Coordinating Resource Usage through Adaptive Service Provisioning in Wireless Sensor Networks

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

Dept. of Computer Science and Engineering
Washington University in St. Louis
Saint Louis, MO, 63105, USA
[liang, roman, lu]@cse.wustl.edu

Abstract. Wireless sensor networks (WSNs) exhibit high levels of network dynamics and consist of devices with limited energy. This results in the need to coordinate applications not only at the functional level, as is traditionally done, but also in terms of resource utilization. In this paper, we present a middleware that does this using adaptive service provisioning. Novel service binding strategies automatically adapt application behavior when opportunities for energy savings surface, and switch providers when the network topology changes. The former is accomplished by providing limited information about the energy consumption associated with using various services, systematically exploiting opportunities for sharing service invocations, and exploiting the broadcast nature of wireless communication in WSNs. The middleware has been implemented and evaluated on two disparate WSN platforms, the TelosB and Imote2. Empirical results show that adaptive service provisioning can enable energy-aware service binding decisions that result in increased energy efficiency and significantly increase service availability, while imposing minimal additional burden on the application, service, and device developers. Two applications, medical patient monitoring and structural health monitoring, demonstrate the middleware's efficacy.

1 Introduction

Coordination in wireless sensor networks (WSNs) is critical due to limited resource availability and high levels of network dynamics. The Service-Oriented Computing (SOC) programming model can facilitate this coordination in an elegant and application-transparent manner. It decouples service consumers and providers enabling the bindings between them to be dynamically adjusted in response to resource and network link availability. In this paper, we investigate the use of SOC to coordinate resource utilization in WSNs. This differs markedly from prior research that focuses on the coordination of applications at the functional level.

In this paper, we investigate two novel methods of coordination that increase energy efficiency and service availability in an autonomous and application-transparent manner. First, we propose dynamic service selection strategies that

enhance energy efficiency. This is important because in a dense WSN, there are often multiple providers available operating at varying levels of energy efficiency. Thus, the selection of a provider affects the application's energy footprint. To account for this, a limited amount of information regarding a provider's energy efficiency is sent to the consumer allowing it to determine and bind to the provider that 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 service execution to be simultaneously delivered to multiple consumers.

Second, we propose adaptive service binding strategies to automatically adjust the binding configurations in response to changes in the network topology. This is important because WSNs exhibit high levels of dynamics due to the use of low-power radios, node mobility, and exposure to a dynamic environment [12]. 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 imposes no additional burden on the application developer.

Significant contributions of this work also lie in the implementation and evaluation of the aforementioned coordination strategies. We have implemented them within a SOC middleware called Servilla [8], which is available under an open-source license from its website.[1] In addition, we have evaluated it on two disparate hardware platforms, the Imote2 [4] and TelosB [17], which highlight the vast differences in energy efficiencies among WSN nodes and demonstrate the need for energy-aware adaptation mechanisms. The evaluation indicates that our midleware does *not* impose undue additional burden on the device, service, and application developers. In addition, two real-world application case studies involving structural health monitoring and medical patient monitoring demonstrate our middleware's ability to enhance energy efficiency and enable perfect invocation success rate despite frequent topology changes due to user mobility.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 presents the problem definition. Section 4 presents the mechanisms for coordinating applicaion resource utilization. Section 5 presents the evaluation. The paper ends with conclusions in Section 6.

2 Related Work

Researchers have traditionally focused on the coordination of applications at the functional level. This has generally taken the form of novel abstractions and calculi for sharing and transmitting data among distributed software components. For example, in wireless ad hoc networks, researchers have investigated the use of tuple spaces [3, 22], process calculi [21], workflow engines [20], publish-subscribe [9], and ambient references [5]. While such coordination is critical, the unique properties of WSNs like limited resources motivate the coordination of

[1] Servilla's website: http://mobilab.cse.wustl.edu/projects/servilla/

applications in terms of resource utilization. The programming model we use to achieve this is SOC. SOC has been used in MANETs in the form of follow-me sessions [13] that decouple services from providers. Like the aforementioned systems, it does not consider resource utilization when establishing binding configurations.

SOC has been used in WSNs for various purposes [16]. One original use is to integrate WSNs with Internet applications [1, 18]. 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 facilitate the integration of WSNs with the Internet, they are not designed to enhance energy efficiency and service availability within the WSN itself.

Another use of SOC is to enable adaptation to network heterogeneity. To this end, we previously developed Servilla [8], a service-oriented architecture (SOA) facilitating 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 is *not* energy-ware or adaptive to network topology changes. The changing of binding configurations is done explicitly by the application. Other systems like eSOA [19] and OASiS [15] use SOC within WSNs but perform service matching and binding *off-line*.

Energy efficiency is critical in WSNs because many nodes operate on batteries. As such, making the SOA energy-aware is essential. Unlike previous systems, this paper uniquely focuses on how energy can be saved through careful service selection and opportunistically merging service executions.

3 Problem Definition

The two problems addressed in this paper are how to support coordination by enabling applications to 1) conserve energy and 2) transparently adapt to changing network topologies. Prior to expanding on these, the system model is described. The system consists of a WSN in which there are many types of nodes. Some are more energy efficient. Others are line-powered and not energy constrained. They also differ in their computational and sensing abilities. Regardless of these differences, they all communicate over the same low-power wireless networking technology like IEEE 802.15.4. The result is a heterogeneous and dynamic environment in which devices differ both in terms of hardware capabilities and energy efficiencies.

The WSN runs a SOA in which each device may host one or more service consumers and/or providers. Consumers are typically platform-independent and contain application logic. Platform-specific functionalities are accessed through services that are bound to consumers. Providers are dynamically discovered, bound to, and invoked by, consumers. A consumer and its providers may reside on the same node or on different nodes. Due to variations in network link quality over time, the set of providers that are within range of a consumer is dynamic.

The service discovery, matching, and binding process is automatic and done based on service-specifications published by both the consumer and provider. The service specification includes both functional and non-functional properties of the service. For example, a functional property of the sensing service may be the sensor type and its accuracy, while its non-functional property may be its location. By comparing the properties required by the consumer to the ones provided by the provider, a match can be made. The matching process ensures that all matching services are functionally interchangeable from the consumer's perspective and will satisfy the application requirements.

WSNs are not general-purpose computing platforms and, as such, exhibit certain common characteristics that are reflected by the SOA. For example, many WSN applications like habitat monitoring operate periodically, each time performing the same set of operations like sensing and data delivery. Other applications remain idle until a particular event like the detection of a phenomenon occurs. To account for these operational characteristics, the SOA has 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. Periodic and event-based invocations involve the provider automatically invoking the service periodically. They differ in that periodic invocations send every result whereas event-based invocations only send interesting results, as defined by the provider, back to the consumer. Both of the latter forms of invocations are more energy efficient than on-demand, assuming the same invocation period, since they do not require the consumer to send the provider a message each time the service is executed.

Given the system configuration described above, the primary objectives of our work are to:

- 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 a mechanism that determines *which* provider to select among a set of potential providers. In this paper, the objective of the energy-aware selection and sharing mechanism is to reduce 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.
- Enhance service availability through application-transparent service rebinding. This is necessary due to the transient connectivity between the consumers and providers. Achieving it requires determining *when* to switch providers. Ideally, the adaptation mechanism should prevent the application from being exposed to service invocation failure when there are available providers within range.

In addition to the above two objectives, the following design goals are needed to enhance the usability and practicality of the coordination middleware. The first is how to ensure the system is responsive to network topology changes while remaining energy efficient. This is a challenging problem because rapid proactive detection of network topology changes requires frequent beaconing, which is energy-intensive. 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 with little to no changes.

4 Adaptation Mechanisms

This section presents the adaptation mechanisms of our coordination middleware for WSNs. Before presenting the details, we first give an overview of the basic service selection and binding process. Service selection involves the consumer analyzing the properties of each known provider and selecting one that best meets its requirements. Upon selection, the consumer binds to the provider by noting its address, which is used to communicate with the provider when it invokes the service. Note that the middleware hides the provider's address from the application by presenting a simple interface for invoking 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.

4.1 Energy-Aware Provider Selection

This section describes *which* provider to select. The selection process must be energy-aware since differences in hardware architectures result in different energy cost. For example, the power draws of the Imote2 and TelosB differ widely, i.e., 145mW versus 9mW, meaning binding to an Imote2 may result 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 is how the energy footprint of a particular binding can be determined. By calculating the amount of energy each potential binding configuration will consume, the middleware can select the provider that will result in the smallest energy footprint.

Determining the energy footprint of a binding requires analyzing the steps of invocation. First consider on-demand and periodic invocations. Both share the same three steps since on-demand is a special case of periodic in which the number of periods is one. As shown in Figure 1, the three steps are 1) initiation, 2) execution, and 3) results delivery. Initiation involves the consumer telling the provider that it wants to invoke the service. If the provider is remote, this involves

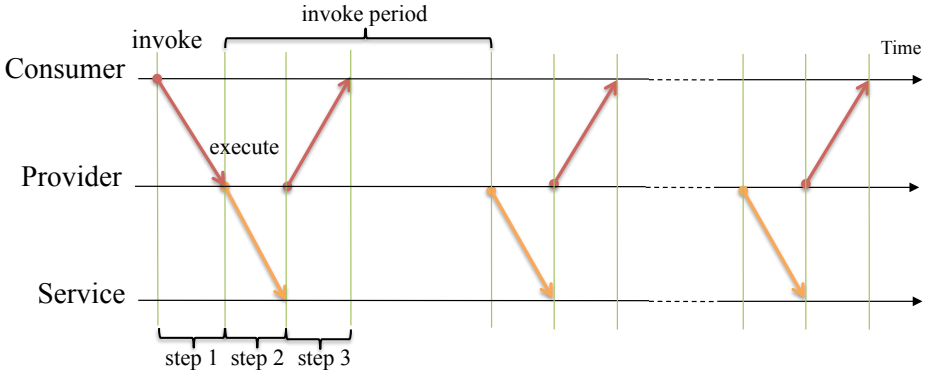


Fig. 1. The actions performed during periodic invocations.

sending an `invoke` message. Execution involves running the service. Finally, results delivery involves sending the results of the execution to the consumer. For periodic invocations, the latter two steps are repeated as specified by the consumer.

Each of the aforementioned steps must be analyzed to determine the energy footprint of a particular binding. The variables used are shown in Table 1. The system-defined variables can be directly measured as will be described in Section 5. The estimated energy footprint is given by equation 1. The first line accounts for the energy in step one while lines 2-3 accounts for steps two and three. Finally, line 4 accounts for the energy consumed while idling. Note that the estimator only considers the energy footprint a consumer has on the overall system. It does not consider the implications of concurrent searches or explicit coordination among consumers or providers, which can result in even greater energy savings. The current equation is used due to its simplicity and ability to estimate actual energy consumption in a real system, as will be shown in Section 5.

$$
\begin{aligned}
E_{periodic} = {} & E_{tx,c} + E_{rx,p} \\
& + \texttt{Count} \cdot (P_{idle,c} \cdot T_{invoke} + T_{invoke} \cdot P_{invoke} + E_{rx,c} + E_{tx,p}) \\
& + (\texttt{Count} - 1) \cdot (P_{idle,c} \cdot (\texttt{Period} - T_{invoke} - T_{rx,c}) \\
& \qquad + P_{idle,p} \cdot (\texttt{Period} - T_{invoke} - T_{tx,p}))
\end{aligned}
\tag{1}
$$

When the binding is local, Equation 1 is simplified since there is no wireless communication. Specifically, equation 2 captures the energy footprint of all forms of local invocation, including those that are event-based. A similar analysis can be done for event-based invocations. It is omitted due to space constraints, details can be found in [7].

Application Developer

Symbol	Meaning	Units
`Period`	Service execution period	ms
`Count`	Number of service executions	n/a

Service Developer

Symbol	Meaning	Units
T_{invoke}	Latency of service execution	ms
P_{invoke}	Power during service execution	mW

Device Developer

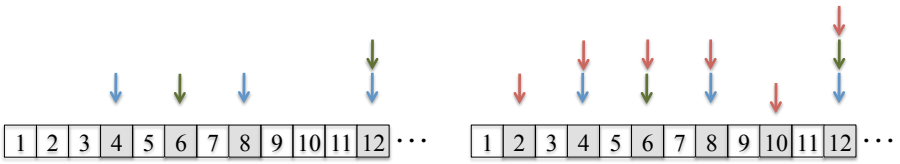
Symbol	Meaning	Units
$T_{rx,c}$	Latency of consumer receiving a message	ms
$T_{tx,p}$	Latency of provider sending a message	ms
$P_{idle,c}$	Consumer idle power	mW
$P_{idle,p}$	Provider idle power	mW
$E_{tx,c}$	Energy cost of consumer sending a message	μJ
$E_{tx,p}$	Energy cost of provider sending a message	μJ
$E_{rx,c}$	Energy cost of consumer receiving a message	μJ
$E_{rx,p}$	Energy cost of provider receiving a message	μJ

Table 1. Variables for deriving the energy cost of service invocation.

$$E_{local} = \texttt{Count} \cdot T_{invoke} \cdot P_{invoke} + (\texttt{Count} - 1) \cdot (\texttt{Period} - T_{invoke}) \cdot P_{idle} \quad (2)$$

If a node is not energy-constrained, the energy cost of the node can simply be set to zero. The original equations can 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 3, the coordination middleware must not impose an unreasonable burden on the device, application, and service developers. In this case, the additional burden is the derivation of the variables shown in Table 1. To understand the actual amount of additional work required, the variables shown in Table 1 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, there is *no additional burden* placed on the application developer when enabling adaptive capabilities.



(a) $P_1 = 4$, $P_2 = 6$ (b) $P_1 = 4$, $P_2 = 6$, $P_3 = 2$

Fig. 2. A visualization of how service utilization is calculated.

4.2 Shared Service Invocations

Periodic and event-based invocations predictably execute a service once every period. This enables *service sharing*, a novel mechanism for saving energy. 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 simultaneously delivered to multiple consumers via wireless broadcast. Energy savings is attained by reducing the number of times a service is executed and the results delivered. This section investigates this possibility.

Before analyzing the details, it is important to note that while not every service is sharable, most are. A service is sharable if its results can be delivered to multiple consumers. This depends on the semantics of the service and whether the invocation parameters are compatible. For example, a sensing service is typically sharable if the invocation periods share common multiples, while a data routing service is usually not. Since the most common service in a WSN is a sensing service, shared service invocation is an important method for increasing energy efficiency. Note that the starting time of a service invocation is assumed to be adjustable to coincide with the discretized time. Most WSN applications, including those that are analyzed in Section 5, meet this assumption since they are only sensitive to the service execution rate.

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 2. Time is discretized and 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. A box is shaded gray if at least one invocation occurs during the interval of time represented by the box. 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 2(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 (lcm) of 4 and 6 because the invocation pattern repeats beyond this. Thus, service utilization

```
1.  Given n periods: P_1, P_2, ..., P_n;
2.  Let lcm = Least Common Multiple of P_1, P_2, ..., P_n;
3.  Let list = [P_1, P_2, ..., P_n]; // These are the base values
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 value;
8.          sort(list);
9.          count++;
10. utilization = count/lcm
```

Fig. 3. An algorithm for calculating service utilization given service sharing.

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 $\frac{4}{12} = \frac{1}{3}$. Figure 2(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 $\frac{1}{2}$, representing an increase of $\frac{1}{2} - \frac{1}{3} = \frac{1}{6}$. Note that this is less than an increase of $\frac{1}{2}$, which would be the case if service invocations could not be shared, demonstrating the benefits of service sharing.

The algorithm for calculating the effects of service invocation sharing is shown in Figure 3. It maintains a sorted list, *list*, that initially contains each period, P_1, P_2, ..., P_n. This initial value is the "base value" 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 *lcm*, they are incremented by their base value. 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 $O(lcm \cdot utilization \cdot n \cdot log(n))$, which is exponential in the number of invocations. However, it is proportional to the utilization, which is usually 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.

The savings achieved through service sharing are incorporated into P_{invoke} and $E_{tx,p}$, 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 $P_{invoke} = 0$ and $E_{tx,p} = 0$ for that consumer. This results in a consumer preferring providers that are better able to share service executions and thus save energy. One limitation is that future changes to the set of bound consumers are not accounted for. This can be remedied by having the provider notify its consumers whenever the degree of sharing has decreased.

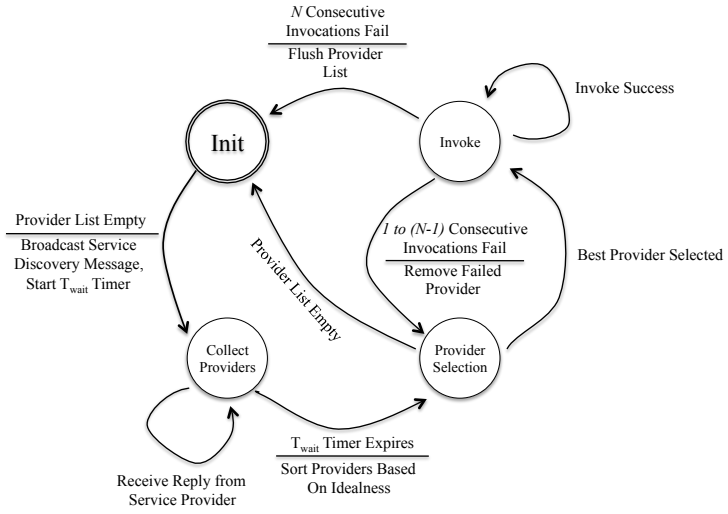


Fig. 4. A finite state machine capturing the behavior of the mechanism used to adapt to network topology changes.

4.3 Adapting to Network Topology Changes

The mechanism for adapting to network topology changes is responsible for switching providers to enhance service availability. As shown in Figure 4, it has only four states imposing minimal overhead. The mechanism only adjusts a service binding when the current one fails. This conservative policy is used due to energy considerations, i.e., to avoid beaconing overhead and needlessly switching providers. For the same reason, the mechanism does not perform service discovery until N previously discovered providers are tried. Note that N must be carefully selected to maximize the likelihood that the energy spent trying to find an alternate provider is less than simply re-running service discovery. Recall that any matching provider is assumed to be interchangeable, enabling the middleware to switch providers transparently from the application. In addition, switching providers is assumed to involve no state transfer from the old provider to the new. This is the common case since typical services like sensing meets this assumption. In the future, this assumption can be removed by including the overhead of state transfer in the energy consumption computations.

The entire adaptation mechanism shown in Figure 4 is conducted by the middleware and hidden from the application developer. By presenting such a simple interface, application development is simplified.

5 Evaluation

We implemented the coordination model by modifying Servilla as shown in Figure 5. The key changes are highlighted in black. Services are augmented

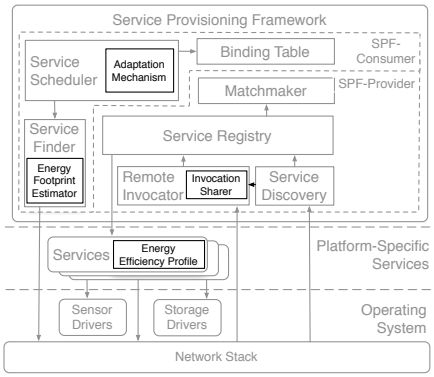


Fig. 5. The middleware architecture.

with their energy efficiency profiles, which include T_{invoke} and P_{invoke}, and various components are modified to enable the coordination of resource utilization. These changes impose minimal memory and network bandwidth overhead. On the TelosB, they consume $20kb$ of ROM and $6.5kb$ of RAM, while on the Imote2, they consume $187kb$ of ROM and $10kb$ of RAM. These are small relative to the amounts of memory available.

In terms of network bandwidth, the service discovery message must contain four additional variables: the invocation type, period, and count, and whether the results need to be delivered reliably, ie., whether the results can be delivered via broadcast. These variables amount to only eight bytes. 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 twelve bytes and can easily fit within a single TinyOS [14] packet. In addition to the messages, the service specifications must include three additional variables: whether it is sharable, T_{invoke}, and P_{invoke}. This amounts to six bytes and can also fit in a single packet.

Recall that the additional burden placed on the developers consists of obtaining the values in Table 1. This can be done using numerous methods the most basic of which is to use an oscilloscope. In this evaluation, this was done for both the TelosB and Imote2 devices. The actual measurements are omitted for brevity, details are available in [7]. Note that this only needs to be done once for each platform or service. In subsequent usages or when an oscilloscope is unavailable, the values can be looked up or estimated [6].

The remainder of this section presents two application case studies: medical patient monitoring and structural health monitoring. They demonstrate the efficacy of our programing model in terms of coordinating resource utilization. While the evaluations are simplified in that they only contain one consumer, they illustrate how our adaptive SOA can (1) increase service availability and (2) enable energy-awareness in applications.

Fig. 6. A map of the WSN testbed at Washington University in St. Louis used in the medical patient monitoring application. The testbed nodes (circles) relay patient data to the monitoring station (triangle). The patient traverses the dotted lines.

5.1 Medical Patient Monitoring

This application was originally deployed at Barnes-Jewish Hospital in St. Louis, Missouri and underwent a successful clinical trial with real patients [2]. It consists of an ambulatory patient wearing a WSN device that monitors and delivers vital sign data to a nurse's monitoring station via a multi-hop WSN infrastructure. The key challenge is overcoming the dynamic network topology caused by patient mobility. Failure to adapt will result in the loss of critical vital sign data jeopardizing the patient's care.

The aforementioned system was implemented in NesC [10] using TinyOS. For this evaluation, we reimplemented it using our adaptive SOA. By using our middleware, the implementation becomes trivial because adaptation to network topology changes is hidden. It consists of a single loop with two lines of code: one for obtaining the patient data, another for invoking a relay service that is provided by the infrastructure nodes. The relay service delivers the data to the monitoring station and is identical to the one used in the original deployment.

We reproduced the results of the clinical deployment using a WSN testbed, a map of which is shown in Figure 6. It runs the Collection Tree Protocol (CTP) [11] to deliver patient data to the monitoring station, which was used in the original implementation. CTP is exposed through our middleware as a service. For each experiment, the patient traversed a fixed $359m$ path as shown in Figure 6. Two walking speeds were used, a slow walk averaging $0.68m/s$ and a fast walk averaging $1.33m/s$.

For a base-line comparison, the application was also implemented using just CTP. Both the adaptive SOA and CTP versions of the application were evaluated using the fixed path shown in Figure 6. While traversing this path, the medical patient's node would attempt to send patient data every 15 seconds, which is sufficient for monitoring most vital signs [2].

	Adaptive SOA	CTP
Fast Walk	$100\% \pm 0\%$	$31.16\% \pm 7.6\%$
Slow Walk	$100\% \pm 0\%$	$40.47\% \pm 11.2\%$

Table 2. Medical patient monitoring service invocation success rate.

The success rates of the adaptive SOA and CTP implementations are shown in Table 2. For all results, the average and 95% confidence intervals over ten rounds are given. The adaptive SOA was able to maintain 100% success rate while the CTP version frequently failed because it was not designed to handle the high level of dynamics due to mobility. In addition the adaptive SOA approach was more energy efficient transmitting an average of 5 packets per invocation relative to CTP's 20. This demonstrates the efficacy of our middleware in terms of coordinating adaptation to network topology changes.

5.2 Structural Health Monitoring

A key challenge of structural health monitoring (SHM) is the need to run for long intervals despite the fact that most SHM algorithms are complex and energy intensive. To address this, a low-power state may be used that simply monitors the structure's vibrations and signals an event whenever they exceed a certain threshold. While previous results demonstrated that this technique saves energy [8], the binding configurations were set manually. This section presents how an adaptive SOA can improve on this technique by automatically determining the energy footprints. The results are validated by comparing the estimated and actual energy consumptions.

The system consists of Imote2s and TelosBs. Both provide a service called `AccelTrigger` that performs low-power monitoring. Among the two, only the Imote2 can perform the complex algorithms for localizing damage. To save energy, the Imote2 relies on `AccelTrigger` during idle periods. 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 TelosB. In this evaluation, we determine how well our middleware predicts the energy footprints of the two binding configurations.

Assuming the service is invoked every second using event-based invocation, the Imote2 must determine the energy footprint of each binding configuration relative to the number of times it is invoked (`Count`). Ultimately, the objective is to determine when remote invocations are more energy-efficient than local invocations. The results are shown in Figure 7. The actual values were obtained using an oscilloscope. Note that the predicted and actual energy footprints closely match, and that both result in the same conclusion: that `Count` must be at least 4 for remote binding to be more efficient. Since `Count` is specified by the application during service invocation, the middleware is able to automatically determine which binding configuration is best.

Implementing this application using the adaptive SOA is simple. It consists of a single call to invoke the `AccelTrigger` service, followed by a callback func-

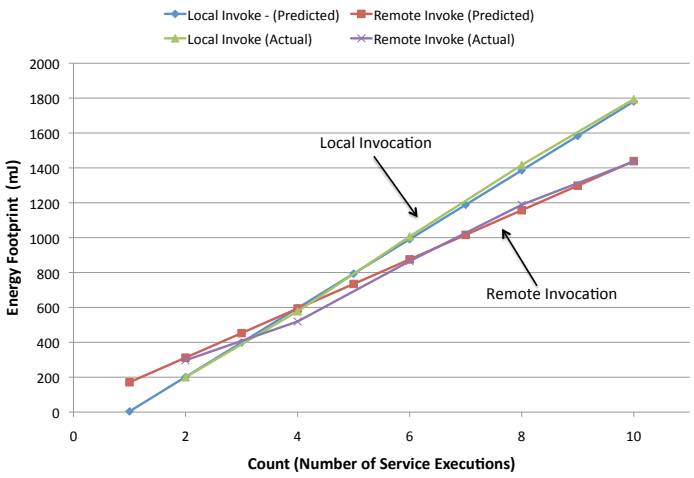


Fig. 7. The predicted and actual energy footprints of the structural health monitoring application when the radio operates at a 10% duty cycle and `Period` $= 1000ms$.

tion implementing the normal energy-intensive SHM algorithm. As intended, the process of binding to the most energy efficient provider is hidden from the application.

6 Conclusion

We present a middleware for coordinating resource utilization among WSN applications through the use of an adaptive service provisioning framework. The framework enhances service availability and energy efficiency automatically imposing minimal additional burden to the developer. 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.

Acknowledgment

This research was supported by the NSF under grants CNS-0520220 and CNS-0708460.

References

1. Arch Rock. Arch Rock PhyNet^{TM}. `http://www.archrock.com/product/`.

2. O. Chipara, C. Brooks, S. Bhattacharya, C. Lu, R. Chamberlain, G.-C. Roman, and T. C. Bailey. Reliable data collection from mobile users for real-time clinical monitoring. In *AMIA Annual Symp.*, November 2009.
3. P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco. TeenyLIME: transiently shared tuple space middleware for wireless sensor networks. In *MidSens'06*, pages 43–48, 2006.
4. Crossbow Technologies. Imote2 datasheet. `http://tinyurl.com/5jrw85`.
5. T. V. Cutsem, J. Dedecker, and W. D. Meuter. Object-oriented coordination in mobile ad hoc networks. In *Coordination'07*, pages 231–248, June 2007.
6. P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler. Energy metering for free: Augmenting switching regulators for real-time monitoring. In *IPSN '08*, pages 283–294, 2008.
7. C.-L. Fok, G.-C. Roman, and C. Lu. Adaptive service provisioning for wireless sensor networks. Technical Report WUCSE-2009-83, Washington University in St. Louis, December 2009.
8. C.-L. Fok, G.-C. Roman, and C. Lu. Enhanced Coordination in Sensor Networks through Flexible Service Provisioning. In *Coordination'09*, pages 66–85, June 2009.
9. D. Frey and G.-C. Roman. Context-aware publish subscribe in mobile ad hoc networks. In *Coordination'07*, pages 37–55, June 2007.
10. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language: A holistic approach to networked embedded systems. In *PLDI '03*, pages 1–11, New York, NY, USA, 2003. ACM.
11. O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *SenSys'09*, November 2009.
12. G. Hackmann, O. Chipara, and C. Lu. Robust topology control for indoor wireless sensor networks. In *SenSys '08*, pages 57–70, 2008.
13. R. Handorean, R. Sen, G. Hackmann, and G.-C. Roman. Supporting predictable service provision in manets via context aware session management. *Int. Journal of Web Services Research*, 3(3):1–26, 2006.
14. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.
15. M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits. Oasis: A programming framework for service-oriented sensor networks. In *COMSWARE'07*, 2007.
16. E. Meshkova, J. Riihijarvi, F. Oldewurtel, C. Jardak, and P. Mahonen. Service-oriented design methodology for wireless sensor networks: A view through case studies. *SUTC'08*, 0:146–153, 2008.
17. J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *IPSN '05*, page 48, 2005.
18. N. B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. In *SenSys '08*, pages 253–266, 2008.
19. A. Scholz, C. Buckl, S. Sommer, A. Kemper, A. Knoll, J. Heuer, and A. Schmitt. eSOA - service oriented architectures adapted for embedded networks. In *IDIN'09*, pages 599–605, June 2009.
20. R. Sen, G.-C. Roman, and C. D. Gill. Cian: A workflow engine for manets. In *Coordination'08*, pages 280–295, May 2008.
21. A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks. In *Coordination'08*, pages 296–314, May 2008.
22. M. Viroli and M. Casadei. Biochemical tuple spaces for self-organising coordination. In *Coordination'09*, pages 143–162, June 2009.