2017

VIRTUALIZATION OF CLOSED-LOOP SENSOR NETWORKS

Priyanka Dattatri Kedalagudde
University of Massachusetts Amherst

VIRTUALIZATION OF CLOSED-LOOP SENSOR NETWORKS

A Thesis Presented

by

PRIYANKA DATTATRI KEDALAGUDDE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

May 2017

Electrical and Computer Engineering

VIRTUALIZATION OF CLOSED-LOOP SENSOR NETWORKS

A Thesis Presented

by

PRIYANKA DATTATRI KEDALAGUDDE

Approved as to style and content by:

Michael Zink, Chair

David Irwin, Member

Lixin Gao, Member

C.V.Hollot, Department Head
Electrical and Computer Engineering

ACKNOWLEDGEMENT

I would like to express the deepest appreciation to my advisor, Professor Michael Zink, who has the attitude and the substance of a genius and who inspired my interest in my work. Without his guidance and persistent help this thesis would not have been possible. He always encourages me and inspires me with lots of valuable insights and ideas. I thank Eric Lyons for providing support to understand the CASA's radar system and its functionalities,test beds and helping with generating traces from these systems. I am heartily thankful to Professor David Irwin and Professor Lixin Gao for their constructive advice and invaluable help on both of my research and future career.

Finally, I appreciate all of the sincere support from my family and my friends, this thesis pales in comparison to what I gained from them.

ABSTRACT

VIRTUALIZATION OF CLOSED-LOOP SENSOR NETWORKS

MAY 2017

PRIYANKA DATTATRI KEDALAGUDDE

B.E, NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY,

BANGALORE,INDIA

M.S.E.C.E, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Michael Zink

The existing closed-loop sensor networks are based on architectures that are designed and implemented for one specific application and require dedicated sensing and computational resources. This prevents the sharing of these networks. In this work, we propose an architecture of virtualization to allow sharing of closed-loop sensor networks. We also propose a scheduling approach that will manage requests from competing applications and evaluate their impact on system utilization against utilization achieved by more traditional, dedicated sensor networks. These algorithms are evaluated through trace-driven simulations, where the trace data is taken from CASA's closed-loop weather radar sensor network. Results from this evaluation show that the proposed scheduling algorithms applied in a shared network result in cost savings, that are the result of being able to multiplex applications onto a single network as opposed to running each application on an dedicated sensor network.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Closed-loop sensor networks represent a sub-class of Cyber-Physical Systems (CPS) [28]. They have the potential to save lives and property from natural disasters and increase national security through new critical infrastructure. In closed-loop sensor networks, the sensing nodes can be actuated remotely and often provide substantial computational and storage capabilities. The data generated by these sensors are subsequently analyzed and then used to determine future actuation of the sensors. Such Dynamic Data-Driven Application Systems (DDDAS) [1, 6] are distinct from regular sensor networks not only by using actuation to perform the sensing but also by the need for a control unit that determines future sensor actuation.

Cost projections for the deployment and operation of such infrastructures are significant (to the order of several billion dollars). Existing closed-loop sensor deployments are based on architectures which are dedicated for only one type of application where sensing resources cannot be shared. A solution to allow the sharing of these sensor networks is of great value since it would allow other applications to use existing infrastructure without redundancy. The sharing of closed-loop sensor networks will thus result in significant reduction of both capital and operational expenditure. To date, no instances of such isolated but fully shared closed-loop sensor networks have been created. One of the major reasons is the general fear that resource sharing might lead to interference between application request, potentially resulting in loss of critical information.

This work makes the following contributions in the area of sensor network vitualization:

- **Architecture:** We propose a virtualization layer as part of this architecture. While this architecture also includes the virtualization of networking and computation components of sensor networks, this work focusses exclusively on the virtualization layer for sensor networks.
- **Scheduling:** As part of the virtualization layer, we propose different scheduling approaches that uses sensor network characteristics like utility of the execution of a sensing task and potential task overlap.
- **Trace-based Evaluation:** We analyze the performance of this architecture based on traces obtained from an CASA's weather sensing sensor network. We use a utility metric that has been specifically developed for this sensor network in our evaluations.

The remainder of this document consists of following chapters. Chapter 2 consists of the related work done in this area. Chapter 3 presents the architecture for the virtualization layer followed by a discussion of the proposed scheduling approaches in the virtualization layer. The experiments and results are explained in Chapter 4, Chapter 5 concludes this work with a discussion about the possible advantages of virtualizating sensor networks.

CHAPTER 2
BACKGROUND AND RELATED WORK

Short-wavelength weather radars networks [18] has the potential to replace the existing radar infrastructures (NEXRAD [19]). CASA has built one such closed-loop sensor network [29]. This system is an example of dedicated non-shared sensor network. However, recent work [12] has shown that the such a system which scans the weather data could also be used to track low lying aircraft. This can be done with the introduction of vitualization layer in the sensor networks. Until now, the focus of the research in sensor network virtualization has been either on the creation of virtual operating systems for sensor nodes ([8, 13, 14]) or the virtualization of the networks that connect the sensor nodes ([3, 16, 15]). Existing approaches for the first case have the aim to use virtualization to abstract from the sensor hardware to simplify application development, but do not address the sharing of resources, and are developed for small sensor nodes (e.g., Atmel or Mica). Bose et al. ([5, 4]) were the first ones to propose a service-oriented sensor network architecture that is based on sensor virtualization. In their approach, sensor virtualization is limited to the fact that a virtual sensor abstracts a set of passive, physical sensors. This concept has been extended by Pumpichet and Pissinou [25] for the case of mobile sensors. Under a similar concept proposed by Pajic and Mangharam, an Embedded Virtual Machine (EVM) programming abstraction [20] has been developed, that allows the composition of a virtual machine across physical nodes. This is different to the approach presented in this paper. First, in our approach, VMs do not span physical nodes but in combination build a virtual slice (as shown in Figure 1). Second, in [20] a physical node can only be part of a single VM making isolated sharing of the sensor between users impossible. This sharing between users (or applications) is one (if not the most prominent) goal of our work. Similar to our approach the SATWARE [17] project provides a middleware for sensor networks. But SATWARE does not allow the isolated sharing of sensors, nor has it been applied for actuation. I.e., the sensors cannot be steered as it is the case for radars or other actuators.

Our approach is different in the way that it will allow the simultaneous sharing of actuable sensor nodes among several users with an emphasis on sensor networks that operate in closed-loop mode. In addition, our approach can combine sensor virtualization with computational virtualization to build end-to-end architectures that enable Sensing as a Service. Existing approaches that focus on network virtualization can be integrated into our virtualization architectures, if necessary. Initially, the existing technologies like TDMA(time division multiple access) can be integrated in virualization of actuable sensors. Work in [27] shows that even slight reordering of user requests for sensing resources can improve the overall efficiency of a virtualized sensor netwrok. So in our approach to integrating TDMA technologies, we also reorder the requests and evaluate the performance of the virtualized sensor networks.

We looked at the existing Disk scheduling algorithms in [10] which try to minimize the latency of seeking disk requests. The FCFS performs operations in order requested without any reordering. This does not provide fastest seek but does not starve the requests. The Shortest Seek Time First (SSTF) reduces the seek time by selecting the disk I/O request that requires the least movement of the disk access arm from its current position. Our scheduling approach is different from the existing ones since we make use of sensor specific properties to make scheduling decisions that will maximize overlap and data sharing between requests.

CHAPTER 3
SYSTEM ARCHITECTURE

The closed-loop architectures built by CASA like the DCAS (Distributed Collaborative Adaptive Sensing) systems [21, 7] have the potential to replace the existing sit-and-spin architecture (NEXRAD). For example, NEXRAD is a sit-and-spin scan system that performs 360 sweeps of the lowest 2 elevations with every heartbeat to sense the their entire sorrounding volume. But the DCAS systems performs targeted sector scanning using optimization algorithms that determine the best sectors to scan that would result in sensing their sorrounding area having only high volume of interested weather data. However, these architectures do not allow different islolated applications to share a network. For example, recent work [12] has shown that the DCAS systems which scans the weather data could also be used to track low lying aircraft. With the present architecture, a sensor network is dedicated only to one application at a given point of time. Also, each of these radar sensor nodes have very high infrastructural and operational costs. The goal of this work is to create a virtualization layer which will enable two isolated applications to be shared on a single sensor network, while reducing replicated infrastructure costs of the network.

The chapter is broadly divided into two sections. The first section deals with the architecture of a sensor virtualization layer and the second section explains the various scheduling approaches.

3.1 Sensor Virtualization Layer

Virtualization in information technology is a software that enables isolated applications to share the use of computer hardware through virtual machines [26, 2, 9]. The existing technologies like TDMA(time division multiple access) are integrated in virualization of actuable sensors. Virtualizing sensor networks can be challenging when two different users might want to actuate their virtualized radar to a different position at the same instant in time. The work presented in this chapter investigates various approaches that offers users to share the sensor networks.

We propose the architecture as illustrated in Figure 3.1 for the virtualization layer. It consists of three major components:

- **Request Manager:** This component creates a request block with a request id for the incoming application's request and submits the request block to the request queue that can be accessed by scheduler component as well. The request block consists of request id, application name, sensors requested, scan angles requested, deadline of the task and utility threshold.

- **Scheduler:** It consists of a class of utility-driven scheduling algorithms that admit or deny tasks from different applications. The goal is to ensure maximum sharing between user requests with maximum utility and minimum usage of sensor resources.

- **Data Manager:** This has two sub components, a sensor registry module that maintains a set of sensors that is up and operational. It periodically checks for the liveness of the sensor. The other sub component maintains the list of merged/shared data and their corresponding request id. When the data is transmitted to the respective application, this component transfers the appropriate data according to the mapping.

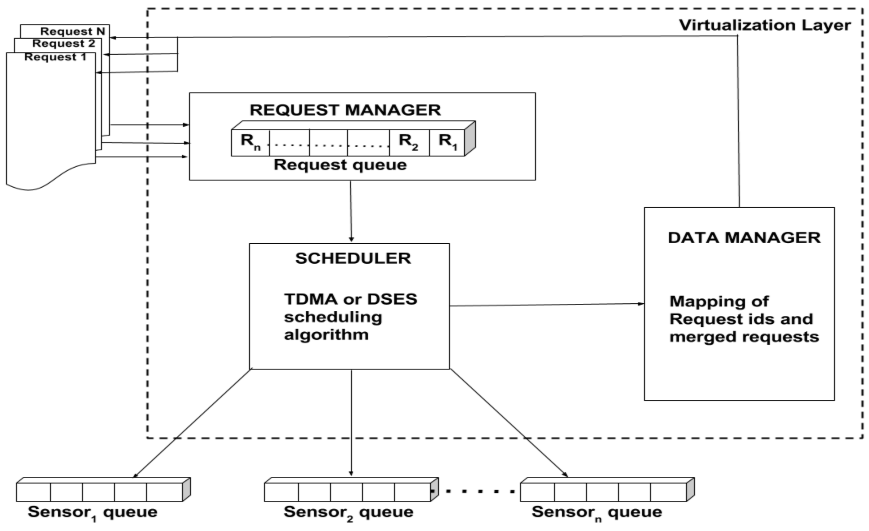


Figure 3.1: Virtualization Framework

3.1.0.1 Utility Functions

Utility functions are a measure of the quality and quantity of data that is of value for a specific user. In [24, 22, 23], the development and implementation of such utility functions in the radar network is illustrated. We have created a simple utility function based on the angle of sectors requested for the scan. We feed the input commands consisting of scan angle along with the other radar parameters to the utility function. The obtained utility value is called the expected utility. We have developed a scheduling algorithm that will determine the amount of data overlap and the actual scan performed. The new utility is calculated based on the actual scans performed. This is called actual utility. With the expected and actual utility values, we can evaluate the performance of different scheduling approaches. The utility function is described in Section 3.2

3.2 Scheduling Approaches

In this section, we introduce a set of scheduling approaches. We use the data set from CASA Integrative Project 1 (IP1) radar sensor network for our evaluation. IP1 is the first test bed deployed by CASA demonstrating the capabilities of a DCAS system. It is a network consisting of four nodes that can be mechanically steered. The four radars are located in Chickasha, Rush Springs, Lawton, and Cyril. Each radar is separated by about 25 km and have a range of about 30 km. In IP1, a set of commands is sent to each radar for steering once with every 60 seconds system heartbeat. For the first 20 seconds of each 60 seconds heartbeat, each radar does a single 360 degree surveillance sweep at the lowest elevation angle to provide a general sense of the weather in the network. The remaining 40 seconds is used to execute scans from the issued commands.

We first investigate if the TDMA access control method can be applied to virtualized sensors/actuators. To investigate the feasibility of this approach, we start by testing different scheduling approaches for multiplexing applications. The advantage of TDMA approaches is the fact that it is easy to implement and allows for a straightforward implementation in existing sensor networks.

3.2.1 TDMA Scheduling Approaches

In this section, we propose a set of dynamic TDMA scheduling approaches in which the time period T is fixed but the slot length can be dynamic. Slot length is defined as the duration for whcih request is executing within a given time period. For the sake of simplicity, we use a sample data set to explain each of the TDMA scheduling approach in this section. However, in the actual experiments we use the data set generated from CASA's test bed, IP1 described in Section 3.2. This test bed has the following system configurations. It takes 60 seconds to perform a scan (one heartbeat).The IP1 radar sensor network is required to always perform a low level surveillance scan in the first 20 seconds of the 60 second heartbeat. In the remaining 40 seconds, targetted scans can be executed. So we consider the time period to be T = 40s. The general approach is to group requests based on certain criteria and schedule tasks that can be executed within the available slot period, T. Then the total utility for an application a_i for a particular scheduling approach is represented by the following equation:

$$u(a_k) = \sum_{i=1}^{n} u_i(a_{k_{hit}}) - \sum_{j=1}^{n} u_j(a_{k_{miss}}), i \neq j, \forall a_k \in A$$

where n represents the number of slots,
$u_i(a_{k_{hit}})$ is the utility of the application in slots it was granted permission to use, and
$u_j(a_{k_{miss}})$ is the utility of the application in slots that it was denied use of.

Slot Number	Application	Angle of scan	Utility	Request ratio
9	B	180	1.3	1
10	B	180	1.7	1
10	C	60	1.8	0.33
11	A	120	1.3	0.25
11	B	180	1.7	0.6
11	C	60	1.8	0.33
12	A	120	1.3	0.28
13	C	60	1.8	0.375

Table 3.1: Sample data set for TDMA approaches

3.2.2 TDMA1 - Dynamic TDMA with scan angle dependent slot sharing

The time taken to execute a slot is directly dependent on the scan angle being requested. For the purposes of this paper, we work on the assumption that it takes 60 seconds to perform a 360° scan. This can vary with each sensor installation, but it is possible to incorporate the change into our approach.

This approach is useful where we want to first schedule all requests that takes shorter time to execute and prevent them from waiting for long executing requests to complete. We can illustrate this scheduling behavior with the example shown in Table 3.1 where three application issue requests for slot 11. First, we group the

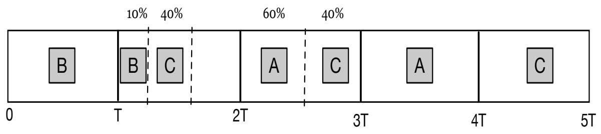


Figure 3.2: Dynamic TDMA with scan angle dependent slot sharing

requested tasks in the increasing order of their scan angle. Lower the sweep angle, the lesser time it takes to execute the scan. Task C's scan angle translates to $(1/4) * T$ of the slot length and task A's to $(1/2) * T$ of the total slot length, respectively, which results in B not getting a share of the slot. Subsequently, in Figure 3.2, slot 2T-3T is shared by A (utilizing 50%) and C (utilizing 25%). The corresponding utility for each application is calculated as follows:

$$\begin{aligned} u_{A_{TDMA1}} =& u_3(A) + u_4(A) \\ u_{B_{TDMA1}} =& u_1(B) + u_2(B) - u_3(B) \\ u_{C_{TDMA1}} =& u_2(C) + u_3(C) + u_5(C) \end{aligned}$$

3.2.3 TDMA2 - Dynamic TDMA with decreasing request ratio dependent slot sharing

In this approach, the requesting tasks in a slot are grouped based on the request ratio of an application. The request ratio is defined as number of requests the application has made to the total number of request raised by all applications till that instant in time. The priority of the applications to be scheduled dynamically changes with changing request ratio every slot. This approach is useful in scenarios where applications monitoring a storm need to be serviced first since they continually require the data. In this case, we need to assign higher priority to the applications that makes more requests. In slot 11 of Table 3.1, the tasks A, B, and C are requested at the same time and are grouped in the decreasing order of the request ratio (not shown in the table and calculated from the traces). We schedule B and C since they have higher request ratios than A (see Figure 3.3). The overall utility for each application in this case is:

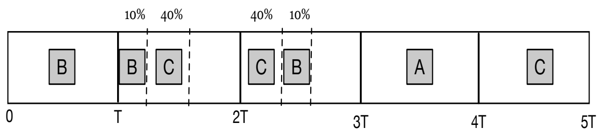


Figure 3.3: Dynamic TDMA with decreasing request ratio dependent slot sharing

$$u_{A_{TDMA3-1}} = u_4(A) - u_3(A)$$
$$u_{B_{TDMA3-1}} = u_1(B) + u_2(B) + u_3(B)$$
$$u_{C_{TDMA3-1}} = u_2(C) + u_3(C) + u_5(C)$$

3.2.4 TDMA3 - Dynamic TDMA with increasing hit ratio dependent slot sharing

In this approach, the requesting tasks in a slot are grouped based on the hit ratio of an application. The hit ratio is defined as number of requests for an application that is executed to the total number of request raised by that applications till that instant in time. It is based on giving a higher priority to requests with low hit ratio in the aim of providing fairness to all applications in a balanced manner. Illustrating this with the same example as the previous section, the tasks are grouped in the increasing order of the hit ratio. A, B, and C from slot 11 have hit ratios that amount to 0, 1, and 1, respectively. A and C are scheduled and executed within the given slot length with this scheduling approach (see Figure 3.4).

$$u_{A_{TDMA3-2}} = u_4(A) + u_3(A)$$
$$u_{B_{TDMA3-2}} = u_1(B) + u_2(B) - u_3(B)$$
$$u_{C_{TDMA3-2}} = u_2(C) + u_3(C) + u_5(C)$$

This approach tries to ensures that no application is indefinitely starved of execution by prioritizing applications based on their instantaneous hit ratios. Two

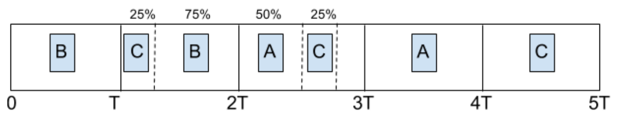


Figure 3.4: Dynamic TDMA with increasing hit ratio dependent slot sharing

applications for example, one to track aircraft and other to detect tornadoes, perform important functions, that need to be executed at fairly regular intervals with strict deadlines. The priority based dynamic approach strives to ensure that application starvation does not occur in the presence of multiple, high priority applications.

Applications	Token count
A	3
B	2
C	1

Table 3.2: Token distribution every six slots for TDMA4

3.2.5 Token based approach

A token bucket based approach can also be adopted to perform admission control. In this static priority based approach, tokens are periodically assigned to certain applications. For example, Table 3.2 illustrates the case where a batch of six tokens is created every six time slots. The distribution of tokens to individual applications is determined based on the overall number of requests for that application. In the aforementioned table, application A is assigned the highest number of tokens since it has the highest ratio of requests and conversely C is assigned the smallest number of tokens. Based on the data shown in Table 3.3, the first three requests from A will be scheduled but only two requests from B. Requests from C are not scheduled for execution since no requests are made in slots 14 and 15 and, thus, the last slot is unused.

Slot Number	Application	Utility	Remaining tokens
10	A	1.4	2
10	B	1.7	2
11	C	1.2	1
11	A	1.4	1
12	A	1.4	0
13	C	1.4	1
13	B	1.7	1
14	A	1.4	0
14	B	1.7	0
15	B	1.7	0
16	A	1.4	0

Table 3.3: Sample data set for TDMA4

While there is no request from application C in this specific case, this approach has the advantage that applications that rarely generate requests do not starve, since the distribution of tokens will determine their priority.

The downside of this token bucket approach is that slots might not be used, as just shown for the case of application C, and this slot could have been used by application B. At the end of the period (six slots in this specific example), all left over tokens are discarded and a new set of tokens are regenerated. We would like to point out that the decision to assign priorities based on overall request rates of the individual applications is experimental and that our approach allows any kind of prioritization. Such prioritization could either be determined by the sensor network operator or negotiated amongst the users of the network. The results for these TDMA approaches are presented in the Chapter 4.

3.2.6 Data-Sharing Enabled Scheduling (DSES)

For a more flexible approach, we propose and develop a scheduling algorithm that enables sharing of sensor data across isolated application requests. In the case of TDMA approaches, two requests from different tasks could actually be identical, or overlap, or could be executed with a slight delay, resulting in redundant sensing activity in consecutive time slots. This scheduling approach without data sharing was inefficient since it did not consider partially overlapping requests.

In sensor networks, the data generated from one application's sensing activity can prove to be of value to other applications. For example, consider one application that measures precise rainfall application and another that tracks severe thunderstorms. Since the thunderstorm also measures rainfall, the measurement data from thunderstorm application can be utilized by both applications.

The sensing requests from different applications may not match perfectly or may not have the same deadlines. In this eventuality, we still want to optimize the utilization of the network by maximizing the data sharing between overlapping requests. In the above example, the thunderstorm sensing application might scan the atmosphere at several elevations, while the rainfall measurement application might require scans only at the lowest elevation. Even though these requests do not align perfectly, the data from the thunderstorm scanner's lowest elevation scan is still useful for the other application. We try and extract this value provision for the application and express it through utility functions described in Section 3.1.

3.2.6.1 The DSES Algorithm

From the data sharing phenomenon discussed in the above section, we illustrate the operation of scheduling algorithm 1. As part of building a tunable framework, we provide the user with two parameters, a utility threshold and an execution deadline. They are discussed below:

- The utility threshold is the minimum quantity of data (utility) that needs to be present between two overlapping requests to share the network. For example, if a user specifies a utility threshold of 0.5 for a request, any potential overlap with another request (executing or otherwise) has to demonstrate a potential utility of atleast 0.5 for this to be considered a viable overlap.

- The execution deadline is the maximum time before which a request has to finish executing.

These two parameters are used to evaluate the feasibility of data sharing between requests by the scheduling algorithm.

Variables	Definition
s_1	Sensor 1
$R1$	Executing request requesting for s_1 at time t_1
$R2$	Incoming request requesting for s_1 at time t_2
$\theta_{s1,1}$	Initial scan angle of the executing request $R1$
$\theta_{s1,2}$	Final scan angle of the executing request $R1$
$\phi_{s1,1}$	Initial scan angle of the incoming request $R2$
$\phi_{s1,2}$	Final scan angle of the incoming request $R2$
$t_{deadline}$	Deadline of the incoming task $R2$
$t_{exectime}$	Execution time of the executing task $R1$
$R2_{overlap}$	Percentage of $R2$ constituting the overlapping set
$R2_{nonoverlap}$	Percentage of $R2$ constituting the non-overlapping set
$Data_{overlap}$	Amount of data overlap between the incoming $R2$ and executing request $R1$
$Data_{nonoverlap}$	Unsharable data from the incoming request $R2$

Table 3.4: Scheduling Algorithm Conventions

Algorithm 1: DSES Algorithm

```
When a request $r$ arrives at sensor $s_i$;
if no requests executing on $s_i$ then
    Execute $r$;
else
    for every scheduled request $j$ in s_i's queue do
        if $j$ has data overlap with $r$ then
            $U \leftarrow ((R2_{overlap} * Data_{overlap}) + (R2_{nonoverlap} * Data_{nonoverlap}))$;
            if $t^{j}_{exectime} \leq t^{r}_{deadline}$ then
                if $U \geq U\_threshold$ then
                    $r \leftarrow$ data from $j$;
                else
                    Delay_task_execution($r$);
            else
                Delay_task_execution($r$);
        else
            Delay_task_execution($r$);
```

Algorithm 2: Delay function

```
Function Delay_task_execution(r)
    if $t^{r}_{delayed\_execution} \geq t^{r}_{deadline}$ then
        if $U_{delayed\_task} \geq U_{threshold}$ then
            Execute $r$;
        else
            Cannot execute task $r$;
    else
        Cannot execute task $r$;
```

The conventions used in our illustration of the scheduling approach are cited in Table 3.4.

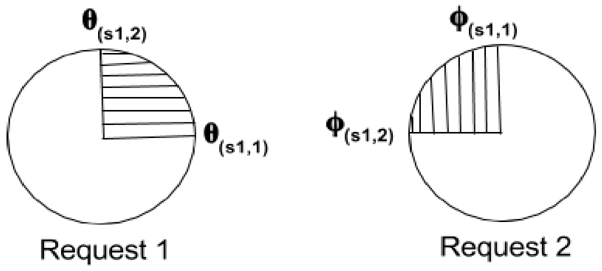


Figure 3.5: No overlap

3.2.6.2 Construction of Overlapping and Non-overlapping Sets

Before contructing the overlapping and non-overlapping sets, the incoming request should satisfy the following criteria.

$$\phi_{s1,2} >= \phi_{s1,1} \wedge \theta_{s1,2} >= \theta_{s1,2}$$

If the above condition is not satisfied, then 360° is added to the lesser scan angle.

$$\text{If } \phi_{s1,2} < \phi_{s1,1}, \text{ then } \phi_{s1,2} = \phi_{s1,2} + 360^\circ$$
$$\text{If } \theta_{s1,2} < \theta_{s1,1}, \text{ then } \theta_{s1,2} = \theta_{s1,2} + 360^\circ$$

The overlapping and non-overlapping sets are constructed as explained below.

3.2.6.3 Case of Non-Overlapping Requests

If there is no scan angle overlap between the two requests $R1$ and $R2$, the execution of $R2$ is delayed, provided the scheduler can guarantee that $R2$ can finish executing before its execution deadline runs out.

$$\phi_{s1,2} <= \theta_{s1,1} \vee \phi_{s1,1} <= \theta_{s1,2}$$

3.2.6.4 Case of Overlapping Requests

If $R1$ and $R2$ are two overlapping requests, then the angle of scan can be divided into an overlapping and non-overlapping set. The construction of these sets for different cases is described below.

- *Case 1: Complete Overlaps* - When the scan angle of the incoming request is a complete subset of an already executing request for a given sensor, the data can shared by both the requests. Since $R1$ is already under execution, this goes with the caveat that scan angle of request $R2$ should be a complete subset of $R1$'s remaining scan. This is shown in Figure 3.6.

  This can be represented by the logical condition as below:

$$\phi_{s1,1} \geq \theta_{s1,1} \wedge \phi_{s1,2} \leq \theta_{s1,2}$$

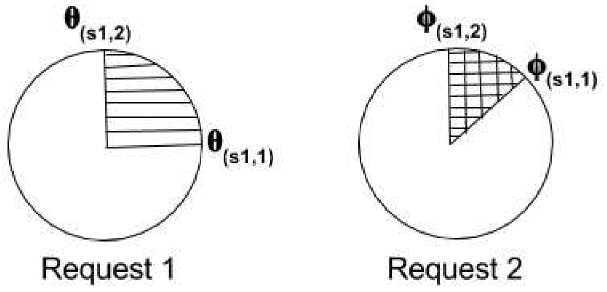


Figure 3.6: Complete Overlap

- *Case 2: Partial Overlaps* - This section deals with the case of partial overlaps. If the incoming request is not a complete subset of an executing request, we can evaluate the viability of data sharing (and also simplify scheduling, if viable), by evaluating the size of the overlapping subset and the utility it provides to the application.

  Once this has been accomplished, data from the overlapping set can be be shared between the two requests and the non-overlapping set can be scheduled for execution later.

  - *Partial Overlap Scenario 1:* We can construct an overlapping set as shown in Figure 3.7. For example, if $(\theta_{s1,1}, \theta_{s1,2}) = (45°, 90°)$ is the scan range of $R1$ and $(\phi_{s1,1}, \phi_{s1,2}) = (0°, 90°)$ is the scan range of $R2$, then (45°,90°)

represents the overlapping set common to both requests and (0°,45°) represents the non-overlapping set specific to $R2$.

$$\phi_{s1,1} < \theta_{s1,1} \wedge \phi_{s1,2} \leq \theta_{s1,2} \wedge \phi_{s1,2} > \theta_{s1,1}$$

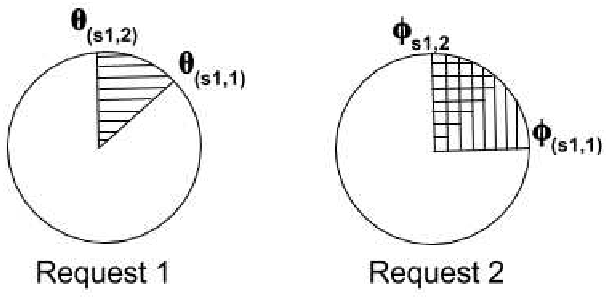


Figure 3.7: Partial Overlap Scenario 1

– *Partial Overlap Scenario 2:* This scenario is shown in Figure 3.8. For example, if $(\theta_{s1,1}, \theta_{s1,2}) = (0°, 90°)$ is the scan range of $R1$ and $(\phi_{s1,1}, \phi_{s1,2}) = (45°, 135°)$ is the scan range of $R2$, then $(45°, 90°)$ represents the overlapping set common to both requests and $(90°, 135°)$ represents the non overlapping set specific to $R2$.

$$\phi_{s1,1} \geq \theta_{s1,1} \wedge \phi_{s1,2} > \theta_{s1,2} \wedge \phi_{s1,1} < \theta_{s1,2}$$

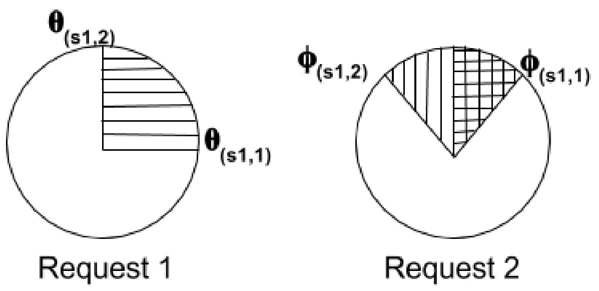


Figure 3.8: Partial Overlap Scenario 2

– *Partial Overlap Scenario 3:* This scenario is shown in Figure 3.9. For example, if $(\theta_{s1,1}, \theta_{s1,2}) = (45°, 90°)$ is the scan range of $R1$ and $(\phi_{s1,1}, \phi_{s1,2}) =$

(0°,135°) is the scan range of $R2$, then (0°,45°) and (90°,135°) of the scan set represents the non overlapping set specific to $R2$ and (45°,90°) represents the overlapping common to both requests.

$$\phi_{s1,1} < \theta_{s1,1} \wedge \phi_{s1,2} > \theta_{s1,2}$$

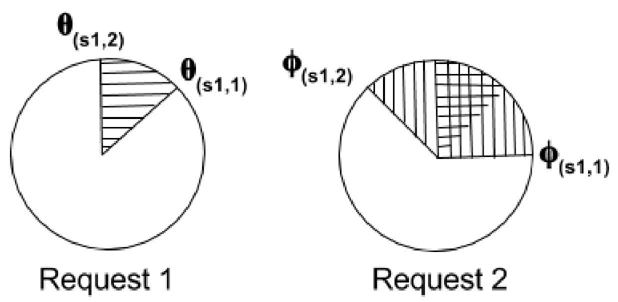


Figure 3.9: Partial Overlap Scenario 3

3.2.7 Calculation the Percentage of Potential Data Overlap

In this section, we calculate the probable quantity of the overlapping data $Data_{overlap}$. For example, if it takes 60 seconds to scan 360 degrees then it takes one sixth of a second to scan a degree. If $R1$ arrives at time t_1 and $R2$ arrives at time t_2, then the portion of $R1's$ scan that would have completed in $t_2 - t_1$ seconds is, $(6 * (t_2 - t_1))$ degrees. When $R2$ is submitted, $R1$ would have completed

$$\phi'_{s1,1} = \theta_{s1,1} + (6 * (t_2 - t_1))^\circ \tag{3.1}$$

where, $\theta_{s1,1}$ is the initial scan angle of the executing request

The $Data_{overlap}$ equations for different cases of overlaps and non-overlaps is shown in Table 3.5.

3.2.8 Calculation of Total Utility

Total utility for the $R2$ is given by,

$$U = ((R2_{overlap} * Data_{overlap}) + (R2_{nonoverlap} * Data_{nonoverlap}))) * (1 - miss) \tag{3.2}$$

Type of Overlap	$Data_{overlap}$
No Overlap	0
Complete Overlap	$((\phi_{s1,2} - \phi'_{s1,1})/(\phi_{s1,2} - \phi_{s1,1}))$
Partial Overlap scenario 1	$((\phi_{s1,2} - \phi'_{s1,1})/(\phi_{s1,2} - \theta_{s1,1}))$
Partial Overlap Scenario 2	$((\theta_{s1,2} - \phi'_{s1,1})/(\theta_{s1,2} - \phi_{s1,1}))$
Partial Overlap Scenario 3	$((\theta_{s1,2} - \phi'_{s1,1})/(\theta_{s1,2} - \theta_{s1,1}))$

Table 3.5: $Data_{overlap}$ for Overlapping and Non-Overlapping requests

where,

$$Data_{nonoverlap} = \begin{cases} utility_{nonoverlap}, & \text{if } t\_deadline \geq t\_exectime \\ 0, & \text{otherwise} \end{cases}$$

utility_nonoverlap is calculated by finding the utility of delayed non-overlapping set which is described in Section 3.2.9.

$$miss = \begin{cases} 0, & \text{if } task_{hit} \\ 1, & \text{otherwise} \end{cases}$$

task_hit is a parameter representing that the task has been scheduled.

$R2_{overlap} * Data\_overlap$ is the percentage of quantity of data from the overlapping set.

$R2_{nonoverlap} * Data_{nonoverlap}$ is the percentage of quantity of data from the non-overlapping set

The utility for different cases of overlap and non-overlap is as follows:

- No overlap

$$R2_{overlap} = 0$$

$$R2_{nonoverlap} = 1$$

$$U = (R2_{nonoverlap} * Data_{nonoverlap}) * (1 - miss)$$

- Complete overlap

$$R2_{overlap} = 1$$
$$R2_{nonoverlap} = 0$$
$$U = (R2_{overlap} * Data_{overlap}) * (1 - miss)$$

- Partial overlap In the case of partial overlap, the utility equation becomes,

$$R2_{overlap} = x\% of (R2_{overlap} + R2_{nonoverlap})$$
$$R2_{nonoverlap} = y\% of (R2_{overlap} + R2_{nonoverlap})$$
$$U = ((R2_{overlap} * Data_{overlap}) + (R2_{nonoverlap} * Data_{nonoverlap})) * (1 - miss)$$

If *utility* is greater than or equal to the utility threshold, the overlapping set of $R2$ is shared with $R1$. If *utility* is lesser than the utility threshold, the execution of $R2$ is delayed as shown in Algorithm 2, if $R2$ can finish execution before its deadline. If not, $R2$ will not be executed. The total utility is calculated as follows,

$$U = utility * (1 - miss) \tag{3.3}$$

where,

$$miss = \begin{cases} 0, & \text{if } task_{hit} \\ 1, & \text{otherwise} \end{cases}$$

3.2.9 Utility of Delayed Tasks

The execution of a request is delayed for a non-overlapping set, or if there is no scan overlap or if the utility of overlapping data is below the utility threshold.

In either of these cases, if the execution of $R2$ is delayed by n seconds, the new initial scan of R2 is:

$$\phi'_{s1,1} = (\phi_{s1,1} + (6 * (n - t - 2))$$

The probable quantity of $R2's$ data when executed with a delay of n seconds is calculated as follows:

$$utility_{delayed} = (\phi_{s1,2} - \phi'_{s1,1})/(\phi_{s1,2} - \phi_{s1,1})$$

If $utility_{delayed}$ is greater than or equal to the utility threshold, $R2$ is executed after $nseconds$.

If $utility_{delayed}$ is lesser than the utility threshold, $R2$ cannot be scheduled for execution.

The total utility is calculated as follows,

$$U = utility_{delayed} * (1 - miss) \tag{3.4}$$

where,

$$miss = \begin{cases} 0, & \text{if } task\_hit \\ 1, & \text{otherwise} \end{cases}$$

$task_{hit}$ is a parameter representing that the task has been scheduled.

3.2.10 Augmented Data Sharing Enabled Scheduling (A-DSES)

In the interest of further optimization, we implement a windowing mechanism on top of the base DSES. This mechanism is is heartbeat based, wherein multiple requests arriving in a single heartbeat are batched and processed together for optimization. This also emulates the behaviour of a real time system.

To measure the data sharing in this approach, we have the following two parameters,

- Utility threshold - As discussed in Sectio 3.2.6.1
- Delay threshold - The maximum number of heartbeats for which a request can be postponed.

In experiments, we use the utility threshold as a tunable parameter and keep the delay threshold fixed to one heartbeat. This delay threshold is to ensure that a request is executed within two heartbeats. if the request is postponed by two heartbeat or more, they cannot be executed.

The A-DSES approach involves the following steps (the algorithm itself is presented in Section 3.2.10.5):

3.2.10.1 Batch Processing

The DSES algorithm executes incoming requests as they arrive and does not batch the requests per heartbeat. Thus, an incoming request that completely overlap, with the executing request will not result in 100% utility since the executing request has already started execution. To emulate the behaviour of a real system, we batch the requests from each heartbeat and then schedule the requests for execution.

3.2.10.2 Request Reordering

In the DSES algorithm, we reordered requests according to the priority of their requesting applications. In this approach, we reorder the batched requests in such a way that the number of overlaps between all the requests are maximized. Consider three requests R_1,R_2 and R_3 consisting of sub-requests requesting for sensors s_1,s_2,s_3 as shown in Table 3.6.

1. Each type of overlap is assigned a weight value based on the degree of overlap between requests as shown in Table 3.7. The weight values range from 0 to 1. A complete overlap with the highest degree of overlap will have the maximum weight of 1 and no overlap with a zero degree of overlap will have the minimum weight of 0. The partial overlap weight value will range from 0 to 1 depending on the degree of overlap.

2. A sub-request's corresponding weights are calculated after comparing it with each of the other sub-requests requesting access to the same sensors in a heartbeat. This is shown in Table 3.8.

3. The total weight for each of the request is calculated as shown in Table 3.9.

Requests	Sub-request (s_1)	Sub-request (s_2)	Sub-request (s_3)
R_1	$r_{1,1}$	$r_{1,2}$	$r_{1,3}$
R_2	$r_{2,1}$	$r_{2,2}$	$r_{2,3}$
R_3	$r_{3,1}$	$r_{3,2}$	$r_{3,3}$

Table 3.6: Request Reordering

Overlap type	Weight - o($r_{n,i}$, $r_{m,i}$)
Complete overlap	1
Partial overlap	f(degree of overlap)
No overlap	0

Table 3.7: Weight for each overlap type

Weight of sub-request	Iteration
$w_{r_{1,1}}$	o($r_{1,1}$, $r_{2,1}$) + o($r_{1,1}$, $r_{3,1}$)
$w_{r_{1,2}}$	o($r_{1,2}$, $r_{2,2}$) + o($r_{1,2}$, $r_{3,2}$)
$w_{r_{1,3}}$	o($r_{1,3}$, $r_{2,3}$) + o($r_{1,3}$, $r_{3,3}$)
$w_{r_{2,1}}$	o($r_{2,1}$, $r_{1,1}$) + o($r_{2,1}$, $r_{3,1}$)
$w_{r_{2,2}}$	o($r_{2,2}$, $r_{1,2}$) + o($r_{2,2}$, $r_{3,2}$)
$w_{r_{2,3}}$	o($r_{2,3}$, $r_{1,3}$) + o($r_{2,3}$, $r_{3,3}$)
$w_{r_{3,1}}$	o($r_{3,1}$, $r_{1,1}$) + o($r_{3,1}$, $r_{2,1}$)
$w_{r_{3,2}}$	o($r_{3,2}$, $r_{1,2}$) + o($r_{3,2}$, $r_{2,2}$)
$w_{r_{3,3}}$	o($r_{3,3}$, $r_{1,3}$) + o($r_{3,3}$, $r_{2,3}$)

Table 3.8: Weight per sensor request

3.2.10.3 Construction of overlapping and non-overlapping sets

This step is same as in 3.2.6.2

3.2.10.4 Scheduling Decisions

The scheduling decisions can be one of the following cases and are explained below:

Total Weight	Equation
W_{R1}	$w_{r_{1,1}} + w_{r_{1,2}} + w_{r_{1,3}}$
W_{R2}	$w_{r_{2,1}} + w_{r_{2,2}} + w_{r_{2,3}}$
W_{R3}	$w_{r_{3,1}} + w_{r_{3,2}} + w_{r_{3,3}}$

Table 3.9: TotalWeight per request

- No Execution - If any of sub-request of a nth request R_n ($r_{n,1}$,$r_{n,2}$,$r_{n,3}$...$r_{n,i}$) cannot be executed, then none of the sub-requests are executed. This method ensures that an application is not left with a partially serviced request in the end.

- Merged Requests - If all of the sub-requests of nth request R_n ($r_{n,1}$, $r_{n,2}$, $r_{n,3}$...$r_{n,i}$) completely overlap, then all the requests are scheduled for execution with 100% utility.

- Delayed Requests - When all the sub-requests of a request R_n ($r_{n,1}$,$r_{n,2}$,$r_{n,3}$...$r_{n,i}$) partially overlap or do not overlap at all, then the request is delayed. If the non-overlapping set's execution time t_{exec} is lesser than the remaining time t_{remain} in the heartbeat as shown in Figure 3.10, the utility for the delayed request is calculated using a delay function such as the one in section 3.2.9.

  $$U_{delayed} = \text{Utility of the delayed request executing for } t_{exec}$$

  If the non-overlapping set's execution time t_{exec} is greater than the t_{remain}, R is scheduled only for t_{remain} along with the delay from above case. This case is illustrated in Figure 3.11.

  $$U_{delayed} = \text{Utility of the delayed request executing for } t_{remain}$$

  If the $U_{delayed}$ is greater than $U_{threshold}$ for all the sub-requests, request R is scheduled to be executed in the same heartbeat with a utilty $Udelayed$, else it is postponed to be executed in the next heartbeat with utility $U_{postponed}$ as shown in Figure 3.12.

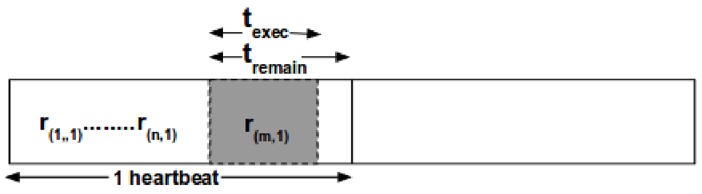


Figure 3.10: Delayed Request - t_{exec} less than t_{remain}

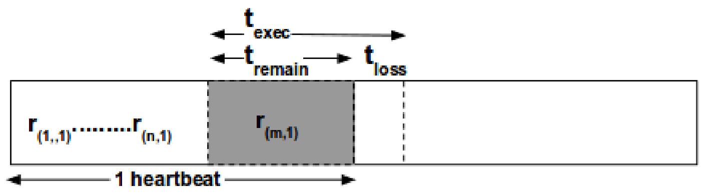


Figure 3.11: Delayed Request - t_{exec} greater than t_{remain}

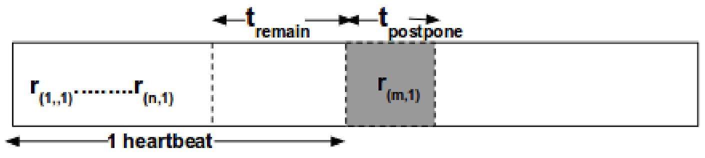


Figure 3.12: Postponed Request

- Deferred/Postponed Requests - In the eventuality of a request R being desired immediate execution (as per the above decision), the possibility of deferring it to the next heartbeat is evaluated. That evaluation follows the selfsame decision process. A request, when postponed, is added to the start of the queue of batched requests in the next heartbeat. This ensures that the postponed requests are considered first for execution. The overlap algorithm is not run on the postponed requests since they need to be prioritized over others and executed irrespective of other conditions. Since the delay threshold is fixed to one heartbeat for these experiments, if the request is postponed by two heartbeat or more, they cannot be executed. The utility of the postponed requests is a direct measure of the utility threshold.

$$U_{postpone} = U_{threshold}$$

Algorithm 3: Postpone function

```
Function Postpone_request($r_{n,1}$,$r_{n,2}$, .... $r_{n,i}$)
    Increment $potspone\_number^r$ by 1;
    Add $r_n$ to $postpone\_queue$;
```

Algorithm 4: Weight function

```
Function Calculate_weight(overlap_type, $s_i$)
    if overlap_type is complete overlap then
        $weight^r_{s_i} = 1$
    if overlap_type is partial overlap then
        $weight^r_{s_i}$ = f(degree of overlap)
    if overlap_type is no overlap then
        $weight^r_{s_i} = 0$
```

Algorithm 5: Reorder requests

```
Function Reorder_requests(req_per_hb)
    for req in postpone_queue do
        if $potspone\_number^r < delay_{threshold}$ then
            $reordered\_request\_queue \leftarrow req$;
    for $r_n$ in req_per_hb_queue do
        Iterate each of the req in re_per_hb_queue and find the type of overlap
          $r_{n,i}$ makes with the other request;
        Calulate_weight(overlap_type,$s_i$);
        $total\_weight_{R_n} = w_{r_{n,1}} + w_{r_{n,2}} + w_{r_{n,3}} ......w_{r_{n,i}}$;
    Reorder requests in req_per_hb in decreasing order of their total_weight;
    Reordered requests appended into reordered_request_queue;
```

3.2.10.5 A-DSES Algorithm

The A-DSES Algorithm 6 comprises of the steps discussed in Section 3.2.10.1 to Section 3.2.10.4

3.2.11 Cost Modeling

We propose a utility based cost model to prevent users from overloading the system by constantly requesting for sensor/actuator resources to maximize utility. This will provide incentives to users to accept sensor access of slightly lower utility with reduced costs and in return, generates opportunities to merge user requests. Users are charged based on the utility factor and the cost for each user is based on the resource usage $c_u(X_{s_i}(t))$, where $X_{s_i}(t)$ represents the resource usage of a sensor s_i at time interval t. The cost for using the sensor/actuator for two overlapping requests A and B can be expressed as $c_A(X_{s_i}(t)^A)$, $c_B(X_{s_i}(t)^B)$.

The resulting revenue for the sensor/actuator resource provider is:

$$R = c_A(X_{s_i}(t)^A) + c_B(X_{s_i}(t)^B)$$

The goals may differ based on the sensor operator policy. A public provider might have the goal to maximize the overall system utility $max(\sum_u u_u(X_{s_i}(t)^u) - c_u(X_{s_i}(t)^u))$, while private sensor resource providers might place greater emphasis on maximizing their overall revenue. This can be achieved for the sensor providers by minimizing the system resource usage and hence minimizing the cost of using the resources for a given user $min(\sum_u c_u(X_{s_i}(t)^u))$. For example, if B is a subset of A, the resulting revenue for the sensor/actuator resource provider is $R = c_A(X_{s_i}(t)^A) + c_B(X_{s_i}(t)^B)$, the cost of system resources used is $c_A(X_{s_i}(t)^A)$, while the utility for each user is $u_A(X_{s_i}(t)^A)$, $u_B(X_{s_i}(t)^B)$, respectively.

Algorithm 6: WDSES Algorithm

```
Reorder_requests($req\_per\_hb$);
for $r_n$ in reordered_request_queue requesting for sensor sensor $s_1$ to $s_i$ do
    if $r_{n,1}$ and $r_{n,2}$ and .... $r_{n,i}$ is a postponed request then
        $U_{postponed}$ is the minimum acceptable utility, $U_{threshold}$;
        $U_{R_n} \leftarrow U_{postponed}$;
    if $r_{n,1}$ and $r_{n,2}$ and .... $r_{n,i}$ is a complete overlap then
        schedule $R_n$;
        $U_{R_n} \leftarrow U_{complete}$;
        $r^{time} \leftarrow t^{sched.}_{exec}$;
    if $r_{n,1}$ and $r_{n,2}$ and .... $r_{n,i}$ is partial or no overlap then
        $t_{remain} = t_{heartbeat} - t^{sched.}_{exec}$;
        if $t_{remain} \geq t_{exec}$ then
            Delayed Utility $U_{delayed}$ = Delay_task_execution($non\_overlap\_data$
             for $t_{exec}$) + $Data_{overlap}$;
            if $(U(r_{n,1}) \wedge U(r_{n,2}) \wedge ....U(r_{n,i})) \geq U_{threshold}$ then
                schedule $R_n$ in the same heartbeat;
                $U_{R_n} \leftarrow U_{delayed}$;
            else
                if $postpone\_number^r \leq Delay_{threshold}$ then
                    Postpone_request$r_{n,1}$,$r_{n,2}$, .... $r_{n,i}$ to next heartbeat;
                else
                    Cannot execute $r_{n,1}$,$r_{n,2}$, .... $r_{n,i}$;
        else
            Actual Utility $U_{delayed}$ = Delay_task_execution$non\_overlap\_data$ for
             $t_{remain}$ + $Data_{overlap}$;
            if $(U(r_{n,1}) \wedge U(r_{n,2}) \wedge ....U(r_{n,i})) \geq U_{threshold}$ then
                Schedule $R_n$ in the same heartbeat;
                $U_{R_n} \leftarrow U_{delayed}$;
            else
                if $postpone\_number^r \leq Delay_{threshold}$ then
                    Postpone_request$r_{n,1}$,$r_{n,2}$, .... $r_{n,i}$ to next heartbeat;
                else
                    Cannot execute $r_{n,1}$,$r_{n,2}$, .... $r_{n,i}$;
```

CHAPTER 4
EVALUATION

In this chapter, we present the results from simulations conducted to evaluate the TDMA and DSES approaches presented in Section 3.2. The simulations use actual traces taken from a four node radar sensor network [11]. This network has different user groups (generally described as applications in this paper) that schedule individual tasks based on application preferences and the past information generated by the radars. The current implementation of the system does not support virtualization and only the task generating the highest utility per heartbeat is executed.

We use these actual traces from the radar sensor network as input to the scheduler, virtually schedule requests according to the approaches presented in Section 3.2, and calculate the utility generated by executing these tasks. We use this resulting utility to compare the performance of the different scheduling approaches.

4.1 Experimental Data Set

Before we present the results of experiments performed with different scheduling approaches, we introduce the dataset used for our trace-based simulations and the notations that go along with it.

This dataset is generated as an output from the CASA system's main control loop called Meteorological Command and Control (MC&C) [21, 7]. The first test bed of CASA that deployed DCAS system is called Integrated Project 1(IP1). The saved features from a past event(in this case meteorological feature like reflectivity and storm cell) are fed into the simulator. The end users used in this data set include Res,NWS,NWP,EM and Storm. NWS, National Weather Service issues severe weather warnings. EM is Emergency Managers that alerts the public about weather hazards. Res is a group of research users(CASA researchers) collecting wide variety of data to improve thier forecast models. NWP are group of users collecting data for their nowcasting algorithm. Storm user detects scans severe storms. The sensing needs of these IP1 users were encoded by CASA into a set of rules called task rule that tells the system what, when and how to scan. As a result, there are eight different applications that generate requests for scanning based on the task rules at locations determined by the sector commands. The naming convention for the eight applications are shown in Table 4.1 The simulator is run individually for each application generating a set of scan tasks, that would be issued by an application when it runs on a dedicated sensor network. The combined set of tasks from all applications is then used as input to the scheduling algorithms presented in Section 3.2.

Name	Application name
P1	P1_NWP_reflectivity
N1	N1_NWS_reflectivity
R1	R1_Storm_reflectivity
S1	S1_Storm_stormcell
T2	T2_Res_stormcell
T1	T1_Res_reflectivity
RHI	RHI_Res_stormcell
E1	E1_EM_reflectivity

Table 4.1: Applications in data set and their naming convention used in the document

The generated data set has the following information:

- Slot number or heartbeat
- Application name requesting for network access in the corresponding heartbeat
- Scanning angle of the radar(s) for the respective task
- Radar configuration parameters

4.2 TDMA approaches

The reduction in utility of an application when compared to its utility on a dedicated network is one of the key performance indicators of efficiency. Any scheduling mechanism that we consider will attempt to minimize this utility reduction. Figure 4.1 plots the utility reduction in percentage against the application request ratio for the various scheduling mechanisms. Using the request ratio as a metric eliminates outliers in the sample space that produce a high reduction in utility due to a low request ratio. The plot in Figure 4.2 for TDMA1 shows a decrease in utility with increase in scan angles of the requests. However since they are prioritized according to the increasing scan angles, they do not decrease with increase in the request ratios. Hence we see a increasing and decreasing values in the barplot 4.1 for TDMA1. This approach is only useful in cases where we do not want to starve the requests that take shorter time to execute. In Figure 4.1, the results show that TDMA3 is more consistent in the small sample space of applications in maintaining an average reduction in utility. Also, TDMA2 looks to perform in accordance with the application request ratio, displaying no reduction for high request applications and a high reduction rate for rarely requested ones. However, for TDMA2 in Figure 4.1 we see that the average utility reduces for 'R1' even though it has a higher request rate. This is because, 'R1' is mostly requested along with other applications that have higher request ratio and longer execution time. So 'R1' cannot be executed within the time period. From Figure 4.3(a) and Figure 4.3(b),we see that even though the request ratio is high, the hit ratio might not be the same due to the fact that they are requetsed along with higher request ratios and longer execution time. This approach is only useful where we need to execute high priority like applications to measure wind speed in tornadoes requests often within small intervals of time. We conclude that TDMA3 is a better approach among the two since it provides a fair distribution of resources to all its applications and a tolerable reduction in utility. The token approach was simulated to observe the impact of different token generation rates on various application utilities. These simulations were run with different token creation periods of 30T, 20T, and 10T respectively. Figure 4.4 illustrates the effect of token generation rates on different applications. The effect of utility appears to degrade with a drop in request rate (the applications are arranged in decreasing order of their request rates from left to right). For lower request rate applications like 'T2','S1','R1' the effect of utility increases with lower token generation rates. We evaluated the impact of utility for the 'T1' application on sharing multiple applications on a sensor network. The average utility for this application decreased by 30% when five applications and 66% when eight applications shared a sensor network in the case of TDMA3 approach.

We see from Figure 4.5, TDMA approaches do not have a high hit rate for applications due to inefficient usage of slots and no sharing of data. We present the results for DSES approach with higher hit rates in the next setion.

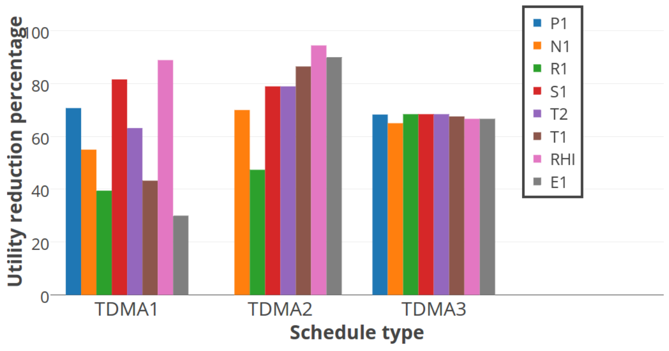


Figure 4.1: Percentage of utility reduction for all TDMA approaches

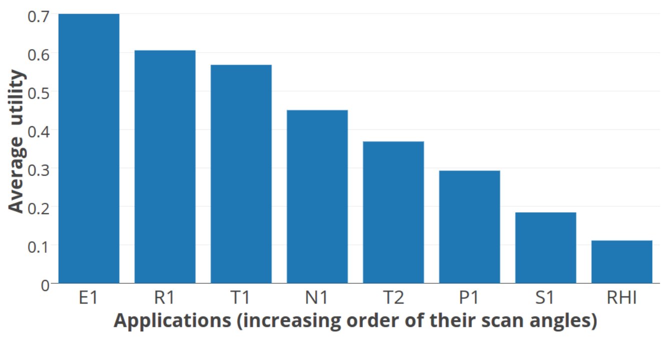
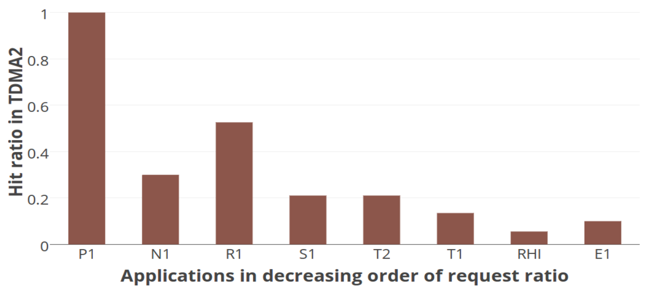
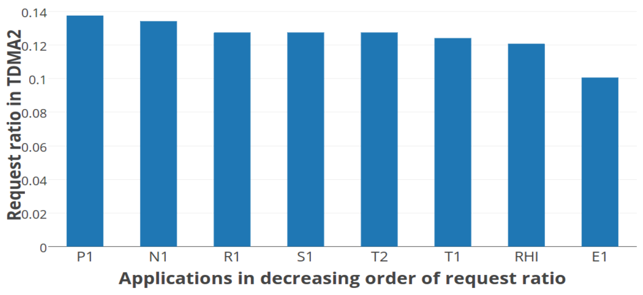


Figure 4.2: Utility vs applications in the increasing order of their scan angles



(a) Hit ratio for applications in TDMA2



(b) Request ratio for applications in TDMA2

Figure 4.3: TDMA2 results

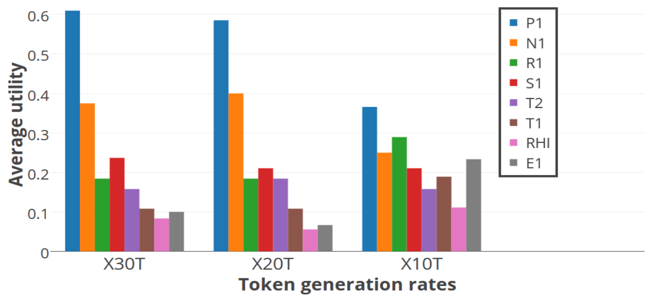


Figure 4.4: Average uility of applications for different token generation rates

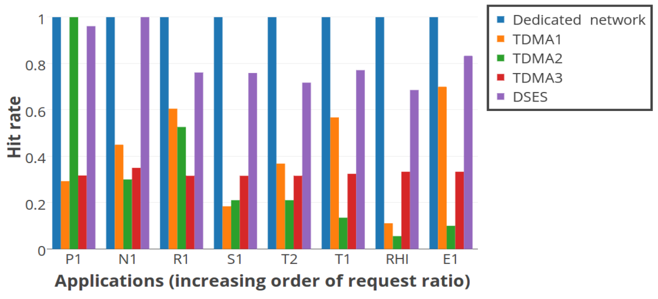


Figure 4.5: Hit rates of applications for TDMA approaches, DSES approach and dedicated network

4.3 DSES approach

In this experiment, our goal is to analyze the impact of average utility of application 'Res' on sharing the network with different applications, using the approach presented in Section 3.2.6.

We first run the traces to allow only application 'T1' to be executed to mimic its performance on a dedicated sensor network. Then the number of applications requesting for network access are gradually increased to mimic a shared network.

In our current design, the tasks are either shared (if they overlap) or are scheduled for execution at a later time if the scheduler can guarantee execution within the request's deadline.

From a service provider's perspective, two requests that overlap completely mean optimal use of system resources. The request that began executing first obtains maximum utility. The second request that was piggybacked onto an executing one receives a portion of the utility from the already executing request.

A request that has to be split into overlapping and non-overlapping portions still results in lower resource utilization because of the overlapping portion that was shared with another request. The dedicated resource usage needs to be taken into account only for the non-overlapping portion.

Figure 4.6(a) illustrates a plot of average utility/request for application 'T1' with an increase in number of applications. From the plot, we see that the average utility for 'T1' decreases with an increase in number of applications. Interestingly, the addition of the fourth application causes the average utility of 'T1' to increase slightly. This may be due to the increase in number of complete overlap requests with the addition of the fourth application in comparison to lesser number of complete overlap requests with the addition of the third application. This characteristic is illustrated in Figure 4.6(b).

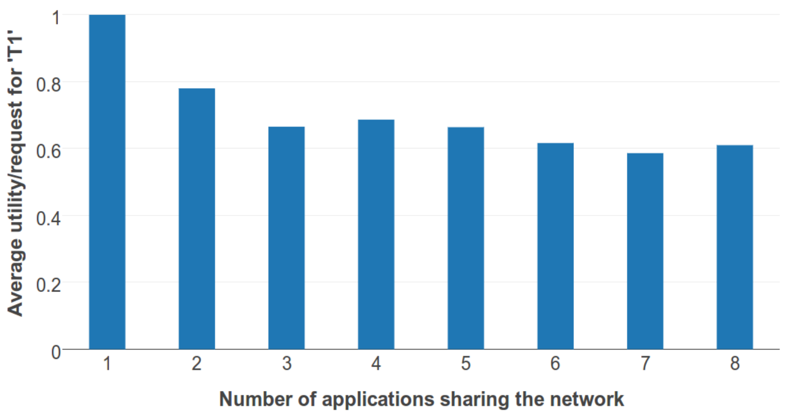
In general, the average utility of 'T1' decreases with an increase in number of applications. This is obviously impacted by the overlap between tasks from different applications. If there is a larger number of non-shared or complete overlapping requests, and lesser number of non executing tasks, the higher will be the corresponding task's utility.

We observed that the average utility for 'T1' decreased by 20% with five applications in Figure 4.7 and 40% with eight applications in Figure 4.6 compared to the dedicated sensor networks. From the results, we see that there was an improvement in the utility of application 'T1' in DSES compared to TDMA approach where the average utility for 'T1' application decreased by 30% with five applications and 66% with eight applications sharing a sensor network.

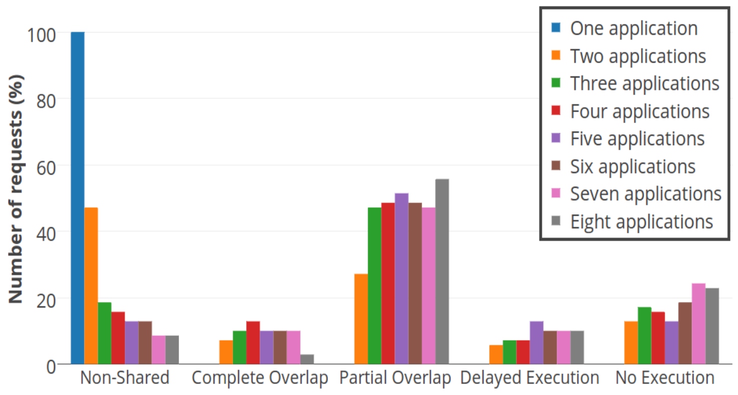
Figure 4.8(a) shows the total utility of all the applications sharing the sensor network. As the number of applications increases, the total utility of all applications in the system also increases. This shows that the data sharing results in executing more number of requests by merging multiple requests. Fig 4.8(b) shows the average cost per application that represents the cost of using system resources by each application. It can be seen that the the cost of using the resources decreases as more applications share a netwrok. This is because many of the requests is a subset of other executing requests which results in merging of requests. Even though the revenue for sensor providers will be the cost of using system resources for both requests individually(two scans),the actual cost of using the system resources is of one merged requests(one scan).

4.4 Synthetic data sets

In order to evaluate the variation in utility for different types of overlaps, we run the DSES and A-DSES on synthetic data-sets. We utilize a complete-overlap data-set, partial-overlap data-set and no-overlap data-set. The complete-overlap data-set



(a) Average utility/request for 'T1'



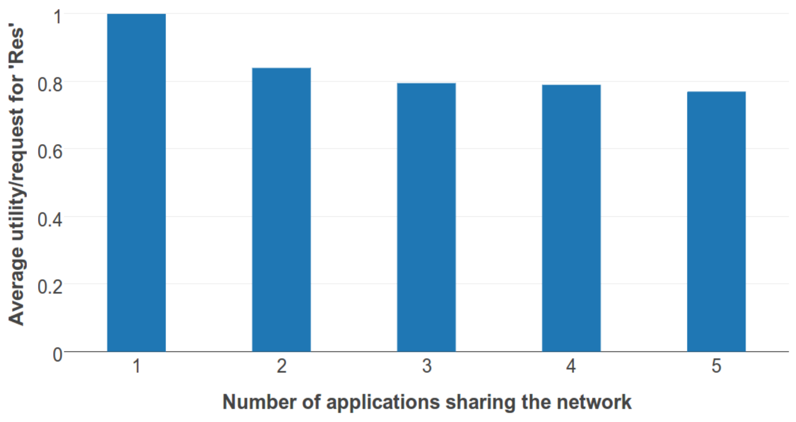
(b) Percentage of non-shared and shared tasks for 'T1'

Figure 4.6: DSES results with eight applications

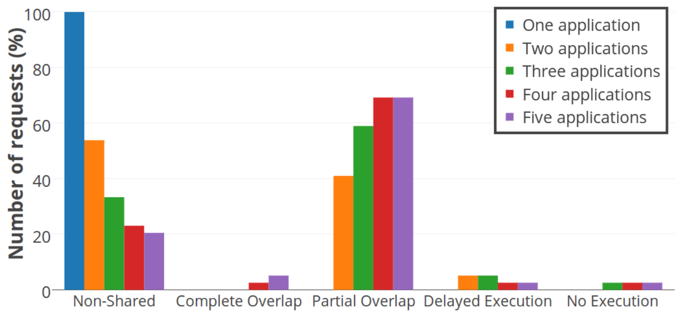
is an ideal representation of the utility's upper bound, wherein all the requests overlap and thus a 100% utility can be acheived. Conversely, no-overlap data-set represents a worst case scenario (with the assumption that all the non-overlapping postponed requests are executed). Typical data-sets should fall between these bounds.

4.4.1 DSES

Figure 4.10 repesents the utility of 'T1' run on synthetic data set with the DSES approach. If the requests are non-overlapping, they are delayed to be executed at a later, if $U_{delayed} < U_{threshold}$. Otherwise, the request is not executed. Hence, we see a decrease in utility in the non-overlap set as the utility threshold increases.



(a) Average Utility for 'Res' application



(b) Percentage of non-shared and shared tasks for 'Res' application

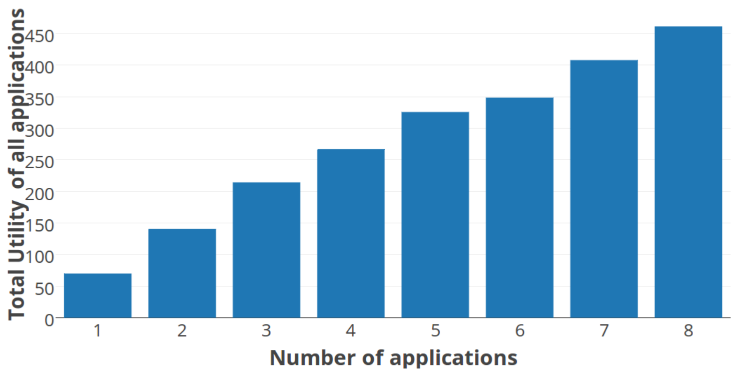
Figure 4.7: DSES results with five applications

4.4.2 A-DSES

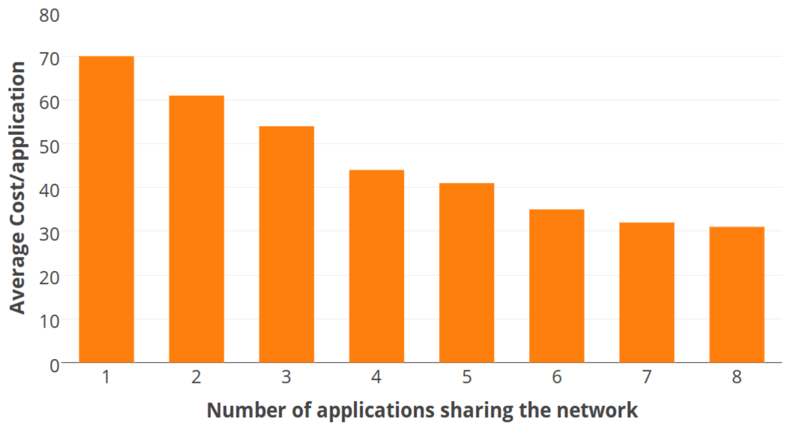
Figure 4.12 repesents the utility of synthetic data-sets with the augmented DSES approach. The non overlapping requests are postponed to the next heartbeat if they cannot be delayed within the same heartbeat due to lower utility constraints. The utility of the postponed request is a direct measure of the utility threshold. Therefore we see an increase in utility for the non-overlap set with increase in utility threshold.

4.5 Real Data-sets

With the upper and lower bound utility obtained from the synthetic data set, we run the base-DSES and augmented-DSES on multiple real workloads. Figure 4.10



(a) Total utility of all applications in the system



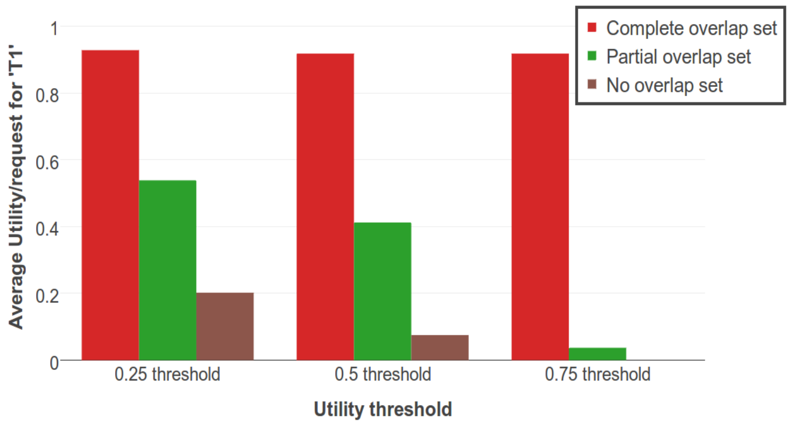
(b) Average cost/application vs number of applications sharing the network

Figure 4.8: DSES results with eight applications

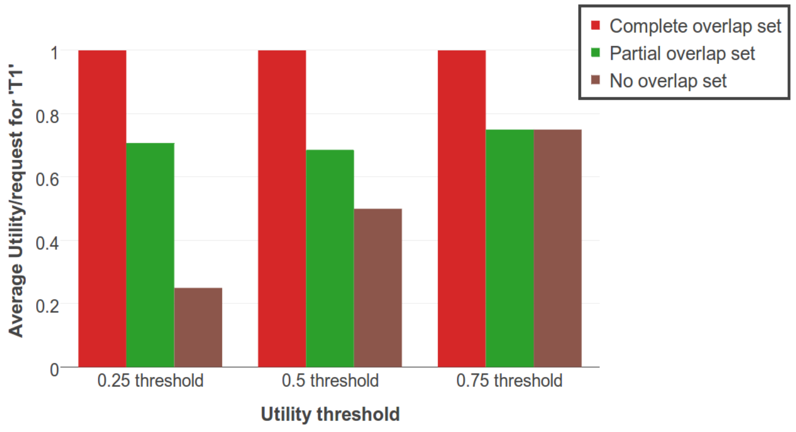
and Figure 4.12 show that the utility on any real workload lie between the upper and lower limits depending on the utility threshold.

4.5.1 DSES

The utility of 'T1' in the real data-set decreses with increase in utility threshold. For higher utility threshold, there are fewer number of partial overlaps since the criteria to execute partial requests is that the overlap should be greater than utility threshold. We see from Figure 4.11(a) and Figure 4.11(b) that the number of partial overlapping requests decreases as utility increases.



(a) DSES



(b) A-DSES

Figure 4.9: Utility of 'T1' on synthetic work loads for different utility threshold

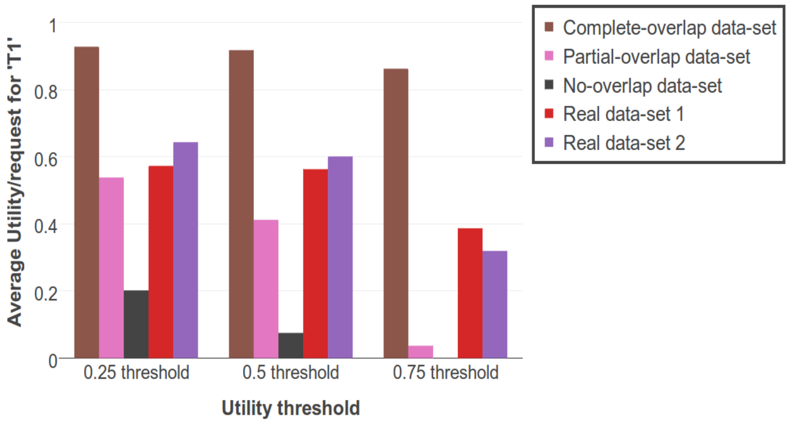
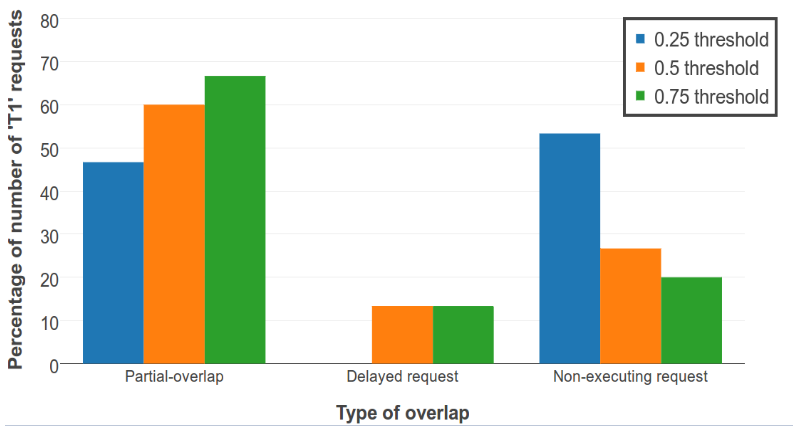
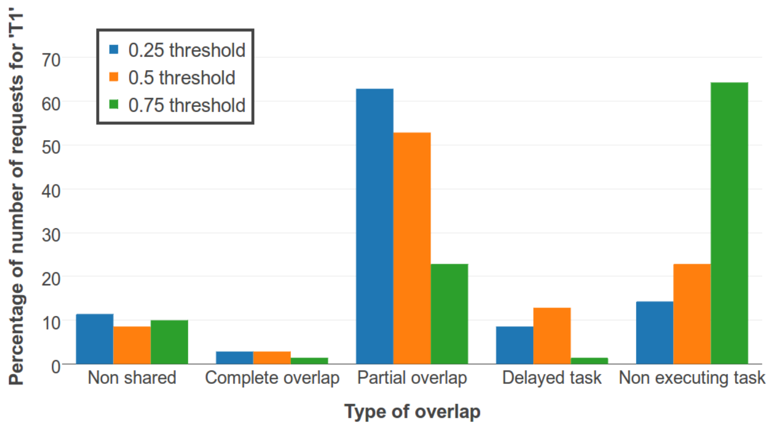


Figure 4.10: Utility of 'T1' on real work loads for DSES



(a) Percentage of number of requests for 'T1' in Real data set 1



(b) Percentage of number of requests for 'T1' in Real data set 2

Figure 4.11: DSES on Real work loads

4.5.2 A-DSES

For the sake of brevity in the Graph 4.13(a) we use the following abbreviations:

- Co(not postponed) - Requests that are complete overlap.
- Po(not postponed) - Requests that are delayed within the same heartbeat due to partial overlap.
- No(not postponed) - Requests that are delayed within the same heartbeat due to no overlap.

- Co(postponed) - One of the sub-request is a complete overlap but postponed due to postponing of one or other sub-requests.
- Po(postponed) - One of the sub-request is a partial overlap but postponed due to postponing of one or other sub-requests.
- No(postponed) - One of the sub-request is a no overlap but postponed due to postponing of one or other sub-requests.

As the number of Co(postponed) requests increase and number of Co(not postponed) requests decrease, the utility decreases. Since the postponed utility of a request is a direct measure of the utility threhsold, the postponed task's utility is higher for a higher utility threshold.

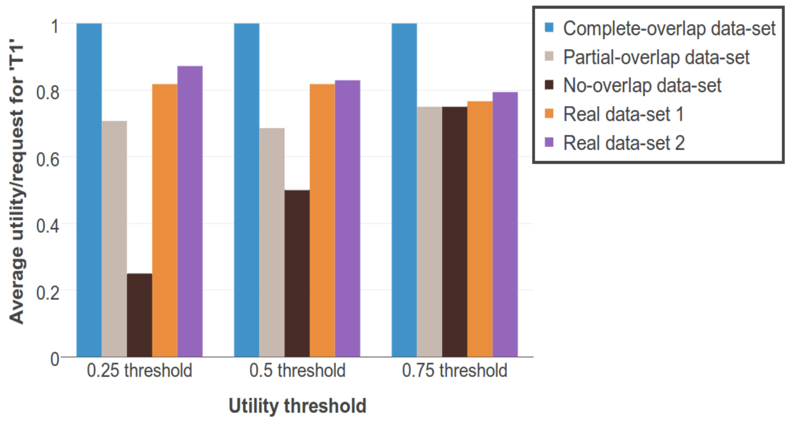
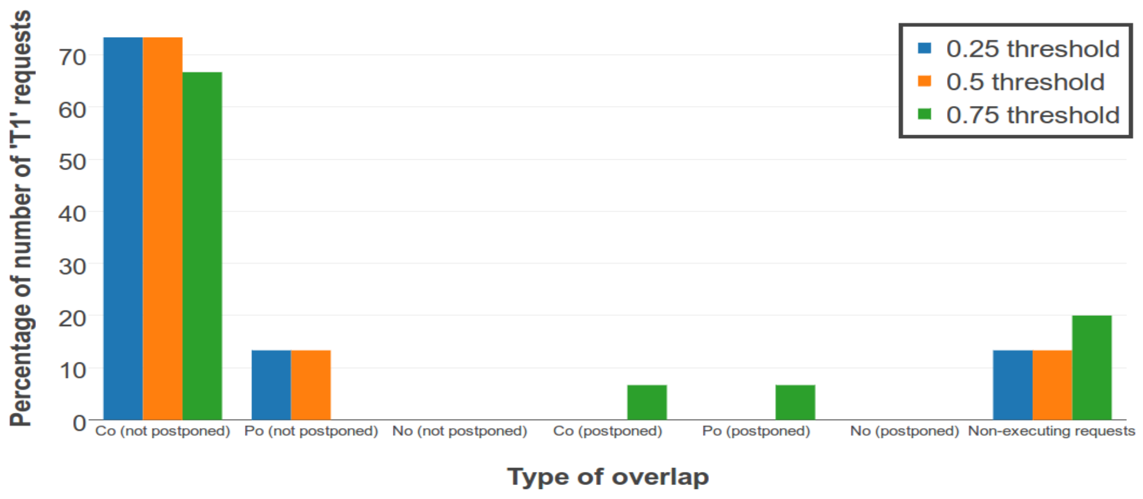


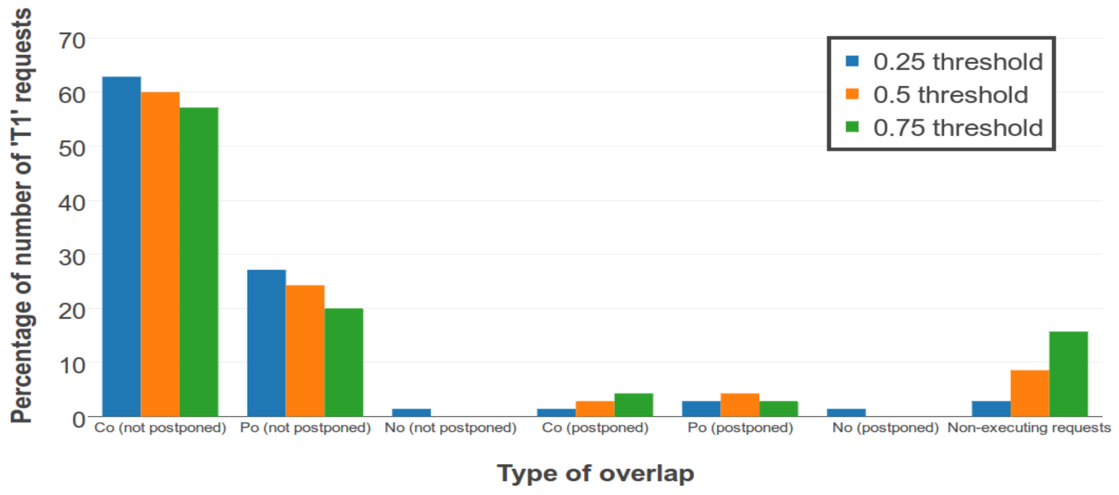
Figure 4.12: Utility of 'T1' on real work loads for A-DSES

4.6 Impact on utility of other applications on the shared network

To ensure that no applicaition is starved, we plot the utility of all the applications in DSES and augmented-DSES approach.



(a) Percentage of number of requests for 'T1' in Real data set 1

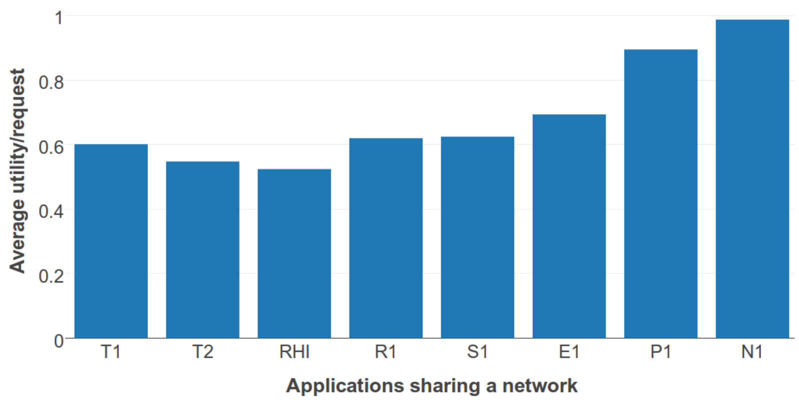


(b) Percentage of number of requests for 'T1' in Real data set 2

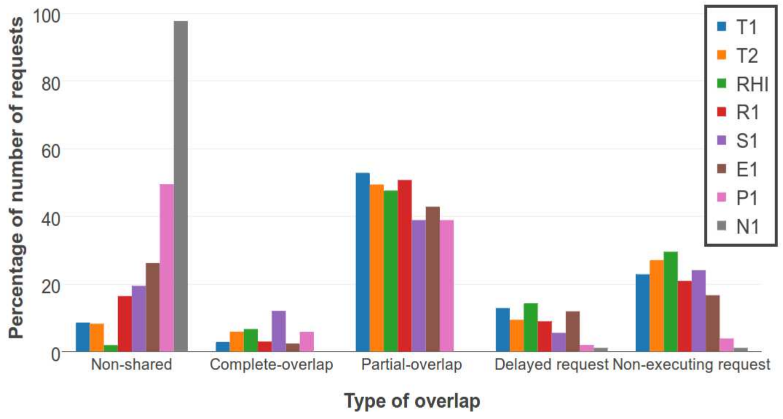
Figure 4.13: A-DSES on Real workloads

4.6.1 DSES

In the base-DSES approach, the requests are first grouped in increasing order of their priority (N1, P1, E1, S1, R1, RHI, T2, T1) and then scheduled. However, their utilities might not increase in the order of their priority since the utility depends on the type of overlap.



(a) Average utility of all the applications in real data set 2



(b) Percentage of number of requests in Real data-set 2

Figure 4.14: DSES results of all applications sharing a network

4.6.2 A-DSES

In this approach, the requests are reordered to maximize the number of overlaps. We can see from Figure 4.15 that there is a more uniform distribution of average utilty among all the applications compared to the DSES approach.

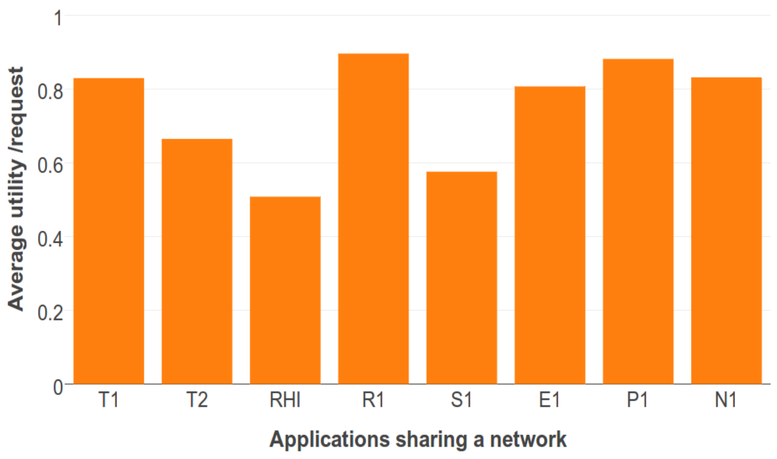


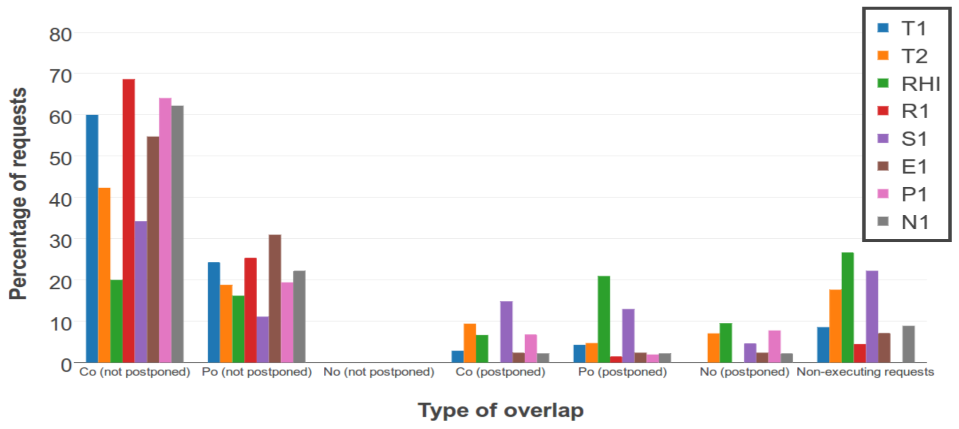
Figure 4.15: Average utility of all the applications in Real data-set 2 - A-DSES

Among all the applications that share the sensor network, RHI and S1 have compaitively lower utilities. From Figure 4.16(a) we observe that the utility of a request reduces in any of the following cases:

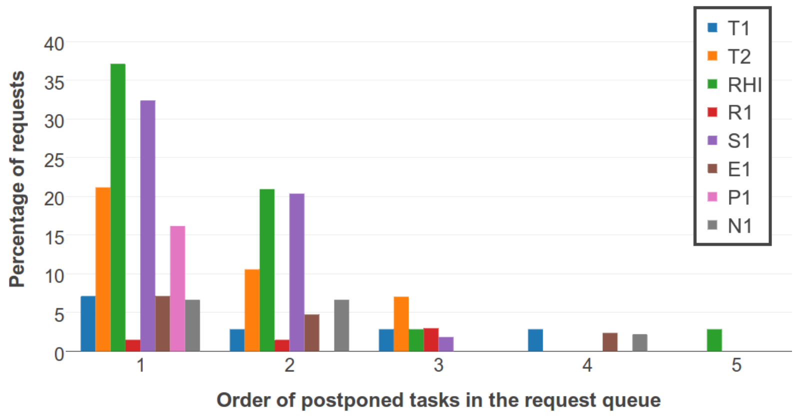
- If Co(postpone) requests increase,
- If Co(not postpone) requests decrease,
- If non-executing requests increase.

The number of non-executing requests increases if:

- The request does not occur first in the queue of postponed tasks. Figure 4.16(b) shows multiple occurances of RHI and S1 in second or later positions in the queue of postponed tasks.
- One of the sensors fail to execute a sub-request, thus disqualifying the entire request from being executed in that heartbeat.



(a) Percentage of number of requests in real set 2



(b) Occurances of requests in the postponed queue

Figure 4.16: A-DSES results of all applications sharing a network

4.7 Comparisons and Concluding Observations

A-DSES performs better than base-DSES for the following reasons,

- A-DSES uses the windowing and batching to maximize the number of complete overlaps in a heartbeat. This ensures merging of more requests and thereby improve the corresponding utility. The average run time to the algorithm is shown in Figure 4.20.

- Since the augmented-DSES requests are batched and processed, instead of executing serially like in DSES, requests with complete overlaps result in 100% utility and do not incur any delay.

- All sub-requests from different sensors for a given request is collaborated and executed together. In this way, augmented approach is more aligned with the way sensor networks work in real time.

- The total utility and the average cost/application in A-DSES is almost the same as DSES. This is seen in Figure 4.18 and Figure 4.19. But we see in Figure 4.17 that the average utility of across most of the applications sharing a network is higher in A-DSES compared to DSES.

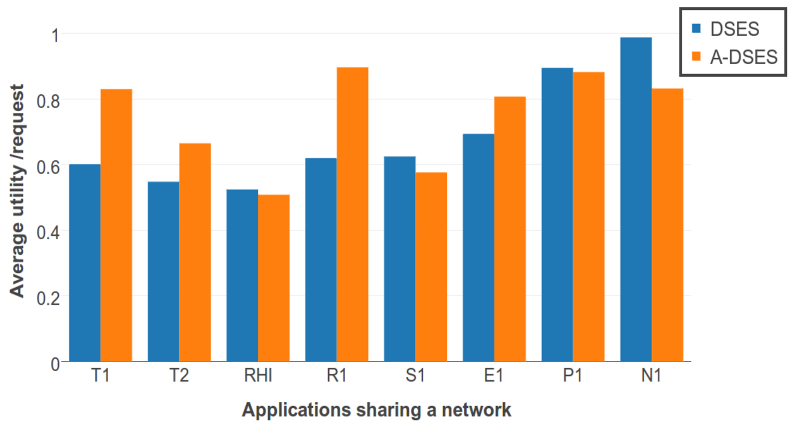


Figure 4.17: Average utility of all the applicaions in DSES and A-DSES

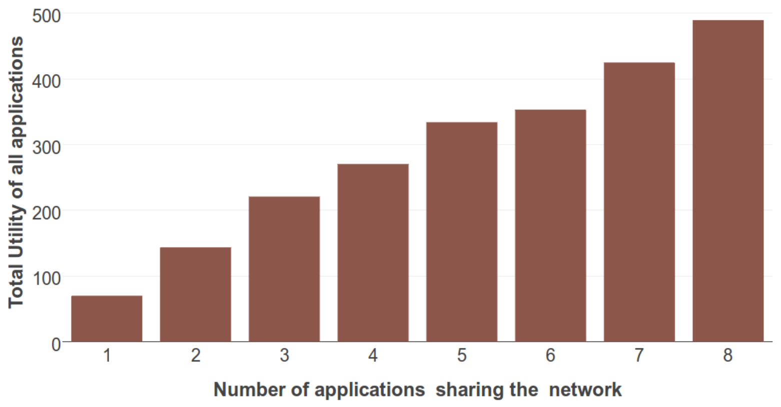


Figure 4.18: Total utility of all applications in A-DSES

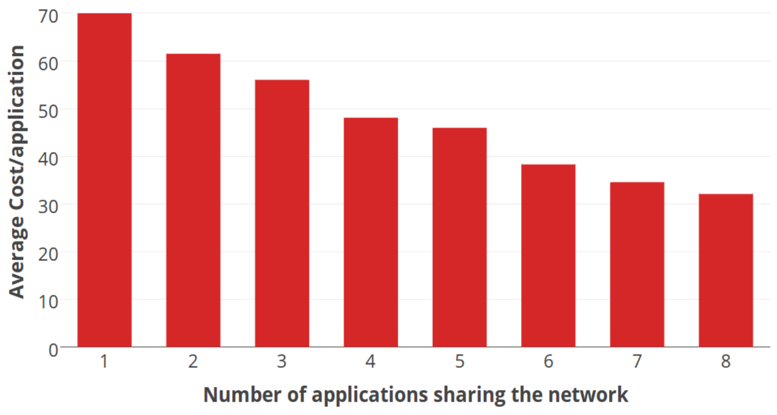


Figure 4.19: Average cost/application in A-DSES

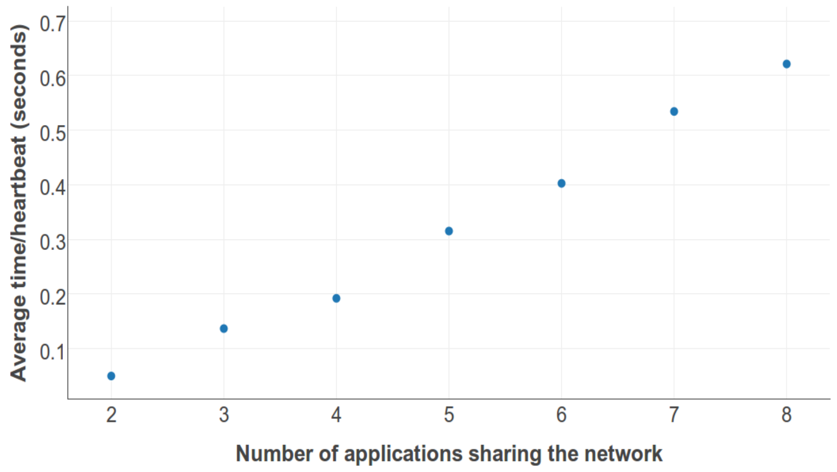


Figure 4.20: Average time/heartbeat to run the overlap algorithm in A-DSES

CHAPTER 5
CONCLUSION

We introduce our architecture for closed-loop sensor network virtualization and our work show that such architecture can lead to the isolated sharing of closed-loop sensor networks. Such shared closed-loop sensor networks will significantly reduce the cost for creating and operating sensing infrastructure, while providing access to a broad set of applications that will run on top of these systems. After introducing the architectures and the virtualization layer for closed-loop sensor networks we present several scheduling approaches for sensor/actuator virtualization, based on TDMA and potential data overlap between requests. We evaluate the approaches through trace based simulations and show how they can support sensor virtualization. Our results show that the Enhanced scheduling approaches with data sharing feature allow the sharing of closed-loop sensor networks with the potential downside that the overall utility is reduced if compared to individual, dedicated networks. The results show that there is a clear trade-off between full utility and cost where applications can share a network. To further optimze the DSES approach, we introduced batch processing and reordering of requests to maximize the number of complete overlaps. Though the total utility and average cost/applications is almost the same in DSES and ADSES, the utility is uniformly distributed across all the applications in ADSES compared to DSES. Also, ADSES design and implementation is better aligned with the working of a real sensor network system.

BIBLIOGRAPHY

[1] G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. J. Dongarra, and P. M. A. Sloot, editors. *Computational Science – ICCS 2009*. Springer-Verlag, May 2009.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebaue, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, USA, October 2003.

[3] T. Baumgartner, I. Chatzigiannakis, M. Danckwardt, C. Koninis, A. Kröller, G. Mylonas, D. Pfisterer, and B. Porter. Virtualising testbeds to support large-scale reconfigurable experimental facilities. In *EWSN*, pages 210–223, 2010.

[4] R. Bose and A. Helal. Distributed mechanisms for enabling virtual sensors in service oriented intelligent environments. In *Intelligent Environments, 2008 IET 4th International Conference on*, pages 1 –8, july 2008.

[5] R. Bose, A. Helal, V. Sivakumar, and S. Lim. Virtual sensors for service oriented intelligent environments. In *Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology*, ACST'07, pages 165–170, Anaheim, CA, USA, 2007. ACTA Press.

[6] J. Brotzge, V. Chandrasekar, K. Droegemeier, J. Kurose, D. McLaughlin, B. Philips, M. Preston, and S. Sekelsky. Distributed collaborative adaptive sensing for hazardous weather detection, tracking, and predicting. In *Proceeding of Computational Science - ICCS 2004*, Krakow, Poland, May 2004.

[7] J. Brotzge, D. Westbrook, K. Brewster, K. Hondl, and M. Zink. The meteorological command and control structure of a dynamic, collaborative, automated radar network. In *21st International Conference on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology*, Jan. 2005.

[8] N. Brouwers, K. Langendoen, and P. Corke. Darjeeling, a feature-rich vm for the resource poor. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 169–182, New York, NY, USA, 2009. ACM.

[9] E. Bugnion, S. Devine, and M. Rosenblum. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Transactions on Computer Systems*, 15(4):143–156, November 1997.

[10] J. R. Celis, D. Gonzales, E. Lagda, and L. Rutaquio. A comprehensive review for disk scheduling algorithms. In *IJCSI 2014*, 2012.

[11] B. P. E. L. M. Z. D. Pepyne, D. Westbrook and J. Kurose. Distributed collaborative adaptive sensor networks for remote sensing applications. In *American Control Conference, 2008*, August 2008.

[12] P. Drake, D. McLaughlin, and M. Nolan. Collaborative and adaptive sensing of the atmosphere (casa) and multi-function sensor services network (mssn). In *Integrated Communications Navigation and Surveillance Conference (ICNS)*, pages 1–33, July 2010.

[13] P. Evensen and H. Meling. Sensor virtualization with self-configuration and flexible interactions. In *Proceedings of the 3rd ACM International Workshop on Context-Awareness for Self-Managing Systems*, Casemans '09, pages 31–38, New York, NY, USA, 2009. ACM.

[14] K. Hong, J. Park, T. Kim, S. Kim, H. Kim, Y. Ko, J. Park, B. Burgstaller, and B. Scholz. Tinyvm, an efficient virtual machine infrastructure for sensor networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 399–400, New York, NY, USA, 2009. ACM.

[15] A. Jayasumana, Q. Han, and T. Illangasekare. Virtual sensor networks - a resource efficient approach for concurrent applications. In *Proceedings of the 4th International Conference on Information Technology: New Generations (ITNG), Las Vegas, NV, USA*, 2007.

[16] H. B. Lim, M. Iqbal, and T. J. Ng. A virtualization framework for heterogeneous sensor network platforms. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 319–320, New York, NY, USA, 2009. ACM.

[17] D. Massaguer, S. Mehrotra, and N. Venkatasubramanian. A semantic approach for building pervasive spaces. In *Proceedings of the 6th Middleware Doctoral Symposium*, MDS '09, pages 2:1–2:6, New York, NY, USA, 2009. ACM.

[18] D. J. McLaughlin and V. Chandrasekar. Short wavelength technology and the potential for distributed networks of small radar systems. In *Radar Conference, 2009 IEEE*, May 2009.

[19] http://www.roc.noaa.gov, 2007.

[20] M. Pajic and R. Mangharam. Embedded virtual machines for robust wireless control and actuation. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 79–88, 2010.

[21] D. Pepyne, D. Westbrook, B. Philips, E. Lyons, and M. Z. J. Kurose. Distributed Collaborative Adaptive Sensor Networks for Remote Sensing. In *Proceedings of 2008 American Control Conference, Seattle, WA, USA*, June 2008.

[22] B. Philips, D. Pepyne, D. Westbrook, E. Bass, J. Brotzge, W. Diaz, K. Kloesel, J. Kurose, D. McLaughlin, H. Rodriguez, and M. Zink. Integrating End User Needs into System Design and Operation: The Center for Collaborative Adaptive Sensing of the Atmosphere (CASA). In *Proceedings of 16th Conf. Applied Climatol., American Meteorological Society Annual Meeting, San Antonio, TX, USA*, Jan. 2007.

[23] B. Philips, D. Westbrook, D. Pepyne, E. J. Bass, and D. Rude. Evaluation of the casa system in the noaa hazardous weather test bed. In *24th International Conf. on Interactive Information Processing Systems (IIPS) for Meteor., Ocean., and Hydrology, 88th American Meteorology Society Annual Meeting, New Orleans, LA, USA*, January 2008.

[24] B. Philips, D. Westbrook, D. Pepyne, J. Brotzge, E. Bass, and D. Rude. User evaluations of adaptive scanning patterns in the casa spring experiment 2007. In *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, volume 5, pages V –156 –V –159, July 2008.

[25] S. Pumpichet and N. Pissinou. Virtual sensor for mobile sensor data cleaning. In *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*, pages 1 –5, dec. 2010.

[26] L. Seawright and R. MacKinnon. Vm/370 - a study of multiplicity and usefulness. *IBM Systems Journal*, pages 4–17, 1979.

[27] N. Sharma, D. Irwin, M. Zink, and P. Shenoy. Multisense: proportional-share for mechanically steerable sensor networks. *Multimedia Systems*, 18(5):425–444, 2012.

[28] J. Sztipanovits and R. Rajkumar, editors. *International Conference on Cyber-Physical Systems*. ACM Press, April 2010.

[29] M. Zink, E. Lyons, D. Westbrook, J. Kurose, and D. Pepyne. Closed-loop architecture for distributed collaborative adaptive sensing: Meteorogolical command & control. *International Journal for Sensor Networks (IJSNET)*, 7(1/2), 2010.