Embedded Virtual Machines for Robust Wireless Control Systems

Rahul Mangharam and Miroslav Pajic
Dept. of Electrical & Systems Engineering
University of Pennsylvania, U.S.A.
{rahulm, pajic}@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 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 the design of a natural gas process plant hardware-in-loop simulation we aim to demonstrate the preliminary capabilities of EVM-based wireless networks.

Keywords: Real-time systems, embedded systems, wireless sensor networks, virtual machines.

1. Introduction

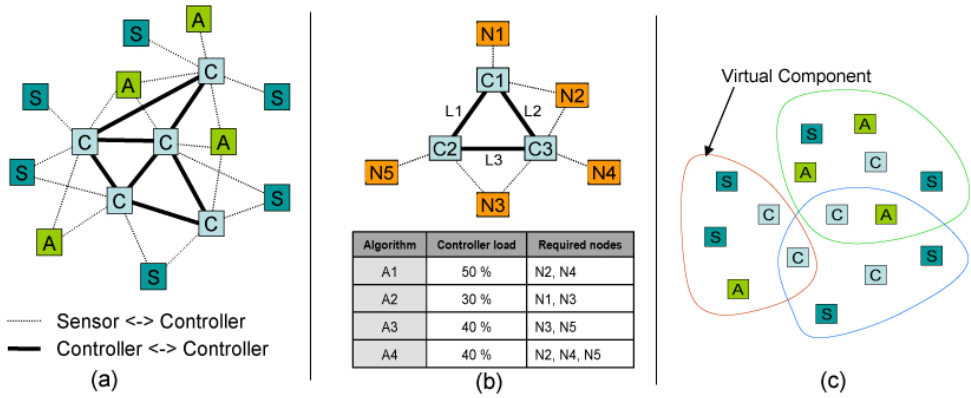
Automation control systems form the basis for significant pieces of our nation's critical infrastructure. Time-critical and safety-critical automation systems are at the heart of essential infrastructures such as oil refineries, automated factories, logistics and power generation systems. Discrete and 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, automation manufacturers and 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.

While WSAC networks facilitate both planned and unplanned mode changes, runtime programmable WSAC networks allow for flexible item-by-item process customization. For example, a high demand for fuel-efficient Toyota Prius' will require major retooling of a traditional wired factory that is designed for the Toyota Camry chassis. With re-programmable WSAC, the assembly line stations can adapt to a schedule where every 3 Camrys are interleaved with 2 Prius' with synchronized changes in operation modes and assembly line operations.



Algorithm	Controller load	Required nodes
A1	50 %	N2, N4
A2	30 %	N1, N3
A3	40 %	N3, N5
A4	40 %	N2, N4, N5

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

1.1. 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. 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.

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. EVMs differ from classical virtual machines (VM). 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 a 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* as shown in Fig. 1. A Virtual Component is a composition of interconnected communicating physical components defined by object transfer relationships. At runtime, nodes determine (via centralized or distributed algorithms) the task-set and operating points of different controllers in the Virtual Component. This machine-to-machine coordination require task-set generation, task migration and remote algorithm activation which are executed via synthesis at runtime.

4. *Adaptive 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. For example, if an assembly line is to process two types of units, red units and blue units, it must ensure that the additional processing time required for blue units does not violate the processing of red units along the shared conveyor belt.

1.2. Research Challenges

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 five 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. It is hard to provide any analytical bounds on the response time, stability and timeliness of tasks in an event-driven regime [3, 4]. Real-time tasks are time-triggered while sensor inputs are event-triggered. It is generally easier to incorporate sporadic tasks in a time-triggered regime than vice versa.

2. Programming of sensor networks is currently at the physical node-level where the tasks are bound to the node at compile-time. This makes it non-trivial to decompose a large control problem into defining components and applications for each mote. In the case of sensor network virtual machines such

as Mate [4], Scylla [5] and SwissQM [6] and runtime programming frameworks such as SOS [7] and Contiki [8], the interaction is assumed to be between an end-user and a single isolated node in a network and not among the nodes themselves.

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. Thus, a change in the link capacity, node energy level or connectivity in the current topology will render the application useless. It is necessary to associate a logical mapping of tasks to nodes and incorporate mechanisms to transfer responsibilities during physical and environmental changes in the network.

4. Fault diagnostics, repair and recovery are manual and template-driven for a majority of networked control systems. Approximately 30% of the code in automation systems is dedicated to fault detection and recovery. In the case of WSAC networks, it is not plausable to exhaustively capture all possible faults at design time and thus provisions must be made for runtime diagnostics and recovery.

5. Template-driven Safety: A majority of automation systems use 'if-then' template-driven statements to detect safety. With frequent code patches, it is hard to provide safety guarantees. Any change in topology or the number of associated nodes may violate the fixed safety rules which are determined at design-time. Nodes must operate in tandem where the performance and operational safety of one node is continuously monitored by others and vice versa.

2. Background and Preliminary Work

The EVM architecture and algorithms are built on a modified version of the FireFly sensor network platform [9] and nano-RK sensor real-time operating system (RTOS) [10]. The EVM is implemented in the form of a virtual machine abstraction layer on top of the RTOS and executes as a special task within nano-RK. As a special task, the EVM has both parametric and programmable control of the entire operating system and hardware resources. We describe below the current developments and experiences of the FireFly platform and nano-RK RTOS and also the preliminary investigations with the EVM.

2.1. Embedded Network Platforms for Time Synchronized Communication

Several platforms, such as Mica2, MicaZ, Telos, ExScale and TinyNode [11], that have enabled sensor networks are available. Many of these platforms are based on the component-based, event-triggered operating system and application framework called TinyOS [3]. While this framework is flexible, timing predictability and fine-grained deterministic resource control were not its primary design objectives.

We use the FireFly platform [9] which is designed to support real-time sensor networking applications [12, 13]. The FireFly node shown in Fig. 2 is a low-cost, low-power, platform that is 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. A FireFly node can also operate with a solar cell driven by ambient light. Each node supports and expansion card with light,

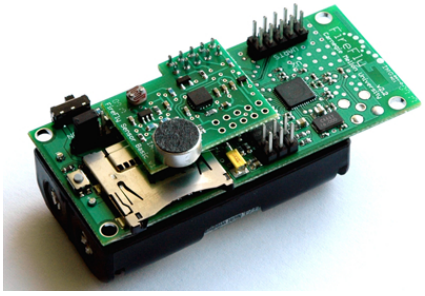


Figure 2. FireFly node with sensors & AM time sync

temperature, audio, passive infrared motion, dual axis acceleration and voltage sensors.

The primary reason we use FireFly for EVMs is for its ability to support tight global hardware-based time synchronization for real-time TDMA-based communication with the RT-Link protocol [12]. FireFly nodes are able to achieve sub-150μs jitter by using a passive AM radio receiver. Through the tight time synchronization of RT-Link, it has been demonstrated to have an effective battery lifetime of 1.8 years with a 5% duty cycle. RT-Link outperforms asynchronous protocols such as B-MAC [14] and loosely synchronous protocols such as S-MAC [15] across all duty cycles and event rates. We have demonstrated real-time two-way interactive voice streaming across multiple FireFly nodes using the RT-Link protocol [13]. With RT-Link, communication for real-time applications is collision-free and is scheduled in well-defined TDMA slots that ensures timely communication between nodes within an EVM's Virtual Component.

2.2. Real-Time Sensor Operating System as a basis for the EVM

To address the need for timing precision, priority scheduling and fine-grained resource management the nano-RK resource kernel [10] was developed with timeliness as first-class citizens. nano-RK is a fully preemptive RTOS with multi-hop networking support that runs on a variety of sensor network platforms (8-bit Atmel-AVR, 16-bit TI-MSP430, Crossbow motes, FireFly). It supports fixed-priority preemptive scheduling for ensuring that task deadlines are met, along with support for and enforcement of CPU and network bandwidth reservations. Tasks can specify their resource demands and the operating system provides timely, guaranteed and controlled access to CPU cycles and network packets in resource-constrained embedded sensor environments. It also supports the concept of virtual energy reservations that allows the OS to enforce energy budgets associated with a sensing task by controlling resource accesses. nano-RK provides various medium access control and networking protocols including a low-power-listen CSMA protocol called B-MAC, an implicit tree routing protocol and RT-Link.

For networked control systems, it is essential that the underlying sensor operating system expose precision timing, scheduled tasks and synchronized networking so that the trade-offs between energy-consumption (node lifetime), reliability and responsiveness are specifiable and enforceable both at design-time and runtime. Support for the above services is required for low-duty cycle and energy-constrained sensor networks too

because the computation and communication are packed into a short duration so all nodes may maximize their common sleep time. As shown in Fig. 3, the EVM is built upon nano-RK and adds the capability for a suite of runtime services with parametric and programmable control.

3. EVM Architecture and Algorithms

The system under consideration includes a number of wireless sensors, actuators and controllers composed into a Virtual Component. The Virtual Component acts as a single entity for the control algorithm execution. The EVM provides a flexible programming abstraction to share state and responsibilities across physical nodes and allows multiple EVM-enabled nodes to be composed into a single logical entity.

Control algorithms are automatically distributed across physical nodes based on computing load and proximity to the corresponding sensors and actuators. Multiple copies of each algorithm are present on the physical nodes and state is shared either passively or actively to enable fault tolerance. Control algorithms 'spawn' automatically proliferating to nodes capable of executing them and maintain a common state at all times. If one of the nodes executing control algorithm fails, another node capable of performing the same control function takes over control execution. Algorithm migration from one physical node to another is a key feature of this system. Control algorithm execution by one node is passively observed by other nodes capable of executing the same algorithm. Control algorithm failure is detected by backup observers and a new master is selected based on an arbitration algorithm.

3.1. EVM Architecture

We now consider the design of the EVM within the nano-RK RTOS framework. The EVM describes its own instruction set for efficient control, task and fault management between nodes. As with Mate, the EVM is based on a FORTH-like interpreter. The interpreter runs within nano-RK as a super task. However, unlike Mate, the EVMs instruction set is extensible at runtime. Furthermore, EVM instructions are focused on node-to-node communication and control rather than PC-to-node control. We describe two main architectural components within the EVM - EVM node-specific operations and object transfers for efficient node-to-node communication.

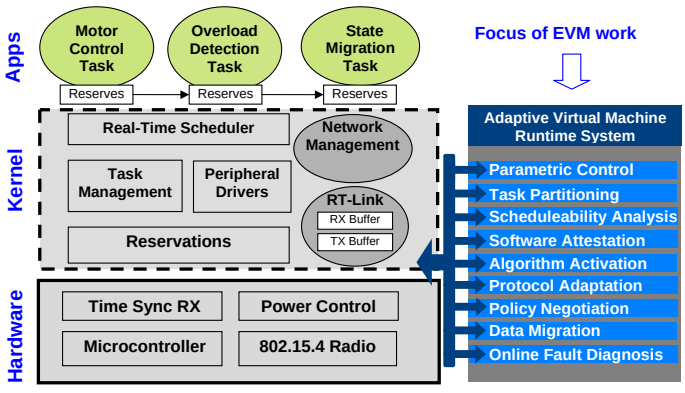


Figure 3. nano-RK sensor RTOS with interfaces to the EVM. EVM includes parametric and programmable control algorithms for runtime logical-task to physical-node mapping.

3.1.1. EVM Node-specific Operations

The EVM is responsible for the following core node-specific operations. The parametric control has been implemented as an EVM library for core pre-defined instructions. The programmable control will be implemented as a runtime service and requires hooks within the kernel, device drivers and link layer.

1. Runtime Task Management This includes basic task allocation, assignment and manipulation. The specific operations supported by the EVM are task assignment to a particular node, task migration from one node to another, task partition from one node to another and itself and finally task replication where an instance of a task is also invoked on another node (using the same state information, stack and register settings).

2. Runtime Resource allocation This operation facilitates allocation or re-allocation of a task control block and reservation with the scheduler and network for a new task or for an existing task on the local node.

3. Scheduling and schedulability analysis This operation is invoked when there has been a change to the scheduler or task-set on a node. The new task-set or schedule will only be activated if the schedulability test is passed. This ensures that all tasks are schedulable within the scheduler's utilization bounds even after a new task is added.

4. Priority assignment This parametric control operation allows a node to re-prioritize its tasks upon the admission of a new task or change in operating conditions.

5. Fault/failure detection and adaptation This handler is activated when a fault message is received by the kernel and the desired action is carried out. An example of this would be when a fault message informs the kernel that the battery is out of energy and the kernel activates a task migration operation to move operations to a more able node.

6. Node membership and data migration The membership of a Virtual Component is not fixed. If new nodes are present they are admitted to the Virtual Component. This operator ensures that the requirements of new nodes or the network state of surviving nodes is stable. Furthermore, this operator invokes the optimization sub-routine if more resources are added to the Virtual Component's resource pool.

7. Run-time optimization This operation executes optimization of resource allocation and task assignment at runtime. We use Binary Quadratic Programming for fixed-point optimization for functional and para-functional requirements across controller nodes. Due to space limitation we will not discuss this in detail.

8. Software attestation When new code or data is received by a node from another node, the node executes a basic attestation test to ensure the code/data is not corrupted and passes the schedulability test.

While the above operations are not exhaustive, we will select the ones that matter the most in our case studies and test them under changing conditions with large dynamic ranges.

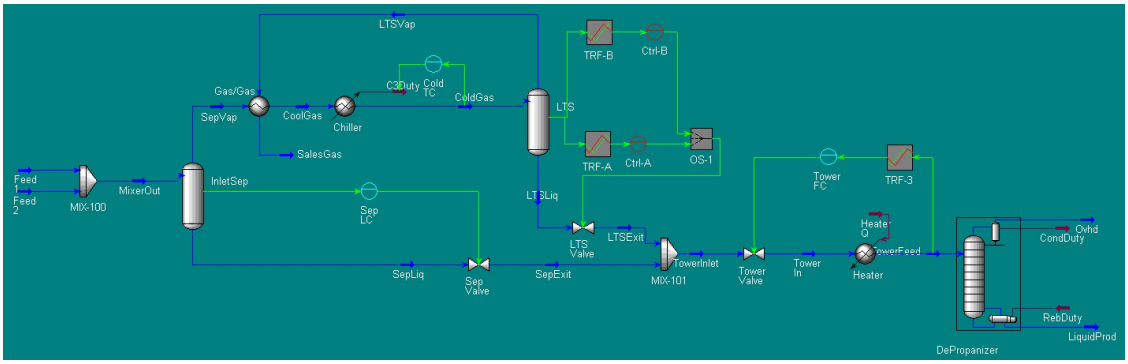


Figure 4. Unisim model for a natural gas process plant

3.1.2. EVM Object Transfers

We now describe the mechanisms used to communicate control, data and fault information between controllers within a virtual component. Five elementary object transfer types are included in the EVM design. These include: disjoint, bi-directional transfers, temporal-conditional transfers, causal-conditional transfers and health assessment.

A disjoint relation between two nodes indicate that the nodes may operate concurrently in both temporal and spatial domains without any shared state. Directional and bi-directional transfers define relationships such as master-slave, publish-subscribe and producer-consumer. This is the basic transfer type for all active controllers within a virtual component. Temporal and causal transfers define the type of relationship between interconnected controllers and enforce a set of restrictions between the controllers. Finally, health assessment transfers are used for monitoring and tracking and define which node is the primary or backup and the nature of response to faults such as trigger alert, trigger backup, halt and local fail-safe operation.

4. EVM Evaluation

We have implemented the parametric control capability of the EVM on the FireFly nodes over the nano-RK sensor RTOS. This allows remote runtime triggering of individual sensor drivers, modification of task reservations and network time-slot assignment. Through a process control case study, we evaluate the programmable control, more specifically the fault tolerant capability, of the EVM. We employ the Honeywell Unisim plant simulator with hardware-in-loop via a a set of six interconnected FireFly nodes, as shown in Fig. 5. Each sensor, controller and actuator node interfaces with a gateway node via RT-Link. The gateway communicates with Unisim (on the workstation) via ModBus. The controllers operate on information generated by the plant simulation and sensor I/O for a realistic closed-loop WSAC evaluation. This allows us to evaluate the network with large dynamic input ranges and dramatic topology changes.

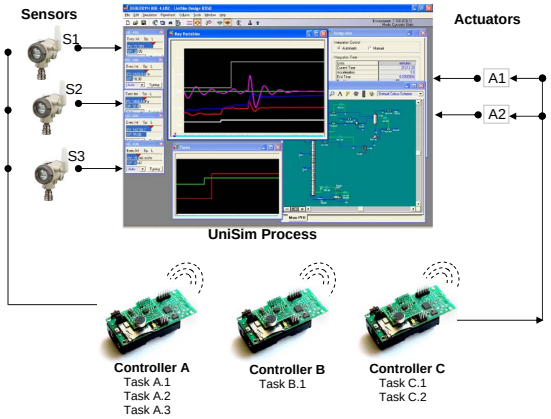


Figure 5. EVM evaluation with wireless networked hardware-in-loop simulation

Our focus is on the fault-tolerance of controllers only, all of which are connected with wireless connections to each other and to the physical sensors and actuators that interface to Unisim. When a particular backup controller detects a series of faults in the primary controller, it triggers a task migration operation to the backup controller. This operation includes a capabilities check and the migration of the task control block, stack, data and timing/precedence-related metadata. The backup controller is activated and the primary controller switches to a passive 'indicator' mode.

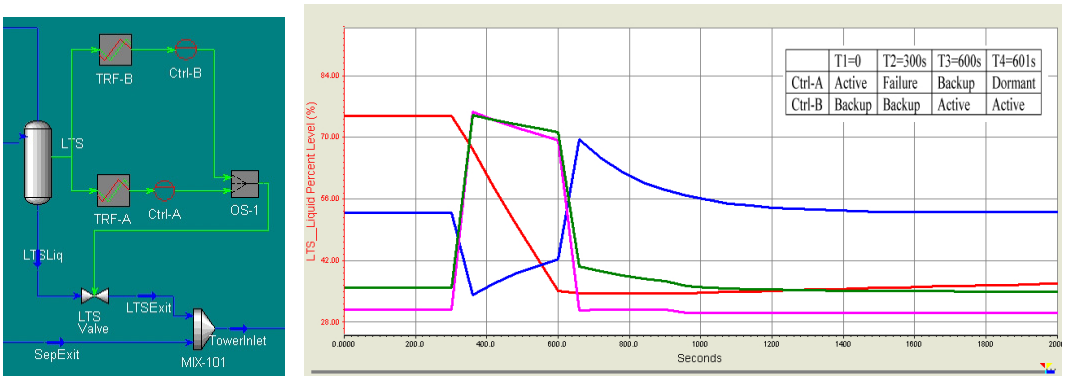
4.1. Natural Gas Plant Model

We employed a Unisim model for a natural gas processing application. This case study models a natural gas processing facility that uses propane refrigeration to condense liquids from the raw natural gas feed and a distillation tower to process the liquids. The flowsheet for this process is in Fig. 4. In this plant, a raw natural gas stream containing N_2, CO_2, and C_1 through n-C_4 is processed in a refrigeration system in order to remove the heavier hydrocarbons. The liquids removed from the input stream yield to a liquid product that has desired propane content.

As shown in Fig. 4, multiple input raw natural gas feed streams are combined before entering Inlet Separator (InletSep) that removes free liquids from them. Overhead gas from the Inlet Separator is combined in the gas/gas exchanger with already cooled gas in order to decrease its temperature. The cold stream from chiller is introduced to the Low-Temperature Separator (LTS), which separates the heavy hydrocarbon liquid from its input stream, while remaining gas is fed back to the gas/gas exchanger. Liquid output of LTS is mixed with free liquids from the Inlet Separator, $InletSep$. These liquids are then processed at the Depropanizer column to produce a low-propane-content bottoms product.

4.2. Fault-Tolerant Wireless Controllers

The plant model has a several control loops (presented with light green connections). In considered application 8 different controllers are used (4 in top level system and 4 in De-



(a) (b)

Figure 6. (a)Primary and backup controllers for the Low Temperature Separator. (b)Process control outputs during primary controller failure (300s), recovery (600s) and activation of backup controller (at 601s). Legend: LTS-Liquid Percent Level (red), SepLiq-Molar Flow (blue), LTSLiq-Molar Flow(magenta), TowerFeed-Molar Flow(green)

Propanizer). These controller algorithms are implemented using EVM across multiple physical nodes.

To show performance of designed EVM we will focus on the controller for valve at liquid flow from LTS output and TowerInlet (Fig. 6(a)). In the presented configuration, 2 physical controllers, Ctrl-A and Ctrl-B, implement the control algorithm as primary and backup controllers respectively. The liquid's percentage level in LTS is used as an input to the controllers, which perform second order filtering with a PID regulator. The operation switch, OS-1, determines which controller's output should be connected to the valve.

To demonstrate fault-tolerant operation with the EVM, the scenario in Fig. 6(b) is used. Before time T1, Ctrl-A is in Active mode and actually controls the valve's output level. This configuration is valid till the moment T2 when, due to a failure, Ctrl-A sets a wrong valve output level (75% instead of 11.48%). This is seen in the rapid drop of the liquid percent level and variation of the liquid level in the separator. At time instance T3, after the node Ctrl-B (which is in the Backup mode), determines inappropriate outputs from Ctrl-A and informs the head of the Virtual Component, the VC sets Ctrl-B in Active mode, while Ctrl-A goes to Backup mode. Finally at the end of this transition, the Ctrl-A node is set to the Dormant mode. After the stable system configuration is restored with the introduction of Ctrl-B output, liquid level in LTS starts to recover slowly. During the on set of the fault, the rapid increase in LTS valve's output level introduced significant changes in molar flows of the LTS, Separator and Tower Feed liquids. But after system's reconfiguration these values were restored to the previous ('stable') values. While the changes in the process are along relatively long time intervals (100s of seconds), our goal is to demonstrate the flexible logical to physical mapping of tasks and the runtime adaptation to system, network and environmental changes. As future work we aim to implement a suite of programmable control runtime capabilities for distributed fault-tolerance and reconfiguration.

In summary, the specific objectives of this effort are:

1. Ability to deploy control algorithms in a virtual component defined over a grid of wireless controllers.

2. On-line capacity expansion where more controllers can be added to share the load and trigger re-distribution of tasks.

3. Algorithm replication to a set of nodes capable of performing the same control function for throughput adaptation.

4. Fault tolerance to node and communication failures.

5. Control algorithm execution with high-speed operation (1/4 second or less control cycle) and with a small latency ($\leq$1/3 of the control cycle).

References

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

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

[3] J. Hill et. al. System architecture directions for network sensors. *ASPLOS*, 2000.

[4] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks . *ACM ASPLOS-X* , 2002.

[5] P. Marbell and L. Iftode. Scylla: A smart virtual machine for mobile embedded systems. In *WMCSA*, 2000.

[6] R. Müller, G. Alonso, and D. Kossmann. A virtual machine for sensor networks. In *ACM EuroSys*, 2007.

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

[8] A. Dunkels and N. Finne and J. Eriksson and T. Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. *ACM SenSys*, 2006.

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

[10] nano-rk sensor rtos. http://nanork.org.

[11] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy. Platforms enabling wireless sensor networks . *Communications of the ACM, 47(6):41-46*, 2004.

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

[13] R. Mangharam, A. Rowe, and R. Rajkumar. Voice over Sensor Networks. *RTSS*, 2006.

[14] J. Polastre, J. Hill, and D. Culler. Versatile Low Power Media Access for Wireless Sensor Networks. *ACM SenSys*, 2005.

[15] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. *INFOCOM*, June 2002.