

# A Peer-to-Peer Collaboration Framework for Multi-sensor Data Fusion

Panho Lee, Anura P. Jayasumana, Sanghun Lim, V. Chandrasekar  
{leepanho, anura, shlim, chandra}@enr.colostate.edu  
Department of Electrical and Computer Engineering, Colorado State University,  
Fort Collins CO 80523 USA

**Abstract—A peer-to-peer collaboration framework for multi-sensor data fusion in resource-rich radar networks is presented. In the multi-sensor data fusion, data needs to be combined in such a manner that the real-time requirement of the sensor application is met. In addition, the desired accuracy in the result of the multi-sensor fusion has to be obtained by selecting a proper set of data from multiple radar sensors. A mechanism for selecting a set of data for data fusion is provided considering application-specific needs. We also present a dynamic peer-selection algorithm, called Best Peer Selection (BPS) that chooses a set of peers based on their computation and communication capabilities to minimize the execution time required for processing data per integration algorithm. Simulation-based results show that BPS can deliver a significant improvement in execution time for multi-radar data fusion.**

## I. INTRODUCTION

Recent advances in technology are introducing means to revolutionize our ability to observe, understand, and predict hazardous events by creating distributed collaborative adaptive sensing (DCAS) systems. These systems use a large numbers of *distributed* powerful sensors, such as radars, to improve spatial and temporal resolution throughout sensing area, operate the sensors *collaboratively* and *adapt* them to changing conditions in a manner that meets competing end-user needs [1]. Unlike mote-based resource poor sensor networks, which are designed to carry out one or few tasks minimizing energy expenditure, the DCAS systems are intended to serve a variety of applications and users with different requirements. Due to the massive amount of data generated by such sensors and computing intensive processing required by applications, the sensor/processing nodes are usually equipped with ample communication and computation resources.

Collaborative Adaptive Sensing of the Atmosphere (CASA) [1~5] is an example of these emerging DCAS systems. CASA is based on a network of short-range weather radars that operate collaboratively to detect and track hazardous localized weather phenomena such as tornadoes. The data generation rate at each of the radars can be several Mbps to tens of Mbps. A suite of meteorological feature detection algorithms that rely only on data from a single-radar have been used for automated identification of the hazardous weather features. To achieve improved accuracies and more specific inferences, it can be required to combine data from the multiple radar sensors using

multi-sensor data fusion algorithms [2].

Client-server is one of the most popular frameworks for realizing multi-sensor data fusion. In a client-server framework, a powerful server acquires data from sensors to perform the algorithms. Although widely used, these frameworks may not be appropriate for these applications, because the demand on the server would increase in proportion to the total number of sensors; quickly overrunning server's limited capacity. To deal with the issue, peer-to-peer (P2P) framework is of interest as an alternative paradigm to the client-server architecture. In contrast to client-server frameworks, each node in peer-to-peer networks can provide bandwidth, storage and computation. For example, in file sharing P2P networks, such as BitTorrent[6], Napster[7] and Gnutella[8], each peer supplies a disk space to store the community's collection of data and bandwidth to move the data to other peers, and obtains others' resources in return. When communication cost becomes too high (e.g., in case of large multimedia content transfers), P2P networks can avoid the bandwidth limitation problem by spreading the cost over several peers. Likewise, resource-intensive sensor applications can benefit from P2P computing. In P2P frameworks, the applications can aggregate and utilize unused resources from peers over the network to achieve better performance on processing sensor data. Furthermore, the P2P mechanism also may reduce the risk of a single-point-of-failure which can be disastrous in client-server architecture.

To use a P2P framework as a feasible solution for multi-sensor data fusion, we must take the mission-critical nature of the sensor applications into account. With multi-sensor data fusion, the peers need to exchange samples to be integrated; thus the samples or data are dynamically replicated among the peers participating in the processing. Since geographically distributed sensors generate substantial volume of samples, the communication cost cannot be ignored. In addition, the large volume of generated samples must be processed with stringent time constraint, so vast amount of computation resources have to be deployed on the fly. Furthermore, in these high-end sensor networks, the peers joining the multi-sensor fusion algorithm can have a wide range of network access speeds, available bandwidth, and variability in system load. Therefore, the main problem is to coordinate a group of peers, each of which may have intensive resource demanding and high variability in performance. That is, when several sensor nodes work together on a multi-sensor data fusion algorithm they need to decide which node would be responsible for providing particular contents with acceptable performance for the multi-

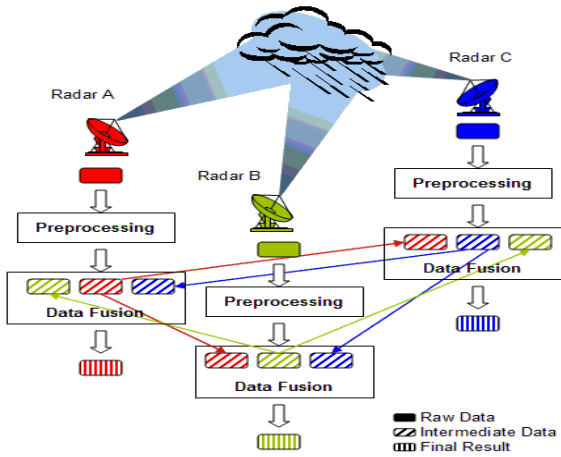


Figure 1. Multi-sensor Data Fusion

sensor data fusion algorithm. Thus, the decision must involve not only locating peers who have the desired contents, but also selecting a peer that can provide the contents within the real-time constraints. Another important issue to be considered is how to select a set of data from multiple sensors when the sampling time and intervals of the sensors are not synchronized. Because in a sensor network, each of the sensors can be assigned to a certain sensing task with different sampling schedule and mode, the synchronization of sample generation time can be a considerable task. In the presence of imperfectly synchronized sample generation, we need to provide a mechanism that notifies the peers participating in a multi-sensor data fusion algorithm when new samples are available for processing. Furthermore, a scheme is needed that selects a set of samples considering the time gap between samples and timeliness of processing.

In this paper, we present a P2P collaboration framework for CASA multi-sensor data fusion algorithms. We provide a simple data selection mechanism that addresses the problem associated with the data synchronization. We propose a dynamic peer selection algorithm, called Best Peer Selection (BPS) that allows the multi-sensor data fusion algorithm to locate desired contents in a scalable, efficient, and distributed manner, and to obtain the desired contents minimizing time. In our radar sensor network, each of the peers maintains a list of peers which are participating in the same multi-sensor data fusion process. The required contents are located by querying peers on the list. Upon receiving the query, the peers estimate the time required for providing the desired contents considering their computation and communication overhead, and respond to the query-initiator node with the estimated time. Based on the response messages, BPS can find a subset of the peers that can collectively provide the given set of contents within the time constraint. We demonstrate the feasibility and efficiency of BPS using simulations. The experimental results show that BPS can deliver a significant performance improvement, especially when the peers and the network have extremely high variability in resource availability.

The rest of the paper is organized as follows. In Section II, we introduce the system model, followed by the software

architecture in Section III. Section IV presents a data synchronization mechanism. Section V details BPS, and Section VI presents the experimental results. Section VII concludes this paper.

## II. TARGET SYSTEM MODEL

A high-performance radar sensor network is anticipated to consist of tens of sensor nodes connected by a combination of wired and wireless networks interconnecting the sensor nodes. The nodes run TCP/IP, and may share links with other Internet nodes. The sensor nodes are not resource constrained in terms of computation and energy compared to mote-based wireless sensor networks, and data generation rates can be several Mbps to tens of Mbps per radar sensor node. Furthermore, the sensor network is designed as a multiple end-user system. Multiple end users/applications may be present that have distinct sensing/communication/computation requirements necessary for their operations. Some of the end users/applications need to combine data from multiple sensors. Because each of the sensor nodes is allowed to conduct in-network processing and provides computation and communication resources responding to user requests, the multi-sensor data fusion can be performed collaboratively and concurrently at each of the sensor nodes [3]. Furthermore, in this real-time system, the radar sensors must be re-tasked every 30 seconds (heartbeat interval of the system) with the system goal of detecting and tracking hazardous meteorological features within 60 seconds [4].

### A. Multi-sensor Data Fusion Algorithm

In our multi-sensor fusion algorithm, data acquired or received from remote sensor nodes are used as supplementary information for correcting sensing errors in the data collected by the local sensor. As illustrated in Fig 1, radar node A requires local data as well as data from node B and C to correct its sensing errors such as those due to attenuation of radar signal, while node B needs the data sampled by nodes A and C. In addition, the multi-sensor fusion algorithm usually requires a number of steps to obtain the final result. As seen in the Fig. 1, the radar data processing algorithm considered consists of two steps, preprocessing and main data fusion processing. In the first step, collected raw samples are pre-processed or sub-sampled to meet the particular requirements of the individual applications. To clarify, we define a sample as a set of radar measurements collected in a particular scanning mode (e.g., 360 degrees wide/2 elevation angle). Due to large amount of the samples collected by each of the radars, even this simple quality control, i.e., preprocessing step, of the samples takes considerable time. We had implemented the multi-sensor data fusion algorithm on a machine (3.16GHz Intel CPU, 2GB RAM) running Linux [2]. The preprocessing for the algorithm requires 1~2 seconds per sample on average. In the second step, the algorithm detects significant features by integrating the pre-processed samples. The amount of computation demand of the second step varies from 1 to 12 seconds depending on the size of the detected meteorological objects.

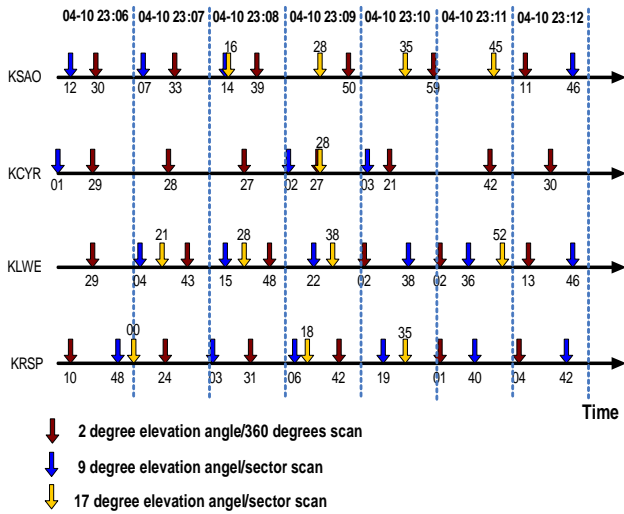


Figure 2. Sample Generation Timing

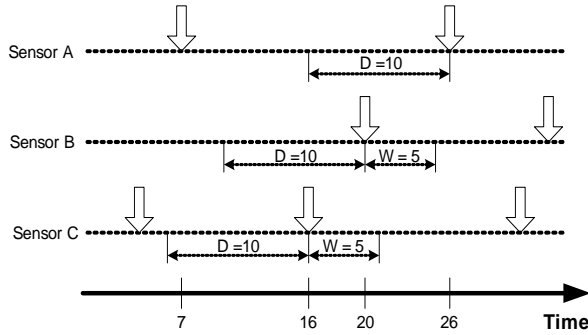


Figure 3. Data Synchronization Example.

### B. Sampling Interval

In the sensor network, some radar scan tasks are triggered only when a particular meteorological feature is detected in the coverage area, while other tasks, such as surveillance scans at lower elevations, are triggered periodically. In addition, the various users are able to express their sensing preferences in terms of area to be scanned and sampling interval as well as the detection of meteorological features, and these preferences must be incorporated into sensing operation. The differing sensing preferences of disparate end users and reactive sensing strategies mean that it may not be possible to satisfy the needs of all end users and the required sampling interval for particular radar algorithms at the same time. Fig.2 illustrates an example of the sampling timing of CASA IP1 test-bed deployed in Oklahoma [5]. The multi-sensor data fusion algorithm considered in this paper currently requires a set of 360 degrees/2 degrees elevation angle scanned radar reflectivity samples from multiple radars.

## III. NODE ARCHITECTURE

Each peer in the sensor network has an event handler, a peer manager and a resource manager. A multi-sensor data fusion application needs to continuously collect and integrate data generated from a distributed group of sensor nodes. There are a

number of sensors exchanging data to be integrated, while all the required data may not be available at the time of processing. Therefore, the sensor applications have to be notified of the availability of new samples whenever new samples are generated. To provide the notification service, the event handler implements three actions:

- Delivering the data generation notifications to other nodes. A notification does not include sampled data itself, but it contains the event source, generation time, data location, and sample size.
- Providing a subscription mechanism to the radar application algorithms (RAA). Each RAA is interested in only a particular type of samples. RAAs express their interests by registering a subscription with the event handler to be notified of any forth-coming events matching the subscription.
- Informing RAAs of the generation of a sample by the local and remote sensors.

The resource manager maintains the following information at a sensor node:

- The list of application tasks the node can offer and the history of execution time for the application tasks.
- The current available bandwidth for incoming/outgoing data transfer. Every time a node communicate with another nodes, the resource manager of the node keeps track of the Exponentially Weighted Moving Average (EWMA) of incoming/outgoing bandwidth based on the time ( $\Delta T$ ) required to transmit/receive a sample and the size (S) of the sample as follows:

$$IN_{EWMA} = \alpha * IN_{EWMA} + (1 - \alpha) * (S_{IN} / \Delta T_{IN}),$$

$$OUT_{EWMA} = \alpha * OUT_{EWMA} + (1 - \alpha) * (S_{OUT} / \Delta T_{OUT}),$$

To avoid too heavily being influenced by temporary bandwidth fluctuations, we currently set  $\alpha$  to 0.8.

- The log of new sample generation events from local and remote sensors. In addition to the sample generation events, the log also contains the information about the outputs of various processing (e.g., preprocessing output, and the final results of the algorithm) and replicated contents from other peers.

As mentioned before, the multi-sensor data fusion algorithm requires obtaining samples from other sensor nodes. Obtaining data involves locating peers who have copies of the desired contents. Peers that have the desired contents are usually restricted to the peers who share same interest of processing. In order to achieve good performance for locating the contents, the peer manager maintains a list of peers who participate in the same multi-sensor data fusion processing.

## IV. DATA SYNCHRONIZATION

When the event handler notifies the algorithm of a new sample generation, the algorithm attempts to start the multi-sensor data fusion. First step of processing is to search the event log to find a proper set of data coming from other sensor nodes to combine them with the local data. However, as seen in Fig. 2, the sampling time of the sensor nodes is not synchronized; the algorithm may not be able to find a proper data set. Some

RAAs may want high sensing accuracy using more samples, while others may prefer timely processing to enhancing sensing accuracy; therefore RAA needs to decide whether it starts processing with the currently available samples or waits sometime to get sufficient samples for enhancing accuracy based on their application-specific processing/sensing goals. Therefore, it is required to allow the algorithms to decide their data synchronization criterion in terms of the minimum number of samples (N) from different sensors for acceptable final result, the maximum possible waiting time for samples (W) when the number of samples is not sufficient, and the maximum tolerance in sampling time difference among the samples to be integrated (D). Fig 3 illustrates an example of data synchronization for multi-sensor data integration algorithms. In the example, a particular algorithm merges samples from three sensor nodes A, B, and C with  $W=5$ ,  $D=10$ , and  $N=3$ . At time  $t=16$ , sensor C generate a new sample, and the event handler running on sensor C posts the notification of available sample. Then the algorithm running at the node C examines the event log whether samples from other nodes are available. As seen in the figure, sensor A generated an appropriate sample at time  $t=7$ , but sensor B has not generated any samples in past 10 seconds. Thus, the algorithm has to wait until the data selection criterion ( $N=3$ ) is satisfied. At  $t=20$ , sensor B generates a new sample, and the difference 20 and 16 is less than  $W=5$ . Because all the criterion is satisfied, the algorithm can start processing at time  $t=20$ .

## V. BEST PEER SELECTION PROTOCOL

After choosing a set of samples to be merged during the synchronization phase, an application starts multi-sensor data fusion by passing a request to the BPS protocol. We define the sensor nodes on which the application starts its data fusion as an *Initiator Node*. Similarly, the peers that participate in the same multi-sensor data fusion processing with the Initiator node are defined as Neighbor peers. The Initiator Node's BPS protocol has four states - *Initiator-Init*, *Initiator-Probe*, *Initiator-Assign*, and *Initiator-Wait*. Likewise, each of the neighbor peers has *Neighbor-Init*, *Neighbor-Probe*, and *Neighbor-Proc* states.

As illustrated in Fig. 1, because individual samples from multiple sensor nodes can be pre-processed independently, we can run preprocessing the individual samples on multiple peers. In addition, the same set of samples can be used by multiple executions of the algorithm on different nodes to correct sensing error in the data collected by individual sensors. Thus, a group of nodes interested in the same set of samples can share intermediate outputs that are obtained by preprocessing instead of raw samples. To exploit those properties of considered multi-sensor fusion processing, the application properly divides the whole process of the algorithm into several sub-preprocesses. After decomposing the whole process, the application passes down an application request that consists of the sub-processes. Each of the sub-processes is composed of a non-empty set of processing items, and each processing item is specified by a type of required processing

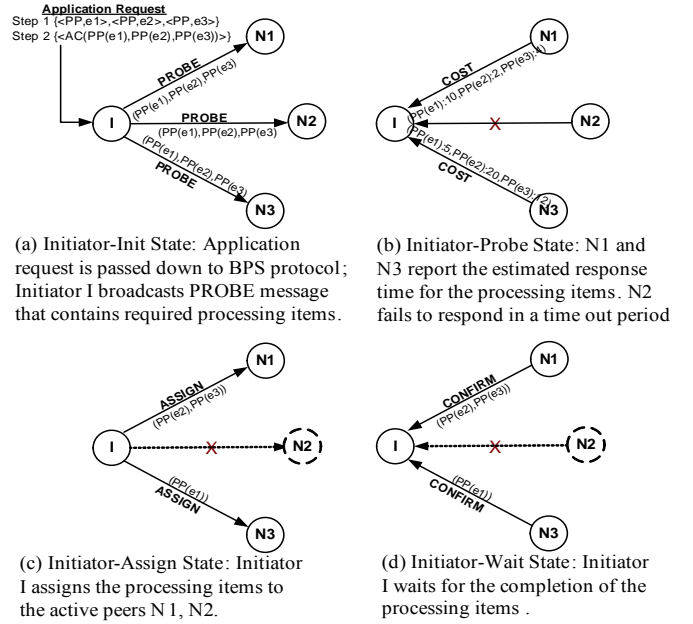


Figure 4. Best Peer Selection Protocol

and an input data set for processing. Fig. 4 illustrates how the BPS supports the multi-sensor data fusion algorithm for better performance. In the figure, the BPS protocol receives an application request which has two sub-processes: Step 1  $\{<PP, \{e1\}>, <PP, \{e2\}>, <PP, \{e3\}>\}$  and Step 2  $\{<AC, \{PP(e1), PP(e2), PP(e3)\}>\}$ . The first sub-process consists of 3 processing items and each of the processing items has a processing type (PP) and a sample (e.g., e1, e2, and e3) as an input for processing. The second sub-process has a single processing item and the outputs of the first sub-processes are used as the inputs for processing.

The Initiator node waits for a request from a radar application in the *Initiator-Init* state. Upon receiving the application request, the Initiator broadcasts a probing message, PROBE, to its neighbor peers. The PROBE message contains the first sub-processes (Fig. 4(a)). After broadcasting the PROBE message, the Initiator moves to the *Initiator-Probe* state, and waits for neighbor peers' response to the PROBE (Fig. 4(b)). The arrival of a PROBE message from the Initiator causes the neighbor peers to move from the *Neighbor-Init* state to the *Neighbor-Probe* state. In the *Neighbor-Probe* state, the Neighbor peers calculate the estimated time for providing the processing items specified by the PROBE message. The details of the time estimation are described in following subsection. After estimating the required time, each of the Neighbor peers responds to the Initiator Node with a COST message, which contains the estimated time for providing the desired contents. As illustrated in Fig 4(b), due to network congestion or outage or high system load, some peers may not be able to respond to the PROBE message in a certain time out period. In these cases, the Initiator node simply assumes that the Neighbor peers not responding are currently unavailable. Alternatively, the neighbor peers which can respond successfully in the time out period are regarded as active peers. When the Initiator receives the COST messages from all the neighbor peers or the

time out period is expired, the Initiator node moves from the *Initiator-Probe* state to the *Initiator-Assign* state. In the *Initiator-Assign* state the Initiator selects a peer among the active peers who has the minimum estimated time for a particular processing item based on the COST message. Once the Initiator node selects the neighbor peers for all the processing items, it assigns processing items to the selected peers by sending ASSIGN messages (Fig. 4(c)). After assigning the processing items, the Initiator moves to the *Initiator-Wait* state. In the *Initiator-Wait* state, the Initiator node waits for CONFIRM messages from the neighbor peers, which are notifications of the completion of the assigned processing items (Fig. 4(d)). When an ASSIGN message arrives at a neighbor peer, the peer moves to the *Neighbor-Proc* state, and provides the desired contents. After providing the desired contents, the neighbor peers notify the Initiator of the completion of the assigned processing items with the CONFIRM messages. After receiving the CONFIRM messages from all the neighbor peers, the Initiator peer starts the next step processing by broadcasting PROBE messages for the next step processing items. These procedures repeat until the final sub-process is done.

#### A. Peer Probing

In order to respond to the Initiator's PROBE message, the neighbor nodes calculate the communication and computation costs using simple estimation schemes. The details of the cost estimation algorithm are illustrated in Fig.5. The first step to the cost estimation is to determine the status of processing items. The status of a processing item can be one of the followings: *Exist*, *ExistRaw* and *NotExist*. If the status of a processing item is "*Exist*" at a neighbor node, then the neighbor has the output of required processing, so the Initiator node needs only the cost of communication to obtain the processing item from the neighbor node. "*ExistRaw*" means that only the raw sample exists at the neighbor; therefore, the cost of computation for required processing and the cost of communication are needed. In the case of the "*NotExist*", the neighbor node has neither the raw sample nor the output of the processing item, so the neighbor node needs to bring the raw sample from other peers and to run required processing. Therefore, in many cases, it is not likely to assign the processing item to a neighbor node when its status of processing item is "*NotExist*". In the BPS, the peers do not use expensive bandwidth and processing power estimation tools to determine communication and computation costs precisely. Instead, the following simplistic approach is used for the cost estimation: the resource manager of each peer keeps track of the EWMA of incoming/outgoing throughput as mentioned in Section III. Similarly, the resource manager also maintains the history of the computation time for particular types of processing. The computation time,  $t_{comp}(item[i])$  for the  $item[i]$  can be decided by the recent computation time for the same type of processing in the history list. Likewise, we define the estimated data transfer time to acquire sample  $e$ , and estimated data transfer time to send  $e$  after the processing as  $t_{in}(e)=e.size/IN_{EWMA}$ , and  $t_{out}(e)=e.size/OUT_{EWMA}$ , respectively.

#### B. Processing Item Allocation

The cost information provided by the neighbor peers is used to assigns the processing items to the active peers. During the iterations of allocation procedure, from the unassigned processing items a new processing item is selected and assigned to an active peer that has the minimum cost for the selected processing item. After assigning the processing item to the peer, the BPS protocol adds the cost of the selected processing item to the other processing items' costs estimated by the selected peer. By updating the other items' estimated cost, the chance for assigning multiple items to the same peer can decrease. The process is repeated until all processing items are assigned to the active peers.

```

Neighbor Node Probe State
Input: Item   item[]; /* from the probe message */
COST   cost[]; /* total execution cost */
int     i, j; /* index variable */
for each item j {
    cost[j].item = item[j];
    cost[j].total_cost = 0;
    for each data i in item[j].dataset {
        LOG log = get_log(item[j].dataset[i]);
        if (log.status == NotExist) {
            cost[j].total_cost =  $t_{in}(item[j].dataset[i]) + t_{comp}(item[j]) + t_{out}(item[j].dataset[i])$ ;
        }
        else if (log.status == ExistRaw) {
            cost[j].total_cost =  $t_{comp}(item[j]) + t_{in}(item[j].dataset[i])$ ;
        }
        else if (log.status = Exist) {
            cost[j].total_cost =  $t_{out}(item[j].dataset)$ ;
        }
    }
}

```

Figure 5. Probe state algorithms for neighbor peers

## VI. EXPERIMENTAL RESULT

SimGrid[10], a discrete time simulator was used in testing. The sensor nodes in our simulation sensor networks were divided into groups of maximum 4 sensor nodes for the multi-sensor data fusion. As described in Section IV, an application can decide a synchronization preference, such as the minimum number of samples (N), the maximum waiting time for samples when the number of samples is not enough(W), and the maximum tolerance in sampling time gap among the samples to be integrated (D). For the experiments, we decided  $D=45sec.$ ,  $W=5sec.$ , and  $N=3$ . During each minute, each sensor collected 1~2 new radar samples in a particular scan mode, and each sample size was about 1MB. We set the execution time for preprocessing on a sample to 1 second. The processing times of the second step are exponentially distributed with 3 seconds mean, and we set the maximum and minimum processing time of the second step to 1 second and 12 seconds respectively. The metric we used for evaluating the performance of the BPS was response time. The response time is defined as the duration from the start of processing to the end of processing including preprocessing and second step processing.

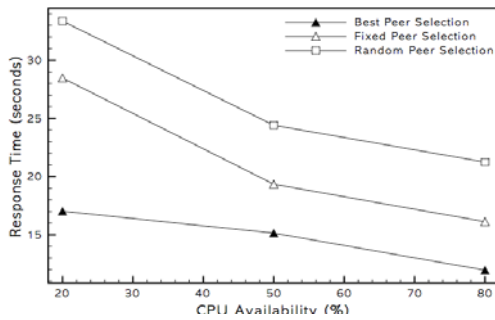
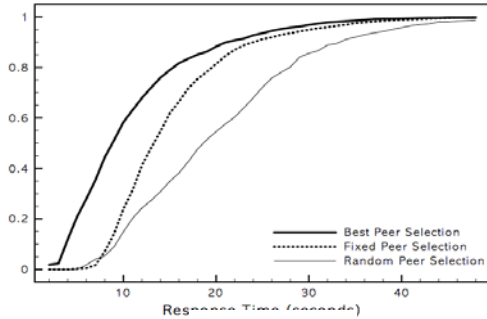
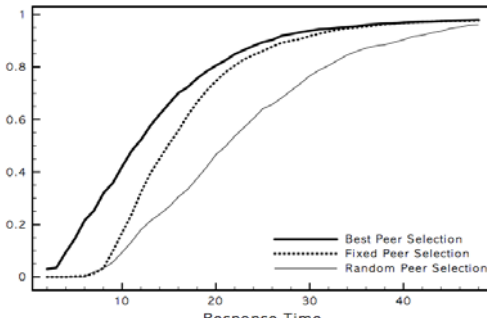


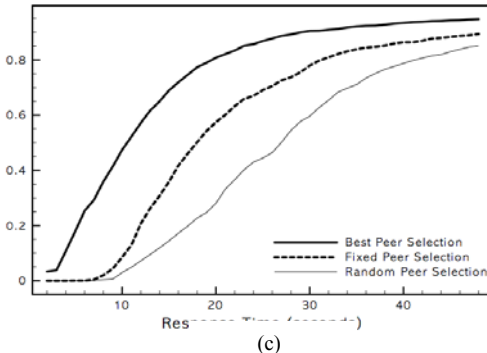
Figure 6. Average Response Time under different CPU availability.



(a)



(b)



(c)

Figure 7. CDF of response times for (a) 30%, (b) 50%, and (c) 20% CPU availability

Thus, low response time represents an improved performance. For the comparison, we also implemented two common heuristic algorithms: random and fixed peer selection algorithms. In the random algorithm, a set of peers was randomly chosen for the processing without considering the resources or contents availability at the peers. In the fixed

algorithm, the peers who generated particular samples were always selected for processing the samples.

In the first set of experiments, we studied the performance of the BPS algorithm when the CPU availability was varied across the peers. In the second set of experiments, we investigated the BPS algorithm's ability to accomplish better performance in the presence of variable cross traffic in the sensor network.

#### A. System Load Variation

For the first set of experiments, we created a network of 12 sensor nodes. In order to simulate the CPU availability variation among the peers, we selected half of the sensors randomly, and set CPU availability to average 20%, 50% and 80% on the selected peers. The other half of the peers were assigned average 95% of CPU availability. The 50% CPU availability means that the peer can deliver only the half of its computing power to the multi-sensor data fusion algorithm. The end-to-end network bandwidth between any two sensor nodes is set to 100Mbps, thus the communication overhead for the processing is small in this set of experiments.

Fig. 6 and Fig. 7 show the simulation results for the experiments. As seen in Fig. 6, the BPS shows better results for all the conditions. The reason is that BPS selects the peers to assign the processing items considering the peers' computing time history for the same type processing but random and fixed do not. In Fig. 7, we plot the cumulative distribution function (CDF) of the response times of multi-sensor data fusion algorithm. As seen in the figures, even the selected peers is highly loaded, almost 80% of multi-sensor data fusion algorithm response times are less than 20 seconds.

#### B. Network Load Variation

We investigated a sensor network with 20 sensor nodes located at the leaf nodes of the network topology. We used the GT-ITM [9] to create a 100-node transit-stub graph as our underlying network topology. The bandwidths on the network links were assigned randomly as 1Gbps, 100Mbps, 10Mbps, and 4Mbps, and the network latency were set randomly to 0.5ms ~ 5ms. To simulate dynamic network conditions, we located network load generators in our sensor network. The load generators injected cross-traffic on shared links. We varied the ratio of the average cross-traffic ( $\mu$ ) to physical bandwidth from 0.1 to 0.9 to investigate the effect of network congestion on the response time. Plentiful evidence suggests that the network traffic being extremely variable and bursty [11,12]. We took the burstiness into account by generating the cross-traffic in a type of self-similar process known as *Fractional Gaussian Noise* (FGN) [12]. The burstiness of network traffic is usually characterized by a self-similarity parameter (Hurst Parameter, H). For the simulations, we fixed  $H = 0.79$ , and the output variance =  $0.3 \cdot \mu$ . Fig. 8 shows the average response time achieved by BPS, fixed, and random under different amount of cross-traffic. The results show that BPS outperformed the other two heuristics under all network conditions.

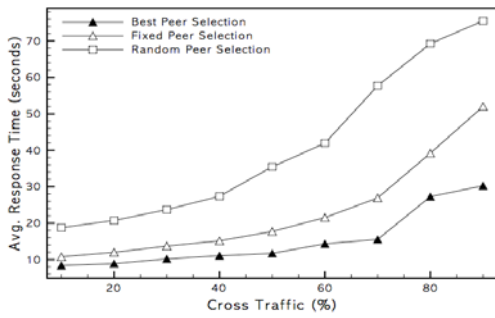
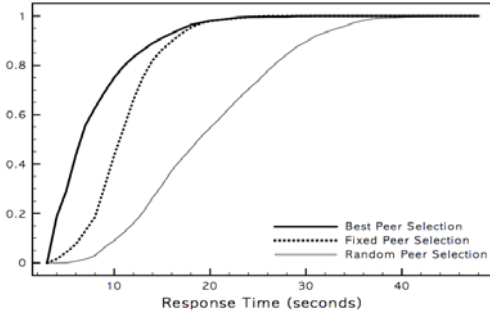
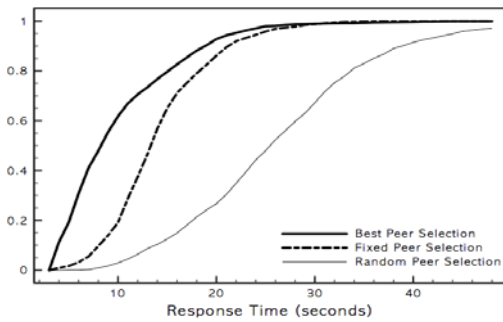


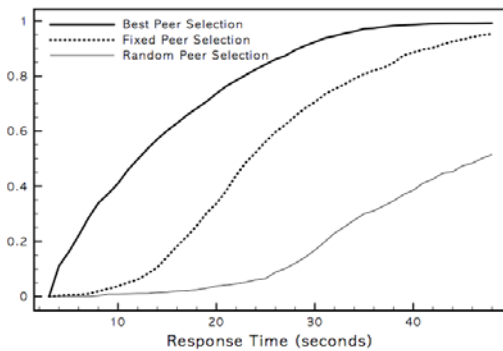
Figure 8. Average Response Time under different cross-traffic.



(a)



(b)



(c)

Figure 9. CDF of response time under (a) 10%, (b) 40%, and (c) 90% network cross-traffic

This is because in the BPS a node always chooses those peers which can provide the desired contents with the minimum cost. While collaborating on a multi-sensor data fusion algorithm, a sample from a bandwidth-poor peer can be replicated by a

bandwidth-rich peer. Once the replication is done, other peers can retrieve a copy of the desired contents from the bandwidth-rich peer in the BPS instead of the bandwidth-poor source node. Fig. 9 plots the CDF of response time that BPS, fixed and random heuristics obtain. Compared to the other heuristics, the majority of the response times are smaller than that of the other heuristics.

## VII. CONCLUSION

In this paper, we proposed a P2P collaboration framework for multi-sensor data fusion in resource-rich radar sensor networks. We proposed a simple data synchronization mechanism and peer selection scheme to coordinate peers for multi-sensor data fusion applications. We have implemented a simulation and our initial simulation results illustrated the effectiveness of the proposed framework and heuristic algorithm. P2P networks could provide an effective framework for the deployment of such real-time multi-sensor data fusion services. Our future plans include implementing a prototype of our framework and test it in a real test-bed environment. We will also investigate how to include other desirable feature such as fault tolerance into our framework.

## REFERENCES

- [1] McLaughlin, D.J., Chandrasekar, V., Droegemeier, K., Frasier, S., Kurose, J., Junyent, F., Philips, B., Cruz-Pol, S., and Colom, J. "Distributed Collaborative Adaptive Sensing (DCAS) for Improved Detection, Understanding, and Prediction of Atmospheric Hazards," in Proc. of AMS IIPS for Meteorology, Oceanography, and Hydrology, American Meteorological Society (AMS), 11.3, Jan 2005.
- [2] Lim, S., Chandrasekar, V., Lee, P., Jayasumana, A. P., "Reflectivity Retrieval in a Networked Radar Environment: Demonstration from the CASA IP-1 Radar Network," Proc. of IGARSS07, Barcelona, Spain. Jul. 2007
- [3] Donovan, B., McLaughlin, D., Kurose, J. V. Chandrasekar "Principles and Design Considerations for Short-Range Energy Balanced Radar Networks", Proc. of IGARSS05, Seoul, Korea, pp. 2058-2061, Jul. 2005.
- [4] Zink, M., Westbrook, D., Abdallah, S., Horling, B., Lyons, E., Lakamraju, V., Manfredi, V., Kurose, J. and Hondl, K. "Meteorological Command and Control: An End-to-end Architecture for a Hazardous Weather Detection Sensor Network," ACM Mobisys Workshop on End-end Sense-and-response Systems, pp. 37-42, Jun. 2005.
- [5] Brotzge, K., Brewster, B., Johnson, B., Philips, M. Preston, D. Westbrook and M. Zink, "CASA'S First Test Bed: Integrative Project #1," AMS 32nd Conf. Radar Meteor., Albuquerque, NM., 2005
- [6] BitTorrent. <http://www.bittorrent.com/>
- [7] Gnutella. <http://gnutella.wego.com>
- [8] Napster. <http://www.napster.com>
- [9] Calvert L. K., Doar, M., Zegura, E., "Modeling Internet Topology", IEEE Communications Magazine, vol. 65, no. 6, Jun. 1997. pp. 160-163
- [10] Legrand, A., Marchal, L., Casanova, H., "Scheduling Distributed Applications: the SimGrid Simulation Framework," Proc. of CCGrid 2003. pp. 138-145, May 2003
- [11] Leland, W.E. et al., "On the Self-Similar Nature of Ethernet Traffic" IEEE/ACM Trans. Networking, vol. 2, no. 1, 1994, pp. 1-15.
- [12] Paxson, V. "Fast, Approximate Synthesis of Fractional Gaussian Noise for Generating Self-similar Network Traffic," Computer Communication Review, vol. 27, pp. 5-18, Oct. 1997.