Predictive Dependency Constraint Directed Self-Healing for Wireless Sensor Networks

Jingyuan Li*, Yafeng Wu*, John A. Stankovic*, Sang H. Son*, Ziguo Zhong†, Tian He†,
Bong Wan Kim‡ and Seong-Soon Joo‡
*Department of Computer Science, University of Virginia, Charlottesville, VA, U.S.A.
Email: {jl3sz, yw5s, stankovic, son}@cs.virginia.edu
†Computer Science and Engineering, University of Minnesota, Minneapolis, MN, U.S.A.
Email: {zhong, tianhe}@cs.umn.edu
‡Electronics and Telecommunications Research Institute, Daejeon, Korea
Email: {kimbw, ssjoo}@etri.re.kr

***Abstract*— Wireless sensor networks are now being considered for mission critical applications, which are often largely unattended and need to operate reliably for years. However, due to the real world communication, sensing and failure realities, clock drift, and node faults, the system performance may degrade significantly over time. It is highly desirable that these natural deteriorations can be monitored continuously and can be corrected with self-healing when necessary. In this paper, we introduce a dependency constraint directed self-healing scheme for wireless sensor networks. We reveal that when self-healing services are being composed, certain dependency constraints, including invocation, parameter consistency, control and implicit assumption dependencies must be carefully identified and respected. We illustrate each of these dependency constraints through case studies in 3 different systems covering the typical functions of wireless sensor networks, including sensing, communication and tracking. Our research indicates that, following the dependency constraints in self-healing design is not only a must for the correctness of self-healing services, but is also a key to energy efficient self-healing.**

***Keywords*— self-healing; dependency; wireless sensor networks; sensing; communication; tracking;**

I. Introduction

Wireless sensor networks (WSNs) are now being considered for mission critical applications such as infrastructure monitoring [10], [11], fire fighting [1], pollution control [15], assisted living [21], military surveillance and tracking [22]. These systems will often need to exist for years, and operate reliably in the context of real world communication, sensing and failure realities. However, due to the negative impact of noisy environments and the unreliable nature of the cheap sensor nodes being used, it has been commonly observed and reported that WSN systems are subject to performance degradations, component faults, and even major system failures in real world deployments [18], [17], [14].

Our key observation is that a WSN system may drift to disorder and lose its key capabilities over time due to faults, performance degradations, node failures, security attacks, workload changes, and natural deterioration such as reduced energy and clock drift, unless proper self-healing services is applied to maintain order, or to recover the system from disorder.

The central idea of our self-healing services is to ochestrate the running of a selective set of existing system protocols in order to prevent the impact of failures and to maintain system performance. The design of our self-healing service is inspired by the "system-wide reboot" and "system-wide reconfiguration" schemes which had been used in WSNs historically to handle malicious distributed failure conditions, e.g., in VigilNet [7]. However, we argue that such simple reboot/reconfiguration schemes can not fulfill self-healing for WSNs in general and are rather energy inefficient.

In modern large scale complex WSNs, it is common practice to have multiple protocols and dozens of components integrated to perform a single application. The inherent complexity of these systems makes composing self-healing services a difficult problem, due to the intricate coupling and complex dependency relationships among different part of the system. Dependency relationships among different components must be carefully studied, explicitly articulated and respected when designing the self-healing services. What's even worse, the system may evolve over time, and components may be added, deleted, and updated, each change potentially impacting the dependency relationships and thus invalidate the previous design of the self-healing solution.

In this work, we propose a novel dependency constraint directed self-healing framework to allow users to compose self-healing services both systematically and consistently, and to be able to perform self-healing services in an energy efficient manner. The framework also permits flexibility in more easily modifying the self-healing services over time rather than re-implementing the system.

The major contributions of our work are: 1) a predictive self-healing approach that monitors for state deterioration and maintains system performance under faults and failure conditions, 2) a novel dependency constraint directed self-healing solution that supports both correct and efficient self-healing, 3) small runtime costs that enable self-healing techniques to execute on minimal capacity sensor nodes, and 4) a set of case studies that demonstrate that the solutions are appropriate for different types of systems and different types of protocols.

The rest of this paper is organized as follows. In Section II we introduce the dependency constraint challenges for self-healing. Section III presents the dependency constraint directed self-healing design. Section IV presents case studies of the self-healing design in 3 different systems, including VigilNet [7], ATPC [12], and SBT [24]. Section V covers an overview of the state-of-the-art self-healing practices in WSNs and related works. Finally, in Section VI, we present the conclusions.

II. Self-Healing Dependency Constraints

There are often intricate coupling and complex dependency relationships among different parts of a system in modern large scale WSNs. Such dependencies incur constraints that must be understood and accounted for when composing self-healing services. Creating a methodology and run time framework that address these issues provides more effective self-healing both in terms of performance (only protocols that have actual or potential problems need be re-executed) and correctness (when re-running protocols to re-establish consistent distributed state, the proper order and collection of protocols are executed). A key element of this approach is the dependency assessment.

We have identified 4 key types of dependencies which we call: *invocation, parameter consistency, control and implicit assumption* dependencies. The invocation dependencies are often considered easy to identify via the explicit dependency relationships implied by function calls. However, even these dependencies are more complex than implied by the top-down call tree (discussed below). Beyond these dependencies, there is a collection of more complex dependencies that exist in many systems. These complex dependencies share similar traits, for example: they are often more implicit and can't be easily traced from explicit function calls. Rather they exist in the form of race conditions among multiple components, or among competing control loops, or due to the assumptions made by the designer/implementer. In these cases, each dependency must be carefully identified when designing self-healing services. This collection of *complex dependencies* includes the parameter consistency, control and implicit assumption dependencies.

A. Invocation Dependency

Imagine that the system performance is being monitored on a continuous basis, and over time more and more nodes were found to be no longer responsive, and certain performance metrics (for example, average end-to-end delay) degrade dramatically. The self-healing framework using the monitoring component decides that certain healing service must be invoked to maintain the performance. However, such healing service can't be trivially defined by the collection of all protocols, because: 1) certain protocols are related to the performance metric, while others are not, so only a portion of the protocols needs to be invoked; 2) due to the dependency constraints among different components of the system, protocols can't be invoked in arbitrary order, but must carefully follow the dependency constraints. We call this type of dependency an "invocation dependency".

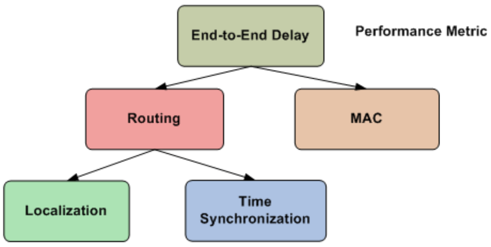

Fig. 1. Invocation dependency.

Invocation dependency primarily refers to the type of dependency that exists due to the vertical integration of a protocol stack or because of a series of function calls. An example of invocation dependency is as follows.

Suppose that average "end-to-end communication delay" is an important state variable being monitored and it starts to degrade to a level that affects losing tracks of vehicles or not having enough time to react to alarms. The increased delay may be due to some failed nodes so that current routes are not effective anymore. Thus, a set of protocols must be re-invoked to perform self-healing. End-to-end delay depends on both the routing protocol and the MAC protocol. Further, the delay performance of the routing protocol may also depend on the time synchronization and the localization protocols. The dependency relationships in this example are illustrated in the Fig. 1. This set of protocols must be re-run to re-establish new and effective routes. Individual protocols do not have the global view to solve this problem.

B. Parameter Consistency Dependency

Parameterized protocols and functions permit flexible systems and adaptive performance by modifying the values of the parameters at runtime. When a problem is detected that needs self-healing, one approach is to adjust some of these parameters. However, this must be performed carefully and consistently. Our framework permits specification of parameter dependencies and the required healing actions. For example, imagine a WSN that is suffering from performance degradation due to node failures and the self-healing service is invoked. The service evaluates the current situation to decide which action to take. Consider the following examples of parameter dependency.

Due to the common practice of node duty cycling in WSNs, there is usually only a portion of total nodes active at given time, and this proportion P is a parameter that is decided by the designer depending on system requirements. Suppose there is another parameter R that determines the number of redundant routes that must be maintained for system communication reliability. Since there are hidden dependencies among these two parameters, we can't set one of the parameters without affecting the other. Thus, in our methodology these parameters are identified and their relationship coded into the self-healing action routines in the framework. The exact settings are application dependent, but with our framework they are set in a consistent manner.

Another example involves a duty cycle choice for a node (rate at which it awakes and senses the environment) and the duty cycle for the radio (rate at which the node is awake for communicating with neighbors). Given deterioration in energy levels available, the system might decide to reduce the communication duty cycle. This decision should not be performed in isolation, rather both duty cycles should be considered in order to implement a system-wide effective solution. Explicitly identifying these dependent parameters and including them into an framework enables correct and efficient decision, and is flexible to future system changes.

C. Control Dependency

Many sensor networks employ control loops within and across protocols, nodes and even the entire system. Often such control loops take a localized view and can operate inaccurately from a system perspective. Designers must consider all the control loops in the system and identify potential control loop dependencies and the proper combined actions to avoid control loops from inappropriately positively reinforcing each other resulting in an overshoot situation, or cancelling each other (when these are not the correct actions).

Imagine a WSN that implements both a communications power control protocol (e.g., ATPC [12]) and a sensor power management protocol. When the RSSI value is found to be low, the power control protocol tries to increase the radio transmission power in order to maintain link quality. Meanwhile, power management protocol may prefer the opposite – decrease radio transmission power in order to conserve energy (if the energy level is low). The dependencies among these competing control loops must be identified and the proper combined action must be defined for the self-healing framework. In this case there may be a compromise that increases the communications power less than it would have if acting alone since the node is low on power.

In the future, as more actuators are added to WSNs, it will become more and more important to identify and support control loop dependencies in a correct manner. Our framework offers this support.

D. Implicit Assumption Dependency

Often, protocols and functions are implemented with unstated assumptions. To support long-lived systems, identifying and making explicit as many of these assumptions as possible will improve the effectiveness of self-healing. In other words, a designer must attempt to understand the implicit assumptions in the system and make them known to the self-healing framework. Once known, the self healing framework uses them in a correct manner.

For example, imagine a WSN that implements a localization protocol which relies on minimum number N of GPS anchor nodes to perform localization to the required level of accuracy. Then, the implicit assumption adopted by the localization protocol and stated to the self-healing framework is that the number of available anchor nodes should not drop below N. At runtime, if localization must be re-run because too many nodes have been moved, this assumption is checked. If the number of current true GPS anchors is greater than or equal to N then self-healing continues. If not, then there must be an action routine that “heals” this problem. Such a routine might activate a GPS device on other nodes (assuming such backups exist) or choose current nodes deemed accurate enough to serve as anchor proxies or inform a system administrator that new anchor nodes must be physically added to the system.

Identifying implicit assumptions is perhaps the most difficult problem for a system designer. However, our approach forces a designer identify such dependencies and make them explicit. This improves the system design as compared to the case where this is not done.

III. Dependency Constraint Directed Self-Healing Design

In self-healing enabled WSNs, a system should have at its core a collection of smart capabilities, including: self-cognizance, decision making and self-healing. These key capabilities allow a WSN to be aware of its health status at run time, such as clock drift, link quality, route accessibility, energy levels, node aliveness and even the integrity of application semantics. Given the health status of the system and its components, the system needs the ability to evaluate this information and to decide whether healing is necessary, when, where and how should the system be healed.

The goal of our self-healing design is to provide systematic self-healing mechanisms to maintain system performance for WSNs in the presence of natural system deterioration due to time, faults, failures and inconsistent states. We design and implement a generic framework to enable the dynamic invocation of self-healing services at run time. Our framework allows individual system functions to specify the required self-healing service, identifying when such a service needs to be activated, and then creating the actual protocols to restore system state for each function.

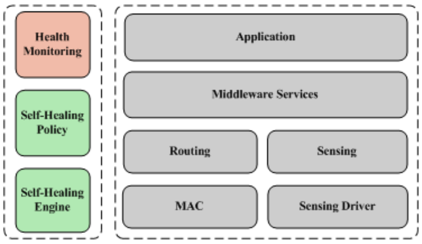

Fig. 2. Self-healing framework.

As illustrated in Fig. 2, our self-healing framework incorporates three components: health monitoring, self-healing policy and self-healing engine. The health monitoring module enables the WSN to be aware of its own health status including the consistency of the distributed state of the network. It monitors for the situation in which state consistency deteriorates to a level that is approaching potential failure and activates the action routines *early* to avoid such conditions. The self-healing policy specifies dependency relationships among system components that must be respected when composing self-healing services, provides decision making mechanisms on when self-healing service should be invoked, and describes how to invoke related action routines. The self-healing engine provides a set of self-healing services that are invoked when the system needs to react based on predicted deterioration or faults and errors that have occurred.

A. *Health Monitoring*

A monitor is implemented to collect the health state information at system execution time. It is entirely implemented by the developers of the self-healing framework, and the states to be monitored are defined by system designers.

There are four types of health states (based on the way they can be accessed), each of the following category represents a particular type, and the corresponding way of implementing health monitoring in WSNs.

- States that are available via directly interfacing with the hardware layer. For instance, the node battery level. The monitor can check these states by calling relevant interfaces.
- States that exist in node software. For instance, the packet buffer utility that exists in the routing protocol. The monitor can use specific functions to access these states.
- States that need cooperation with other nodes. For instance, the link quality between two neighboring nodes. To support accessing these states, the monitor uses the communication facilities to exchange information with its counterpart at neighbors.
- Global states that only the base station can process. For instance, the end-to-end delay. For these states, the monitor at nodes use existing network layer protocols to send states to the base station.

B. *Self-healing Policy*

1) Dependency Specification: Dependency specification is designed off-line. How to identify all dependency constraints among system components is a separate research question that's beyond the scope of this paper. Either the system designer or the developer of the self-healing framework presents the dependency specification in a file. Each line of the file specifies a dependency relationship and the type of dependency between two system components. The dependency specification is then parsed by a script (a Perl script) to generate a dependency component that contains corresponding representations of dependency constraints in nesC to be used by the run-time invocation component. The developer of the self-healing framework then links the dependency component with the run-time invocation component manually.

In our self-healing design, invocation dependency and complex dependencies are represented with different run-time data structures according to their different natures. Invocation dependency is represented with a set of dependency trees. Each parent node of the tree depends on its child nodes. At run time, the run-time invocation component searches the list of trees and checks against the root of each tree to determine if a dependency exists. The complex dependencies are represented with a linked list, each item in the list includes a set of components with complex dependency constraints and the type of each dependency.

2) Policy Library: The policy library is basically a set of tables that are referred to once the monitor identifies changes in system health status. Each entry in the table represents a policy, which describes the state being monitored, the condition (e.g., threshold values) under which action routines must be activated and the corresponding self-healing action under particular failure situations. Coordination among a set of policies is enabled via user defined priority bits. Such coordination requires the system to resolve certain dependencies with higher priority than the others.

C. *Self-healing Engine*

A set of run-time invocation routines is embedded in the main loop of the application to implement the self-healing engine. Once the system finishes the initialization, it enters both the functional state and the health monitoring state. When major changes in system health status are identified, depending on the self-healing policy and the type of dependency constraints involved, self-healing commands are sent out by the initiator (e.g., the base station). If only invocation dependencies are involved, the system invokes related self-healing protocols according to the dependency specification; if complex dependencies also exist, the system uses a *resolver* component to request a set of parameters to be adjusted correspondingly, or a set of control loops to take consistent actions accordingly, or a self-healing action to be taken in response to violated system assumptions.

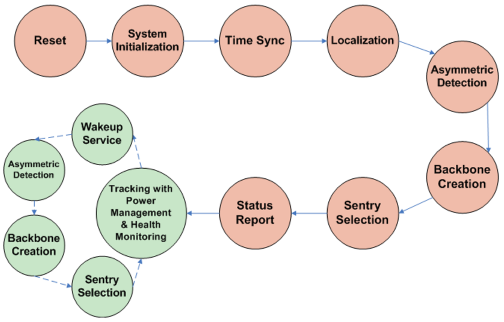

Fig. 3. Self-healing for communication and sensing coverage in VigilNet.

A key advantage of the modularized design of our self-healing framework is that it allows users to deal with system updates and changes relatively easily. Dependency specifications, policies and self-healing action routines can be added,

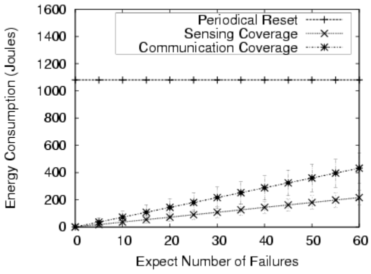

Fig. 4. Impact of the number of failures on energy consumptions.

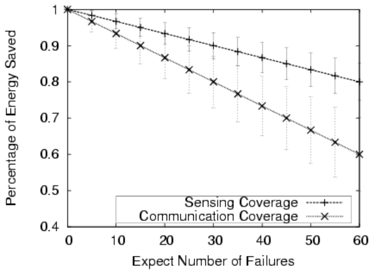

Fig. 5. Impact of the number of failures on energy savings.

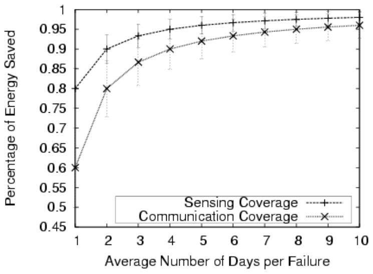

Fig. 6. Impact of failure intervals on energy savings.

deleted or modified easily to allow the self-healing design to evolve with the system over time.

IV. Case Studies

In this section we present case studies for self-healing in 3 different systems, to illustrate each of the above dependencies. Our case studies cover the typical aspects of WSNs, including sensing, communication and tracking.

A. Self-healing in VigilNet: Invocation and Parameter Consistency Dependencies

In this section, we illustrate the invocation and parameter consistency dependency constraints in the context of self-healing design in VigilNet [7]: a military surveillance application.

VigilNet is a large-scale WSN for long-term military surveillance with significant system complexities. The primary design goals of the VigilNet system are to detect events and moving targets, keep track of the position of the moving targets and to correctly classify the detected targets in an energy-efficient and stealthy manner.

Of critical importance to its application level performance is the health status of a set of system functions, including time synchronization, sensing coverage and communication coverage. The major cause of performance degradation and failures are natural clock drifts over time, link failures due to radio communication realities [25] and node failures. Inspired by the motivation to combat clock drift among all the nodes, as well as to deal with node failures and disconnected sub-networks, VigilNet implemented a rotational scheme to reinitialize the entire network for all the services once per day.

Such a rotational re-initialion scheme suffers from two major problems. First, since re-initialization happens periodically on a daily basis, any faults or failures only get resolved at the end of the day when the system reinitializes. Second, the re-initialization process interrupts the normal operations of the system and introduces a constant overhead in energy consumption, regardless of whether there are failures occurring in the system or not.

To address these two issues, our self-healing solution allows the system not only to be responsive to failures, but also to be able to perform self-healing in an energy efficient manner. In this case study, we explore self-healing for two important functions in VigilNet – the communication coverage and sensing coverage.

Due to the unreliable nature of the battery powered sensor nodes being used, node failure occurs at times. If the failed node happens to be a backbone node of the network spanning tree, then it disconnects a sub-network from the system, leading the system to lose communication coverage; on the other hand, if the failed node happens to be a sentry node (VigilNet incorporates a node duty cycling based scheme to save energy, active nodes are selected strategically to guarantee sensing coverage, while other nodes can go to sleep to save energy. Such active nodes are called sentry nodes in VigilNet.), then the system loses sensing coverage.

In either case of failures, the system performance can be affected significantly. Thus, we monitor the health status of the communication backbone nodes and sentry nodes on a continuous basis, using periodic beacons. Once node failures are detected, the system invokes a series of pre-installed self-healing protocols to heal the system. Such invocation sequences represent the minimum number of protocols that need to be invoked, and are carefully ordered to satisfy the dependency constraints described in Section II-A.

As illustrated in Fig. 3, when the system detects a communication backbone node failure and determines that a self-healing service must be invoked according to the self-healing policy, it first uses the "wakeup service" to wakeup all the nodes in sleep mode, re-runs the "asymmetric detection" protocol to select good links, and then re-runs the "backbone creation" protocol to pick the backbone nodes and create the communication backbone, and finally, due to the change in network topology, a "sentry selection" protocol needs to be re-run to maintain sensing coverage based on the recovery of communication coverage. If the failed node is a sentry node, then the self-healing service only needs to invoke the "wakeup service" and the "sentry selection" protocol to re-establish sensing coverage.

By following the dependency constraint directed self-healing design, our system is not only more responsive to node failures (it heals the system almost immediately after the failure has been detected), but also achieves significant energy efficiency, because our self-healing only invokes the minimum set of protocols needed to self-heal and only when failures have actually occurred, rather than on a periodic basis.

To illustrate the benefit of our self-healing design, we carried out a simulation to study the energy consumption in contrast with the original system. Our simulation assumes the same energy model as used in [7]. In a simulated scenario of 60 days of deployment, VigilNet reinitializes the system 60 times, consuming 1079.9 Joules (99.9936 mAh) of energy, which is about 4.5% of the node's battery capacity (2200 mAh). During the 60 days, we assume node failures follows a Poisson distribution, with expected number of failures from 0 to 60.

The energy overhead of our self-healing service is presented in Fig. 4 in contrast with the original VigilNet system, which uses periodic reset to achieve the same purpose. We can see that the energy consumptions of our self-healing services increase linearly with the number of failures, but are significantly smaller than the original system. Fig. 5 illustrates the impact of the number of node failures on energy savings. The energy savings decrease linearly with the number of node failures, but are still significant (at least 60%) with even 60 node failures. Fig. 6 presents the impact of the average number of days per failure on energy savings. As indicated in Fig. 6, even if the failures occur on a daily basis (on average), we can still save 60% of the energy for self-healing in communication coverage and 80% of the energy for self-healing in sensing coverage.

As an example of the parameter consistency dependency, we found that when more and more node fails in VigilNet, it becomes more and more difficult to maintain the network lifetime without adjusting a set of system parameters to meet the energy constraints. To maintain the targeted network lifetime, the system must sacrifice the sensing coverage to reduce energy consumption, thus the number of sentry nodes, as a parameter, must be decreased during the self-healing process. Similarly, the system must also reduce the level of route redundancy to save energy, thus the number of redundant routes, as another parameter, must also be decreased to adapt to the new failure realities. However, these two parameters are not independent. Thus, when self-healing action is initiated, these two parameters must be adjusted consistently via the resolver, based on the self-healing policy and the current failure realities (e.g., proportion of node failures).

B. *Self-healing in ATPC: Control Dependencies*

In this section, we study a control dependency example in WSN protocol stacks to illustrate how control dependency constraints are addressed in our self-healing design.

It is well-known that WSNs suffer from unstable and poor wireless links, and high packet loss ratio in many cases. One effective way to address this issue is to use transmission power control protocols, such as ATPC [12], which apply feedback control to adjust transmission power levels according to packet reception ratio. On the other hand, packet loss can also be caused by collisions and congestions of concurrent traffic. Backoff interval control methods are used to address this issue, which apply feedback control to adjust backoff interval of CSMA protocols to reduce the chances of collisions[8]. Both control loops take the packet reception ratio as input, and improve communication qualities individually. However they may conflict with each other when they exist in the same system. For example, when packet losses are caused by collisions, ATPC increases the transmission power in response, but higher transmission power can't reduce packet loss due to collisions, rather, it aggravates collisions. On the other hand, when packet losses are caused by poor link quality, the backoff control loop increases the backoff window size, but it can't improve the link quality and packet reception ratio, rather, it increases the transmission delay and reduces the throughput.

To address this control dependency, our self-healing approach implements a resolver that utilizes the control coordinator on top of both control loops. This control coordinator manages control in two steps: First, it differentiates whether packet losses are caused by poor link quality or by collisions, via measuring RSSI values (in reality, when collisions occur at receivers, high RSSI fluctuations can be observed); Second, the coordinator invokes the corresponding control loop to maintain communication quality. Fig. 7 illustrates the design of this self-healing resolver.

We implemented both control loops and the resolver in TinyOS, and conducted experiments on Telosb motes to study their performance. Our experiment uses two pairs of senders and receivers, which are placed close to each other initially. Each experiment is divided into two time periods. In the first period ($0 \sim 20s$), the first pair starts to transmit 50 packets/s. At the same time, the sender moves away from the receiver. In the second period ($20 \sim 30s$), both pairs send 50 packets/s, and the first sender stops moving. During the experiment, we measured the average packet reception ratio and the transmission power level every two seconds, and the results are shown in Fig. 8 and 9. Each data point is an average of 5 experiments.

In the first period, when the sender moves away from the receiver, the link quality degrades, ATPC is triggered to increase the transmission power to maintain the packet reception ratio. In the second period when two senders start to transmit packets concurrently, without the self-healing resolver, ATPC still runs and increases transmission power (as shown in Fig. 8), which not only leads to approximately 9.5% of energy waste, but also causes 7% more packet loss in the worst case (as shown in Fig. 9). In contrast, our self-healing resolver can avoid such problems, when packet losses are caused by collisions only, it stops the ATPC loop and triggers the backoff loop to maintain the communication quality.

Overall, dependent control loops can cause complicated problems if not handled properly. By addressing the control dependency via self-healing resolver in this case study, our self-healing design not only maintains system performance, but also eliminates the problems which dependent control loops can otherwise cause.

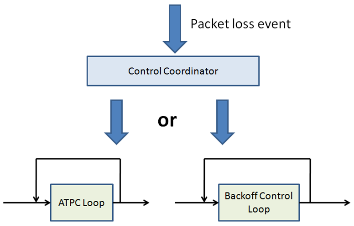

Fig. 7. Self-healing resolver for control dependency.

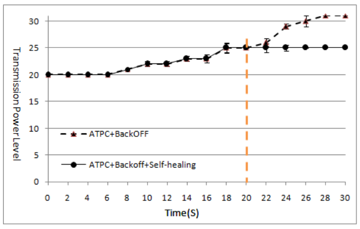

Fig. 8. Transmission power level over time.

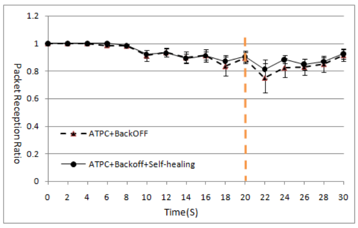

Fig. 9. Packet reception ratio over time.

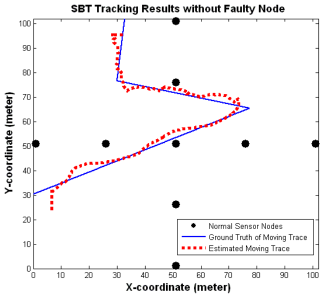

Fig. 10. SBT tracking results without faulty node.

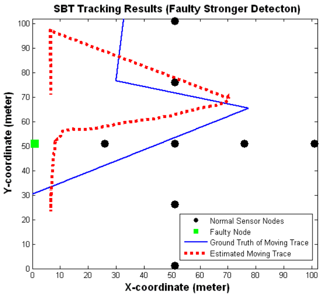

Fig. 11. SBT tracking results (with faulty stronger sensor reading).

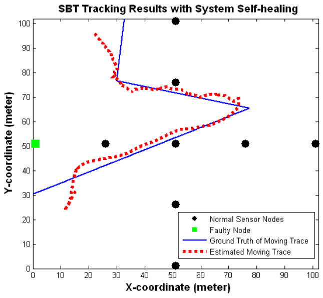

Fig. 12. SBT tracking results with system self-healing.

C. Self-healing in SBT: Implicit Assumption Dependency

To demonstrate our approach to implicit assumption dependencies, consider a tracking application for WSNs called Sequence-based Tracking (SBT) [24]. SBT is a system proposed to localize and track mobile objects with a WSN. Given the location information of all the nodes in the network, SBT divides the whole map into a large number of small regions with distinct sequence signatures, called faces. Each ordered sequence signature reflects the distance relationships of sensor nodes in the network to the corresponding face. Once a target enters the network area, sensors detect the target with a certain physical modality, following an implicit assumption that a node will detect a weaker signal when it is further away from the moving object. By comparing a series of detection sequences obtained from ordering in-the-field sensor nodes according to their sensing results with the signature sequences of faces in the map, the mobile target can be localized and tracked despite of strong sensing noise. However, the basic assumption of SBT can be violated due to faulty sensor readings, e.g., a malfunctioning sensor node always outputs stronger sensing result than the ground truth of the signal. In such situations, the performance of the SBT system deteriorates significantly.

We use simulation to illustrate the impact of violated assumptions and to reveal the benefit of our self-healing design. In our simulation, 9 senor nodes are deployed in the map, a mobile target moves in and out of the area under surveillance with its movement trace depicted with a solid curve, as shown in Fig. 10-12. The dashed curve in Fig. 10-12 shows the estimated track of the target from the SBT system. We can see from Fig. 10 that if every node works fine, the system gives a sound result. However, if one faulty node appears, i.e., it always gives significantly stronger sensing results, the SBT system is not able to give good tracking results, as shown in Fig. 11. As we can observe from Fig. 11, the estimated locations of the target are shifted towards the faulty node, this is because the faulty node reports a continuous stronger detection result creating the illusion that the target is close to the faulty node. According to the simulation results in Fig. 11, we can see that the violation of the implicit assumptions can cause significant errors in the tracking performance.

To deal with such problems, we apply self-healing in the SBT system. Our self-healing procedure first evaluates the sensor readings from each node in the network. If symptoms of sensor malfunctioning is detected, the self-healing protocol eliminates the faulty sensor node from the tracking system so as to re-establish satisfaction of the implicit assumptions. Next, a sequence of self-healing routines including map division, signature sequence assembling, and detection sequence matching [24] are invoked following the invocation dependencies. Fig. 12 presents the result of tracking after removing the faulty sensor node from the tracking system and self-healing. We can see that the tracking performance is greatly improved after self-healing.

V. Related Work

Self-healing, as an important research topic, has been explored both within and outside the context of WSNs. IBM initiated *Autonomic Computing* [9] in 2001, which aims to develop novel computing systems that can self-configure, self-heal, self-optimize and self-protect. Self-healing, as a property of this computing paradigm, has been explored in the context of personal computing [16]. However, these solutions can't be directly applied to WSNs since they are too heavy weight for the resource constrained sensor nodes.

A few recent works have investigated self-healing in WSNs. Some were targeting the architectural design of self-healing, others explored self-healing for a particular system or protocols.

SASHA [2] proposed an immunology inspired self-healing architecture for sensor networks, featuring adaptive network monitoring, automatic fault recognition and coordinated response. However, the system's ability to self-heal was only demonstrated in terms of dealing with faulty sensor readings. In BiSNET [3], self-healing (in terms of detecting and eliminating false positives in sensor readings) is a property derived from the biologically inspired architecture. However, the system's ability to self-heal is not demonstrated in general. GS^3[23] 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. However, self-healing is only derive from the particular structure, and is not applicable to other systems in general.

A number of papers explored the possibility of self-healing (in terms of bridging holes in routing or node coverage) by means of mobile nodes [5], [19], [6]. Although node mobility is possible and anticipated by researchers, it is still a luxury for current sensor network practices in general. POSH [4] proposes a self-healing protocol that allows sensors to collectively recover from compromises via infusion of secure randomness, however, its focus is only on backward secrecy (recover from prior compromise). [13], [20] proposed processes for a ZigBee based sensor network to repair itself after node failure or communication breakdown by allowing a disconnected subnet to rejoin the network. They both improved the standard self-healing scheme described in ZigBee specification [26], however, their focus is only on network connectivity.

VI. Conclusion

In this paper, we present a predictive dependency constraint directed self-healing scheme for WSNs. We explored 4 types of dependency constraints (including invocation, parameter consistency, control and implicit assumption dependencies) and their impact on self-healing for WSNs. We proposed a self-healing framework and a dependency constraint directed design approach to deal with these dependencies in self-healing. We illustrated each of these dependency constraints through case studies in 3 different systems covering the typical functions of WSNs, including sensing, communication and tracking. Our research indicates that, following the dependency constraints in self-healing design is not only a must for the correctness of self-healing service, but is also a key to energy efficient self-healing in WSNs.

VII. Acknowledgment

This work was supported in part by the Ministry of Knowledge Economy (MKE) of the Republic of Korea under grant 2008-F-052, and NSF CNS-0614870 and CNS-0626632.

References

[1] Berkeley FireBug. http://firebug.sourceforge.net/.

[2] T. Bokareva, N. Bulusu, and S. Jha. Sasha: Toward a self-healing hybrid sensor network architecture. *The Second IEEE Workshop on Embedded Networked Sensors*, pages 71–78, May 2005.

[3] P. Boonma and J. Suzuki. Bisnet: A biologically-inspired middleware architecture for self-managing wireless sensor networks. *Computer Networks*, 51(16):4599–4616, 2007.

[4] R. Di Pietro, D. Ma, C. Soriente, and G. Tsudik. Posh: Proactive co-operative self-healing in unattended wireless sensor networks. In *SRDS*, Oct. 2008.

[5] X. Du, M. Zhang, K. Nygard, M. Guizani, and H.-H. Chen. Distributed decision making algorithm for self-healing sensor networks. In *ICC*, June 2006.

[6] B. Haynes, M. Coles, and D. Azzi. A self-healing mobile wireless sensor network using predictive reasoning. *Sensor Review*, 28(4):326–333, 2008.

[7] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. *ACM Transactions on Sensor Networks*, 2(1):1–38, 2006.

[8] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating congestion in wireless sensor networks. In *SenSys*, 2004.

[9] IBM Autonomic computing white paper - An architectural blueprint for autonomic computing, 2005. IBM Corp.

[10] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Wireless sensor networks for structural health monitoring. In *SenSys*, 2006.

[11] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *IPSN*, pages 254–263, 2007.

[12] S. Lin, J. Zhang, G. Zhou, L. Gu, J. A. Stankovic, and T. He. Atpc: adaptive transmission power control for wireless sensor networks. In *SenSys*, 2006.

[13] W. Qiu, P. Hao, and R. Evans. An efficient self-healing process for zigbee sensor networks. In *ISCIT*, Oct. 2007.

[14] N. Ramanathan, K. Chang, L. Girod, R. Kapur, E. Kohler, , and D. Estrin. Sympathy for the sensor network debugger. In *SenSys*, November 2005.

[15] D. Shin, S. Y. Na, J. Y. Kim, and S.-J. Baek. Fish Robots for Water Pollution Monitoring Using Ubiquitous Sensor Networks with Sonar Localization. In *ICCIT*, Nov. 2007.

[16] Sterritt, R. and Bantz, D.F. Personal autonomic computing reflex reactions and self-healing. In *IEEE Transactions on Systems, Man, and Cybernetics*, May 2006.

[17] R. Szewczyk, J. Polastre, A. M. Mainwaring, and D. E. Culler. Lessons from a sensor network expedition. In *EWSN*, pages 307–322, 2004.

[18] G. Tolle, J. Polastre, R. Szewczyk, D. E. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A macroscope in the redwoods. In *SenSys*, 2005.

[19] N. Vlajic and N. Moniz. Self-healing wireless sensor networks: Results that may surprise. In *Globecom Workshops*, Nov. 2007.

[20] J. Wan, W. Chen, X. Xu, and M. Fang. An efficient self-healing scheme for wireless sensor networks. In *FGCN*, Dec. 2008.

[21] A. Wood, J. Stankovic, G. Virone, L. Selavo, Z. He, Q. Cao, T. Doan, Y. Wu, L. Fang, and R. Stoleru. Context-aware wireless sensor networks for assisted living and residential monitoring. *IEEE Network*, 22(4):26–33, July-Aug. 2008.

[22] T. Yan, T. He, and J. A. Stankovic. Differentiated Surveillance for Sensor Networks. In *SenSys*, November 2003.

[23] H. Zhang and A. Arora. Gs^3: scalable self-configuration and self-healing in wireless sensor networks. *Computer Networks*, 43(4):459–480, 2003.

[24] Z. Zhong, T. Zhu, D. Wang, and T. He. Tracking with Unreliable Node Sequence. In *INFOCOM*, Apr. 2009.

[25] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Impact of radio irregularity on wireless sensor networks. In *MobiSYS*, pages 125–138, 2004.

[26] ZigBee Specification Version 1.0, 2004. ZigBee Alliance.