

# An Adaptive Network/Routing Algorithm For Energy Efficient Cooperative Signal Processing In Sensor Networks<sup>1</sup>

Jay L. Gao  
Jet Propulsion Laboratory  
4800 Oak Grove Drive, M/S: 161-260  
Pasadena, California 91109-8099  
818-354-9528  
[Jay.L.Gao@jpl.nasa.gov](mailto:Jay.L.Gao@jpl.nasa.gov)

*Abstract-* In this paper, we describe an adaptive network/routing algorithm that facilitates both coherent and non-coherent event-based cooperative signal processing. The core of this algorithm is a distributed election procedure that produces one or multiple winners based on a context-dependent election metric. In scenarios where non-coherent signal processing techniques are applied, a central processing node is selected, and highly compressed sensor data will be gathered for processing. Energy efficiency is improved by reducing algorithm overhead because the actual sensor traffic volume is light compared to the messaging overhead of the algorithm. For coherent processing, raw data streams must be relayed from each sensor to the central processing node, producing large data streams. A multi-winner election process is initiated first to select only a limited number of sensors that will provide the raw data; then a second election process will use an energy-based metric to find the optimal central processing node whose location minimize the total relaying cost. Simulation result is provided to demonstrate the inherent overhead-delay trade-off and compare the scalability of the algorithm under different scenarios.

improve the quality of raw data by canceling noise from sensor measurement using side information supplied by other co-located sensors. It is a valuable application to sensor networks deployed for scientific, tactical, as well as space/planetary exploration purposes. Having many advantages over traditional signal processing techniques, there is an interest in the research community to develop new networking technology that can support cooperative signal processing. The most significant challenge in developing such technology is overcoming the energy constraint, since for most wireless sensor networks energy is the most expensive resource.

While energy-efficient algorithms for network self-organization and routing have long been studied [4][5][6], they are not yet integrated with cooperative sensor operations. In [1], the basic concept and general approach for cooperative sensor operation has been described. In this paper, we will provide more detail description and discussion of a distributed algorithm that can serve as the enabling technological platform on top of which cooperative signal processing can take place.

## TABLE OF CONTENTS

1. INTRODUCTION
2. SENSOR NETWORK OPERATIONS
3. DISTRIBUTED ELECTION ALGORITHM
4. SIMULATION STUDY
5. CONCLUSION

## 1. INTRODUCTION

Cooperative signal processing, such as blind beam forming [3] and data fusion, is one of the most important application in modern sensor networks. By making several simultaneous observations on the same phenomenon and exploit the correlation in the combined data set, a great deal of information, which otherwise would be hidden, becomes available. Cooperative processing can also dramatically

## 2. SENSOR NETWORK OPERATIONS

Cooperative signal processing is essentially a form of hierarchical information processing where raw sensor data is first collected and processed by individual nodes to generate a parametric or filtered version of the original data, and later gathered at a single location for combined processing. The benefit of such processing is that it eliminates the communication cost for relaying the raw data to some entity outside of the sensor network for processing. It can also improve the quality of sensor data, probability of detection and false alarm, and possibly the resolution of data analysis.

### *Cooperative Signal Processing*

There are two categories of cooperative signal processing techniques: (1) coherent combining and (2) non-coherent

---

<sup>1</sup> 0-7803-7231-X/01/\$10.00/©2002 IEEE

combining. For both techniques, each sensor gathers data generated by a target event, performs varying degrees of filtering or pre-processing on the data, and then sends the data to a near-by node for combined processing. The terms “coherent” and “non-coherent” mark the degree to which temporal information is removed from the data. There are advantage and disadvantage to both approaches. For non-coherent processing, the raw data is often parameterized and/or highly compressed such that the communication cost can be significantly reduced. However, subtle target features hidden in the correlation between time domain waveforms may be lost. For coherent processing, the raw data is only mildly filtered before combined processing takes place, but the communication cost associated with relaying long data streams can be prohibitively high because of energy resource limitation.

Due to the significant difference in traffic volume, network/routing algorithm works differently for coherent and non-coherent signal processing. For coherent processing, the bulk of energy cost will come from relaying data traffic; therefore we focus on finding the optimal processing node and the minimum energy routes. For non-coherent processing data traffic is lower; energy minimization is best achieved by reducing the overhead in the algorithm itself.

#### Event-Based Sensor Operation

Traditional sensor operations are time-based, where each sensor measures and records data from the environment based on a pre-determined schedule. This is perfectly suitable for applications that are interested in monitoring the environment over time rather than looking for specific event or phenomenon. For sensors that are designed to look for randomly or even rarely occurring events, an event-based operation is more energy efficient because it allows the sensors to operate primarily in a low power mode and only expend energy when necessary. This approach has the potential of prolonging the total lifetime of a sensor network over those that are regularly operating in an active state.

The primary function of a network/routing algorithm is to facilitate an efficient and timely transition from the low-power, inactive state to a highly interactive state where multiple nodes are collecting and processing sensor data in coordinated fashion. **Figure 1** shows the basic operational flow for such network/routing algorithm.

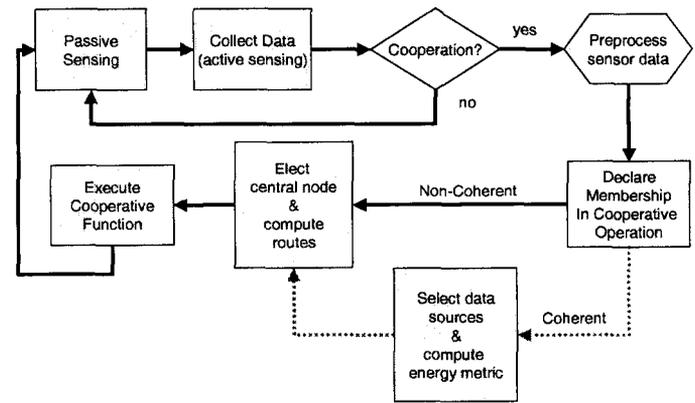


Figure 1: Basic Operations

A sensor will spend most time in the passive state where its sensing capability is only partially activated to look for the signatures of a limited class of events. The transition from passive to active sensing state, where sensor circuitry is fully activated to collect data, relies on the whether the signal received in the passive mode represent strong statistical inference on the occurrence of an interested event. A simple example is to use a threshold method applied on the Signal to Noise Ratio (SNR) in certain set of frequency bands that comprise the signature of an event.

Once data collection is completed, some assessment must be made to determine whether cooperative processing is necessary and if so what technique should be used. Such decisions can be made in a distributed fashion by pre-programmed algorithm in each sensor or in a centralized fashion by an outside user. In the former case, the sensor will simply proceed to declare to its neighbors the intent to participate in cooperative processing; in the latter case, a remote user or separate software agent can disseminate its decision to each sensor through a multi-hop network.

The core of the algorithm lies in the distributed process of electing one or multiple sensor nodes to perform specific tasks. Depending on the goal of each election process, different election metric is used to find the most suitable candidates. In the coherent case, the algorithm is designed to reduce the relaying cost of raw data from the sensors to the processing node. There are two ways to accomplish this. First, the set of sensors that will provide the raw data must be pruned. For example, a distributed election can select a limited number of sensors nodes that have best SNR on the target signal to provide the data. Then a second election process will find the optimal processing node such that the relaying cost to gather the data can be minimized.

In the non-coherent case, the driver for choosing the central processing node is to achieve low algorithm overhead because the actual data traffic is small. Therefore as long as a node has sufficient computational capability, energy resource and close proximity to the target and other sensors in the cooperative group, it would be suitable for the task. There is no need to try to minimize relay cost because the

additional exchanges of routing and energy information would probably offset any gain in energy saving.

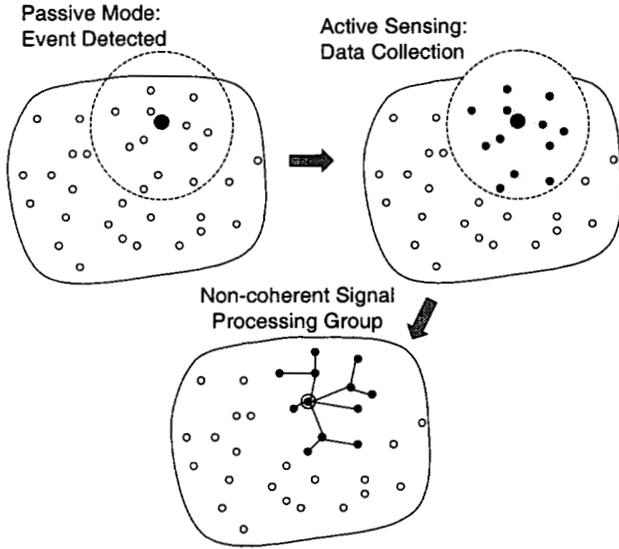


Figure 2: Non-Coherent Signal Processing

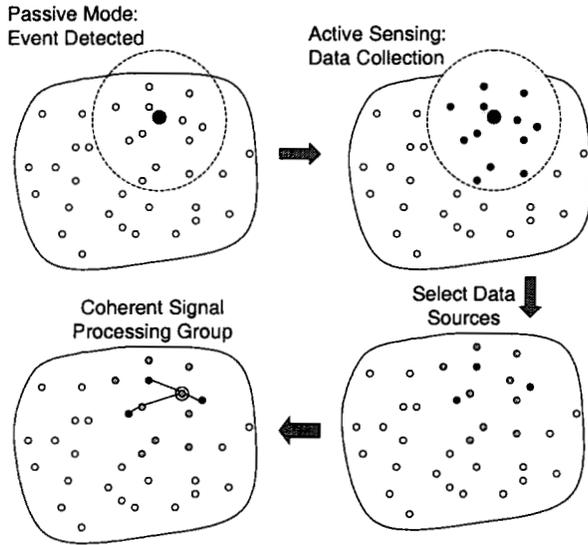


Figure 3: Coherent Signal Processing

### 3. DISTRIBUTED ELECTION ALGORITHM

One way we can select the central processor in a distributed fashion is by flooding the election metrics for each node throughout the network so that all candidates can be compared. However, this approach requires, assuming optimal flooding, at least  $N(N-1)$  transmissions or  $N-1$  messages per node, which is not scalable as the network size increases. To improve scalability, we propose two modifications that will lower messaging overhead: (1) breakdown a large election into smaller local elections, whose results are exchanged to derive the global winner, and (2) impose delay to suppress activities of those candidates that are likely to lose the election.

#### Localized Elections

If we consider breaking down an election among  $N$  nodes to two elections, one with  $m (\geq 1)$  nodes, one with  $N - m (\geq 1)$  nodes, then the minimum overhead required, assuming optimal flooding, would be:

$$\underbrace{m(m-1) + (N-m)(N-m-1)}_{\text{Local Elections}} + \underbrace{m + (N-m)}_{\text{Exchange Results}} \quad (1)$$

$$= m^2 + (N-m)^2$$

The overhead reduction is:

$$N(N-1) - (m^2 + (N-m)^2) \quad (2)$$

$$= N(2m-1) - 2m^2 \geq 0$$

Intuitively we can see that having two smaller elections and then exchanging the election results makes the algorithm more efficient by eliminating information propagation for nodes that have lost the local election. One can further divide each local election into smaller sub-elections to reduce overhead even more. Taking this approach to the smallest unit, we can have local elections that occur just between two neighboring nodes.

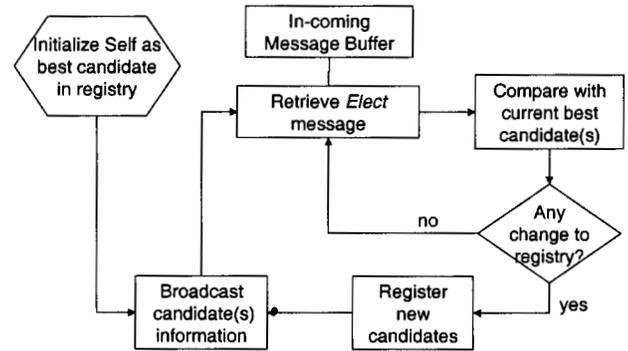


Figure 4: Distributed Election Algorithm

Figure 4 shows the flow chart of the election algorithm for each node. Candidate information is exchanged using an *Elect* message, which contains the node ID and election metric. Other routing information is piggybacked on the *Elect* message so that a minimum-hop spanning tree can be built from each sensor node to the eventual winner(s) of the election. Each sensor will have a registry designed to hold the information regarding the best candidate(s) it knows.

In the beginning, each sensor will initialize the registry with its own ID and election metric and multicast this information to all neighbors in the cooperative group. In response to an incoming *Elect* message, each node will compare the proposed candidate(s) with those in its own registry; when better candidates are found, the registry will be updated and all 1-hop neighbors belonging to the cooperative group will be notified. Each *Elect* message sent may spawn further exchange of *Elect* message as each sensor continue to compare candidates and update its own registry. Message exchange will eventually terminate when all sensors choose the same winner(s).

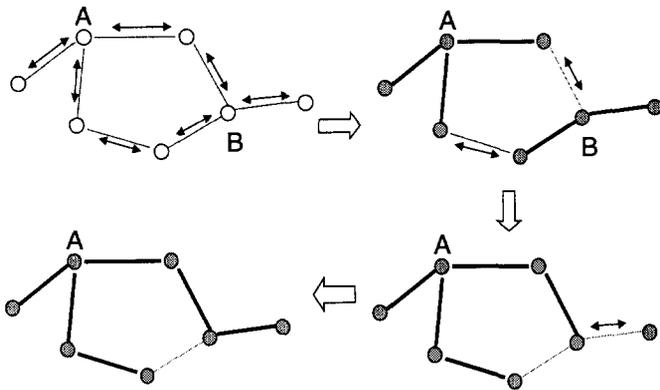


Figure 5: A Single Winner Election  
(A is the best candidate)

**Figure 5** shows a single winner election where node A and node B are respectively the best and second best candidates. After the first round of local election, A and B emerged as winners in their individual 1-hop neighborhood; however, as *Elect* messages from node A continues to propagate and win each local contest, eventually all nodes will choose A as the winner. In this case we have 10 local elections before the final winner emerges.

#### Overhead-Delay Trade-off

Further efficiency gain is possible if the local elections can be initiated in a sequential manner, rather than occurring simultaneously, such that the better candidates are given early starts. If the best candidate is given a sufficient head start over others, it is possible that its *Elect* messages will propagate throughout the network before other candidates have the chance to voluntarily initiate their own local elections. In the ideal scenario, very *Elect* message exchanged will only carry the best candidate's information, thus achieving the minimum overhead.

**Figure 6** shows the same election with voluntary delay imposed on the starting time of each sensor. Since node A is the best candidate, it will impose on itself the shortest delay and therefore starting issuing *Elect* messages earlier than all other nodes. We can see that by the time node B starts to initiate its own election process, node A has already dominated half the network. The result is that node B will quickly loose to the challenge from node A and the total number of local elections is less than the previous case where every candidate starts its own process simultaneously. In this case only 8 local elections took place before the winner is found.

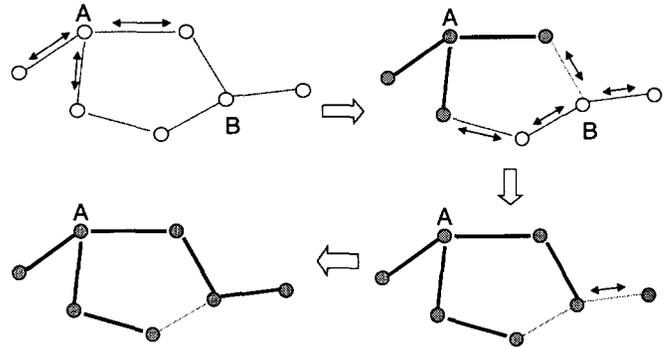


Figure 6: Single Winner Election with  
Voluntary Time Delay

Since there are no *a priori* knowledge which node is the best candidate, each node will usually begin its election process with some non-zero delay. This will create the *absolute delay*, the initial period of inactivity where no nodes are active, which increase the duration of the entire election process. However, overhead reduction comes from the *differential delay*, the amount of delay spread between the best candidate and all other nodes. Increasing the delay spread will, in general, increase the absolute delay as well. Hence we have a overhead-delay trade-off. Properly managing this overhead-delay trade-off will ensure the cooperative processing is performed in an efficient as well as timely manner.

#### Spanning Tree Computation

Because the ultimate goal of the election requires that each sensor find a route to the winner(s) of the distributed election, it is natural to piggyback routing information in the *Elect* messages such that routing computation can be performed simultaneously. The routing algorithm used in our scheme computes a minimum-hop spanning tree connecting each participating sensor to the winner(s) of the election. Due to the similar characteristic in message propagation between the election processes and a distributed minimum hop spanning tree computation, no additional complexity is added to the algorithm complexity except for a slightly larger payload in each *Elect* message. This approach ultimately shortens the duration of the entire network/routing algorithm and may also cut overhead by compressing election and routing information into a single message.

#### Election Termination

Another key component of the election process is that the winner of the election should be able to detect the end of the election process. This allows the winning node to direct a swift transition from the election process to the next phase of the operation. For single winner election, a distributed termination detection process can be implemented by using a  $(N+1)$  state algorithm, where  $N$  is the number of participating sensor nodes in the election.

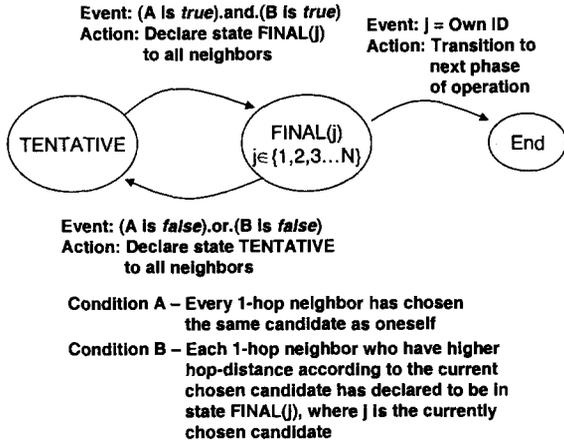


Figure 7: Termination Detection Algorithm

Each node starts the election process in state TENTATIVE. When both condition A and B are satisfied, a node will enter state FINAL(j) – if currently the best candidate is node j. In the case the node j enter FINAL(j), the election has ended and node j realize it has won the election. From this point on, it can trigger other actions necessary for the next phase of operation. Note that for condition B, the spanning tree algorithm, which is executed simultaneously with the election process, provides the knowledge regarding hop-distance with respect to any chosen candidate. Each neighbor will update each other of changes of state information {TENTATIVE, FINAL(j),  $j=1,2,\dots,N$ } by using the *Elect* message, which as a “state” field in its payload. If the election process is won by node j, eventually each node will select node j as its candidate. Due to condition B, nodes that are farther away from the winner will enter state FINAL(j) first and followed by those that are closer in hop-distance, until eventually node j itself enters state FINAL(j), which ends the election process. **Figure 8** illustrates this process graphically.

The termination procedure for elections that produces multiple winners is more complex to design due to the large number of states and the difficulty of coordinating the winners. To avoid this additional layer of complexity, we use a passive method of detection; namely, we monitor the *Elect* message traffic and allow each node to infer whether the election has terminated. For example, a timer can track the time elapsed since the last *Elect* message is received. If sufficient time has elapsed without observing any further message exchanges, then a node will declare the election terminated and assume the candidate recorded in its registry as the final winner. The time out must be carefully chosen to balance the possibility of pre-mature election termination and high latency.

#### Failure Recovery

When link or node failure occurs, the network may be severed topologically and/or create invalid routes that causes a deadlock in the election process. One remedy is to have the data link or physical layer report such occurrence and either execute a local recovery procedure or re-start the

entire election process. Another possibility is to use a timer so that each sensor does not wait indefinitely for the process to end. Some sort of reporting mechanism can also be used to request instruction from a remote user when the election process fails. In our simulation study, however, we will ignore the effect of random link or node failures.

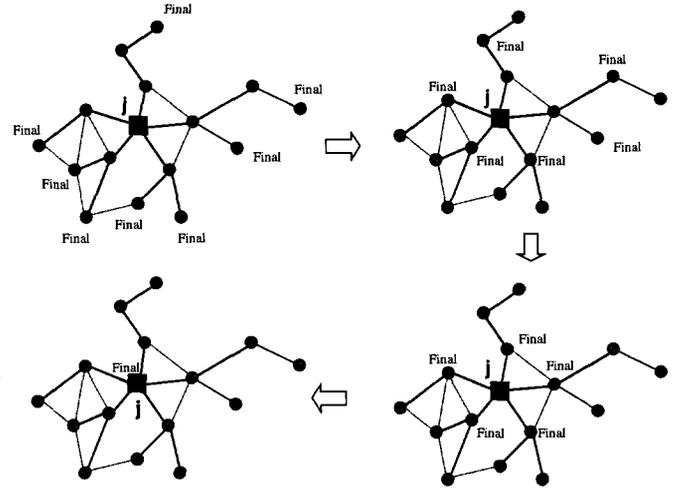


Figure 8: Termination Detection

## 4. SIMULATION STUDY

In this section we describe a simulation study of the network/routing algorithm and explore the overhead, latency, and scalability of the algorithm under different scenarios.

#### Assumptions and Scenario Setup

In our simulation study, we made the following assumptions:

1. At any time instance, there is only one event occurring in the environment.
2. There are two types of signal source generated as the result of an event:
  - a. Near-field Line-of-sight source (NL) – We assume an event occurs at a random location in the near field of a sensor network and that the received signal is dominated by the line-of-sight component. The average received signal strength at each sensor is modeled by an inverse-square law based on the distance between an event and a sensor such that only sensor sufficiently close to the event center will be activated. In our simulation, the average received signal strength at 1 meter varies from 3dB (2) to 17dB (50).
  - b. Strong Far-field Multi-path source (SFM) – We assume a far-field event occurs such that the received signal is dominated by the mulit-path propagation component. Such events will generate sufficiently strong signal to active most of the sensors even though the source is far field. The received signal strength at each sensor is modeled

by i.i.d. Raleigh random variables with mean value of 50.

3. For NL sources, we use a fixed network of 99 nodes, and the average network size is determined by the number of sensors close enough to detect the event. For sensors with received signal strength greater than the threshold of 0.4, it will become active and participate in cooperative signal processing.
4. For SFM sources, we received signal is sufficiently strong to awaken most nodes in the sensor network because the mean value is 50 while the threshold is fixed at 0.4. We examine the performance of the algorithm under different network sizes by varying the number of sensor from 10 to 99 while keeping the node density fixed.

We studies the following four scenarios:

1. Non-coherent processing with NL sources
2. Non-coherent processing with SFM sources
3. Coherent processing with NL sources
4. Coherent processing with SFM sources

For non-coherent processing, received signal strength is used as the election metric for choosing the central processing node. We choose this particular election metric because it serves as an general indicator of the likelihood that it will be in close proximity to the signal source, and therefore in a good location to gather sensor data from other sensors observing the same phenomenon.

For coherent processing, the election metric for selecting data source nodes is the received signal strength, which serves as a proxy for data quality. Data set from each sensor has random length, and a maximum of 5 sensors will be selected to provide the raw data. For the central processing node, total energy cost for relay data the sensor is the election metric. The transition from the first election to the second election uses a timer scheme that incurs a fixed extra latency of 20 simulation time units.

#### Data Link/MAC Layer

We use a simple model for the data link layer. We assume that a contention-free TDM-like MAC schedule exists and error-free transmissions. Each transmission will incur average frame latency of 1 simulation time unit and have the same energy consumption.

#### Voluntary Time Delay and Election Metric Calculation

The voluntary time delay as a function of received signal strength is an inverse function given by,

$$D(S) = \frac{D_o}{S} \quad (3)$$

where  $S$  represents the received signal strength and  $D_o$  is a coefficient. Since the goal of imposing delay is to favor those with larger  $S$ ,  $D(S)$  is an inverse function. The voluntary delay for coherent case is computed as a result of the relay energy cost (represented by the number of transmissions to relay all selected sensor data to node  $j$ ) is given by:

$$D(E_j) = D_o \frac{E_j}{E_{\max}} \quad (4)$$

Since the goal is to *reduce* the energy cost, the delay imposed is proportional to the energy cost.  $E_j$  is the total number of transmission required to relay all sensor data to node  $j$ . It is given by  $\sum_{i \in SN} L_i d_{i,j}$  where  $L_i$  is the number of

packets node  $i$  has to send, and  $d_{i,j}$  is the hop-distance from node  $i$  to  $j$ .  $SN$  is the set of sensors that have been selected to provide the raw data for the coherent combining.

#### Simulation Results

**Figure 9** shows the overhead-delay trade-off of our algorithm for non-coherent processing.

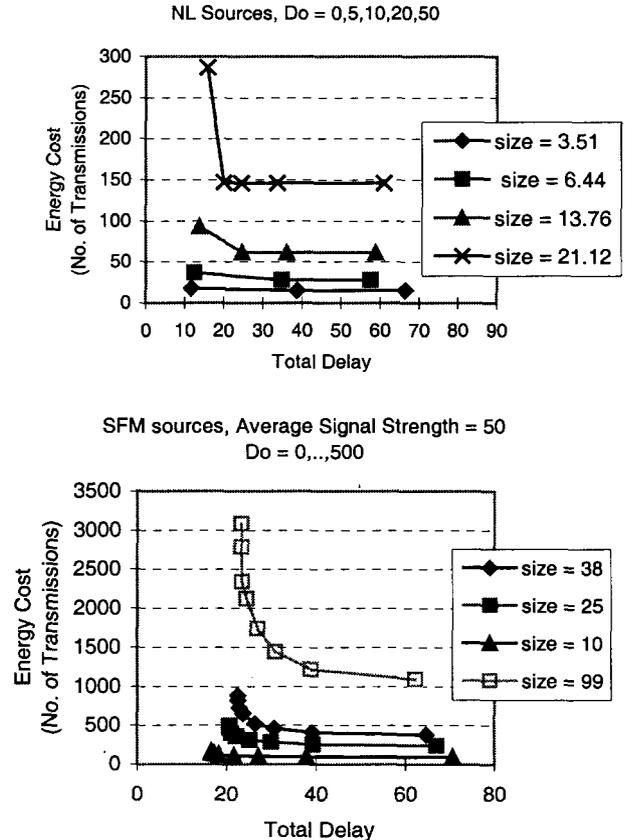


Figure 9: Delay-energy Trade-off for Non-coherent Processing

The first observation we can make is that for SFM sources the overhead reduction is more gradual as delay increases; while for NL sources the trade-off is stronger, and the overhead reaches its minimum quicker.

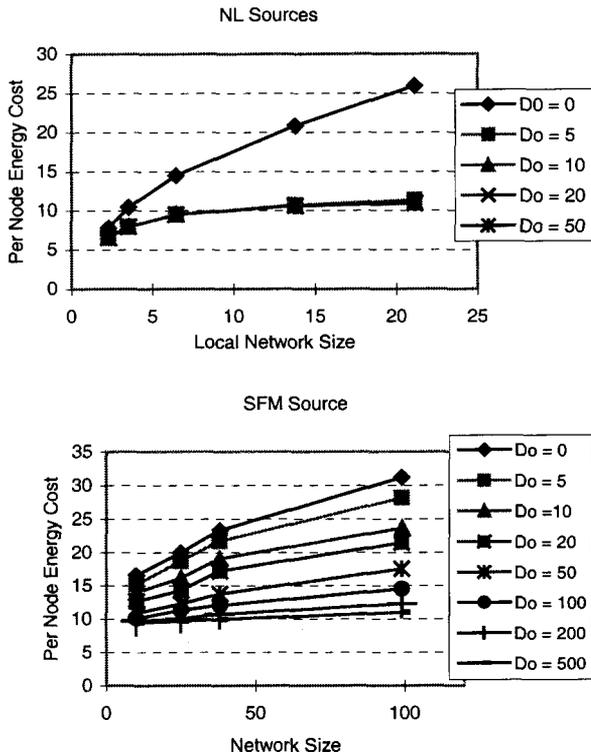


Figure 10: Scalability for Non-Coherent Processing

Figure 10 illustrates a key difference in scalability for NL and SFM sources. We can see that for NL source, the non-coherent process is very scalable (nearly flat average energy cost for each node with respect to network size) except when no voluntary delay is imposed. While for SFM sources, the algorithm shows only slow and gradual improvement in scalability as the delay coefficient  $D_o$  increases. One possible explanation for this difference is that for NL sources the most competitive candidates, those with highest received signal strengths, tend to cluster in close proximity. Therefore small differential delays will give the best candidate sufficient head start over other strong competitors. For SFM sources, nodes that have comparable received signal strengths are more spread out due to multi-path signal propagation. Therefore longer delay is required to suppress message exchanges initiated by losing candidates.

Figure 11 and Figure 12, shows the overhead, latency and scalability of the algorithm for coherent processing. In this case, the scalability is still better with NL sources. However, the overhead-delay trade-off for SFM sources is much weaker than the non-coherent case. This means one must be willing to tolerate very long delay in order to minimize overhead. The average energy cost is in general higher than the non-coherent case because it requires two distributed elections. Although the network/routing algorithm for

coherent case has higher overhead and delay than its non-coherent counterpart, the less-scalable algorithm will generate significant performance pay-off by finding the optimal central node and routing paths to relay large volume of sensor data. Therefore saving energy in the long run. In reality, it is difficult to compare the relative merit of the algorithm under the coherent and non-coherent cases because the underlying traffic characteristics for these two types of processing are inherently different.

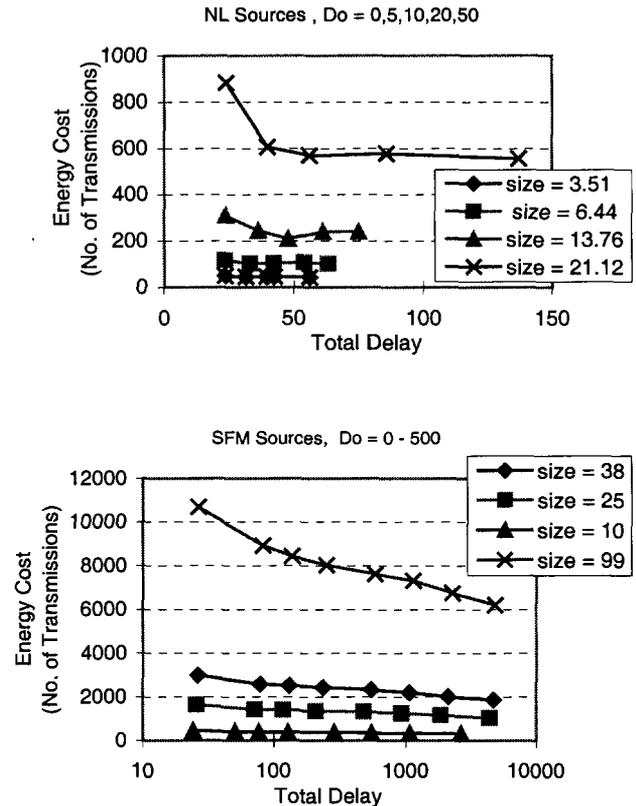


Figure 11: Delay-energy Trade-off for Coherent Processing

## 5. CONCLUSION

In this paper, we have described a network/routing algorithm that facilitates adaptive event-driven cooperative signal processing applications in sensor network. We explored the inherent overhead-delay trade-offs and scalability of this algorithm under both coherent and non-coherent cases where the target signal is dominated by either a line-of-sight or multipath mode of signal propagation. We conclude that for non-coherent scenario, our algorithm is very scalable and has low latency. For coherent processing, we achieved the objective of finding a central processing node such that the energy cost of relaying raw data can be minimized at the expense of higher algorithm complexity, latency, and lower scalability.

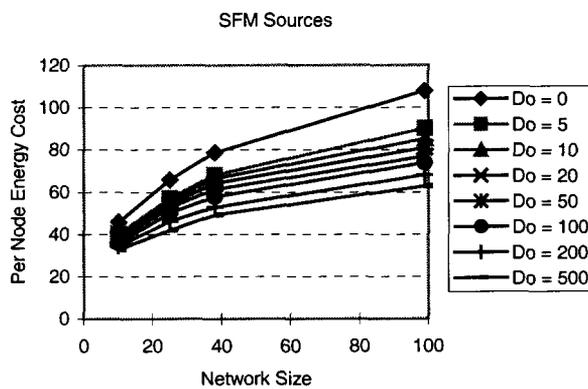
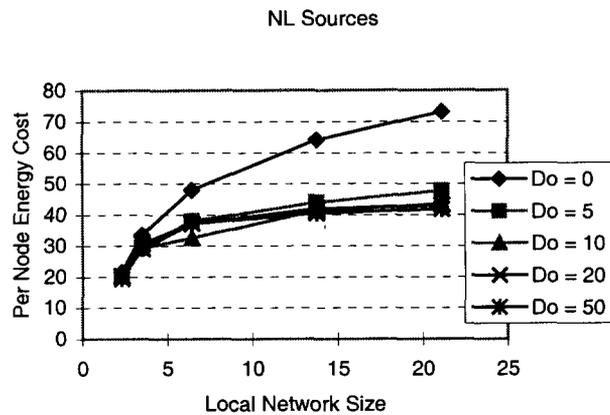


Figure 12: Scalability for Coherent Processing

## REFERENCE

- [1] J. Gao, K. Sahrabi, V. Ailawadhi, and G. Pottie, "Protocols for Self-Organization of a Wireless Sensor Network," *IEEE Personal Communications Magazine*, October 2000.
- [2] E. W. Dijkstra and C.S. Scholten, "Termination detection for Diffusing Computations," *Information Processing Letters*, vol. 11, no.1, 1-4, August 1980.
- [3] K. Yao, R.E. Hudson, C.W. Reed, D. Chen, and F. Lorenzelli, "Blind Beam-forming on a Randomly Distributed Sensor Array System," *IEEE Journal On Selected Areas in Communications*, vol. 16, no. 8, 1555-67, October 1998.
- [4] K. Scott, "Control and Routing in Self-Organizing Wireless Networks," *Ph.D. Dissertation*, Department of Electrical Engineering, UCLA, 61-76, 1997.
- [5] S. Singh, M. Woo, C.S. Raghavendra, "Power-Aware Routing in Mobile Ad Hoc Networks," *MOBICOM'98*, 181-190, Dallas, TX.
- [6] V. Rodoplu and T.H. Meng, "Minimum energy mobile wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, 1333-1344, August 1999.

Jay L. Gao is a research staff with the Communication Systems and Research Section of the Jet Propulsion Laboratory. He is currently working on protocol development and performance evaluation for in-situ surface-to-surface and surface-to-orbit communications in planetary missions, as well as developing energy efficient network and routing protocols for sensor networks. He received his B.S., M.S., and Ph.D. in Electrical Engineering from UCLA in 1993, 1995, and 2000, respectively.

