An In-Field-Maintenance Framework for Wireless Sensor Networks ⋆

Qiuhua Cao and John A. Stankovic

Department of Computer Science
University of Virginia
{qhua, stankovic}@cs.virginia.edu

Abstract. This paper introduces a framework for in-field-maintenance services for wireless sensor networks. The motivation of this work is driven by an observation that many applications using wireless sensor networks require one-time deployment and will be largely unattended. It is also desirable for the applications to have a long system lifetime. However, the performance of many individual protocols and the overall performance of the system deteriorate over time. The framework we present here allows the system or each individual node in the network to identify the performance degradation, and to act to bring the system back to a desirable coherent state. We implement and apply our framework to a case study for a real system, called VigilNet [5]. The performance evaluation demonstrates that our framework is effective and efficient.

1 Introduction

Many applications [5] in wireless sensor networks (WSN) typically initialize themselves by self-organizing after deployment. At the conclusion of the self-organizing stage it is common for the nodes of the WSN to know their locations, have synchronized clocks, know their neighbors, and have a coherent set of parameter settings such as consistent sleep/wake-up schedules, appropriate power levels for communication, and pair-wise security keys. However, over time these conditions can deteriorate.

The most common (and simple) example of this deterioration problem is with clock synchronization. Over time, clock drift causes nodes to have different enough times to result in application failures. While it is widely recognized that clock synchronization must re-occur, this principle is much more general. For example, even in static WSN some nodes may be physically moved unexpectedly. More and more nodes may become *out of place* over time. To make system-wide node locations coherent again, node re-localization needs to occur (albeit at a much slower rate than for clock sync).

Many WSN protocols have similar characteristics with respect to the need for re-applying computation. Reasons for this include the decentralized nature

⋆ Supported in part by NSF grants CNS-0435060, CNS-0626616, and CNS-0720640.

of many protocols in WSN, the large scale of these systems and a high degree of loss and uncertainty experienced in these systems. WSN systems are in need of systematic mechanisms to dynamically adjust themselves at runtime based on the current performance. Moreover, many systems require mechanisms to understand which set of protocols/services are contributing to the degradation of the system performance. For example, typical tracking applications [1] might notice that the accuracy of the object classification is not acceptable, and many of these applications consider that the classification protocol is the only cause. However, the routing and localization protocols also have the impact on the classification accuracy. The unacceptable accuracy may be due to a low packet deliver ratio or incorrect location information in the detection report messages from the nodes to the base station.

In this paper we consider the entire set of services that keep or restore system coherence as *in-field-maintenance services*. We call these maintenance services since they follow the concept that systems tend to disorder unless explicit action is applied to keep them ordered. In other words, specific energy (in the form of executing specific code) has to be applied to keep or regain system-wide coherency.

Our framework allows users to (1) define the coherency requirements of the states of protocols, (2) specify the maintenance/repair policy (i.e., when to self-heal the system, and who invokes the maintenance service), and (3) define which set of protocols need maintenance services in order to satisfy a system performance requirement and the dependency constraints among the services, according to the system specification and constraints. And our framework (1) enforces the dependency requirements among maintenance services, (2) supports different memory requirements for different maintenance services, (3) implements an online monitor to measure system performances and states of protocols, and (4) provides mechanisms for on-demand real-time self-healing of the system.

However, the resource constraints in terms of memory, energy, and bandwidth of WSN systems present challenges to implement such a framework. We address the challenges in our framework by supporting different memory management strategies to satisfy different maintenance policies specified by users, by enabling both global (the whole network) and local (a specified region) maintenance services so that global maintenance services are only invoked when needed, by executing only the necessary maintenance services via the dependency enforcement instead of maintaining all the protocols in stack at each time a maintenance service required, and by implementing both global and local monitors and piggybacking report messages in the monitors.

2 Our In-Field-Maintenance Framework

Our framework works side by side with system protocols and application code as shown in Figure 1. We present the details of the two main components of our framework in subsections 2.1 and 2.2, respectively.

2.1 Maintenance Policy

The maintenance policy component takes the system specification and defines: (1) the conditions under which a maintenance service is to be invoked, (2) the dependency constraints of a particular maintenance service, (3) the region a maintenance service is to be applied, i.e., global or local, and (4) the state update policy to define when the states of a protocol are to be updated. The first 3 sub-components take energy efficiency into consideration when defining what to support in the framework. The 4th sub-component deals with the memory constraints imposed by WSN systems.

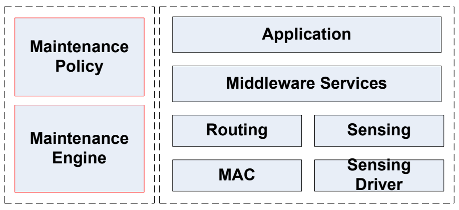


Fig. 1. System Architecture

Conditions. Maintenance services might be triggered when there is (1) a system sensing coverage failure, or (2) a system communication coverage failure, or (3) a system density coverage failure or periodically.

Dependency Constraints. Dependency constraints define (1) the relationship between a system performance measurement and the protocols contributing to the measurement, and (2) the dependencies among protocols in a WSN system. Consequently, the constraints define the dependencies among the maintenance service for each system measurement and for each protocol. The dependency specifications are defined according to the system requirements and look similar to dependency graphs in the real-time scheduling research domain (but cannot be shown here due to lack of space). Briefly, the specifications define the sequence of a set of protocols to be executed.

As a result of dependency specifications, two energy inefficient situations are avoided. The first is avoiding executing unnecessary maintenance services. Two, avoiding invoking a maintenance service unnecessarily.

Regions. The region specification for a maintenance service gives the system the flexibility to accommodate different strategies for different services to minimize the energy consumption of each maintenance service. The coherent states of the protocols in WSN are organized into two categories:

Global View of Coherent States – requires the states of the protocols to be the same value within some acceptable range for every node in the system. For example, time synchronization requires every node in the system to have the consistent time view with the base station within some ϵ.

Local View of Coherent States – the local view of coherent states is defined from the node's point of view, i.e., by whether a node perceives the true state in the system. For example: node A thinks that node B is its neighbor, but actually node B is either out of its communication range due to the change of the communication quality, or unexpected movement (i.e, wind blows node B

away), or node B is not functional due to running out of energy. In this case, node A does not have a coherent view of its local neighborhood states.

The two different types of coherency requirements require different capabilities to bring the system back to coherent states. Our in-field-maintenance framework is designed to provide the flexible and efficient approaches to handle both cases.

Guidelines for choosing a global service are when: (1) all the nodes in the network demonstrate similar properties over a period of time. If each node in the system has similar clock skew or drift, then it is reasonable to globally run the time synchronization maintenance service to resynchronize the clocks; (2) applications or maintenance policies require all the nodes in the network to contain the same information or to take the same action. If an application or a maintenance policy requires updating the report rate of all the nodes in the network with a new rate; (3) replacing a protocol with a new implementation. In order to defend against a newly expected security attack to the routing algorithm currently deployed in the network, it is desirable to globally disseminate a new routing algorithm to all the nodes. Otherwise, the nodes with different routing algorithms may not be able to communicate.

The recommendations to choose local services are when: (1) the states of concern only have local meanings. For example, the link quality of a node to its neighbors only makes sense to be defined as a local parameter; (2) an application or a maintenance policy only requires local repair.

Update Policies. In WSN, the memory of sensor nodes (128K ROM, 4K RAM for MICA series devices) is very limited. When applications become more complex, the memory is even a bigger concern. Moreover, some applications may require that the system does not stop from its normal processing while maintaining the system. The maintenance service requires extra available memory to be allocated. Our framework supports three update policies to give the system the capability to balance the trade-off between memory constraints and event detection delays.

Delay Update until Commit Time: This policy delays the update of the states to right before the conclusion of the maintenance process for a given protocol. It does not stop a system from its normal processing, but requires the maintenance service to allocate memory space to back up the old states.

Immediately Update, but Disable Sensor Interrupts: Here when a maintenance service is invoked, all sensor interrupts are disabled. The result is that the maintenance service does not demand substantial memory and race conditions don't exist. But the applications can not detect any new sensor events during the execution of the service.

Immediately Update, but do not Disable Interrupts: This policy delays the sensor reports until the end of the maintenance service, but allows sensor interrupts during the execution of the maintenance service. The sensor readings are stored in a buffer, so that no event is missed, but the time to report the events to the applications is delayed.

2.2 Maintenance/Repair Engine

The maintenance engine implements the policies discussed in subsection 2.1. It is composed of 4 parts: monitors, dependency checks, memory management, and in-field-maintenance services. We next discuss the functionality of each.

Monitors. Monitors collect the specified states of concern and initiate maintenance services according to the policy specification. Our framework designs a two tier monitoring architecture as shown in Figure 2. The global monitor enables the base station to collect and process the performance and/or state information from each individual node in the network via the collector. The processed information then feeds into the controller. The controller generates the list of the protocols to execute maintenance services and floods the list to the network. Each node runs a local monitor. The local monitor acquires the information on what to monitor from the base station, collects the requested information through the local collector, and reports the requested information back to the base station if required.

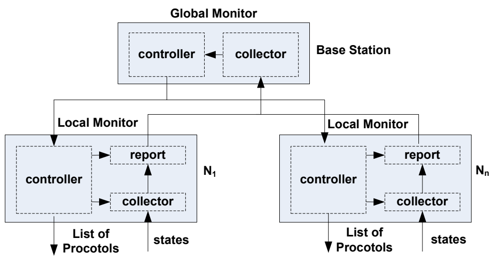


Fig. 2. Two Tire Monitor Architecture for the Global In-Field-Maintenance Policy Enforcement

The monitors are capable of collecting specified state information in 5 categories. And the 5 categories comprehensively consider different data required and system conditions so that our framework can flexibly be integrated with many existing systems. (1) States that are available via directly interfacing with the hardware layer, for instance, energy remaining, and the clock. (2) States that are obtainable through the interfaces provided in the original system without the need to interact with the nodes in the neighborhood, such as the maximum number of neighbors in a node's neighbor table or the maximum number of parents of a node. (3) States that require the cooperation among the nodes in a neighborhood with explicit message exchanges. Link quality is one example of those states. (4) States that are specified as the states to be monitored but the original system does not provide interfaces to expose those states. (5) States that are not maintained by any components in the original system.

Dependency Checks. This component checks if a maintenance service has to be executed together with some other services according to the dependency constraints specified in subsection 2.1, and generates the correct execution flow for the set of services. It also provides mechanisms to enforce the dependencies when the maintenance services are invoked.

Memory Management. To support the "Delay Update until Commit Time" policy specified in subsection 2.1, our memory management provides three primitives to manage the states of maintenance services. The primitives are:

– MaintenanceAlloc: allocates memory to backup the state information of a protocol before executing the maintenance service for the protocol.
– MaintenanceCommit: makes the new state information accessible to the applications right before the maintenance service finishes its execution.
– MaintenanceRelease: releases the allocated memory locations after the commitment.

In-Field-Maintenance Services. This component provides the mechanisms to execute the requested maintenance services.

3 A Case Study

In this section, we present one implementation of our framework for a real application, VigilNet [5] [6] [7]. We choose VigilNet because it is a typical surveillance and tracking application. Note that our framework is applicable to many other types of applications.

3.1 Brief Overview of VigilNet

VigilNet is a recent major effort to support long-term military surveillance using large-scale micro-sensor networks. The primary design goals of the VigilNet system are to detect events or moving targets appearing in the system, to keep track of the positions of moving targets, and to correctly classify the detected targets in an energy-efficient and stealthy manner. Power management and collaborative detection are two key research problems addressed in VigilNet. As discussed in [8], VigilNet provides a three level power management service, namely tripwire service, sentry service, and duty cycle scheduling.

After initialization, the tripwire service divides the system into multiple tripwire sections. A tripwire section can be either in an active or a dormant state. The uniform discharge of energy in a section is achieved through rotation strategies based on the remaining energy within individual nodes. Together with the motivation to balance the energy consumption among all the nodes in the network, at the same time to synchronize the time of all the nodes in the network with the base station, and also to heal any transient node failures, VigilNet implemented a rotation scheme to reinitialize the whole network for all the services once per day.

The timeline to control the rotation is given by the phase transition graph [5]. VigilNet starts with system initialization at phase I and follows the phase transition graph through phase VIII. The duration of each phase is a control parameter that is dynamically configurable at the base station. The initialization process from phase I to phase VII normally takes 3 minutes, but it is a tunable parameter according to the network size and system requirements. As we can see that during the rotation/reinitialization process, the system stops functioning to reinitialize itself.

3.2 One Application on VigilNet

Different from the execution sequence of the original VigilNet system, the new VigilNet system we implemented, called SelfHealingVigilNet, executes as shown in Figure 3. When the system transits into the tracking phase (phase VIII), SelfHealingVigilNet also enters to the maintenance service phase which is in parallel with the tracking phase. There is no rotation service in SelfHealingVigilNet as it would be redundant.

Maintenance Policies in SelfHealingVigilNet: We define the maintenance policy in SelfHealingVigilNet based on the system specification of VigilNet, (1) Longevity, (2) Effectiveness, (3) Adjustable Sensitivity, and (4) Stealthiness. In this case study, our maintenance policy focuses on the longevity and the effectiveness of the system. More specifically, time synchronization and localization services are important for the effectiveness of the system. VigilNet requires the clock drift to be confined to the millisecond range. We use the same maintenance policy for the time synchronization protocol, once per day. Due to practical constraints, each node in VigilNet obtains its location at system deployment time.

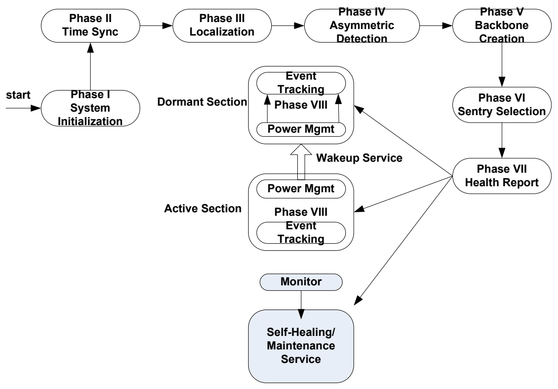


Fig. 3. Phase Transition in SelfHealingVigilNet

Besides time synchronization (Time Syn) and localization protocols, sensing coverage (Sensing Cov) and communication coverage (Comm Cov) are also crucial to the system performance. The sensing coverage depends on the sentry selection protocol. The communication coverage involves protocols such as a asymmetric detection protocol (Asym D), and a backbone creation/robust diffusion tree protocol (R. Diff Tree). The definitions of sentry selection, asymmetric detection, neighbor discovery and backbone creation are the same as defined in [5]. For the case study of our framework, we specify the minimum sensing coverage requirement as 1 (100%), and the minimum communication coverage requirement as 1 (100%).

We also observe that time synchronization, backbone creation/robust diffusion tree creation, and self-configuration of the system need system-wide broadcast. Sentry selection (Sentry Sel) can be global or local. The region of a local maintenance service (Local R) is defined by hop counts k, meaning that a local maintenance process takes effect in its initiator's k hop neighborhood.

In summary, the maintenance policies in SelfHealingVigilNet for the case study are shown in Table 4(a). And the maintenance policies also demonstrate that our framework is capable of supporting combinations of different condi-

	Time Syn	Sensing Cov	Comm Cov
Condition	1day	1	1
Dependency	No	Sentry Sel	Asym D R. Diff Tree
Region	Global	Local R	Global
Update P	Delay	Disable Int.	Immediate U

(a) Maintenance Policy.

```
BPeriod=beaconP;CollectorBTimer(CBT)=T-beaconP;
ReportTimer = T; ConnectivityTable(CT) = ø;
CBT.fired() {Broadcast(CollectorBeaconMsg (CBM));}
RecvedCollectorBeaconMsg(CBM) {Update CT;}
ReportTimer.fired() {
    signal ConnectivityReady to the report component}
```

(b) Pseudo Algorithm of the Local Collector.

tions, different dependency check requirements, different maintenance regions, and different state update strategies to invoke a maintenance service. In Table 4(a), Delay means "Delay Update until Commit Time", Disable Int. stands for "disable sensor interrupts", Immediate U is for "Immediately Update but do not Disable Interrupts".

Monitors in SelfHealingVigilNet: The base station initiates the maintenance process on time synchronization when the invocation condition is satisfied, which is the pre-calculated period based on the one-hop clock drift/skew and the number of the hops in the network. For the communication coverage, we implement a two-tier monitor as discussed in subsection 2.2. The local collector on each individual node collects the connectivity information using the algorithm as shown in Figure 4(b) and reports the information back to the collector on the base station very period T. Period T and beacon period (BeaconPeriod) are specified by the base station and tunable.

The communication coverage of the network is computed at the base station by constructing a connectivity graph based on the connectivity information reported from the nodes. As we observe from the VigilNet system, the nodes on the communication backbone are also the sentry nodes. To be more energy efficient, we piggyback the collecting of the states of the sentry selection protocol and the connectivity.

Dependency checks in SelfHealingVigilNet: The maintenance service for the time synchronization protocol is independent from all the other protocols in VigilNet. The maintenance service for the sentry selection protocol also can be independent. However, when the base station starts the maintenance process to heal the communication coverage, it has to run the service for the asymmetric detection protocol first, then the backbone creation/robust diffusion tree protocol. These dependencies are declared and then followed by the in-field-maintenance framework.

4 Performance Evaluation

We organize our performance evaluation into two parts. First, we demonstrate that our framework works with VigilNet effectively by implementing SelfHealingVigilNet on XMS2 motes, the supported platform by VigilNet. Second, we use the overhead, energy saved, system blackout time, and event detection probability as the performance metrics to analyze our framework. We choose the latter 3 metrics because they are critical to VigilNet. Because it is not easy to deploy a large system unattended for days we use simulations to analyze SelfHealingVigilNet.

4.1 Experiments

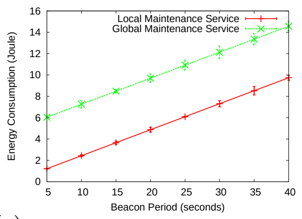
In total, our framework uses 2,197 bytes of code memory and 115 bytes of data memory and there were some modifications to the original VigilNet mainly in the top level components (MainControlM.nc and MainControlC.nc).

In the experiments we enable the magnetic sensors. We deploy 10 XMS2 motes programmed with SelfHealingVigilNet in two lines (2 by 5) in our lab and a base station connecting to a PC. We define that maintenance period is the time between two maintenance services and maintenance phase is the time to execute a maintenance service. We set the maintenance phase for both the time synchronization and the sensing coverage to be 20 seconds and the communication coverage to be 40 seconds, to be compatible with the default parameters defined in VigilNet. However, the maintenance phase is a tunable parameter depending on the network size. In order to speed up the experiments and without effecting the correctnesses of the experiments, based on observations, we set the beacon period to 10 seconds and the beacon rate to 0.5 (one beacon message every 2 seconds) for the monitors, the maintenance period to 10 minutes, and we manually create the sensor and communication coverage failures. All experiments start the system from the initialization phase to the tracking phase as shown in Figure 3. The real testbed experiments demonstrate that our framework can (1) execute in parallel with the original system, (2) efficiently monitor specified performance measurements, (3) enforce different maintenance policy specifications, and (4) effectively manage the memory constraints. Due to lack of space the details are not presented.

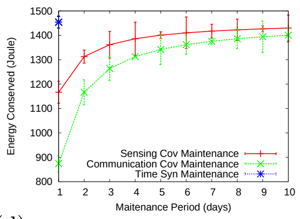
4.2 Analysis

In this section, we analyze the performance of SelfHealVigilNet using the metrics defined above. The performance of VigilNet is the baseline. In our analysis, each node has a radio range of 30 meters and a sensing range of 10 meters as in VigilNet. Nodes are uniformly distributed over an area with 300 meters by 300 meters. The average density of the deployment is 10 nodes per radio range. Radio consumes $48mW$ at transmit state and $24mW$ at receive state as studied in. When a message is transmitted, the radio switches to the transmit state for 30 milliseconds, a typical time required by XSM motes to send a message under the MAC contention. The beacon period for monitors and the maintenance phases for each maintenance services are the same as in the experiments. The report period and report rate for monitors are 20 seconds and 0.2 (one report every 5 seconds), respectively. Each battery of XSM motes has an energy capacity uniformly chosen between 2,848mAh and 2,852mAh, voltage at 3V. Each analysis result is the mean of 10 runs, with a confidence interval of 95%.

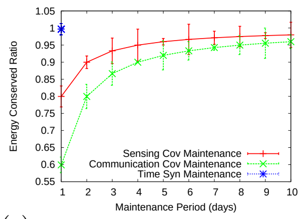
Overhead: The energy consumption of the monitors is the main overhead of our framework. And the main energy consumer of the monitors is the radio,



(c) Overhead per Maintenance Service per Node.



(d) Absolute Energy Saved per Node.



(e) Ratio of Energy Saved per Node.

Fig. 4. Energy Analysis of the In-Field-Maintenance Framework

exchanging beacon messages around a neighborhood to collect the states information of concern and reporting the states. We do not consider the overhead to compute the connectivity graph in the communication coverage maintenance service because the computation is done at the base station, and energy is not a severe concern of the base station. Meanwhile, the computation overhead of a node to calculate the sensing coverage is minimal as compared to the energy consumed by the radio. During beacon and report periods, the radio is at either the transmit state or receive state. For local maintenance services, the energy consumed by a local collector is the overhead. For global maintenance services, the local collector and reporting cause the overhead.

Figure 4(c) shows that the overhead of a node for both a local and a global maintenance service at different beacon periods, while we fix the beacon rate at 0.5 (one beacon message every 2 seconds). We can see that the beacon period has an approximately linear impact on the overhead. But the overheads for both local and global services are minimum (less than 1.2mAh) by comparing with a battery's capacity (2,848mAh).

Energy Conserved Per Node: We analyze the impact of the maintenance period to the energy consumption of SelfHealVigilNet. Different from the original VigilNet system, our framework allows different maintenance periods for different maintenance services. We investigate different maintenance periods over a period of 60 days system lifetime. In VigilNet, all the protocols are reinitialized once per day without considering if a protocol needs the reinitialization. During these 60 days, VigilNet reinitializes the system 60 times, where each reinitialization takes approximately 24.3Joules calculated using the data available in [5]. Figure 4(d) and Figure 4(e) show that the absolute energy saved per node can be as high as 1,483.6Joules (137.4mAh), which is around 5% of a node's battery capacity. And the ratio of the energy saved is always above 60% and can save up to 98%, which is significant. The ratio of the energy conserved is the difference between the energy consumed during rotation services (eR) and the energy consumed by individual maintenance services (eE) divided by eR.

Our experiments and analysis results show that our framework can be effectively incorporated with a complex legacy system (VigilNet) to improve the performance of the system. For example, SelfHealingVigilNet saves up to 5% of a node's battery capacity, decreases the system blackout time to less than 11%

of the original system, and always detects an intruder, with minimum overhead (less than 1.2mAh).

5 Related Work

SASHA [2] proposed a self-healing architecture for WSN. While SASHA is a centralized approach, our framework supports both global and local maintenance services. A timeout control mechanism with two timers was studied in [10] to enable the control center of a WSN system to be constantly aware of the existence/health of all the sensor devices in the network. The two-tier monitors in our framework can collect 5 different categories of information of concern both globally or locally.

Sympathy [13] proposed a prototype tool for detecting and debugging failures for WSN applications in both pre-deployment and post-deployment. It detects failures based on data quantity, rather than data quality at a centralized location. Different from Sympathy, our framework not only supports globally and locally collecting failure detection information, it also provides mechanisms to recover the system. LiveNet [3] provides tools to understand the behavior of or to provide a global view on the dynamics of WSNs. Passive monitoring infrastructure sniffers are implanted into the networks to collect the traffic traces. Offline trace merging and analysis tools are developed to reconstruct the network connectivity, to infer the routing path, and to identify hotspots of the system. In our system, different from LiveNet, the state information are collected, distributed or processed online by the sensor nodes themselves. Memento [14] is designed for inspecting the health state of nodes in WSNs. It provides failure detection and symptom alerts via an energy efficient protocol to deliver state summaries. Our monitored state information not only is about node failure detection, but also includes the performance and resource usage information of protocols in the stack.

[15] presented a distributed algorithm to configure the network nodes into a cellular hexagonal structure. The self-healing under various perturbations, such as node joins, leaves, deaths, movements, and corruption, is achieved by the property of the hexagonal structure. And [9] [12] [11] [4] achieve the reliability or resilience toward failures or dynamics in the system via model or analysis redundancy. However, these solutions are designed for individual protocols.

6 Conclusion

In this paper, we study an efficient and effective in-field-maintenance framework for WSN under the resource constraints. The framework is composed of two components, a maintenance policy and a maintenance engine. As far as we know, our in-field-maintenance service framework is the first effort to build an efficient and effective framework for WSN that supports the following collection of features:(1) efficiently monitor both system performance measurements and states of protocols of a system, locally and globally, (2) enforce dependency constraints,

(3) provide mechanisms to recover the system, and (4) provide different strategies to manage memory. Based on the case study, both experiments and analysis, for a real application VigilNet, our framework demonstrates that it improves the performance of the original VigilNet system, such as energy conservation, system blackout time, detection probability, with minimum overhead.

References

1. A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M Miyashita. A wireless sensor network for target detection, classification, and tracking. In *Computer Networks (Elsevier)*, 2004.
2. T. Bokareva, N. Bulusu, and S. Jha. Sasha:toward a self-healing hybrid sensor network architecture. In *EmNetS-II*, 2005.
3. B. Chen, G. Peterson, G. Mainland, and M. Welsh. Livenet: Using passive monitoring to reconstruct sensor network dynamics. Technical report, Harvard University, 2007.
4. L. S. Crawford, V. Sharma, and P. K. Menon. Numerical synthesis of a failure-tolerant, nonlinear adaptive autopilot. In *CCA*, 2000.
5. T. He, S. Krishnamurthy, L. Luo, T. Yan, B. Krogh, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, and J. Hui. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Transactions on Sensor Networks*, 2(1):1 – 38, 2006.
6. T. He, S. Krishnamurthy, J. A. Stankovic, T. F. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, and B. Krogh. An energy-efficient surveillance system using wireless sensor networks. In *ACM MobiSys*, 2004.
7. T. He, L. Luo, T. Yan, L. Gu, Q. Cao, G. Zhou, R. Stoleru, P. Vicaire, Q. Cao, J. A. Stankovic, S. H. Son, and T. F. Abdelzaher. An overview of the vigilnet architecture. In *RTCSA*, 2005.
8. T. He, P. Vicaire, T. Yan, Q. Cao, G. Zhou, L. Gu, L. Luo, R. Stoleru, J. A. Stankovic, and T. F. Abdelzaher. Achieving long-term surveillance in vigilnet. In *IEEE INFOCOM 2006*, 2006.
9. G. Hoblos, M. Staroswiecki, and A. Aitouche. Optimal design of fault tolerant sensor networks. In *CCA*, 2002.
10. C. Hsin and M. Liu. A distributed monitoring mechanism for wireless sensor networks. In *Proceedings of the 3rd ACM workshop on Wireless security*, 2002.
11. K. Marzullo. Tolerating failures of continuous-valued sensors. In *ACM Transactions on Computer Systems*, 1990.
12. G. Provan and Y. Chen. Model-based fault tolerant control reconfiguration for discrete event systems. In *CCA*, 2000.
13. N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *ACM SenSys*, 2005.
14. S. Rost and H. Balakrishnan. Memento: A health monitoring system for wireless sensor networks. *SECON*, 2006.
15. H. Zhang and A. Arora. Gs^3: Scalable self-configuration and self-healing in wireless networks. In *ACM PODC*, 2002.