Essentia: Architecting Wireless Sensor Networks Asymmetrically

Tian He[§], John A. Stankovic[†], Radu Stoleru[‡], Yu Gu[§] and Yafeng Wu[†]

[§]Department of Computer Science and Engineering, University of Minnesota
[†]Department of Computer Science, University of Virginia
[‡]Department of Computer Science, Texas A&M University
Email:{*tianhe,yugu*}@*cs.umn.edu,* {*stankovic,yw5s*}@*cs.virginia.edu, stoleru*@*cs.tamu.edu*

***Abstract*—In this paper, we advocate *asymmetric function placement* as one of guiding principles to architect sensor network systems. We demonstrate its generic applicability and effectiveness by applying this principle to three typical sensor network technologies, namely, localization (*Spotlight*), sensing (*uSense*) and communication (*mNets*). These technologies have very dissimilar features, representing a wide spectrum of system design requirements. We have invested significant effort to design, implement and evaluate our techniques on TinyOS/Mote testbeds. The results from several running systems indicate that asymmetric function placement is a powerful guiding principle to achieve *efficiency* and *high-performance* simultaneously in wireless sensor networks. At the end, we exam the system features that discourage the use of asymmetric function placement and approaches to address them.**

I. Introduction

Wireless Sensor Networks (WSN) support many promising applications, such as military surveillance, infrastructure protection, scientific exploration and smart environments. Researchers in this area have accumulated a large portfolio of designs, effectively addressing a wide range of case-by-case problems. However, we are still searching for guiding principles to efficiently compose sensor network systems as a whole. In this paper, we attempt to address this issue by (i) advocating the design principle of *Asymmetric Function Placement*, and by (ii) designing and prototyping several sensor network technologies based on this design principle, and (iii) by presenting the insights, limitations, challenging issues learned from our study.

As evident in the Internet evolution, good design principles contribute to a successful architecture. The famous "end-to-end argument", proposed by Saltzer, Reed and Clark [28], is one of these design principles that led to the success of the Internet. It suggests pushing functions, which require end-to-end guarantees (e.g., reliable delivery), to the edge of the network in order to avoid redundancy and inflexibility. We note that although the "end-to-end argument" is proposed as a design principle independent of the TCP/IP architecture, it provides important guidance as to where each network function should be implemented. Having observed the importance of design principles to the success of the Internet, we raise the following questions: What are the key design principles for sensor network architectures? And how can we efficiently utilize these design principles in existing and future sensor systems?

A sensor network is fundamentally different from Internet in two key aspects. First, sensor nodes are very constrained devices with limited computation and communication resources. Therefore efficiency of protocol designs should be a major architecture concern in sensor networks. Second, a sensor network interacts with physical environments, while the Internet builds a logical world where human-computer interaction dominates. These differences make new design principles necessary.

In general, sensor network architecture involves multiple design principles, guiding the design of architecture-level service and control interfaces, decomposition and composition methodologies of functional components. In this paper, we do not intend to enumerate every design principle that is potentially useful, but to advocate the principle of *Asymmetric Function Placement – a sensor-network equivalence to the end-to-end argument found in the Internet*. It should be emphasized first that asymmetric design by itself is not new in general context - it has been widely proposed and well articulated in the context of Client/Server architecture, and implicitly used in many existing sensor network designs [10], [11], [31], [39]. The main novelty of this work is to 1) demonstrate systematically that this design principle can be applied in various aspect of wireless sensor networks, using several novel solutions, 2) explain how to apply this design principle under the right sensor network settings appropriately, and 3) reveal Asymmetric Function Placement can lead to *generic*, *efficient* and *powerful* designs across different applications.

In the rest of the paper, we first explain the concept of *Asymmetric Function Placement*. And then we present three sensor network designs and physical implementations, namely *Spotlight*, *uSense* and *mNets*, which use asymmetric function placement as the guiding design principle. We carefully choose these designs so that they have very dissimilar features and represent a wide spectrum of requirements to evaluate the generality of asymmetric function placement. In each design, we qualitative and quantitatively explain why this design principle plays a key role not only in simplifying the sensor-level design but also in improving the system-level performance as a whole. We note that it is unfair to claim asymmetric function placement can be applied in every sensor system design. Therefore, we discuss the system features that discourage the use of this design principle and possible approaches to address them at the end of this paper.

II. The Concept of Asymmetric Function Placement

Essentially, function placement advises *where* each function should be implemented in sensor network systems to achieve overall architecture and performance objectives efficiently. The guiding principle we identify here is called *Asymmetric Function Placement*. It suggests decoupling the non-essential functions from the core of the sensor network, an approach to obtain a slim architecture that can be supported on the resource constrained sensor nodes. The principle of asymmetric function placement deems a function *non-essential*, if sensor applications operate well without direct support from this function. Non-essential functions, if embedded within the sensor network, not only increase the footprint (code size) of the implementation, but also consume valuable resources such as bandwidth and computation. By decoupling the non-essential functions and implementing them outside the network core, we achieve *efficiency* and *performance* simultaneously, because of following two reasons: First, with fewer functions sharing the limited resources on a node, we can build the essential functions, the ones that must be embedded into individual sensor nodes, in a less stringent design space. Second, we can design and implement non-essential functions outside of sensor networks free of resource constraints inherent in the sensor nodes. These functions, therefore, can be very sophisticated and powerful, leading to a significantly improved overall performance.

We note that the decoupling of non-essential functions is not straightforward, but a rather challenging task for two reasons. First, most functions seem to be essential if considered *as a whole* and we cannot simply re-implement them entirely outside of the sensor nodes. We will demonstrate this later in the paper. Second, application features would affect whether a function is essential or not. It would also affect how decoupling should be done. It is a demanding task to conceive application-specific efficient decoupling mechanisms.

Although asymmetric design is not new in computer science field, it is new to clarify, validate and consolidate this design principle from system perspective in the context of sensor networks, using several running systems. More specifically, our contributions lie in two main aspects.

- We reveal asymmetric function placement's ability to reduce the in-network complexity and enhance system performance *simultaneously*. To indicate how widely this principle can be used, we have designed a set of novel asymmetric solutions, demonstrating their architecture and performance superiority in several TinyOS/Mote testbeds.
- We analyze the differences and integration between our design and other existing architectures such as Tenet [10], SNA [25], COMPASS [27] and WaveScope [20]. This contributes to the ultimate architecture design convergence in the future.

III. Applications of Asymmetric Function Placement

Architecture design can be regarded as a process of systematically decomposing and composing of functions. Since a monolithic system leads to difficulties in interoperability and extensibility, it is an indispensable step to first decompose a system into different functions and answer *where* each function should be realized. A thoughtful function placement principle would lead to an effective implementation as indicated by the "end-to-end argument". Another notable function placement principle is put forward by UCLA's Tenet [10] project, which advocates that multi-node data fusion functionality and complex application logic should be implemented only at the second tier (e.g., master nodes), because the cost and complexity of in-network processing would overshadow their benefits. Here we advocate the asymmetric function placement principle that is different, however complementary. We decouple sensor network functions into two categories: essential and non-essential functions. Essential functions are the ones that have to be supported within the network, while non-essential functions are the ones that need not be embedded as a part of the network. Different from Tenet, we do not dictate a tiered architecture and we address function placement at a *finer* granularity.

Since we do not expect asymmetric function placement to apply to all application domains, we scope our work by assuming the nodes are stationary (no mobility), which is the case for most existing deployed sensor systems [5], [13], [30], [32], [34], [38]. In a highly-mobile sensor network, we expect the benefit of using asymmetric function placement might not apply. Since mobile nodes are normally more capable than the types of devices we are considering, traditional architectures might be used for them.

To illustrate the architectural benefits of asymmetric function placement, in the rest of the section, we present three examples, namely, *Spotlight*, *uSense* and *mNets*. We organize each example as follows: We first briefly describe the technical design, and then illustrate how to apply the principle of asymmetric function placement. At the end of each example, we evaluate the design through quantitative experiments and qualitative description of architecture insights.

A. Spotlight Localization

In this section, we use Spotlight localization system [31] as the first example for asymmetric function placement. To support location-aware applications, we need two functions, namely localization and a location service. Localization is a process of obtaining the (x, y, z) location information of individual nodes, while the location service defines how to expose and use these information. We note that localization is an indirect mechanism while the location service is essential function, since applications only use location information as inputs and they do not care about how the location is obtained.

To decouple the non-essential localization function, the Spotlight system employs an asymmetric architecture, in which sensor nodes do not need any special hardware. We push all the sophisticated hardware and computation into a powerful device, called Spotlight, which is not part of the network. This asymmetric function placement leads to simplified sensor-level architecture as well as a much more powerful localization design.

Fig. 1. Scenario

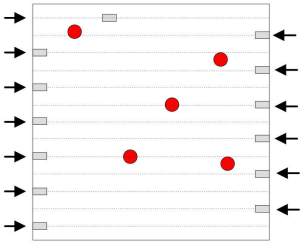

Fig. 2. Point Scan

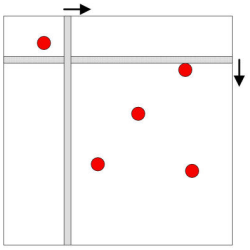

Fig. 3. Line Scan

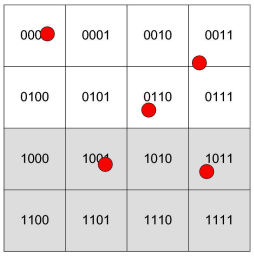

Fig. 4. Area Cover

The main idea of the Spotlight localization system is to use a Spotlight device to generate *controlled events* in the field where the sensor nodes are deployed. An event could be, for example, the light projected into an area. Using the time when a controlled event is detected by a sensor node and the space-time correlation of the generated events, location information regarding the sensor node can be inferred.

- **Advanced Design at the Spotlight Device:** We illustrate the Spotlight idea further with Figure 1. Figure 1 shows a sensor localization scenario: After deployment, the sensor nodes time synchronize with each other. An (aerial or terrestrial) vehicle with a Spotlight device generates a sequence of controlled light events. For example, a controlled sequence of point events can be described as the event locations (x, y, z) over time t: $f : \mathbb{R} \rightarrow \mathbb{R}^3$, where $f(t) = (x, y, z)$. A sensor node detects the event at time t and reports t to the Spotlight device. Since the spatiotemporal property $f(t)$ of the controlled events is known to the Spotlight device, it can compute the location of the sensor nodes by inputting the time-stamp t into the sequence $f(t)$. Besides disseminating of point events as shown in Figure 2, we can also use line and area events to obtain the nodes' locations. Figure 3 shows how line-scan works in a 2-D space. Instead of disseminating a single spot, line-scan create a line of light and scans the area twice, first with a vertical event line followed by a horizontal event line. After two scans, each node gets two time stamps: t_1 and t_2, which can be used by the Spotlight device to obtain x and y coordinates of a node. Figure 4 shows how the area-cover method works. The 2-D area is divided into multiple sections, each with a unique code. Events are disseminated according to the code (1/0 denotes the existence/void of the event). The 0/1 detection sequence of a node can be used to identify in which section this node is located. For example, as shown in Figure 4, if a node reports a 0011 event sequence, this node must be at the right upper corner of the field.
  The Spotlight device is designed to be powerful enough to support many types of controlled events, allowing the tradeoff between the cost and time to achieve localization. Figure 5 shows both indoor and outdoor version we implemented. The indoor version (left) uses an Infocus LD530 projector connected to an IBM Thinkpad laptop. The outdoor version (right) uses diode lasers, a computerized telescope mount (Celestron CG-5GT), and an IBM Thinkpad laptop.

Fig. 5. μSpotlight System and Spotlight System

- **Simple and Generic Detection Function at Sensor Nodes:** In contrast, sensor-level design is very simple and generic. Sensor nodes need only to support simple detection, time-stamping and report functions. In fact, sensor nodes are blissfully unaware of which type of events a Spotlight device generates. This lightweight and generic design not only reduces the memory and energy during the operation, but also decreases the cost for reprogramming and the overhead of debugging.

1) Evaluation of Spotlight: We evaluated the performance of Spotlight localization both indoors and outdoors (Figure 6). Here we only present the relevant results. Based on the asymmetric design, Spotlight localization achieves (i) a centimeter-level accuracy as shown in Figure 7, which is $1 \sim 2$ orders of magnitude more accurate than well-known range-free schemes such as centroid [2] and amorphous localization [21], and (ii) over 1000 meters localization range as shown in Figure 8, which is two orders of magnitude longer than state-of-the-art range-based localization systems such as Cricket [26]. And we note that long range and high accuracy are achieved simultaneously without any additional cost at sensor nodes. The only process performed at a node is simple event detection with time-stamping. The cost of a spotlight device (e.g., 800$ currently) can be amortized per node over multiple usages.

2) Architecture Insights: We conclude that the significant improvements in the Spotlight system come from the architecture design. Previous symmetric solutions such as [2], [12], [21], [26] implement the localization function completely within the sensor networks. Developing these symmetric solutions has limited design choices due to the resource constraints. For examples, existing localization protocols normally fall into one of two categories: *range-based* and *range-free*. The former is defined by protocols that use absolute point-to-point distance estimates (range) or angle estimates for calculating location. The latter makes no assumption about the availability

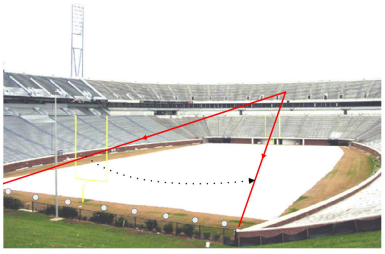

Fig. 6. Deployment Scenario

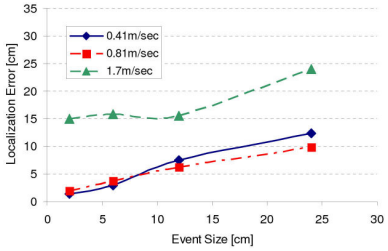

Fig. 7. Localization Accuracy

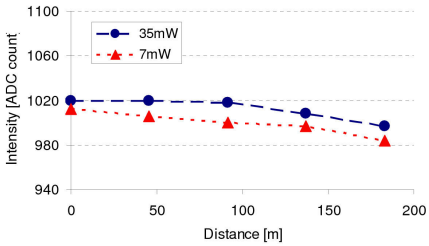

Fig. 8. Effective Range

or validity of pair-wise distance information. In both categories, cost and accuracy are at odds with each other. In order to obtain a high accuracy, range-based protocols need to equip the sensor nodes with sophisticated hardware devices capable of ranging. In order to reduce cost, range-free protocols have to use only connectivity information, resulting in a large error in localization.

In contrast, our asymmetric function placement allows non-essential functions (e.g., localization) to be designed in a resource-rich design space (Spotlight device), hence these functions can be powerful enough to improve the performance of sensor systems. For example, outdoor experiments [31] (Figure 6) show that the Spotlight system can achieve as low as 5 centimeters accuracy over a 170-meter range without any additional hardware on sensors.

In addition, the asymmetric function placement improves architecture flexibility by removing application-specific mechanisms outside of sensor nodes. As shown in Spotlight, the time-stamp function is *generic*. Three localization methods (i.e., point-scan, line-scan and area-cover) can be supported by the Spotlight device, requiring no changes to the sensor devices.

B. uSense Coverage Framework

In this section, we introduce a coverage framework, called uSense [11], using asymmetric function placement. The objectives of the uSense design are the flexibility in configuration and significant energy efficiency in sensing coverage. More importantly, we use uSense as another example to illustrate the design philosophy of asymmetric function placement.

It is often the case that a sensor network needs to support multiple operating scenarios. For example, a military surveillance network could be required to provide robust full coverage during a red alert (a spatial coverage requirement), while tolerating a certain detection delay (a temporal coverage requirement) at other times to conserve energy. Two algorithms ([33] and [3]) have been successfully designed to meet these two design goals. However, neither of them, unfortunately, is flexible enough to meet both requirements. Evidently, with these two algorithms, we can achieve flexibility by reprogramming the coverage component through Deluge [15] and other wireless download tools [18]. However, reprogramming is expensive in communication cost, making flexibility and efficiency at odds with each other.

We observe that the wakeup/sleep schedule of individual sensor nodes can be described by a set of parameters, which are independent of the mechanisms to obtain the schedules. Therefore, we suggest a conceptual decoupling of *scheduling* from *switching*. The scheduling function calculates the parameters of a working schedule (wakeup/sleep durations) for individual nodes, while the switching function turns on and off the sensors according to these scheduling parameters. As shown in Figure 9, uSense features an asymmetric solution, which consists of a lightweight algorithm running at the sensor and a sophisticated design and implementation outside the sensor network. To support this, a two-way communication is needed between the outside entity and the sensor network.

- **Generic Switching Algorithms at the Sensor:** We develop a switching algorithm that is lightweight enough to run on the resource constrained nodes and generic to accommodate various types of schedules. We use two parameters in the switching algorithm design, namely schedule bits S and switching rate R:
  (1) The schedule bits S are an infinite binary string in which 1 denotes the active state and 0 denotes the inactive state. The duty cycle of a node is the percentage of 1s in S. Since the node schedule is normally periodic, it can be concisely represented by a regular expression (to save dissemination overhead). For example, $(0010)^*$ can be used to denote a repeated off-off-active-off schedule.
  (2) The switching rate R defines the toggling rate between states. For example, a switching rate of 2HZ requires a node to read 2 bits from the schedule per second. Theoretically, when the switching rate approaches infinity, schedule bits can precisely characterize any on/off behavior generated by any coverage algorithm. In reality, the switching rate is finite, therefore in the worst-case, a node might need to extend the wake-up period by $\frac{1}{S}$ seconds to guarantee the desired coverage. To build an efficient switching algorithm, we build a Timed Finite Automaton (TFA) to interpret the schedule S, associating a time value to each 0/1 and 1/0 state transition. For example, Figure 10 shows a state-transition of a TFA, which interprets the schedule $(10000010)^*$. In this TFA, $A0$ and $A2$ are two active states and $S1$ and $S2$ are two sleep states. The state transitions are triggered by the expiration of the time.
- **Global Scheduling Algorithms at the Outside Entities:** As shown in Figure 9, we develop sensor-scheduling algorithms outside the network (e.g., multiple computational nodes can be used to distributively collect data and execute the algorithm). Free of resource constraints, this outside entity can support a large number of sophisticated coverage algorithms [4], [29], [6], [16], [3], [35], [36]. We design a generic parameter translation engine, which takes

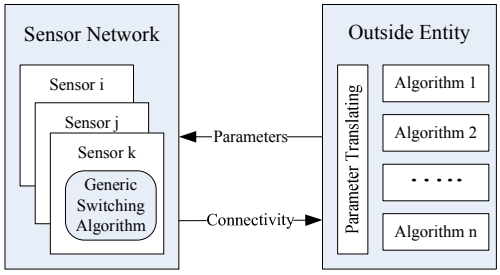

Fig. 9. Overview of uSense Architecture

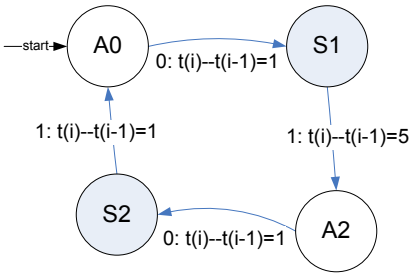

Fig. 10. Time Finite Automata

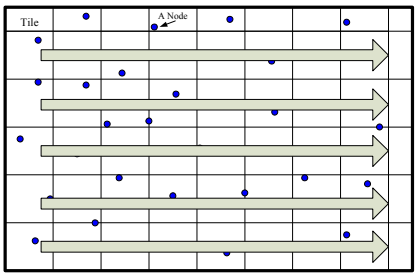

Fig. 11. Scan with Tessellations

the output of a sensing algorithm and converts it to the S and R parameters. We note that uSense's asymmetric architecture is especially friendly to global scheduling algorithms, which might require excessive communication if implemented within the core of network. Since global scheduling allows more nodes to be activated than scheduling schemes that only schedule nodes within the neighborhood, it leads to a significant power savings compared with the localized coverage algorithm. As proofs of concept, we have designed and implemented two global scheduling algorithms: line scan and systolic scan. To illustrate the idea, we only use a simple example here. Figure 11 shows that an area is partitioned into some small tiles. Instead of covering all tiles 100% of time, we only cover a column or a row of tiles in a certain interval of time during one round of horizontal or vertical scan. Because only a small number of tiles are sensed at a specific point of time, many more nodes are put into sleep mode than a localized scheduling algorithm can. Therefore, it reduces energy consumption significantly. One key research challenge we have addressed is to optimize the node-to-tile assignment according to the spatiotemporal features of scanning algorithms. Due to the space constraint, we omit this non-architecture related detail.

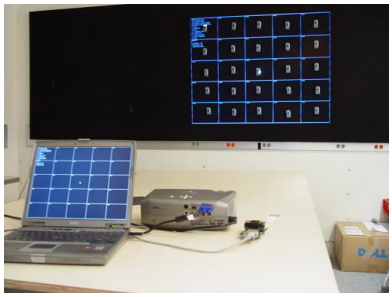

Fig. 12. uSense System Setup

1) Evaluation of uSense: We have implemented a compete version of uSense as designed in previous section. As shown in Figure 12 to evaluate uSense, we attach 25 MicaZ motes on a veltex board (4 feet by 12 feet) using velcro straps. We use a DELL 2300MP projector to generate light spots on the veltex board. These light spots are used to emulate static and mobile events. Due to the space constraints, we summarize our results as follows: uSense can significantly improve the average life for

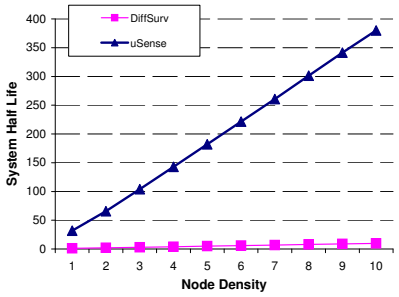

Fig. 13. System Life Time

single node and the half life for the whole network, compared with the full coverage algorithm [35] as shown in Figure 13, because its ability to support global scheduling without requiring coordination between neighboring nodes. Second, due to asymmetric design, uSense implementation has a very small footprint. The generic switching algorithm is written with the NesC language [8], running on the TinyOS/Mote platform. The compiled image of a full implementation occupies 21,040 bytes of code memory and 907 bytes of data memory. Third, other than the cost to disseminate the scheduling bits, there is no need for coordination among nodes.

2) Architecture Insights: uSense has several nice features originating from the architectural design: (i) The *generic* design supports a fast and efficient policy change. We only need to disseminate a few new parameters to switch between different coverage algorithms. This property is similar to the Spotlight design where multiple event distribution functions can be supported without changing the code at sensor nodes. (ii) It allows an efficient implementation of the global scheduling algorithm, which coordinates nodes within a large area, leading to significantly better performance compared with the localized solutions. (iii) Only a simple switching algorithm is implemented on the sensor nodes, which leads to savings in memory and energy consumption.

C. mNets Communication Paradigm

To demonstrate the design philosophy of asymmetric function placement further, in this section we introduce our third example: a new multi-frequency communication protocol, called mNets. This communication design leverages multiple frequencies to reduce congestion and improve network throughput in dense wireless sensor networks.

Figure 14 shows the main idea of mNets: We separate a dense network into several connected sparse sub-networks,

which use different frequencies to support parallel data transmission through multiple sub-networks. We decouple multi-frequency communication into two sub-functions, *frequency assignment* and *data forwarding*. Using asymmetric function placement, we implement a set of sophisticated algorithms to do near-optimal frequency assignment outside the networks, and a lightweight data forwarding mechanism at sensor nodes.

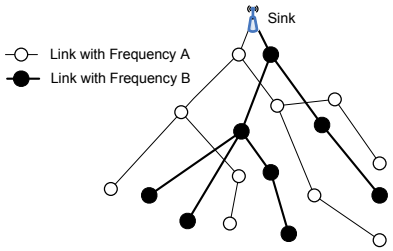

Fig. 14. The Conceptual Design of mNets

- **Frequency assignment outside:** Frequency assignment function is to partition the original network into sub-networks and assign different channels to sub-networks. Our design objectives for mNets are: (1) Every sub-network is k-connected ($k \geq 1$ and connects to at lease one base station. (2) Sub-networks should overlap in the field and can be easily reached by all sensor nodes. (3) Every sub-network should be sparse enough to reduce congestion and also be dense enough to allow robust connectivity. Distributed algorithms within sensor networks can hardly make a global partition and frequency assignment to meet these requirements without requiring excessive communication, especially for the last requirement. Based on the asymmetric function placement principle, we implement this function outside the network. In the system initialization, we gather the connectivity information of nodes, and build a graph model outside the networks and design centralized and sophisticated graph algorithms to solve corresponding graph theory problems. Specifically, we modify one classical MAX-CUT algorithm to produce a network partition such that each sub-network is k-connected (condition 1), with minimizing the total numbers of links within sub-networks (condition 3). We also modify legacy dominating set algorithms to compute sub-networks which are connected (condition 1), are k-dominating sets of the graph (condition 2) and satisfy degree constraint (condition 3). Using the strong computation ability outside the sensor networks, these algorithms compute better network partition and frequency assignment. The computed results are sent back to the sensor network for the purpose of data forwarding.
- **Data forwarding:** Since data has to be sent over the network, we deem data forwarding as an essential function that should be placed at nodes. Data forwarding operation of mNets is very simple and generic, in which a node only needs to use the pre-assigned frequency to forward packets to destinations. Since the sophisticated intelligence of the frequency selection is located outside of sensor nodes, the complexity is significantly reduced, hence leading to less in-network debugging issues.

1) Evaluation: We have implemented mNets in GloMoSim [37] and TinyOS/Mote testbed, and conducted extensive experiments to evaluate its performance. Detailed description of mNets performance is out of the scope of this paper. Here we only focus on three important performance metrics: 1) throughput, 2) packet delivery ratio and 3) packet delivery latency. We adopt two typical sensor network scenarios: (i) multiple sensor nodes report their readings to one base station and (ii) the base station disseminates information to multiple sensor nodes.

The experimental results indicate that mNets outperforms original network protocols in the following aspects: First, as shown in Figure 15, by using advanced network partition and frequency assignment algorithms, mNets achieves 1.6 and 2 times higher average aggregate throughput than the network without mNets support. Second, by splitting the traffic into different sub-networks, mNets decreases the chance of collisions, leading to higher packet delivery ratios and low latencies as shown in Figure 16 and Figure 17, respectively.

2) Architecture Insights: mNets gains several nice properties from the asymmetric function placement principle due to the following reasons: (i) It uses the strong computation ability outside the networks to compute globally optimal frequency assignment solutions, which can provide better network performance with small communication overhead, compared with distributed algorithms. (ii) As uSense and Spotlight, the *generic* design allows the flexible exchange of different frequency assignment polices according to their requirements; It also allows the definition of specific frequency selection rules for corresponding traffic patterns. At the sensor side, a generic forwarding technique is used to accommodate different frequency assignment. (iii) It supports efficient parameter tuning and replacement of frequency assignment algorithms, which makes mNets adapt to various network scenarios with good scalability. (iv)Generic data forwarding protocols can be implemented at sensor nodes with low overhead and footprint.

IV. Analysis of the Asymmetric Function Placement

Unlike evaluating a specific algorithm, a design principle is difficult to assess and can only be appreciated over a long time and over a large number of users. In this paper, we present three new protocols which have very dissimilar features, representing a wide spectrum of system design requirements. This allows us to demonstrate the generality of asymmetric function placement to some degree. However obviously, with three concrete examples, our effort alone is by no means sufficient to evaluate the design principle of sensor networks.

Despite the difficulty in evaluation, we attempt to reveal the benefits of asymmetric function placement by comparing the code size, computation complexity and communication overhead between the asymmetric solutions with the legacy symmetric solutions. Since the protocols under comparison do not share identical assumptions and settings, the comparison can only serve as a qualitative indication of the benefits of asymmetric function placement. More specifically, 1) we

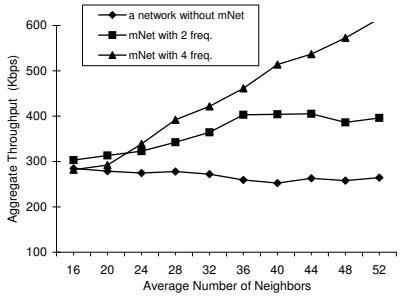

Fig. 15. Aggregated Throughput

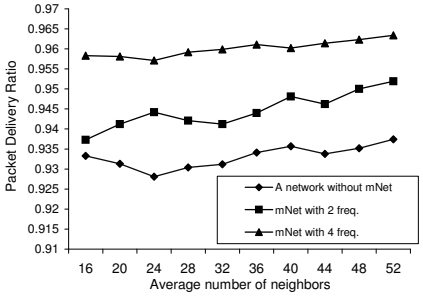

Fig. 16. Delivery Ratio

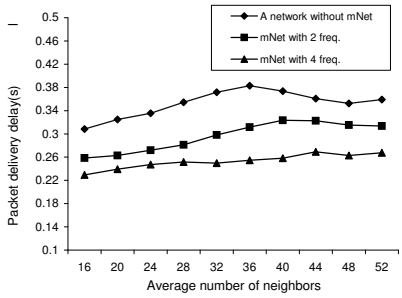

Fig. 17. Delivery Delay

	Overhead at a sensor			External Overhead				Overhead at a sensor		
	memory	Compu.	comm.	memory	Compu.	comm.		memory	Compu.	comm.
Spotlight	$O(1)$	$O(1)$	$O(H)$	$O(N)$	$O(N)$	$O(N)$	DV-Hop [23]	$O(A)$	$O(A^2)$	$O(NA)$
uSense	$O(1)$	$O(1)$	$O(H)$	$O(N)$	$O(N)$	$O(N)$	K-COVER * [1]	O(D)	O(D)	O(1)
mNets	$O(1)$	$O(1)$	$O(H)$	$O(ND)$	$O(fND^2)$	$O(ND)$	MMSN [40]	$O(D)$	$O(D)$	O(D)

TABLE I
THE BENEFIT OF ASYMMETRIC FUNCTION PLACEMENT.

In the table I, N denotes the number of nodes within a network. D denotes the average degree of connectivity (i.e. the node density). H is the diameter of the network in the number of hops. f is the number of frequencies available. A is the number of anchor nodes used for localization.
$*$We note Set K-cover is a well-known NP-Hard problem, therefore, we compare only the greedy versions in uSense and [1].

compare Spotlight with the DV-Hop localization [23], which localizes the sensor nodes using the hop counts between the nodes to several known beacons; 2) We compare uSense with the greedy version of SET K-COVER [1], which assigns a sensor to the cover set that it can contribute most. 3) mNets with MMSN [40], which allocates communication frequency by comparing the random numbers generated within two-hop neighboring nodes.

A. Memory Requirement

Constrained by the form factor and energy supply, the code and data memory available in sensor nodes are normally very small. For example, MicaZ has only 4K data memory available. Asymmetric function placement suggests the decoupling of sophisticated components from the sensor nodes to reduce the footprint of various kinds of functions. As shown in Table I, the memory requirements of Spotlight, uSense and mNets are constant at the sensor side, regardless of the size of the network and the node density. The protocols under comparison require a memory size proportional to either network density [1], [40] or the anchor density [23].

B. Computation Complexity

As shown in Table I, the principle of asymmetric function placement simplifies the computational complexity of the sensor network protocols. For examples, Spotlight only requires a simple time-stamp function after the detection, uSense only needs to turn on and off a node according to the scheduling bits, and mNets only performs simple transmission using preassigned frequency. With asymmetric function placement, the expensive computation is located outside of sensor nodes. In contrast, DV-Hop, K-Cover and MMSN demand the sensor nodes to perform all operations within the network. In addition to the computation overhead, the sophisticated in-network process introduces two detrimental side-effects: large code size and difficulties in debugging.

C. Communication Cost

The principle of asymmetric function placement necessitates communication between sensor and off-net entities. The cost of communication is proportional to the diameter of the network, which can be reduced by placing multiple second-tier nodes. In contrast, the communication cost of in-network protocols could vary significantly. For example, K-Cover only needs one message to notify its sensing neighbor about its decision, while DV-Hop requires multiple global flooding to estimate the distances between sensors and anchors. We argue that the communication costs of distributive algorithms are small only if communication is confined within neighborhood. However, when coordination beyond local interaction (e.g., in uSense and mNets case) is needed, in-network solutions would introduce surprisingly more communication cost than asymmetric designs.

V. DISCUSSION OF LIMITATIONS AND SOLUTIONS

On one hand, Spotlight, uSense and mNets demonstrate several key architecture advantages: (i) simple and generic processing within the nodes, (ii) a flexible and advanced design space outside sensor networks, and (iii) high-performance design as a whole. On the other hand, we do not expect that asymmetric function placement can be applied to every possible sensor network scenario. We believe there are several critical issues that discourage the use of asymmetric function placement. In this section, we discuss these issues and possible approaches to address or avoid them.

- **In-Network Adaptability:** Adaptability is a major concern of asymmetric function placement. High dynamics within some sensor networks would invalid an off-net design quickly. For this reason, we hypothesize that asymmetric function placement does not apply to highly-dynamic scenarios (e.g., mobility), because the frequent changes in network states could offset the benefit of

asymmetric design. On the other hand, for most stationary sensor networks, dynamic is moderate and can be addressed smartly. For example, in stationary networks, nodes normally do not move, which warrants a low-cost one-time localization such as Spotlight, instead of addition of extra hardware. As another example, if a node fails in the uSense design, it recovers by changing only the scheduling bits of the nodes in the neighborhood of failed node. There is no need for coordination among nodes and no need to refresh the schedule globally. As rules of thumb, system designers shall take the dynamics of target systems (i.e., how frequently an off-net computation could be invalided) into account, and design possible solutions (as we did in uSense) to alleviate the impact.

- **Off-Network Scalability:** If an asymmetric design is supported by a single off-net entity, it is subject to computation scalability issues and single-point of failure. In addition, if the diameter of the network is large, asymmetric design would suffer communication scalability issues. Legacy system experience indicates computation scalability issues can be addressed by designing a conceptually centralized, physical distributed computation platform. Similarly, communication scalability issues can be resolved by placing multiple sinks within the network to reduce in-network communication. We also note that different protocols normally share the same network states (e.g., the locations of nodes), therefore, the communication cost can be amortized by several coexisting asymmetric functions.
- **Principle Applicability:** The success of asymmetric function placement as a design principle is partially decided by how widely this design principle can be applied and how much benefit we can obtain from such decoupling. We have shown a $1 \sim 2$ orders of accuracy and range gain from the Spotlight example, however, three examples do not entitle us to make the judgment. We note that asymmetric function placement is a rather challenging task, because most functions are essential as a whole and we cannot simply re-implement them outside of the network. The core issue here is to decouple a function into several subfunctions and remove the non-essential parts, pushing the complexity outside of the network core as much as possible. In the future work, we plan to apply decoupling on a wide range of functions. For example, we suggest investigating whether 1) routing can be decoupled from forwarding, 2) coordination from transmission, 3) calibration from adaptation, and 4) topology control from transmission control. These decouplings can be achieved by removing indirect or costly underlying mechanisms away from the core service, as we have shown in Spotlight, uSense and mNets cases.

VI. Related Sensor Network Architecture

Recently, sensor network research is evolving very quickly. We are now reaching a critical point to decide the issue of architecture. Although several sensor network architectures [10], [25], [27], [20] have been proposed, it is still unclear which architecture will dominate eventually. In this section, we explain architecture commonality and differences between our design principle and other representative and well-known architectures.

- **TinyOS:** TinyOS/T2 [14], [19] is currently the most widely adopted architecture for sensor applications. In TinyOS, an application is defined by the modules and the connections among them. Two-level scheduling allows concurrency-intensive operations driven by on-demand events. Since TinyOS defines mostly the composition of functions within a single node, it is currently enhanced by the SNA architecture.
- **SNA:** The SNA architecture [25] aims at providing a unified abstraction (SP) at the link layer, the sensor network equivalence of internet IP layer. This unified abstraction greatly facilitates the rapid integration between the diversified link layer design and the network layer design. It effectively isolates the design concerns, making an architecture highly flexible and extensible. SP focuses on the design of a concrete layer SP, specifying the interfaces for this particular layer. In contrast, function placement is a general design principle for the placement of functions.
- **TENET:** The TENET architecture [10] advocates that multi-node data fusion functionality and complex application logic should be implemented only at the second tier (e.g., master nodes). In general, asymmetric function placement shares the same thoughts as TENET, i.e., move the complexity to a more powerful entity. Differently, TENET focuses on the elimination of application-specific complex logic at the node level, treating *functions as a whole*. In contrast, asymmetric function placement uses *function decoupling* to reduce the in-network complexity a step further. We believe the sensor nodes should be free from not only application-specific complex logic, but also non-essential application-independent functions.
- **COMPASS:** COMPASS [27] is a high-level architecture that supports collaborative, multi-scale data processing. From the information processing perspective, it provides a new multiscale wavelet transform for feature extraction. From the networking perspective, it provides a multi-overlay network structure. These two design perspectives align the communication hierarchy with the information flow. Our design principle can be applied in the COMPASS architecture to reduce the complexity of in-network processing and the COMPASS architecture is an excellent platform for us to evaluate the applicability of asymmetric function placement.
- **WaveScope:** WaveScope [20], [9] is a specific architecture for continuous query processing and signal processing over high-data rate sensor data streams. WaveScope argues that there is no single architecture currently working for all application domains, it is often necessary to place architecture design in a targeted context to achieve better performance. Different from WaveScope, our principle is generic guideline for various type of sensor systems, including WaveScope-like sensor systems.

TinyOS/T2 [14], [19], SNA [25], TENET [10], COMPASS [27] and WaveScope [20] have demonstrated their architectural superiority from different perspectives. There are also a set of special purpose architecture for storage [7], power management [17], inference [24] and filtering [22]. Until now it is too early to judge which one is better and which path sensor network researcher should follow. But we are confident that our design principle has identified a niche position to assist the effort of final convergence.

VII. Conclusion

In this paper, we advocate *Asymmetric Function Placement* as the guiding principle to architect the sensor network systems, which removes one-time, indirect, specific or costly underlying mechanisms away from the core service. We apply this design principle on three key sensor network technologies, which have very dissimilar features and hence represent a wide spectrum of system design requirements. Through the system design, physical implementation and test-bed evaluation/simulation of Spotlight, uSense and mNets, we demonstrate that asymmetric function placement is a powerful guiding principle for building high-performance and lightweight sensor network systems if it is used in appropriate settings. We hope this work can assist the research community to identify the key design principle that is conceptually useful for building concrete wireless sensor systems.

ACKNOWLEDGEMENTS

This research is supported by NSF grants CNS-0626614 and CNS-0626609.

References

[1] Z. Abrams, A. Goel, and S. Plotkin. Set K-Cover Algorithms for Energy Efficient Monitoring in Wireless Sensor Networks. In *IEEE IPSN*, 2004.

[2] N. Bulusu, J. Heidemann, and D. Estrin. GPS-less Low Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications Magazine, Special Issue on Networking the Physical World*, 7(4):28–34, August 2000.

[3] Q. Cao, T. Abdelzaher, T. He, and J. A. Stankovic. Towards Optimal Sleep Scheduling in Sensor Networks for Rare Event Detection . In *IPSN'05*, 2005.

[4] M. Cardei, M. T. Thai, Y. Li, and W. Wu. Energy-Efficient Target Coverage in Wireless Sensor Networks. In *IEEE INFOCOM*, 2005.

[5] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology. In *Proc. of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[6] W. Choi and S. K. Das. A novel framework for energy - conserving data gathering in wireless sensor networks. In *IEEE INFOCOM*, 2005.

[7] P. Desnoyers, D. Ganesan, and P. Shenoy. TSAR: A Two Tier Storage Architecture Using Interval Skip Graphs. In *IPSN'05*, 2005.

[8] D. Gay, P. Levis, R. v. Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *PLDI'03*, 2003.

[9] L. Girod, K. Jamieson, Y. Mei, R. Newton, S. Rost, A.Thiagarajan, H. Balakrishnan, and S. Madden. WaveScope: A Signal-Oriented Data Stream Management System (Poster). In *SenSys 2006*, November 2006.

[10] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The Tenet Architecture for Tiered Sensor Networks. In *SenSys 2006*, November 2006.

[11] Y. Gu, J. Hwang, T. He, and D. H. Du. uSense: A Unified Asymmetric Sensing Coverage Architecture for Wireless Sensor Networks. In *ICDCS'07*, June 2007.

[12] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-Free Localization Schemes in Large-Scale Sensor Networks. In *MOBICOM'03*, September 2003.

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

[14] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System Architecture Directions for Networked Sensors. In *Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 93–104, 2000.

[15] J. Hui and D. Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In *SenSys 2004*, November 2004.

[16] J. Hwang, T. He, and Y. Kim. Exploring in-situ sensing irregularity in wireless sensor networks. In *SenSys '07*, 2007.

[17] K. Klues, G. Xing, and C. Lu. Demo Abstract: A Unified Architecture for Flexible Radio Power Management in Wireless Sensor Networks. In *SenSys'06*, 2006.

[18] J. Koshy and R. Pandey. VM*: A Scalable Runtime Environment for Sensor Networks. In *SenSys'04*, November 2004.

[19] P. Levis, D. Gay, V. Handziski, J. Hauer, M. Turon Be. Greenstein, J. Huio, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonnao, L. Nachman, G. Tolleo, D. Cullero, and A. Wolisz. T2: A Second Generation OS For Embedded Sensor Networks. In *Technical Report TKN-05-007*, 2005.

[20] MIT. *An Adaptive Wireless Sensor Network System for High Data-Rate Applications*, 2005. Available at http://wavescope.csail.mit.edu/.

[21] R. Nagpal and D. Coore. An Algorithm for Group Formation in an Amorphous Computer. In *PDCS'98*, 1998.

[22] S. Nath, Y. Ke, P. B. Gibbons, and S. Seshan B. Karp. A Distributed Filtering Architecture for Multimedia Sensors. In *BaseNets'04*, 2004.

[23] D. Niculescu and B. Nath. Ad-Hoc Positioning System. In *IEEE GLOBECOM*, 2001.

[24] M. Paskin, C. Guestrin, and J. McFadden. A Robust Architecture for Distributed Inference in Sensor Networks. In *IPSN'05*, 2005.

[25] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A Unifying Link Abstraction for Wireless Sensor Networks. In *SenSys 2005*, November 2005.

[26] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *MOBICOM'00*, pages 32–43, August 2000.

[27] Rice University. *COllaborative Multiscale Processing and Architecture for SensorNetworkS*, 2005. Available at http://compass.cs.rice.edu/.

[28] J. H. Saltzer, David. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. In *ACM Transactions on Computer Systems*, pages 277–288, November 1984.

[29] S. Shakkottai, R. Srikant, and N. Shroff. Unreliable sensor grids: coverage, connectivity and diameter. In *IEEE INFOCOM*, 2003.

[30] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton. Sensor Network-Based Countersniper System. In *SenSys'04*, November 2004.

[31] R. Stoleru, T. He, J. A. Stankovic, and D. Luebke. High-Accuarcy, Low-Cost Localization System for Wireless Sensor Networks. In *SenSys'05*, November 2005.

[32] R. Szewczyk, A. Mainwaring, J. Anderson, and D. Culler. An Analysis of a Large Scale Habit Monitoring Application. In *SenSys'04*, 2004.

[33] D. Tian and N.D. Georganas. A Node Scheduling Scheme for Energy Conservation in Large Wireless Sensor Networks. *Wireless Communications and Mobile Computing Journal*, May 2003.

[34] G. Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, S. Burgess, D. Gay, P. Buonadonna, W. Hong, T. Dawson, and David Culler. A Macroscope in the Redwoods. In *SenSys'05*, November 2005.

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

[36] F. Ye, G. Zhong, S. Lu, and L. Zhang. PEAS: A Robust Energy Conserving Protocol for Long-lived Sensor Networks. In *ICDCS'03*, May 2003.

[37] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks. In *the 12th Workshop on Parallel and Distributed Simulations*, 1998.

[38] F. Zhao, J. Shin, and J. Reich. Information-Driven Dynamic Sensor Collaboration for Tracking Applications. *IEEE Signal Processing Magazine*, March 2002.

[39] Z. Zhong and T. He. Msp: multi-sequence positioning of wireless sensor nodes. In *SenSys '07*, November 2007.

[40] G. Zhou, C. Huang, T. Yan, T. He, and J. A. Stankovic. MMSN: Multi-Frequency Media Access Control for Wireless Sensor Networks. In *IEEE Infocom*, April 2006.