Adaptive Service Provisioning for Wireless Sensor Networks

Authors: Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu

Service provisioning has gained significant attention as a promising programming model for heterogeneous wireless sensor networks. Its key idea is to exploit the decoupling of service providers and consumers to enable platform-independent applications that are dynamically bound to platform-specific services. We explore novel adaptive service binding strategies that are able to cope with network dynamics and to promote energy conservation. To achieve this goal, we developed policies and algorithms that automatically switch providers in response to network topology changes and adapt application behavior when opportunities for energy savings surface. The latter is accomplished by providing limited information about the energy consumption associated with using various services, by systematically exploiting opportunities for sharing service invocations, and by exploiting the broadcast nature of wireless communication in WSNs. The policies and algorithms have been implemented and evaluated on two disparate WSN platforms, the TelosB and Imote2. Empirical results show that adaptive service provisioning can significantly increase service availability and enable energy-aware service binding decisions that result in increased energy efficiency, while imposing minimal additional burden on the... Read complete abstract on page 2.

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation
https://openscholarship.wustl.edu/cse_research/34
Adaptive Service Provisioning for Wireless Sensor Networks

Complete Abstract:

Service provisioning has gained significant attention as a promising programming model for heterogeneous wireless sensor networks. Its key idea is to exploit the decoupling of service providers and consumers to enable platform-independent applications that are dynamically bound to platform-specific services. We explore novel adaptive service binding strategies that are able to cope with network dynamics and to promote energy conservation. To achieve this goal, we developed policies and algorithms that automatically switch providers in response to network topology changes and adapt application behavior when opportunities for energy savings surface. The latter is accomplished by providing limited information about the energy consumption associated with using various services, by systematically exploiting opportunities for sharing service invocations, and by exploiting the broadcast nature of wireless communication in WSNs. The policies and algorithms have been implemented and evaluated on two disparate WSN platforms, the TelosB and Imote2. Empirical results show that adaptive service provisioning can significantly increase service availability and enable energy-aware service binding decisions that result in increased energy efficiency, while imposing minimal additional burden on the application, service, and device developers. Applications involving medical patient monitoring and structural health monitoring are used in the evaluation process.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/34
Adaptive Service Provisioning for Wireless Sensor Networks

Authors: Chien-Liang Fok, Gruia-Catalin Roman and Chenyang Lu

Abstract: Service provisioning has gained significant attention as a promising programming model for heterogeneous wireless sensor networks. Its key idea is to exploit the decoupling of service providers and consumers to enable platform-independent applications that are dynamically bound to platform-specific services. We explore novel adaptive service binding strategies that are able to cope with network dynamics and to promote energy conservation. To achieve this goal, we developed policies and algorithms that automatically switch providers in response to network topology changes and adapt application behavior when opportunities for energy savings surface. The latter is accomplished by providing limited information about the energy consumption associated with using various services, by systematically exploiting opportunities for sharing service invocations, and by exploiting the broadcast nature of wireless communication in WSNs. The policies and algorithms have been implemented and evaluated on two disparate WSN platforms, the TelosB and Imote2. Empirical results show that adaptive service provisioning can significantly increase service availability and enable energy-aware service binding decisions that result in increased energy efficiency, while imposing minimal additional burden on the application, service, and device developers. Applications involving medical patient monitoring and structural health monitoring are used in the evaluation process.

Type of Report: Other
Adaptive Service Provisioning for Wireless Sensor Networks

Chien-Liang Fok, Gruia-Catalin Roman and Chenyang Lu
Washington University in St. Louis
{liang, roman, lu}@cse.wustl.edu

Abstract—Service provisioning has gained significant attention as a promising programming model for heterogeneous wireless sensor networks. Its key idea is to exploit the decoupling of service providers and consumers to enable platform-independent applications that are dynamically bound to platform-specific services. We explore novel adaptive service binding strategies that are able to cope with network dynamics and to promote energy conservation. To achieve this goal, we developed policies and algorithms that automatically switch providers in response to network topology changes and adapt application behavior when opportunities for energy savings surface. The latter is accomplished by providing limited information about the energy consumption associated with using various services, by systematically exploiting opportunities for sharing service invocations, and by exploiting the broadcast nature of wireless communication in WSNs. The policies and algorithms have been implemented and evaluated on two disparate WSN platforms, the TelosB and Imote2. Empirical results show that adaptive service provisioning can significantly increase service availability and enable energy-aware service binding decisions that result in increased energy efficiency, while imposing minimal additional burden on the application, service, and device developers. Applications involving medical patient monitoring and structural health monitoring are used in the evaluation process, demonstrating the system's efficacy.

I. INTRODUCTION

The standardization of low-power wireless communication combined with the continuous evolution of wireless and embedded technologies, result in device heterogeneity in wireless sensor networks (WSNs) [6]. This complicates application development since it requires developers to account for a vast range of devices when creating an application, preventing platform-specific optimizations that are traditionally included to increase energy efficiency.

Service-Oriented Computing (SOC) [17] has been recognized as a promising approach to deal with the complexity of heterogeneous WSNs. SOC consists of service consumers, providers and a service registry. Its primary advantage stems from the decoupling of the consumers and providers. Specifically, consumers and providers each submit service specifications that are used by the service-oriented architecture (SOA) to automatically match and bind consumers to providers. Traditionally, the decoupling was used to enable the consumer and provider to be independently created by different organizations. For example, it enables an Internet-based application running on a web server to access data produced by another application executing within a WSN [20], [2]. More recent research on SOC middleware has enabled heterogeneous sensors to collaborate within the WSN itself [7], [22], [13].

However, despite the promising prior works on SOC for WSNs, they adopted traditional service binding schemes for Internet applications (though with significantly simplified implementations). In this paper, we propose novel service binding schemes to enhance the service availability and energy efficiency of WSNs in an autonomous and application-transparent manner. First, we propose adaptive service binding strategies to automatically adjust the bindings between service providers and consumers in response to changes in the network topology. This is important because WSNs exhibit high levels of dynamics due to node mobility, exposure to a harsh and dynamic environment, and the use of low-power radios susceptible to fluctuations in link quality [11]. A key advantage of our adaptive service binding scheme is that it enables application-transparent handling of network topology changes in a SOC framework, and thus greatly simplifies application development despite network dynamics.

Second, we propose novel service selection strategies to enhance the energy efficiency of the WSN. This is important because when multiple providers are available in a heterogeneous network, each may be configured differently resulting in the consumption of differing amounts of energy. Thus, the selection of a provider affects the energy footprint of an application. To make the SOA energy-aware, a limited amount of information regarding the energy efficiency of a provider is included in the provider’s response to a service discovery request, allowing the consumer to determine which provider will result in the highest energy efficiency. Furthermore, opportunities for sharing service executions are automatically identified and exploited to further increase energy efficiency. This is particularly useful when combined with the broadcast nature of wireless communication, which enables the results of a single execution to be simultaneously delivered to multiple consumers, thus reducing energy consumption.

Significant contributions of this work also lie in the implementation of the adaptive service binding strategies in an SOC middleware specifically designed for heterogeneous WSNs, and comprehensive empirical evaluation through both microbenchmarks and two application case studies. We have implemented the adaptive service binding strategies within an SOC middleware called Servilla [7] on two disparate hardware platforms, the Imote2 [5] and TelosB [19]. These two platforms highlight the vast differences in energy consumption...
that can exist between WSN nodes, demonstrating the need for energy-aware adaptation mechanisms.

To evaluate the efficacy, feasibility, and usability of our adaptive SOA, a detailed analysis of how the adaptation mechanisms are configured for the Imote2 and TelosB platforms is performed. They indicate that using the adaptation mechanism does not impose undue additional burden on the device, service, and application developers. In addition, two real-world application case studies involving medical patient monitoring and structural health monitoring are implemented. The medical patient monitoring application involves a user moving through a region covered by a 74-node WSN testbed spread across two buildings at Washington University in St. Louis [23] periodically invoking services provided by nodes in the testbed. Adaptive service provisioning achieved 100% service invocation success rate despite frequent topology changes caused by user mobility. The structural health monitoring application involves a WSN dedicated to detecting and localizing damage in a structure. It demonstrates the ability of adaptive service provisioning to enhance energy efficiency through energy-aware service selection and sharing.

The remainder of this paper is organized as follows. Section II presents related work. Section III presents the problem definition, including the underlying assumptions. Section IV presents the middleware’s adaptation model that enables automatic rebinding and energy-awareness. Section V presents an evaluation of the additional burden the adaptive service provisioning framework places on the device, service, and application developers. Section VI presents the implementation and evaluation of two applications using the adaptive service provisioning framework, medical patient monitoring and structural health monitoring. The paper ends with conclusions in Section VII.

II. RELATED WORK

SOC has been used in WSNs for various purposes [15]. One original use is to integrate WSNs with Internet applications [2], [20]. To do this, the WSN is hidden behind services that provide sensor data. Using SOC, traditional Internet applications can bind to these services and access information generated by the WSN. While these systems represent major steps toward the integration of WSNs with the Internet, they adopted traditional service binding schemes that are not specifically designed to enhance energy efficiency and service availability, which are common concerns in WSNs.

In addition, SOC has also been used for enabling adaptation to network heterogeneity. To this end, we previously developed Servilla [7], a SOA that facilitates the development of applications that execute efficiently in heterogeneous WSNs. Its key idea was to present platform-specific functionalities as services that are dynamically bound to platform-independent applications. Servilla differs from the system presented in this paper in that it does not provide adaptive service provisioning. Service binding and unbinding is done explicitly by the application and energy efficiency is not automatically considered when selecting a provider — the application had to include it as a required attribute in a service specification, and manually select the most energy-efficient provider. Other systems that use SOC to adapt to network heterogeneity include eSOA [22] and OASIS [13]. They differ from our work by performing service matching and binding off-line on the base station.

In-network reprogramming [18], [14] enables adaptation via code updates. By replacing the code in the WSN, almost any form of application behavior can be added, resulting in maximum adaptation flexibility. However, they differ from the system presented in this paper in that the adaptation decisions are made by the user at a centralized gateway and require disseminating code from the gateway onto the WSN nodes, which is an energy-intensive process. In contrast, the system presented in this paper enables each consumer to automatically make local adaptation decisions in an energy-efficient manner.

Macro-programming [12], [3], [10], [24] is another mechanism for adaptation in WSNs. It enables application developers to treat the entire WSN as if it were a single device by automatically decomposing the application written by the developer into micro-programs that are distributed among the WSN nodes. Adaptation capabilities are achieved via the decomposition process, i.e., it adjusts the decomposition based on the WSN topology in a manner that is transparent to the user. A key difference between macro-programming systems and the system presented in this paper is the fact that the adaptation is done at compile-time before the micro-programs are deployed onto individual WSN node. In contrast, the adaptive SOA presented in this paper performs on-line adaptation within the WSN.

Energy efficiency is another key focus of this paper. It is so important that nearly every aspect of the WSN software stack contains mechanisms for increasing energy efficiency [1]. For example, Santini et. al. [21] presents an adaptive algorithm for predicting sensor data readings, enabling energy to be conserved by decreasing the amount of sensor data that needs to be transmitted. It differs from the system presented in this paper by focusing on optimizing a specific type of data (sensor readings), whereas an adaptive SOA optimizes operations performed by services in general. Given the necessity to consider energy consumption in all aspects of WSNs, making the SOA energy-aware is essential. Unlike previous systems that increase energy efficiency, the system presented in this paper uniquely focuses on how energy can be saved through careful service selection and opportunistically merging service executions.

III. PROBLEM DEFINITION

The two problems addressed in this paper are how SOC can be used to enable applications 1) to transparently adapt to changing network topologies and 2) to conserve energy. This section explains the system model and presents the design goals.

A. System Model

The system consists of a WSN in which there are consumers and providers. Consumers are controlled by applications that
require and invoke services. Providers provide services that are dynamically discovered, bound to, and invoked by consumers. The consumers and providers communicate locally when they share the same node, or over a wireless link when they are located on different nodes. The limited wireless range results in a consumer only being able to communicate with a subset of all matching providers in the network. Since wireless links change over time, this subset of matching providers is dynamic. Currently we only support service invocations over a single hop.

Switching providers is assumed to involve no state transfer from the old provider to the new. Many services like sensing and data routing can be offered in a manner that meets this assumption, though some services like data storage cannot. In the future, this assumption can be removed by including the overhead of state transfer in the energy consumption computations, and implementing a mechanism that determines when a provider is about to disconnect and transfers the data before actual disconnection occurs.

WSNs are different from traditional networks in that they are energy-limited and rate-based. They often remain idle until a particular event like the detection of a phenomenon occurs. To account for these differences, SOC in WSNs have three forms of service invocations: on-demand, periodic, and event-based. On-demand is what is traditionally provided by most SOAs in which an invocation is similar to a remote procedure call. That is, the consumer initiates a service invocation by sending the provider a message, and waits for the provider to respond with results. Unfortunately, the two-way message exchange is energy inefficient if the service needs to be invoked many times. To account for this, periodic and event-based invocations involve the provider automatically invoking the service periodically. They differ in that periodic invocations send every result back whereas event-based invocations only send interesting results, as defined by the provider, back to the consumer. Both forms of invocations are more energy efficient since they do not require the consumer to send the provider a message each time the service is invoked.

While most nodes operate on batteries and are energy-constrained, some nodes are not. For example, in a medical patient monitoring application, a network of nodes that relay data from the patient to the nurse’s central monitoring station can be embedded in the walls and ceilings of the hospital, enabling them to be powered by the building’s electrical grid [4]. Since not all nodes are energy-constrained, the SOA must consider this fact when accounting for energy costs. Specifically, energy consumed by non-energy-constrained nodes should be ignored.

B. Design Goals

The primary objectives of an adaptive SOA are to:

- Enhance service availability through application-transparent service rebinding. This is necessary due to the transient connectivity between the consumers and providers. Achieving it requires developing an algorithm that determines when to switch providers.
- Reduce energy consumption through energy-aware service selection and sharing. The selection of a particular provider affects the amount of energy consumed due to device heterogeneity and differences in wireless link qualities between the consumer and provider. Achieving this objective involves developing an algorithm that determines which provider to select when switching providers.

The adaptation mechanism is considered successful if it hides provider disconnection from the application. Thus, the adaptation mechanism should prevent the application from being exposed to service invocation failure when there exists available providers within its neighborhood. Achieving this requires solving different problems depending on the invocation type. Specifically, a successful adaptation mechanism must ensure that the results of an on-demand invocation are always returned successfully. For periodic invocations, the number of invocation results received must be the number expected. For event-based invocations, the service must be continuously invoked in a periodic manner despite changes in the actual provider providing the service.

In addition to network topology changes, the adaptive mechanism should also conserve energy. In this paper, this is done by reducing an application’s “energy footprint,” which is the total energy an application consumes invoking services. This includes the energy spent on wireless communication and service execution on all energy-constrained nodes in the network (including the hosts of both consumers and providers).

We also have the following additional design goals to enhance the usability and practicality of the SOC system. The first is how to ensure the system is responsive in terms of adapting to network topology changes. Second, the problem of additional overhead for achieving adaptation must be addressed. Specifically, they must not outweigh the energy efficiency gained through adaptation. Finally, the problem of additional burden imposed on the application, device, and service developers must be addressed. Ideally, their software components can be integrated with the adaptive SOA without any changes.

IV. Adaptation Mechanisms

This section presents the adaptation mechanism we’ve developed and integrated into a SOA for WSNs. Before presenting the details, we first give an overview of the basic service selection and binding process. Service selection is the process of selecting one provider from among the set of all known providers that provide the desired service. It involves the consumer analyzing the properties of each provider, and selecting the one that it believes best meets its requirements. Upon selecting a particular provider, the consumer binds to it by noting the provider’s address. This address is used to communicate with the provider when the consumer invokes the service. Note that the address of the provider is hidden from the application by the SOC middleware, which presents to the application a simple interface enabling it to invoke the service.
The remainder of this section is divided into three parts: 1) selecting the most energy-efficient provider, 2) optimizing energy efficiency via shared service invocations, and 3) increasing service availability by adapting to network topology changes.

A. Energy-Aware Provider Selection

This section describes which provider to select. Provider selection must be energy-aware since it impacts energy consumption due to differences in hardware architectures and wireless link qualities. For example, the Imote2 and TelosB differ widely in terms of power draw, i.e., 145mW versus 9mW. Thus, binding to an Imote2 can potentially result in an order of magnitude greater energy consumption relative to a TelosB.

Fundamentally, deciding which provider to select is simple: choose the one that results in the smallest energy footprint. The problem thus becomes how the energy footprint of a particular binding can be determined. Doing this requires analyzing the various steps of invocation, which depends on the type of invocation being performed and whether the provider is local or remote.

First consider on-demand and periodic invocations. Since on-demand invocations are a special case of periodic invocations in which the number of periods is one, both share the same basic steps: 1) initiation, 2) execution, and 3) results delivery. Figure 1 contains a visualization of these steps. Initiation involves the consumer telling the provider that it wants to invoke the service. If the provider is local, this consumes negligible energy since it essentially amounts to a method call. However, if the service is remote, this involves the consumer sending an invoke message to the provider. Execution involves actually running the service. This includes all energy associated with executing the service. Finally, results delivery involves the provider sending the results of the invocation to the consumer. If the provider is local, this consumes negligible energy since it essentially amounts to a method call. However, if the service is remote, this involves one message transmission if it is remote. For periodic invocations, steps two and three are repeated a certain number of times as specified by the consumer.

To determine the energy footprint of a particular binding state, each invocation step must be analyzed. The variables used in the analysis are shown in Table I, and the equation deriving the energy footprint is as follows:

\[ E_{\text{periodic}} = E_{\text{tx,c}} + E_{\text{rx,p}} + \left(\text{Count}\right) \left( \left( P_{\text{idle,c}} \cdot T_{\text{invoke}} \right) + T_{\text{invoke}} \cdot P_{\text{invoke}} + E_{\text{tx,c}} + E_{\text{rx,p}} \right) + \left(\text{Count} - 1\right) \left( \left( P_{\text{idle,c}} \cdot \left( \text{Period} - T_{\text{invoke}} - T_{\text{rx,c}} \right) \right) + P_{\text{idle,p}} \cdot \left( \text{Period} \cdot T_{\text{invoke}} - T_{\text{tx,p}} \right) \right) \]

The first line of equation 1 accounts for the energy in step one. Lines 2-5 account for steps two and three, i.e., the energy consumed during each service execution and results delivery. Note that it is multiplied by Count since that is the number of times the service is invoked. Finally, lines 6-9, account for the energy consumed when either the consumer or provider are idling.

Equation 1 can be simplified when the binding is local, since in this case there is no energy cost associated with wireless communication. Specifically, the equation for local invocation is as follows:

\[ E_{\text{periodic}} = E_{\text{tx,c}} + E_{\text{rx,p}} + \left(\text{Count}\right) \left( P_{\text{idle,c}} \cdot T_{\text{invoke}} + T_{\text{invoke}} \cdot P_{\text{invoke}} + E_{\text{tx,c}} + E_{\text{rx,p}} \right) + \left(\text{Count} - 1\right) \left( P_{\text{idle,c}} \cdot \left( \text{Period} - T_{\text{invoke}} - T_{\text{rx,c}} \right) + P_{\text{idle,p}} \cdot \left( \text{Period} \cdot T_{\text{invoke}} - T_{\text{tx,p}} \right) \right) \]
There is no designation of whether $P_{idle}$ is a consumer or provider, since in a local invocation, they are the same. In addition, equation 2 captures the energy footprint of all forms of local invocation, including those that are event-based, since there is no network communication cost.

The energy cost of remote event-based service invocation is not captured by equation 1 since uninteresting results are not delivered back to the consumer. The sequence of actions performed during remote event-based service invocation is shown in Figure 2. Note that in an actual deployment, the number of invocations that must occur before an interesting one is found may not be known. In this case, the application programmer must estimate the likely number of service executions necessary before one of interest occurs. The equation for deriving the energy footprint of event-based service invocations is given by equation 3.

$$E_{local} = \text{Count} \cdot T_{invoke} \cdot P_{invoke} + (\text{Count} - 1) \cdot (\text{Period} - T_{invoke}) \cdot P_{idle}$$

$$E_{event} = E_{tx,c} + E_{rx,p} + E_{tx,p} + E_{rx,c}$$

The first line of equation 3 captures the energy consumed during steps one and three. Lines 2-3 capture the energy spent in step two. Finally, lines 4-5 capture the energy spent idling between service invocations.

By implementing equations 1, 2, and 3 into the adaptation mechanism described in Section IV-C, the set of matching providers can be automatically sorted according to the amount of energy they will consume. This enables the adaptation mechanism to select the provider that will result in the smallest energy footprint, which is essential in energy-constrained WSNs.

To capture situations in which a node is not energy-constrained, the energy cost of the node can simply be set to zero. Equations 1, 2, and 3 can still be used without modification. For example, if the provider is line-powered, $E_{tx,p}$, $E_{rx,p}$, and $P_{invoke}$ should be set to zero. This will effectively remove non-power-constrained nodes from the energy cost calculation.

As mentioned in Section III, one important requirement of the adaptive middleware is that it does not impose too much burden on the device, application, and service developers. In this case, the additional burden is the derivation of the variables shown in Table I. To understand the actual amount of additional work required of each party, the variables shown in Table I are divided based on who needs to provide them. The device developer needs to specify eight variables related to the energy efficiency and latency of wireless communication and idling. This only needs to be done once for each platform type. The service and application developers each need to specify only two additional variables. In the application developer’s case, the two variables, Count and Period, need to be specified anyway when invoking a service periodically or in an event-based manner. In other words, in most circumstances, there is no additional burden placed on the application developer when enabling adaptive capabilities. The feasibility of deriving these values is shown in Section V, while the validity of the equations are shown in Section VI.

### B. Increased Energy Efficiency via Shared Invocations

Periodic and event-based invocations predictably execute a service once every period. This enables a novel mechanism for saving energy: service sharing. The idea is that multiple service execution requests can be combined into one. In addition, depending on whether reliability is needed, the results can be delivered to multiple consumers simultaneously via wireless broadcast. By reducing the number of times a service needs to be executed and the results delivered, energy savings is possible. This section investigates this possibility.

To understand how energy can be saved via service sharing, consider the impact a particular invocation has on a service’s...
utilization, as shown in Figure 3. Time is discretized into an array of boxes in which each box may or may not execute the service. Thus, when a consumer invokes a service periodically or in an event-based manner, each service execution will fall into a unique box in the array. If a least one invocation occurs during the interval of time that is represented by a box, the box is shaded gray. The number of arrows pointing at each box is the number of consumers that are sharing the same service execution. Thus, the more arrows pointing at a box, the greater the degree of sharing, and the more energy is saved.

Figure 3(a) shows the service utilization when there are two consumers, $C_1$ and $C_2$, invoking at periods $P_1 = 4$ and $P_2 = 6$, respectively. $C_1$ thus executes the service at times 4, 8 and 12, as indicated by the blue arrows, while $C_2$ executes the service at times 6 and 12, as indicated by the green arrows. Note that the length of the array is equal to the least common multiple of 4 and 6 because beyond this, the invocation pattern repeats. Thus, service utilization can be calculated by only considering the block of times leading up to the least common multiple.

Calculating service utilization involves dividing the number of shaded boxes by the total number of boxes, which in this case is $\frac{12}{12} = \frac{1}{2}$. Figure 3(b) shows the utilization when a new consumer, $C_3$, invoking with period $P_3 = 2$, arrives. With this additional consumer, the new utilization is $\frac{1}{2} + \frac{1}{2} = 1$, representing an increase of $\frac{1}{2} - \frac{1}{2} = \frac{1}{6}$. Note that this is less than an increase of $\frac{1}{2}$, which would be the case if service invocations could not be shared, further demonstrating the benefits of service sharing.

Unfortunately, calculating the utilization of a service in general is complex. Consider the algorithm shown in Figure 4. It maintains a sorted list, $list$, that initially contains each period, $P_1$, $P_2$, ..., $P_n$. This initial value is the “base amount” that is continuously added to itself until it reaches $lcm$. With each round, the list is sorted and, if the smallest values are less than the least common multiple, they are incremented by their base amount. This process repeats until all values in $list$ equal $lcm$. The number of rounds in the algorithm is equal to the number of positions in the timeline in which a service execution occurs, meaning the utilization is the number of rounds divided by $lcm$. The time complexity of this algorithm is $O(lcm \times utilization \times n \times log(n))$, which is exponential in the number of invocations. However, it is proportional to the utilization, which may be small, and the number of consumers is also expected to be small due to the limited wireless range of WSN nodes, meaning this algorithm is feasible in most situations.

In the current implementation, the savings achieved through service sharing is incorporated into $P_{\text{invoke}}$ and $E_{\text{tx,p}}$, which are included in the response to a service discovery message. For example, if adding a consumer results in no change in the utilization of the service, and the results can be delivered via broadcast, then $P_{\text{invoke}} = 0$ and $E_{\text{tx,p}} = 0$ for that consumer. This results in a consumer preferring providers that are better able to share service executions and thus save energy. One limitation to this approach is that it does not account for future changes to the set of bound consumers. To account for this, the

1. Given $n$ periods: $P_1$, $P_2$, ..., $P_n$;
2. Let $lcm = \text{Least Common Multiple of } P_1, P_2, ..., P_n$;
3. Let $list = [P_1, P_2, ..., P_n]$;
4. Let $count = 0$;
5. sort(list);
6. while smallest value(s) in list are less $lcm$
7. increment smallest value(s) in list by base amount;
8. sort(list);
9. count++;
10. utilization = count/$lcm$

Fig. 4. A algorithm for calculating service utilization when service sharing is possible.

provider can notify its consumers that the degree of sharing has changed whenever it has decreased.

C. Adapting to Network Topology Changes

The mechanism for adapting to network topology changes is responsible for switching providers to enhance service availability. It is necessary due to the transient connectivity between nodes in a WSN. As shown in Figure 5, the adaptation mechanism has only four states, imposing minimal computational overhead. The system maintains a list of known providers in a provider list, and a count of the number of consecutive failures using the providers in the current provider list. The system begins in the Init state and instantly transitions to the Collect Providers state while transmitting a service discovery message and setting timer $T_{wait}$, since the provider list is initially empty.

When a provider receives the service discovery message, it checks whether it provides the required service by comparing the service specification contained within the message to the specifications of the services it provides. If it finds a match, it replies notifying the consumer that it provides the service.

In addition to containing a service specification, the service discovery message also contains specifications on how the consumer is going to use the service, i.e., the type of invocations that will be performed is indicated. The service provider uses this to calculate the energy footprint the consumer will
have on the provider, which is sent back to the consumer. The consumer records this information in the provider list, and uses it to select the “best” provider, which is by default the one with the smallest energy footprint. The actual criteria for determining which provider is best, can be customized via software modules that can be plugged into the SOA’s middleware.

After broadcasting the service discovery message, the consumer remains in the Collect Providers state accepting and recording responses from service providers until timer \( T_{wait} \) expires. When this occurs, the consumer sorts the list based on the aforementioned criteria for selecting the best provider, and enters the Provider Selection state. From this state, the consumer either selects the best provider and transitions into the Invoke state, or transitions back into the Init state if the provider list is empty.

Once in the Invoke state, the consumer invokes the service while remaining in the same state. If the invocation fails, it discards the provider, returns to the Provider Selection state, where it selects the next-best provider. This process of discarding the currently selected, and theoretically most ideal, provider and switching to the next best provider can repeat up to \( N \) times consecutively before the consumers gives up by flushing the provider list and returning to the Init state. The reasoning behind this is that \( N \) consecutive failures is indicative of a major change in network topology, e.g., when the consumer moves out of range of all previous providers. When this happens, the most logical action is to clear the provider list and re-discover new providers. The value \( N \) is exposed as a tunable parameter. It reflects the expected reliability of receiving a response from a provider, assuming one exists.

The entire adaptation mechanism shown in Figure 5 is conducted by the middleware and hidden from the application developer. Other than the ability to invoke services, the only aspects revealed are those that tune the provider selection algorithm. Specifically, the adaptive SOC middleware allows the developer to specify the algorithm for determining which provider is best, and the values of \( T_{wait} \) and \( N \). By presenting such a simple interface, application development is simplified.

The method of detecting invocation failure differs depending on the type of invocation being performed. On-demand invocations fail if a provider does not respond within a certain amount of time after an invoke message is sent. Periodic invocations fail if the consumer does not receive the expected number of invocation results. Event-based invocations fail if the system does not continue to send interesting events back to the consumer. This can be detected when the current provider is removed from the neighbor list, which is maintained by lower-level services like a link estimator [8].

One important aspect of the adaptation mechanism is the fact that it is reactive. That is, it does not actively seek to change providers so long as the current provider remains available. The reasoning behind this is the fact that energy efficiency is of paramount importance to most WSN nodes, and needlessly searching for new providers when the current one is still available waists energy. In addition, there is no guarantee that a more efficient provider exists, so pro-actively searching for a provider when the current one is available is risky in terms of wasting energy. Finally, some applications like habitat monitoring may infrequently invoke services. In this case, proactive adaptation is wasteful if the application doesn’t even invoke the service between multiple adaptations. For these reasons, we use a passive mechanism that reacts to application invocations and provider disconnections.

V. Evaluation

The actual implementation of the adaptive SOA imposes minimal overhead in terms of memory and network bandwidth. On the TelosB, it consumes \( 20kb \) of ROM and \( 6.5kb \) of RAM, while on the Imote2, it consumes \( 187kb \) of ROM and \( 10kb \) of RAM. These are small relative to the amounts of memory available.

In terms of network bandwidth overhead, the adaptive SOA requires additional information related to energy efficiency to be included in certain messages. The service discovery message must contain four additional variables: the invocation type, period, and count, and whether the invocation results should be delivered reliably. This amounts to 8 bytes of data. The reply message to a service discovery must include six additional variables: \( T_{tx,p}, E_{tx,p}, P_{idle,p}, E_{rx,p}, T_{invoke} \), and \( P_{invoke} \). This amounts to 12 bytes of data and can easily fit within a single TinyOS packet. To support the adaptive SOA, the service specifications must include three additional variables: whether it is sharable, \( T_{invoke} \), and \( P_{invoke} \). This amounts to six bytes of data, and can also fit in a single packet.

The remainder of this section evaluates the additional burden placed on the application, service, and device developers in terms of what they must do to use the adaptive SOA presented in this paper. Understanding the SOA’s ease-of-use is important since a primary objective is to maintain usability and simplify application development. In the process, the variables presented in Section IV are derived. These variables will be used in Section VI.

In this evaluation, two types of nodes are examined: the Imote2 [5] and TelosB [19]. They represent two extremes in energy consumption among current WSN devices, and are often used in today’s WSNs. In addition, one service called AccelTrigger is evaluated. It involves sensing the accelerometer and is used by the structural health monitoring application discussed in Section VI.

Using the adaptive SOA consists of determining the latency, power, and energy values listed in Table I. The remainder of this section analyzes how these values can be derived. It is divided in to four parts: the derivation of the variables associated with idling, sending, receiving, and sensing.

A. Derivation of Variables Associated with Idling

Let \( P_{idle} \) be the amount of energy a device consumes when idle. It is affected by the duty cycle at which the radio operates. Ideally, \( P_{idle} \) is given by equation 4. To determine whether
this is true, the actual power draw of each device operating at various duty cycles is measured.

\[
P_{\text{idle}} \approx \frac{\text{DutyCycle}}{100} \cdot P_{\text{radio-on}} + \left(1 - \frac{\text{DutyCycle}}{100}\right) \cdot P_{\text{radio-off}}
\]

The results are shown in Figure 6. The results indicate that equation 4 does not hold, meaning directly measuring \( P_{\text{idle}} \) at various duty cycles is necessary, especially with the Imote2. By fitting a trend line to the measured data, an equation for \( P_{\text{idle}} \) is obtained. Figure 6(b) contains two sets of data with the sensor board on and off. This is because driver limitations prevent disabling the sensor board between sensor readings.

B. Derivation of Variables Associated with Transmitting

The media access control (MAC) protocol has a significant impact on the efficiency of wireless transmission. In TinyOS 2.1, the default MAC layer is called BoxMAC-2 [16], which utilizes asynchronous duty cycling. This means nodes must synchronize with the receiver each time they transmit a message, as shown in Figure 7. The figure shows that there are three stages to message transmission: search, send, and wait. The search stage consists of the device continuously retransmitting the first packet until it is acknowledged by the receiver. This is necessary due to the use of asynchronous duty cycling. The second stage, send, consists of sending the four remaining messages. The last stage, wait, notifies the receiver that it should expect no more packets and can go back to sleep.

Let \( E_i \), \( T_i \) and \( P_i \) be the average energy, latency, and power draw of performing task \( i \). The goal is to find \( E_{tx} \), the energy consumed during transmission, and \( T_{tx} \), the transmission latency. From the sequence of steps shown in Figure 7, equations 5 and 6 are derived.

\[
E_{tx} = P_{\text{search}} \cdot T_{\text{search}} + P_{\text{send}} \cdot T_{\text{send}} + P_{\text{wait}} \cdot T_{\text{wait}} \tag{5}
\]

\[
T_{tx} = T_{\text{search}} + T_{\text{send}} + T_{\text{wait}} \tag{6}
\]

Among the time variables used in equations 5 and 6, the only one that is dependent on the duty cycle is \( T_{\text{search}} \), which has a range of 0 to DutyCycle. The other time variables are not dependent on DutyCycle. Specifically, \( T_{\text{send}} \) is a function of the network bandwidth, and \( T_{\text{wait}} \) is hard-coded into the MAC layer.\(^1\)

Assuming a normal distribution of clock asynchrony, the average \( T_{\text{search}} \) is theoretically given by equation 7.

\[
T_{\text{search}} = \frac{\text{DutyCycle}}{2} \tag{7}
\]

To determine whether equation 7 is true, the actual \( T_{\text{search}} \) is measured using an oscilloscope for a variety of duty cycles and the results are shown in Figure 7. The results show that equation 7 is a valid characterization of \( T_{\text{search}} \).

The remaining variables in equations 5 and 6 can also be measured using an oscilloscope. Table II contains the results of the measurements assuming one packet per message and no message retransmissions. Note that \( P_{\text{send}} \) and \( T_{\text{send}} \) are both zero because only one packet is being sent, and it is delivered at the conclusion of the wait stage. By plugging in the values

\(^1\)See $TOSROOT/tos/chips/cc2420/lpl/DefaultLpl.h$, constant DELAY\_AFTER\_RECEIVE.
of Table II and equation 7 into equations 5 and 6, the equations for $E_{tx}$ and $T_{tx}$ are obtained.

C. Derivation of Variables Associated with Receiving

Receiving a message consists of two stages, receive and wait, as shown in Figure 9. Thus, the following equations can be created for $E_{rx}$ and $T_{rx}$, the energy and latency of reception, respectively.

$$E_{rx} = (P_{receive} \cdot T_{receive}) + (P_{wait} \cdot T_{wait}) \quad (8)$$

$$T_{rx} = T_{receive} + T_{wait} \quad (9)$$

Using an oscilloscope, the power draws and latencies of the Imote2 and TelosB receiving a message can be measured. The results are shown in Table III. By plugging in the values shown in Table III into equations 8 and 9, the energy cost and latency of message reception is obtained.

D. Derivation of Variables Associated with Sensing

The AccelTrigger service primarily consists of accessing the accelerometer. Thus $P_{invoke} = P_{sense}$ and $T_{invoke} = T_{sense}$. Using an oscilloscope to measure the actual power draw and latency of accessing the accelerometer, the results are shown in Table IV.

All of the values in Table I have now been derived, demonstrating the feasibility of using these variables in the evaluation of the energy footprint of a particular binding state. The ability to use these measurements for estimating the energy footprint of a service binding will be analyzed in Section VI. Note that the values derived in this section assume one packet per message and no message retransmissions. While this holds true for the case studies given in Section VI, if this is not true, the values must be re-derived.

VI. Applications

This section presents two applications implemented and evaluated using our adaptive SOA: medical patient monitoring and structural health monitoring. The medical patient monitoring application focuses on the ability to adapt to changing network topologies, while the structural health monitoring application focuses on energy-awareness.

A. Medical Patient Monitoring

The medical patient monitoring application consists of a mobile user (patient) wearing a WSN device that monitors vital signs and periodically delivering the data to a central monitoring station. The delivery is done via a fixed WSN infrastructure consisting of relay nodes embedded within the

<table>
<thead>
<tr>
<th>Variable</th>
<th>TelosB</th>
<th>Imote2</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{search}$</td>
<td>51.49 ± 0.11</td>
<td>184.44 ± 0.24</td>
<td>mW</td>
</tr>
<tr>
<td>$P_{send}$</td>
<td>0</td>
<td>0</td>
<td>mW</td>
</tr>
<tr>
<td>$T_{send}$</td>
<td>0</td>
<td>0</td>
<td>ms</td>
</tr>
<tr>
<td>$P_{wait}$</td>
<td>54.56 ± 0.06</td>
<td>182.81 ± 0.29</td>
<td>mW</td>
</tr>
<tr>
<td>$T_{wait}$</td>
<td>78.43 ± 2.59</td>
<td>86.5 ± 1.71</td>
<td>ms</td>
</tr>
</tbody>
</table>

TABLE II

THE LATENCY AND POWER ATTRIBUTES OF SENDING ONE PACKET. THE AVERAGE AND 95% CONFIDENCE INTERVALS OVER TEN MEASUREMENTS ARE SHOWN.

<table>
<thead>
<tr>
<th>Variable</th>
<th>TelosB</th>
<th>Imote2</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{receive}$</td>
<td>53.63 ± 0.60</td>
<td>194.18 ± 0.38</td>
<td>mW</td>
</tr>
<tr>
<td>$T_{receive}$</td>
<td>16.45 ± 1.51</td>
<td>22.99 ± 2.13</td>
<td>ms</td>
</tr>
<tr>
<td>$P_{wait}$</td>
<td>54.03 ± 0.09</td>
<td>182.22 ± 0.15</td>
<td>mW</td>
</tr>
<tr>
<td>$T_{wait}$</td>
<td>93.25 ± 0.45</td>
<td>99.64 ± 0.33</td>
<td>ms</td>
</tr>
</tbody>
</table>

TABLE III

THE LATENCY AND POWER ATTRIBUTES OF RECEIVING A PACKET.

<table>
<thead>
<tr>
<th>Variable</th>
<th>TelosB</th>
<th>Imote2</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{sense}$</td>
<td>102.90 ± 0.31</td>
<td>176.67 ± 0.45</td>
<td>mW</td>
</tr>
<tr>
<td>$T_{sense}$</td>
<td>18.49 ± 0.07</td>
<td>21.87 ± 1.28</td>
<td>ms</td>
</tr>
</tbody>
</table>

TABLE IV

THE LATENCY AND POWER ATTRIBUTES OF ACCELTRIGGER.
hospital building. As the patient moves, the monitoring device must adapt to the changing network topology. If it fails to adapt, critical patient data may not be delivered.

A similar clinical monitoring system has been deployed at the Barns and Jewish Hospital in St. Louis, and a clinical trial with real patients is currently underway. While the system deployed in the hospital was implemented in native nesC, reimplementing the system using our adaptive service provisioning framework demonstrated the efficacy of the simple programming model enabled by our middleware system.

For this evaluation, the WSN testbed at Washington University in St. Louis [23] serves as the relay network for delivering patient data to the base station. It consists of 73 TelosB nodes and spans the 5th floors of Jolley and Bryan Halls. A map of the testbed is shown in Figure 10. Each node in this network is line-powered, meaning they are not energy-constrained. For this evaluation, all default settings and configurations were used. Since CTP technically does not “invoke” a “service,” this study focuses on how reliably the patient node is able to send patient data to its parent, which is a relay node. Since the 4BLE decides CTP’s parent, the remainder of this section compares our adaptive SOA to the 4BLE.

For a base-line comparison, the medical patient monitoring application was also implemented using just CTP. This represents a native implementation that involves no SOC. Note that by default, CTP uses the 4-bit link estimator (4BLE) and an exponentially-decaying algorithm for determining beaconing frequency, both of which are included with TinyOS 2.1. For this evaluation, all default settings and configurations were used.

Both the 4BLE and adaptive SOA versions of the application were run using fast and slow walks along the path shown in Figure 10. While traversing this path, the medical patient’s node would attempt to send patient vital sign information consisting of a single 28-byte packet to the base station every 15 seconds, which is sufficient for monitoring most vital signs [4]. Each experiment was run ten times, enabling the calculation of average statistics and 95% confidence intervals.

The success rates of the adaptive SOA and 4BLE implementations are shown in Table V. In both the fast and slow walk scenarios, the adaptive SOA was able to maintain 100% success rate, while the 4BLE failed a significant percentage of times (its success rate was only $40.4 \pm 11.2\%$ and $31.2 \pm 7.5\%$ for the slow and fast walking scenarios, respectively). Our adaptive SOA clearly outperforms the 4BLE, demonstrating the need to adapt to changing network topologies, and the efficacy of our adaptation mechanism.

In addition to success rate, consider the network bandwidth overhead, which is the number of packets transmitted by the patient’s device per service invocation. The non-beacon portion of the network bandwidth overhead is shown in Figure 11.
The average and 95% confidence interval over 10 experimental rounds are shown. Note that the adaptive SOA out-performs the 4BLE transmitting less than ten packets per invocation while the 4BLE transmits about 15-25. This indicates that our adaptive SOA saves energy by transmitting fewer packets, while providing higher service availability.

The average number of beacons emitted per invocation is shown in Table VI. Clearly, the 4BLE emits many more beacons than the adaptive SOA, while delivering lower success rate. The 4BLE emits more beacons because it uses the link estimator for discovering the parent, which rapidly re-broadcasts beacons whenever it detects dynamics in the network. As the patient node moves, the link estimator running on the node may detect changes in the network (based on beacons from new providers) resulting in additional beacons being emitted. More importantly, CTP tells the 4BLE every time it fails to invoke the service, causing the link estimator to emit beacons at a faster rate. The net result is the 4BLE sending about 1.77 ± 0.72 additional beacons per service invocation.

The average latency of invoking the relay service is shown in Figure 12. 95% confidence intervals are included based on the ten experimental rounds. The results indicate that the adaptive SOA has much higher latency than the 4BLE. This makes sense since the adaptive SOA has an adaptation mechanism that continuously retries the service invocation with different relay nodes until it succeeds. Since this process may take many rounds, depending on whether any providers are within range, its latency may sometimes be high. However, from the application’s perspective, the higher latency is usually justified by the 100% success rate of invoking services and lower network overhead provided by the adaptive SOA.

B. Structural Health Monitoring

Structural health monitoring (SHM) is a class of WSN applications that use WSNs to monitor the health of structures like buildings and bridges. A key challenge of SHM applications is the need to run for long periods of time, ideally for the life of the structure, despite having limited energy. The fact that most SHM algorithms are computationally heavy and energy intensive only magnifies the problem. To address this, one solution we explored in the past, and revisit now, is to use a low-power state that simply monitors the vibrations in the building, and signals an event whenever the vibrations are large enough to result in structural damage [7]. With this low-power state, the energy-intensive algorithms do not have to continuously run, thus saving energy.

While previous results demonstrated that this technique can save energy, the process of selecting the node that performs the low-power monitoring was done manually, and the system did not automatically determine the energy cost of selecting a particular node. This section presents how an adaptive SOA can improve on this technique by automatically determining the energy footprint of selecting a particular node. The results are validated by comparing the estimated energy consumption to the actual energy consumption.

The system configuration is as follows. There are two nodes in the network, an Imote2 and a TelosB. The Imote2 is a high-powered but energy-inefficient node that is both a consumer and provider, while the TelosB is a low-powered but energy-efficient node that is just a provider. Both nodes provide a service called AccelTrigger, which performs the low-power monitoring. As a consumer, the Imote2 must bind to and invoke the AccelTrigger service to save energy. Given this setup, there are two binding states: 1) the Imote2 can bind to the AccelTrigger service locally, or 2) it can bind to the service remotely by using the one provided by the TelosB. In addition, there are two variables that need to be supplied by the consumer, Period, and Count, as specified in Table I. The challenge, then, becomes how to determine the energy footprint in terms of the binding state, Period, and Count.

Predicting the energy footprint requires using equations 3 and 2 with values specific to the Imote2 and TelosB platforms, which were derived in Section V. Assuming DutyCycle = 10 and Period = 1000, the question becomes what is the energy footprint of each binding state relative to Count, and when is remote invocations is more energy-efficient than local invocations. The results are shown in Figure 13. The actual values were obtained by directly measuring the energy consumption of the system using an oscilloscope. Note that the predicted energy footprints closely match the actual energy footprints, and that both result in the same conclusion: that Count must be at least 4 for remote binding to be more energy efficient than local binding. Specifically, the measured intersection point between the energy footprint of local vs. remote binding is Count = 3.45, which is close to the predicted 3.95.
Implementing this application using the adaptive SOA is also simple. It consists of a single call to the AccelTrigger service, followed by a callback function implementing the normal energy-intensive structural health monitoring application. As intended, the process of selecting and binding to the most energy efficient provider is hidden from the application.

VII. CONCLUSION

We present an adaptive service provisioning framework that enhance service availability and energy efficiency transparently from applications. Our framework features three novel adaptation strategies specifically designed for service provisioning in WSNs: 1) energy-aware service selection, 2) opportunistic service sharing, and 3) adaptive service rebinding in response to network dynamics. Naturally incorporated into an SOC paradigm, our adaptive strategies are hidden from the device, service, and application developers and thereby simplify application development. Empirical results from implementations on TelosB and Imote2 platforms and an evaluation of two applications, medical patient monitoring and structural health monitoring, demonstrate the systems efficiency and efficacy.

REFERENCES