for Real-Time Systems: A Survey. *J. Real-Time Sys.*, 2004.

[18] R. Mangharam, A. Rowe, and R. Rajkumar. FireFly: A Cross-layer Platform for Real-time Embedded Wireless Networks. *Real-Time System Journal*, 2007.

[19] EVM website - http://mlab.seas.upenn.edu/evm, 2009.

[20] D.R. Lewin et. al. *Using Process Simulators in Chemical Engineering*. Wiley, 2009.

[21] D.M. Prett and M. Morari. The shell process control workshop. *Butterworths*, 1986.

[22] P. Levis and D. Culler. Mate: A Tiny Virtual Machine for Sensor Networks . *ACM ASPLOS-X* , 2002.

[23] P. Marbell and L. Iftode. Scylla: A Smart Virtual Machine for Mobile Embedded Systems. In *WMCSA*, 2000.

[24] R. Muller, G. Alonso, and D. Kossmann. A Virtual Machine for Sensor Networks. *ACM EuroSys*, 2007.

[25] A. et al. Dunkels. Contiki - A Lightweight Operating System for Tiny Networked Sensors. *IEEE LCN*, 2004.

[26] S. Bhatti et. al. MANTIS OS: An Embedded Multithreaded Operating System. *Mob. Netw. Appl.*, 2005.

[27] K. Lorincz and et. al. Resource Aware Programming in the Pixie OS. *ACM SenSys*, 2008.

[28] Q. Caoand and et. al. The LiteOS: Towards Unix-Like Abstractions for WSNs. *IEEE IPSN*, 2008.

[29] J. Hill et. al. System Architecture Directions for Network Sensors. *ASPLOS*, 2000.

[30] M. Welsh and G. Mainland. Programming Sensor Networks using Abstract Regions. *NSDI*, 2004.

[31] R. Gummadi et al. Macro-programming Wireless Sensor Networks Using Kairos. In *Dist. Comp. in Sensor Sys.*, 2005.

[32] R. Newton et al. The Regiment Macroprogramming System. *ACM IPSN*, 2007.