# Mobile-agent-based Collaborative Signal and Information Processing in Sensor Networks

Hairong Qi, *Member, IEEE,* Yingyue Xu, and Xiaoling Wang, *Student Member, IEEE*
Electrical and Computer Engineering Department, University of Tennessee

***Abstract*— In this paper, we develop an energy-efficient, fault-tolerant approach for collaborative signal and information processing (CSIP) among multiple sensor nodes using a mobile-agent-based computing model. In this model, instead of each sensor node sending local information to a processing center for integration, as is typical in a client/server-based computing, the integration code is moved to the sensor nodes through mobile agents. The energy efficiency objective and the fault tolerance objective always conflict with each other and present unique challenge to the design of CSIP algorithms. In general, energy-efficient approaches try to limit the redundancy in the algorithm so that minimum amount of energy is required for fulfilling a certain task. On the other hand, redundancy is needed for providing fault tolerance since sensors might be faulty, malfunctioning, or even malicious. A balance has to be struck between these two objectives. We discuss the potential of mobile-agent-based collaborative processing in providing progressive accuracy while maintaining certain degree of fault tolerance. We evaluate its performance compared to the client/server-based collaboration from perspectives of energy consumption and execution time through both simulation and analytical study. Finally, we take collaborative target classification as an application example to show the effectiveness of the proposed approach.**

***Index Terms*— energy efficiency, fault tolerance, collaborative signal and information processing, mobile-agent-based computing**

## I. Introduction

The infusion and maturation of the Micro Electro Mechanical System (MEMS), computation, and wireless communication technologies has advanced the development of sensor networks. A large amount of low cost, intelligent microsensors can be rapidly deployed in an environment of interest. These sensors can individually sense the environment. They can also collaborate with each other and achieve complex information gathering and dissemination tasks.

Although potentially powerful, sensor networks have presented unique challenges to many aspects of network design and information processing, which can be summarized as: (1) The scalability challenge. The proliferation of low cost sensors enables large amounts of sensor deployment. As more sensors are put into the field, more data is captured which can enhance decision making. The risk is, however, large data transfer and information overloading. (2) The reliability challenge. Sensors communicate through low-bandwidth and unreliable wireless links compared to wired communication. An individual sensor may suffer intermittent connectivity due to high bit error rate (BER) of the wireless link, and it can be further deteriorated by environmental hazard. The challenge is to provide reliable information based on potentially unreliable wireless communication networks and unreliable sensor nodes. (3) The dynamics challenge. Since sensors are usually rapidly deployed in large amount, it is very difficult, if not impossible, to maintain a pre-designed network structure. Sensors may be static or mobile. They may come and go because of new sensor deployment and node failure. All these dynamic features indicate that sensor networks tend to be infrastructureless and require the underlying network services and applications to be adaptive. (4) The energy challenge. In sensor networks, the major constraint to individual sensor performance is energy [1], [2]. The lifetime of the sensor node is mainly determined by the power supply since battery replacement is not an option. The longer the sensors last, the more stable the network, and the less dynamic the network. To meet the energy requirement, redundant activities should be reduced if not eliminated [3], [4], [5], [6]. However, redundancy is desired in providing robust and fault-tolerant information, especially when individual sensor nodes are potentially unreliable.

The focus of this paper is collaborative signal and information processing (CSIP). In order to respond to the challenges posed by sensor networks, the underlying CSIP techniques need to be scalable, adaptive, energy-aware, and capable of delivering reliable information in real time. Furthermore, CSIP algorithms should provide progressive accuracy as the collaboration process could be terminated upon achieving desired accuracy to conserve energy. In order to realize these goals several approaches have been developed [7], such as the information-driven dynamic sensor collaboration technique proposed by Zhao et. al [8] and the relation-based approach by Guibas [9]. In this paper, we develop a mobile-agent-based computing model to carry out the collaborative processing. Instead of focusing on either fault tolerance or energy efficiency, the mobile-agent-based model aims to resolve the conflicts between these two objectives.

The outline of this paper is as follows: Sec. II explains the difference between the mobile-agent-based computing model and the client/server-based computing model in collaborative processing. Sec. III uses both analytical analysis and simulation tools to evaluate the performance of these two models from energy consumption and execution time points of view. Sec. IV presents a multi-sensor integration algorithm that can be realized by mobile agents. It also discusses the degree of fault tolerance that the mobile-agent-based collaborative processing can provide. Experimental results on collaborative classification of three targets in a network of 70 sensors are

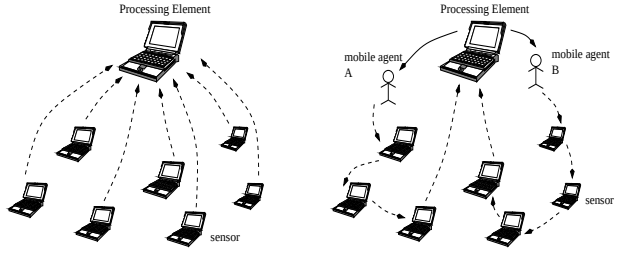
This research was supported in part by DARPA under grant N66001-001-8946.

presented in Sec. V.

## II. Mobile-agent-based Computing Model

In collaborative processing the most commonly used computing model is client/server-based where individual sensors (the clients) send raw data or pre-processed data to a processing center (the server) and data integration is carried out at the center [10], [11]. There are a few drawbacks with this model that might prevent it from being used in sensor networks. First of all, the client/server-based computing generally requires many round trips over the network in order to complete one transaction. Each transaction consumes network bandwidth and communication energy. The network connection needs to be alive and healthy the entire time of the transaction, otherwise, the transaction has to restart if it can at all. Secondly, there have to be some kind of super-nodes in the sensor network served as the processing centers which have bigger storage, higher computing capabilities, and more energy. But in some automatic and homogeneous sensor networks, this is not always the case. Given the unreliability and low bandwidth of the wireless link used in sensor networks, the client/server-based computing is not appropriate to carry out the collaborative processing between multiple sensor nodes.

In this section, we present a new computing model based on the mobile agents. In this model, instead of each sensor node sending raw data or pre-processed data to the processing center, the processing code is moved to the data locations through mobile agents. Figure 1 illustrates the difference between the client/server-based computing and mobile-agent-based computing.




(a) Client/server-based computing. (b) Mobile-agent-based computing.

Fig. 1. Different computing models.

The mobile-agent-based computing has the following features that respond to the unique challenges posed by the sensor network: (1) *Scalability*. The performance of the network is not affected when the number of sensor nodes is increased. Agent architectures that support adaptive network load balancing could do much of a redesign automatically [12]; (2) *Reliability*. Mobile agents can be sent when the network connection is alive and return results when the connection is re-established. Therefore, the performance of the mobile-agent-based computing is not affected much by the reliability of the network; (3) *Extensibility and task-adaptivity*. Mobile agents can be programmed to carry different task-specific integration processes which extends the functionality of the network; (4) *Energy-awareness*. The itinerary of the mobile agent is dynamically determined based on both the information gain and energy constraints. It is tightly integrated into the application and is energy-efficient; and (5) *Progressive accuracy*. A mobile agent always carries a partially integrated result generated by nodes it already visited. As the mobile agent migrates from node to node, the accuracy of the integrated result is constantly improved assuming the agent follows the path determined based on the information gain. Therefore, the agent can return results and terminate its itinerary any time the integration accuracy satisfies the requirement. This feature, on the other hand, also saves both network bandwidth and computation time since unnecessary node visits and agent migrations are avoided.

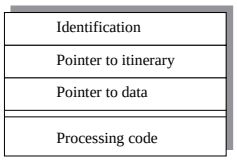


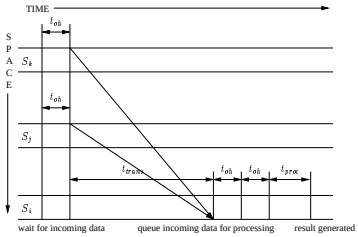

Fig. 2. A mobile agent.

Generally speaking, mobile agent is a special kind of "software". Once dispatched, it can migrate from node to node performing data processing autonomously. Figure 2 shows the structure of a mobile agent that has four attributes: identification, itinerary, data, and processing code. Identification uniquely identifies each mobile agent. Data is the agent's data buffer which carries a partially integrated result. Itinerary is the route of migration. It can be fixed or dynamically determined based on the current network status and the information gain. Processing code carries out the integration whenever the mobile agent arrives at a local sensor node.

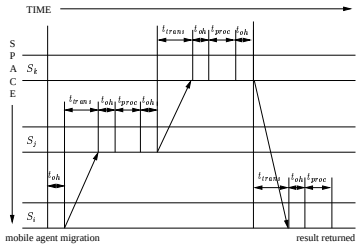
In order to better illustrate how these two computing models perform integration, Figure 3 presents a temporal and spatial comparison of the life cycle of their migration units. In the client/server-based model, the migration unit is "data", while in the mobile-agent-based model, the migration unit is "mobile agent". In both examples we assume there are three sensor nodes ($S_i$, $S_j$, and $S_k$) and we assign $S_i$ as the processing center in the case of client/server-based computing and the cluster head that dispatches mobile agents in the case of mobile-agent-based computing.

The life cycle of both units is composed of three components, $t_{trans}$, $t_{oh}$, and $t_{proc}$. We use $t_{trans}$ to represent the time spent in transferring the migration unit from one node to the other. $t_{oh}$ represents the overhead time. In the case of client/server-based model, it is the time spent on file access; and in the case of mobile-agent-based model, it is the time used to create, dispatch, and receive the mobile agent. Finally, $t_{proc}$ represents the processing time. Note that the length of $t_{trans}$, $t_{oh}$, and $t_{proc}$ does not reflect the actual time spent. They are used here as symbols to show the sequence of events in the life cycle.

In the client/server-based model (Fig. 3 (a)), the clients ($S_j$ and $S_k$) first read the data files into memory using $t_{oh}$, then transfer them to the server ($S_i$) using $t_{trans}$. Here, we assume $S_j$ and $S_k$ are identical nodes and can start data transfer at the same time. Therefore, these data files might arrive at the server simultaneously but can only be processed serially. It takes $S_i$ $t_{oh}$ amount of time to receive each data file and open it. After receiving all the incoming data files, the server can




(a) Data migration.




(b) Mobile agent migration.

Fig. 3. Life cycle of data (client/server-based) and mobile agent (mobile-agent-based) migration in time and space.

start processing, which would take $t_{proc}$ amount of time.

In the mobile-agent-based model (Fig. 3 (b)), a mobile agent is created at node $S_i$ using $t_{oh}$. It then migrates to $S_j$ using $t_{trans}$. After $S_j$ receives the mobile agent, it spends $t_{oh}$ to read data from the mobile agent and $t_{proc}$ to do data integration. Then the agent migrates to $S_k$. After the same procedure, it finishes migration and returns to $S_i$.

From Fig. 3, we observe that the mobile-agent-based computing might not always perform better than the client/server-based computing since mobile agents also introduce overhead, which mainly comes from the agent creation and dispatch time. However, for the client/server-based computing, there will be increased queuing delay as the number of clients increases. As a result, it may cause longer processing delay and more potential drops at the server side. Unfortunately, in sensor networks, the number of nodes could be hundreds or even thousands. In order to evaluate the performance of these two computing models objectively, we conducted a series of analytical study and simulation as discussed in the next section.

## III. Performance Evaluation

We choose to use two metrics, the execution time and the energy consumption, to evaluate the performance of the client/server-based and mobile-agent-based computing models in collaborative processing.

### A. The Execution Time

The *execution time* is the time spent to finish a processing task. In the mobile-agent-based model, it starts from the time a mobile agent is created to the time the mobile agent returns with results. In the client/server-based model, it is from the time the clients send out data to the time the data processing is finished and results are generated at the server. The execution time consists of three components, $t_{trans}$, $t_{oh}$, and $t_{proc}$ as illustrated in Fig. 3. A few factors that can affect the execution time include the network transfer rate $v_n$, the data processing rate $v_d$, the data file size $s_f$ (the size of raw data each node collects), the mobile agent size ($s_a$), the overhead of file access $o_f$ (the time used to read and write a data file), the overhead of mobile agent $o_a$, the number of sensor nodes $p$, the number of agents $m$, and the number of sensor nodes $n$ that each agent migrates (the processing center is not included). Notice that $p = mn$.

Thus, for the client/server-based computing, the data transfer time is $t_{trans} = mns_f/v_n$; the overhead time is $t_{oh} = 2mno_f$ (assuming the time used to read and write the data file is the same); and the data processing time is $t_{proc} = mns_f/v_d$. Therefore, the total execution time using the client/server-based model is

$$t_{cs} = \frac{mns_f}{v_n} + 2mno_f + \frac{mns_f}{v_d} \quad (1)$$

For the mobile-agent-based computing, the time used to transfer the agents is $t_{trans} = (m+n)s_a/v_n$ since it takes $ns_a/v_n$ for the $m$ mobile agents to migrate among the $n$ sensor nodes simultaneously and it takes $ms_a/v_n$ additional time for the cluster head to receive the agents in sequence after they finish the migration; the agent overhead time is $t_{oh} = 2(m+n)o_a$ as it takes $2mo_a$ for the cluster head to dispatch and receive $m$ mobile agents and $2no_a$ for all the local nodes to send and receive each mobile agent; and the time used to execute the processing code locally is $t_{proc} = (m+n)s_a/v_d$. Therefore, the total execution time using the mobile-agent-based model is:

$$t_{ma} = \frac{(m+n)s_a}{v_n} + 2(m+n)o_a + \frac{(m+n)s_a}{v_d} \quad (2)$$

In the analytical model described above, the component that is most difficult to measure is the data transfer time, where retransmission and error control are not considered. Unfortunately, these factors occur quite often in sensor networks because of the use of wireless link. Therefore, we develop simulation models for more accurate estimation of the data transfer time $t_{trans}$.

### B. The Energy Consumption

Sensor nodes are normally composed of four basic units [1]: a sensing unit, a processing unit, a communication unit, and a power unit. Among these units, communication and sensing consume most of the energy. However, since the energy consumed in sensing is the same for both models we choose to neglect this factor.

Similar to the formulation of the execution time, the energy consumption for the two computing models depends on three components, energy consumed in data transfer ($E_{tran}$), overhead processing ($E_{oh}$), and data processing ($E_{proc}$). Since no matter where the data processing is taken place, be it at the local sensor node or the processing center, the energy consumed for the entire sensor network is the same for both computing models, we choose to neglect $E_{proc}$ as well.

According to [13], the energy consumed in data transfer can be modelled using a linear equation, $E_{tran} = c \times size + d$ where $d$ is a fixed component associated with device state changes and channel acquisition overhead, $size$ is the size of data being transferred, and $c$ is a coefficient indicating the amount of energy consumed by transferring 1 byte of data. The values for $c$ and $d$ are different between data transmission and data receiving. Therefore we separate $E_{tran}$ into two parts, $E_{tx} = c_{tx} \times size + d_{tx}$ for transmission and $E_{rx} = c_{rx} \times size + d_{rx}$ for receiving.

We use a similar linear equation to model the energy consumption in overhead processing, $E_{oh} = c_{proc} \times size$ where $c_{proc}$ is the coefficient indicating the amount of energy consumed in processing 1 byte of data. Since we only have knowledge of time spent for overhead processing and the amount of data that can be processed in 1 second, the so-called equivalent data size ($s_e$), we can derive the size of data that takes $o_a$ or $o_f$ amount of overhead time to process, that is, $s_e o_a$ for mobile-agent-based model and $s_e o_f$ for client/server-based model. We assume the data processing routine read/writes data files using a 4KB records and one byte of data takes one operation to process. Since $s_e$ differs when the number of clients changes we choose an average value $s_e = 470$KB for simplicity purpose [14].

Similar to Eq. 1, the energy consumption model we use for the client/server-based computing is

$$E_{cs} = (mn)[(c_{tx}s_f + d_{tx}) + (c_{rx}s_f + d_{rx}) + 2(c_{proc}s_e o_f)] \quad (3)$$

Correspondingly, the energy consumption model for the mobile-agent-based computing is

$$E_{ma} = m(n+1)[(c_{tx}s_a + d_{tx}) + (c_{rx}s_a + d_{rx}) + 2(c_{proc}s_e o_a)] \quad (4)$$

where each of the $m$ mobile agents migrates through $n$ sensor nodes and then returns which accounts for $n+1$ migration segments.

We adopt the values for $c_{tx} = 1.9$, $d_{tx} = 454$, and $d_{rx} = 356$ from [13] which are measured based on a Lucent IEEE 802.11 WaveLAN PC Card using 2.4GHz direct sequence spread spectrum. We derive the rest of the constants based on the energy consumption ratio between transmission, receiving, and computation, which is roughly $4:3:1.5$ [15]. That is, $c_{rx} = 1.9 \times 3/4 = 1.425$ and $c_{proc} = 1.9 \times 1.5/4 = 0.7$.

### C. Simulation Results

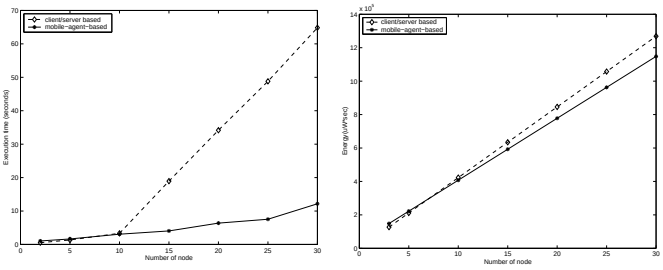
We use a simulated sensor network written by GloMoSim [16] to obtain the amount of time spent for data transfer. We then use the analytical models developed in Eqs. 1-4 to calculate the execution time and energy consumption.

In our simulation, the basic network consists of wireless nodes randomly distributed within a $100 \times 100$m$^2$ area. The mobility model we use is the random waypoint model where nodes randomly choose a destination and move at a speed of 10m/s. Once it arrives at the destination, it pauses for 5 seconds and then continues moving. The radio frequency we used is 2.5GHz. Parameters that determine the data processing time and the overhead in different computing models are listed in the following table with their typical value selected for the basic network.

| $n$ | $m$ | $s_f$ | $s_a$ | $o_f$ | $o_a$ | $v_n$ | $v_d$ |
|---|---|---|---|---|---|---|---|
| 10 | 1 | 10KB | 1KB | 0.0125s | 0.05s | 2Mbps | 100Mbps |

We design four experiments to evaluate the effect of four parameters ($p$, $m$, $s_a$, and $o_f/o_a$) on the execution time and energy consumption using the two computing models. In each experiment, we only change the value of one parameter but keep others fixed.

*1) Effect of the number of nodes ($p$):* In this experiment, we change the number of nodes $p$ from 2 to 30 and use 1 mobile agent. The result is shown in Fig. 4. We observe that both the execution time and the energy consumption using either computing model grow as the number of nodes increases. But the execution time of the client/server model grows much faster than the mobile-agent-based model. This is because as the number of nodes increases, the server has to deal with more connections requested by the clients at the same time which elongates the execution time. On the other hand, the mobile agent model is less influenced by the number of nodes because there are much less connections at one time for the mobile agent model. The figure also shows that, for $p < 8$, the client/server model performs a little better than the mobile agent model from both the execution time and energy consumption perspectives. This happens when the mobile agent model needs more connections than the client/server model in order to send and receive mobile agents. It also happens when the overhead of the mobile agent surpasses the overhead of the client/server model.

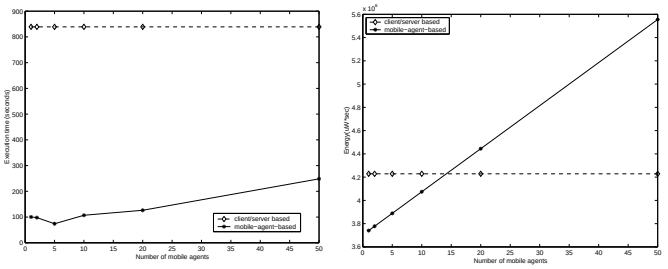



(a) Execution time. (b) Energy consumption.

Fig. 4. Effect of the number of nodes ($p$).

*2) Effect of the number of mobile agents ($m$):* In this experiment, we fix the node number at 100 but change the number of mobile agents from 1 to 50. Without loss of generality, we assume each agent migrates the same number of nodes. We expect a constant profile from the client/server model since it is irrelevant to the number of mobile agents. We can see from Fig. 5 that the execution time of the mobile agent model is always less than that of the client/server model because the node number is large. Interestingly, the execution time of the mobile agent model decreases as the number of mobile agents increases and reaches the lowest point when there are 5 mobile agents. Then the execution time begins to climb. This is because more mobile agents will reduce

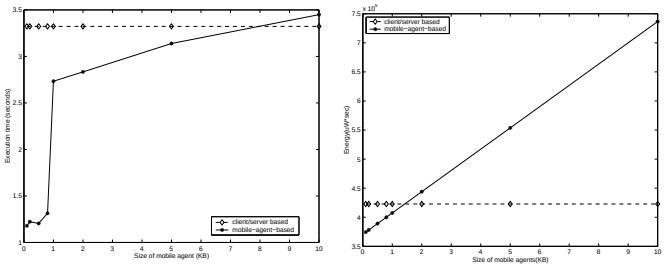
the number of nodes each agent migrates, thus reducing the execution time. But more mobile agents also cause more connections and more overheads. As the number of mobile agents increases, the energy consumption also increases in linear and the mobile agent model actually consumes more energy when $m > 15$.




(a) Execution time. (b) Energy consumption.

Fig. 5. Effect of the number of mobile agents ($m$).

*3) Effect of the mobile agent size ($s_a$):* In this experiment, we change the size of the mobile agent $s_a$, but fix the other parameters and let $s_f = 10$K. Figure 6 shows that both the execution time and energy consumption of the mobile agent model grow as $s_a$ increases. When $s_a > 1.5$K the mobile agent starts to consume more energy and when $s_a > 8$K the client/server model spends less execution time.

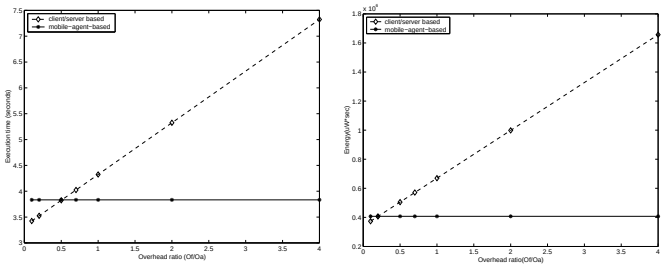



(a) Execution time. (b) Energy consumption.

Fig. 6. Effect of the mobile agent size ($s_a$).

*4) Effect of the overhead ratio ($o_f/o_a$):* In this experiment, we fix all other parameters and observe the effect of the overhead ratio $o_f/o_a$ changing from 0.1 to 4.0. We can see from Fig. 7 that when the ratio is greater than 0.5, the client/server-based computing starts to spend more execution time since the larger the $o_f$, the more the execution time. On the other hand, from the energy consumption point of view, when the overhead ratio is greater than 0.2, the client/server-based computing starts to consume more energy already.

The above simulation results show that the mobile-agent-based model does not always perform better than the client/server-based model and in different scenarios, the energy consumption is usually the more contingent resource. However, in the context of sensor networks with hundreds or even thousands of nodes, unreliable communication links, and reduced bandwidth, the mobile-agent-based computing




(a) Execution time. (b) Energy consumption.

Fig. 7. Effect of the overhead ratio ($o_f/o_a$).

provides solutions to energy-efficient collaborative processing with less execution time.

## IV. Using Mobile Agents for Multi-sensor Collaborative Processing

Since individual sensors can only sense a portion of the sensor field using certain sensing modalities, information provided by single sensor might very well be biased or inaccurate. If the sensor is malfunctioning or even tempered by malicious adversaries, the information can be contradictory. Therefore, collaborative processing among multiple sensor nodes is important to complement for each others missing information and tolerate faults.

Having shown the advantages of using the mobile-agent-based computing model in sensor network through simulation and analytical analysis, this section discusses how the mobile agent can be used to carry out collaborative processing among multiple sensor nodes. We also comment on the degree of fault tolerance that mobile-agent-based collaborative processing can provide.

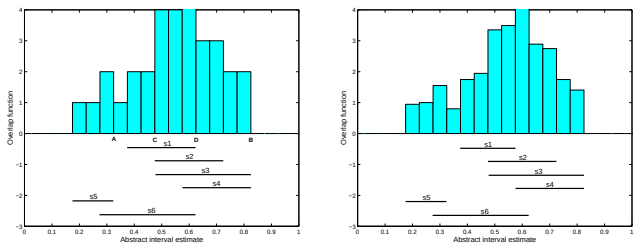
### A. Formulating the Overlap Function

Let us assume each sensor can generate a *confidence range* over some information derived from the data collected locally. For example, in the classification of a certain target, the sensor output might be expressed as "I am 40 to 70 percent sure that the target just went by me is a diesel truck." We define this range as the confidence interval estimate, e.g. from $a = 40\%$ to $b = 70\%$. The *confidence* itself can be modelled by different stochastic distributions, the simplest of which would be a uniform distribution, where equal weight has been put on each confidence within the confidence range. Other appropriate distributions could be a Gaussian (more weight on the central confidence) or a Rayleigh (more weight on the low confidence).

Let $\zeta_i$ represent the confidence distribution provided from node $i$ over a confidence range $[a_i, b_i]$. A uniform distribution can then be written as $\zeta_i(x) = 1$, if $x$ is in $[a_i, b_i]$, otherwise, $\zeta_i(x) = 0$. In order to integrate the confidence range distribution $\zeta_i$ from individual sensor nodes, we developed a distributed multi-resolution integration (MRI) algorithm [17]. The original MRI algorithm was proposed by Prasad, Iyengar, and Rao in 1994 [18], where a processing center collects

the outputs of the sensors and constructs an *overlap* function $\Omega(x) = \sum_{i=1}^{n} \zeta_i(x)$ ($n$ is the number of sensors). The overlap function can be resolved at successively finer scales of resolution to isolate region over which the correct sensors lie. In [18], an *abstract sensor* is defined as a sensor that reads a physical parameter and gives out an abstract interval estimate which is a bounded and connected subset of the set of real number. Based on this definition, a *correct sensor* is an abstract sensor whose interval estimate contains the actual value of the parameter being measured. Otherwise, it is a *faulty sensor*. A faulty sensor is *tamely faulty* if its interval estimate overlaps with a correct sensor, and is *widely faulty* if it does not overlap with any correct sensors.

Figure 8 illustrates the construction of overlap function for a set of 6 sensor nodes. Assume the correct parameter interval estimate is [0.5, 0.6] we then observe that the tamely faulty sensors ($s_4$) cluster around correct sensors ($s_1$, $s_2$, $s_3$, $s_6$) and create high and wide (maximal) peaks in the profile of $\Omega(x)$ while wildly faulty sensors ($s_5$) do not overlap with correct sensors, and therefore contribute to smaller and narrower peaks. The actual value of the information derived lies within regions over which the maximal peaks of $\Omega(x)$ occur with the largest spread. These regions are defined as the "*crests*".




(a) Uniform (b) Gaussian

Fig. 8. The overlap function for a set of 6 sensors using two distribution models (resolution is 0.05).

### B. Modified Overlap Function and Fault Tolerance

In mobile-agent-based multi-sensor collaborative processing, we make three modifications based on the original formulation of overlap function:

First, the abstract interval estimate is not a physical parameter reading but a confidence range derived from local signal processing.

Second, the overlap function is not generated at the processing center. Once an event occurs, local processing will be initiated automatically at each sensor node. In the meanwhile, mobile agents will be dispatched by the cluster head and migrate among these sensor nodes, integrating local processing results, which we call the partially integrated result [17]. A 1-D array serves as an appropriate data structure to store this result. The size of the array depends upon the resolution used. The lower the resolution, the smaller the array. For example, if the resolution is 0.05, then within a [0,1] interval, the buffer size will be 21 (1/0.05+1). The advantage of using mobile-agent-based model is that it provides *progressive accuracy*. When the accuracy requirement has been reached, the mobile agent can return to the cluster head immediately without finishing the scheduled route which saves network bandwidth as well as conserves power.

The third modification concerns the reliability of the integrated result. The original MRI algorithm picks a "crest" from the overlap function and resolves only the crest in the next finer resolution level. The process will continue until the finest resolution is reached. Take the overlap function in Fig. 8 (a) as an example, the crest picked at the current resolution (0.05) would be $[A, B]$. In [18], the authors show that the algorithm is robust and satisfies a Lipschitz condition [19] which ensures that minor changes in the input intervals cause only minor changes in the integrated result. Cho *et al.* [20] improves the original MRI algorithm to only return the interval with the overlap function ranges $[n - f, n]$ where $f$ is the number of faulty sensors. Once again, take Fig. 8 (a) as an example, where among the $n = 6$ sensor nodes, there are $f = 2$ faulty sensors. Thus the final integration result using Cho's approach will be $[C, D]$ where the overlap function ranges $[4, 6]$. This algorithm also satisfies Lipschitz condition and the biggest advantage of which is that it is able to reduce the width of the output interval in most cases and produce a narrower output interval when the number of sensors involved is large, which is the case for sensor networks in general. However, the problem with Cho's approach is that the value of $f$ is normally unknown. We make the following improvement in order to solve this problem.

We assume before the mobile agent is dispatched, the accuracy requirement and the resolution used are already known. The resolution requirement tells the mobile agent the size of buffer it needs to carry and the accuracy requirement tells it when to stop migration. In order for the mobile agent to automatically determine when it can return, a "*partially integrated overlap function*" needs to be generated at every stop of the mobile agent migration. We use $c = h \times w \times acc$ to pick the "*crest*" where $h$ is the height of the highest peak in the overlap function, $w$ is the width of the peak, and $acc$ is the confidence at the center of the peak. The peak with the largest $c$ is selected as the crest. For example, in Fig. 8 (a), the crest selected is the rectangle between $C$ and $D$. The height of the crest is $h = 4$, the width is $w = 3$, and the central is $acc = 0.55$, therefore, $c = 6.6$. An "*intermediate accuracy*" ($acc$) can then be derived which is used to determine if the mobile agent has achieved the required accuracy or not. We design a *protocol* for decision making which concerns both the degree of fault tolerance and the accuracy achieved.

According to the Byzantine generals problem [21], the maximum number of faults ($f$) that certain amount of sensor nodes ($n$) can tolerate is

$$f = \lfloor \frac{n-1}{3} \rfloor \tag{5}$$

We define the degree of fault tolerance *df* as the ratio between $f$ and $f + n$, that is, $df = f/(f + n)$. As the mobile agent migrates from node to node, the maximum number of faulty sensors that can be tolerated will change, so does the integration result $[n - f, n]$, but the degree of fault tolerance

maintains the same as that of the Byzantine generals problem. The protocol says that if and only if the following three criteria are satisfied, the mobile agent has to continue migration:

1. The overlap function has its highest peaks ranging from $[n-f, n]$, where $f$ is calculated from Eq. 5;

2. The intermediate accuracy $acc$ calculated from the partial integration result at each sensor node has to be equal to or larger than the median of the estimated interval. For example, if the estimated interval is $[0, 1]$, then the integrated accuracy cannot be less than 0.5.

3. Both 1 and 2 have to be satisfied in adjacent two migrations excluding the first sensor node in order to add stability to the decision.

We will provide an application example to show how this protocol is applied.

## V. Collaborative Target Classification

In this section, we describe the usage of the mobile-agent-based collaborative processing in multi-sensor target classification. Experimental results on classification of three targets in a network of 70 sensors are presented. In order to use the mobile-agent-based computing, the local result needs to be formulated in the form of a confidence range, where in the application example of target classification, it is the range of classification confidence over each possible class.

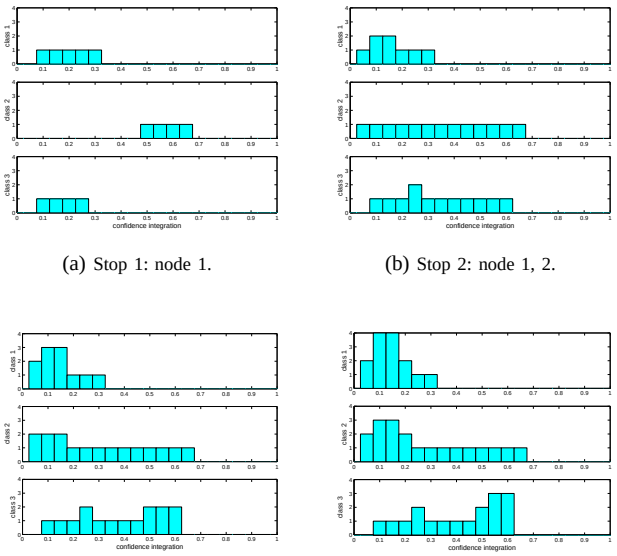
We start with an example where the mobile agent migrates within a cluster of 4 sensor nodes in order to integrate classification results on three possible targets as shown in the following table. The three intervals associated with each node indicate the classification confidence range of that node *thinking* the target might be of class 1, 2, or 3. For example, node 1 classifies the target as class 1 with a confidence ranges from 0.10 to 0.29. In this example, node 1 provides a tamely faulty result.

| nodes | class 1 | class 2 | class 3 |
|---|---|---|---|
| 1 | [0.10, 0.29] | [0.46, 0.65] | [0.10, 0.21] |
| 2 | [0.05, 0.14] | [0.05, 0.41] | [0.22, 0.58] |
| 3 | [0.05, 0.15] | [0.05, 0.15] | [0.49, 0.59] |
| 4 | [0.08, 0.16] | [0.08, 0.16] | [0.51, 0.60] |

Figure 9 illustrates how mobile agent generates the partially integrated confidence range by migrating from node 1 to node 4. We assume the resolution requirement is 0.05, then the size of the 1-D array is 21*3=63 to hold results for the three targets. The four sub-figures show the content of this array while the mobile agent migrates. Table I summarizes the mobile agent's decision making procedure at each stop of its migration. We observe that the integration result at stop 1 and 2 is "class 2" but changes to "class 3" at stop 3 and 4.

### A. Generate the Confidence Range

Among all the supervised classification algorithms, we choose to use the k-nearest-neighbor (kNN) technique since it can classify data in any kind of distributions. kNN is also easier to modify in order to generate a classification confidence range. Simply speaking, kNN uses a so-called "nearest neighbor rule", to classify the unknowns. It assigns




(a) Stop 1: node 1. (b) Stop 2: node 1, 2.

(c) Stop 3: node 1, 2, 3. (d) Stop 4: node 1,2,3,4.

Fig. 9. Multi-sensor integration result at each stop of migration.

| | stop 1 | | stop 2 | | stop 3 | | stop 4 | |
|---|---|---|---|---|---|---|---|---|
| | $c$ | $acc$ | $c$ | $acc$ | $c$ | $acc$ | $c$ | $acc$ |
| class 1 | 1 | 0.2 | 0.5 | 0.125 | 0.75 | 0.125 | 1 | 0.125 |
| class 2 | **2.3** | **0.575** | **4.55** | **0.35** | 0.6 | 0.1 | 0.75 | 0.125 |
| class 3 | 0.7 | 0.175 | 0.5 | 0.25 | **3.3** | **0.55** | **3.45** | **0.575** |

TABLE I

Mobile agent's decision making procedure at each stop.

the unknown to the same class as the class of the nearest neighbor in the training set. In specific, assume the number of possible classes is $C$, among the first $k$ training samples that are the closest to the unknown, if we use $k_i$ $(i = 1, \cdots, c)$ to indicate the number of training samples that belong to class $i$, then the *a-posteriori* probability of the unknown belonging to class $i$ is $k_i/k$ and the classification result is based on the largest probability.

In order to provide the local processing results in the format of $\zeta_i(x)$, we modify the classic kNN algorithm to generate a classification confidence range for each class at each sensor node. Let $k$ change in a closed-form integer range, such as $[5, 15]$. For each specific $k$, classification is performed using the classic kNN algorithm and a confidence (or probability) of the unknown belonging to every possible class is generated. The confidence range is then formed by choosing the minimal and the maximal values of these confidence.

### B. Experimental Results

The data set we use for testing collaborative target classification is provided by DARPA SensIT (Sensor Information Technology) program from a field demo (SITEX02) held at 29 Palms, California in November 2001. In the SITEX02 field demo, around 70 sensors are deployed in a field with a $150\times200\text{m}^2$ intersection connecting to a N-S road and an

E-W road. There are 3 possible target classes, AAV, Dragon Wagon (DW) and HMMWV. The time-series signals are collected from two sensing modalities on-board each sensor node, a geophone to collect seismic signals down-sampled at 512Hz and a microphone to collect acoustic signals down-sampled at 1024Hz. A three-way cross validation is conducted by dividing the whole data set into three partitions, with two of the partitions used as the training set and the other one the test set. We adopt two representations, classification accuracy and confusion matrix, to illustrate the performance of the collaborative classification as shown in Figs. 10 and 11 respectively.

In Fig. 10 the four pairs of bar charts compare the classification accuracy on target AAV, DW, HMMWV, and the overall average accuracy between single node classification and collaborative classification. In all cases, the multi-sensor classification performs better than the single sensor by a percentage of 3.6, 1, 27.8, and 13 respectively. The average classification accuracy can reach as high as 96.4%.

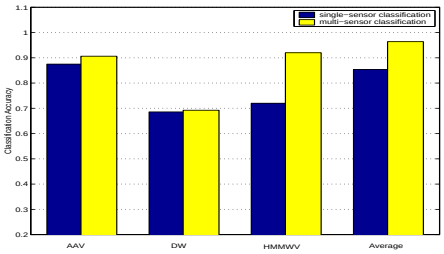



Fig. 10. Classification accuracy comparison between single sensor processing and multi-sensor collaborative processing.

Figure 11 compares the classification accuracy using a 3-D illustration of the *confusion matrix* listed in Tables II and III. Confusion matrix is a square matrix used to compare the classification results with the ground truth where the rows indicating the ground truth and the columns the actual classification result. Ideally, the confusion matrix should be diagonal that indicates a 100% classification accuracy. However, no classification algorithms can be perfect and misclassifications happen all the time. This results in a 3-D illustration of the confusion matrix where the cylinders along the diagonal are significantly higher than the rest cylinders that are contributed from misclassification. We observe from Fig. 11 that the misclassification cylinders of the multi-sensor classification are much shorter than those from the single-sensor classification.

| | AAV | DW | HMMWV | accuracy |
|---|---|---|---|---|
| AAV | 70 | 9 | 1 | 0.8750 |
| DW | 12 | 48 | 10 | 0.6857 |
| HMMWV | 11 | 10 | 54 | 0.7200 |

TABLE II

CONFUSION MATRIX FROM SINGLE NODE CLASSIFICATION.

## C. Classification Discussion

Although the focus of this paper is not about classification, indeed, we only use classification as an example, it is still worth mentioning that the considerably high average classification accuracy (96.4%) is not just because of the usage of multi-sensor collaborative processing. If the local processing result is not reasonably good, no matter how superior a collaborative processing algorithm is, the integrated result still couldn't be improved. The local classification results cited in this paper are based on three levels of processing: time-series data classification (apply the modified kNN algorithm on feature vectors extracted from both the frequency and time-frequency analysis of 1-sec time-series data), event fusion (majority voting on classification results generated from 1-sec time-series data that belong to the same event), and multi-modality fusion (fusion of classification results from different sensing modalities). Interested readers are referred to [22] for details on these techniques.

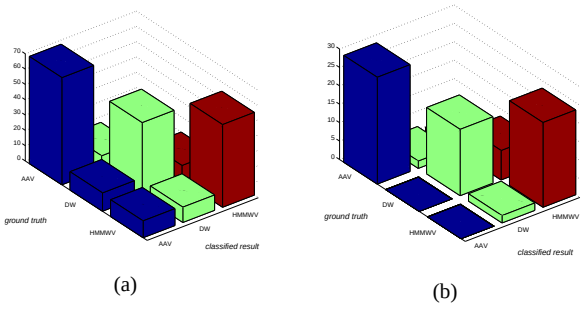



Fig. 11. 3-D illustration of the confusion matrix generated from (a) single-sensor classification and (b) multi-sensor classification.

| | AAV | DW | HMMWV | accuracy |
|---|---|---|---|---|
| AAV | 29 | 2 | 1 | 0.9062 |
| DW | 0 | 18 | 8 | 0.6923 |
| HMMWV | 0 | 2 | 23 | 0.9200 |

TABLE III

CONFUSION MATRIX FROM MULTI-SENSOR CLASSIFICATION.

Figure 12 summarizes the performance gain of conducting classification at different levels of processing. We observe that the performance of the event fusion and multi-modality fusion might vary for different target classes, but in all cases, the multi-sensor collaborative processing is *always* better than the local processing result.

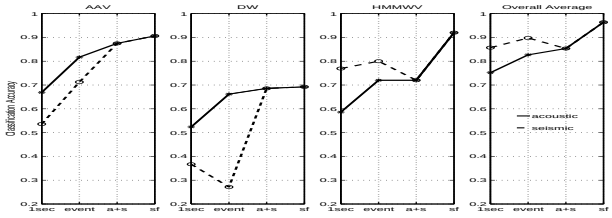



Fig. 12. Performance evaluation of different levels processing. From left to right: AAV, DW, HMMWV, and overall average. Solid line: acoustic signal; Dash line: seismic signal; "1sec": averaged accuracy using 1-second sub-events; "event": event fusion result; "a+s": multi-modality fusion result; and "sf": multi-sensor collaborative processing result using mobile agent.

## VI. Conclusion

In this paper, we presented a mobile-agent-based computing model for collaborative signal and information processing (CSIP) in sensor networks. We compared the performance of the mobile-agent-based model with the classic client/server-based model from the execution time and energy consumption perspectives through both simulation and analytical study. We conclude that in the context of sensor networks where the number of sensor nodes is very large, the communication bandwidth is considerably low, and the energy resource is contingent, the mobile-agent-based computing model is more suitable for conducting collaborative processing. We further discussed the degree of fault tolerance that the mobile-agent-based collaborative processing can achieve. We applied this computing model in collaborative target classification in a ground sensor network and the results clearly show the superior performance of the collaborative classification over single node classification.

## Acknowledgment

The authors would like to thank Professor Yu Hen Hu and his students at the University of Wisconsin-Madison for the tremendous amount of time they spent in generating the cross-validation data set based on the SITEX02 field demo. Without their data set, our research approach would have been difficult to validate. The authors are also grateful to the anonymous reviewers for the very instructive suggestions which led to the much improved quality of the paper.

## References

[1] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Processing Magazine*, pp. 40–50, March 2002.

[2] M. Bhardwaj, A. Chandrakasan, and T. Garnett, "Upper bounds on the lifetime of sensor networks," in *IEEE International Conference on Communications*, 2001, pp. 785–790.

[3] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next century challenges: scalable coordination in sensor networks," in *Intl. Conf. on Mobile Computing and Networking (MobiCom)*, Seattle, WA, August 1999, pp. 263–270.

[4] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient routing protocols for wireless microsensor networks," in *Proc. 33rd Hawaii Intl Conf. on System Sciences (HICSS00)*, January 2000, pp. 3005–3014.

[5] S. Meguerdichian, S. Slijepcevic, V. Karayan, and M. Potkonjak, "Localized algorithms in wireless ad-hoc networks: location discovery and sensor exposure," in *MOBIHOC*, 2001.

[6] A.Wang and A. Chandrakasan, "Energy-efficient DSPs for wireless sensor networks," *IEEE Signal Processing Magazine*, pp. 68–78, July 2002.

[7] S. Kumar, F. Zhao, and D. Shepherd, "Collaborative signal and information processing in microsensor networks," *IEEE Signal Processing Magazine*, pp. 13–14, March 2002.

[8] F. Zhao, J. Shin, and J. Reich, "Information-driven dynamic sensor collaboration for tracking applications," *IEEE Signal Processing Magazine*, March 2002.

[9] L. J. Guibas, "Sensing, tracking, and reasoning with relations," *IEEE Signal Processing Magazine*, pp. 73–85, March 2002.

[10] R. Viswanathan and P. K. Varshney, "Distributed detection with multiple sensors: Part i - fundamentals," *Proc. IEEE*, vol. 85, no. 1, pp. 54–63, January 1997.

[11] R. S. Blum, S. A. Kassam, and H. V. Poor, "Distributed detection with multiple sensors: Part ii - advanced topics," *Proc. IEEE*, vol. 85, no. 1, pp. 64–79, January 1997.

[12] T. Sundsted, "An introduction to agents," *Java World*, June 1998.

[13] L. M. Feeney and M. Nilsson, "Investigating the energy consumption of a wireless network interface in an ad hoc networking environment," in *IEEE INFOCOM*, Anchorage, AK, 2001.

[14] IOzone, "Iozone filesystem benchmark," http://www.iozone.org/.

[15] D. Estrin, A. Sayeed, and M. Srivastava, "Mobicom 2002 tutorial - wireless sensor networks," http://nesl.ee.ucla.edu/tutorials/mobicom02/, 2002.

[16] X. Zeng, R. Bagrodia, and M. Gerla, "GloMoSim: a library for parallel siumulation of large-scale wireless networks," in *Proc. of the 12th workshop on Parallel and Distributed Simulations (PADS '98)*, May 1998, pp. 154–161.

[17] H. Qi, S. S. Iyengar, and K. Chakrabarty, "Multi-resolution data integration using mobile agents in distributed sensor networks," *IEEE Trans. Syst., Man, Cybern. C*, vol. 31, no. 3, pp. 383–391, August 2001.

[18] L. Prasad, S. S. Iyengar, and R. L. Rao, "Fault-tolerant sensor integration using multiresolution decomposition," *Physical Review E*, vol. 49, no. 4, pp. 3452–3461, April 1994.

[19] L. Lamport, "Synchronizing time servers," Digital System Research Center, Tech. Rep. Technical Report 18, 1987.

[20] E. Cho, S. S. Iyengar, K. Chakrabarty, and H. Qi, "A new fault tolerant sensor integration function satisfying local lipschitz condition," *Submitted to IEEE Trans. Aerosp. Electron. Syst.*, 2000.

[21] L. Lamport, R. Shostuk, and M. Pease, "The Byzantine generals problem," *ACM Transactions of Programming*, vol. 4, no. 3, pp. 382–401, July 1982.

[22] X. Wang, H. Qi, and S. S. Iyengar, "Collaborative multi-modality target classification in distributed sensor networks," in *Proceedings of the Fifth International Conference on Information Fusion*, vol. 2, Annapolis, MA, July 2002, pp. 285–290.

**Hairong Qi** received her Ph.D. degree in Computer Engineering from North Carolina State University in 1999, B.S. and M.S. degrees in Computer Science from Northern JiaoTong University, Beijing, in 1992 and 1995 respectively. She is now an Assistant Professor in the Department of Electrical and Computer Engineering at the University of Tennessee, Knoxville. Dr. Qi is a member of IEEE and Sigma Xi. Her current research interests are collaborative signal and information processing in sensor networks, biomedical imaging, and automatic target recognition.

**Yingyue Xu** received his B.S. and M.S. degrees in Electrical Engineering from Tianjin University, China in 1999 and 2001 respectively. Since 2001, he has been pursuing a Ph.D. degree in Electrical and Computer Engineering at the University of Tennessee, focusing on energy efficient computing paradigms design and simulation in sensor networks. He is a student member of the IEEE.

**Xiaoling Wang** received her B.S. and M.S. degrees in Electrical Engineering from Northeastern University, Shenyang, in 1997 and 2000 respectively. She is now a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Tennessee, Knoxville. Her current research interests are information processing in distributed sensor networks, distributed data fusion for target tracking and classification, pattern recognition.

## LIST OF FIGURES



## LIST OF TABLES