In this chapter, we have presented MA assisted data fusion in WSNs (MADFWs). In MADFW, data stay at the local site, while the fusion process (code, i.e., MA) is moved to the data sites. By transmitting the MA instead of data, network bandwidth requirement is largely reduced and the performance of the MA is more stable. One of the key issue is implemented in this model is how to give identity and the itinerary for a MA in order to achieve progressive fusion accuracy in wide area distributed sensor networks with consumption of minimum amount of resources - energy, computing power, end-to-end and communication delay, etc.

Rest of the chapter is organized as follows. Issues are explored in Section 6.1. Agent Architecture is presented in Section 6.2. Section 6.3 gives the Agent/SN Namespace. System Model and Assumption are presented in Section 6.4. System architecture is given in Section 6.5. Mobility Management will be found in Section 6.6. Route planning for ad hoc network is presented in Section 6.7 and Section 6.8. Algorithm for managing the BS and BN is discussed in Section 6.9 Implementation and Performance study of WSNs under the leadership of AMS(SAP) is explored in Section 6.10. Section 6.11 discusses the outcome of the implementation. Finally chapter is summarized in Section 6.12.

6.1 Issues

There are several issues related to the design of MADFWs [195]. The major issues design of agent architecture, itinerary, distributed data integration and agent/SN namespace, etc. In [196] author presented a scheme which updates the programs located in the WSN nodes, each node occasionally advertises the most recent version of the program it has available to whatever nodes that can hear its local broadcast. Scheme allows application developers to update the program located in a specific PE by specifying the identification number of that PE. This updated program code is sent from the BS and the batteries of the PEs (SNs/BNs)
deployed near the BS become strained when programs are frequently updated, because sending the updated program to the necessary nodes (PEs) requires relaying messages back and forth with the BS. In [196] presented scheme only allows for two programs to coexist on a node(PE). Mate [197] allows for more lightweight agents, but does not actually resolve the problem of energy consumption being focused on nodes near the BS.

Agilla [179], ActorNet [198], SensorWare [199] and SAP allow agents to migrate within a WSN, enabling the update of a program on any node without consuming the energy of one specific node. In addition, multiple applications can be constructed using MAs [204]. However, there is one problem that makes MAs impractical when an agent (on one node) that monitors a specific container, if that container were to move out of the sensing range of the node the agent is on, it is difficult for the agent to know the new location of the container. In this situation, there are two general, naive approaches for the agent to figure out where the container is: random move/search and message passing. The former approach introduces a potentially significant number of migrations, causing undesirable levels of energy consumption as the agent searches blindly. The latter approach also has an energy problem when obtaining the new location, but, more than that, message passing using multihop communication in WSNs is not as reliable as that of wired networks. This problem may also be resolved by organizing a group of agents. However, this group must be able to migrate together, thus raising problems of group migration. Group migration of agents are not supported in Agilla, ActorNet and SensorWare, hence, a application developer must coordinate message passing between agents and synchronization their migration in the application logic. As a consequence, this approach introduces unreliable message passing and complex coding work. Hierarchical MA groups and their migration are proposed in wired networks [200-201]. These works assume a reliable, broadband network, and rely heavily on message passing to manage the group and its hierarchy. Thus these approaches are not applicable for WSNs.

The system must be designed in such away that it should consume minimum amount of resources for completion of a particular task. These resources are onboard SN power, computing time, integration time, and update program are critical in designing a high quality WSN.
6.2 Agent Architecture

We have developed a novel hybrid agent architecture based on the mobility and itinerary patterns of PMADE [194]. It is an adaptable model which allows agents to interact and modify external, as well as internal states. It also differentiates between an agent and a clone in order to locate clones by providing them with separate identities. The SAP MA consists of agent code, agent container, results, dynamic data and itinerary type. Thus,

\[ MA = \text{Agent Code, Agent Container, Results, Dynamic Data, Itinerary}; \]

\[ = \text{Control Code, Main Code, Task Code, Agent Container, Results, Dynamic data, Itinerary}; \]

Agent code is the set of instructions describing the task of the agent. It is further divided into several parts- such as main code, control code, task code, etc. The main code is the main part of the agent code, which initiates execution of agents on remote sites. Control code helps agents to take decisions about mobility and runtime solutions at the remote site when a fault occurs. Task Code represents the work agent has to perform at the remote site. It also contains its static data to accomplish the assigned task. Agent container is an information storage area that can be used by the agent at any time for reading and writing and always travels with it. It is also used to carry the Task objects (methods). Results are the output generated by the agent.

Dynamic Data represents the state variables which are required for the intermediate computation of the agent code at the next destination. Itinerary provides two options to the agent programmer depending on how the agent is managed. SAP agents can be categorized as BS/BN managed or agent managed.

A single SAP agent code supports one or more itinerary patterns. The itinerary patterns implemented in the agents’ code are external or internal. An External (explicit) itinerary is one in which the list of nodes to be visited is maintained outside the agent object (method list) in a separate data structure. It describes a set of nodes to which the MA is to travel and the work to be accomplished at each node. In an Internal (implicit) itinerary the list of nodes to be visited is maintained inside the agent object and processed by the agent by calling appropriate supporting SSA. Any agent roaming in this itinerary can dynamically change its itinerary type whenever required. This feature makes the agent fault
tolerant, while roaming on the network. We discuss these itineraries in more detail below.

6.2.1 Option-A: External (Explicit) Itinerary

Here the agent code is split into several pieces. Each piece of code and itinerary are secured using the protocols of PMADE. For agents with this option, their launcher can choose any mobility pattern (serial, virtual serial or parallel) [10,194].

In Virtual Serial and Parallel patterns an agent is controlled and managed by the BS/BN on which it is initially submitted, while in Serial pattern it is controlled and managed by every BS/BN in the itinerary one after another, i.e., when an agent finishes its task on one BS/BN, its control is transferred to the next BS/BN in the itinerary.

The main code is executed in all PEs in the serial itinerary, but in virtual serial and parallel itineraries, it is only executed at the BS/BN where the agent was initially submitted. There are as many task objects as there are nodes in the itinerary, each one to be executed at a particular PE. This feature makes the SAP agent flexible for accommodating different applications and makes it very useful in applications where execution is application dependent. The main code, which is needed to start the execution of a task object at every PE, is embedded in the BS/BN itself and used by the specific itinerary controller to execute the specific task object at the BS/BN. It need not be carried on the network and thus reduces network load.

6.2.2 Internal (Implicit) Itinerary

Here agent management methods are included in the agent code itself and are part of its architecture. This allows delegating specific mobility functions to the agent, freeing the BS/BN from this responsibility. Two cases are possible- Agent itinerary in pure implicit form or implicit with explicit property. In pure implicit agent architecture, the agent itinerary is merged in the agent code itself. A simple method call allows the agent to migrate to another BS/BN. The same code is executed by every BS/BN, so there is no need to keep it secret. Third parties cannot spy on it if migration is made secure. The code of such a MA is
In implicit with explicit architecture, the agent code is split into several pieces and secured as discussed earlier. There is a main code which is different from the one discussed earlier. It is application dependent and depends on the choice of an application developer. It executes in all nodes (PEs). There are as many task objects as there are nodes (PEs) in the itinerary, each one to be executed at a particular PE. This feature also makes SAP very useful in applications where execution is application dependent. The agent changes after every migration. This dynamic aspect of the agent allows several security mechanisms to be applied. Agent programmer uses complex agent mobility pattern in the design of the agent. The structure of the agent code is

\[ MA_i = \text{Control Code, Main Code, Agent Container, Static Data, Results, } S_o (H(\text{Control Code, Main Code})), PE_i, \text{Object}_{RH}, PE_{RH} | \text{NULL} \]

\( PE_i \) is the node where the agent migrates next. The agent that is sent to the next hop of the itinerary \( (MA_{i+1}) \) has the same structure. The last PE is identified because it has a NULL for next agent. The main code is executed by all BS/BNs before the specific task object. Programming is simplified by using this main code. This code does not depend on the BS/BN and is included only once. \( H() \) is the hash function.

This architecture allows several protection mechanisms in addition to the mechanism presented in pure implicit itinerary architecture to protect the agent code and data on an insecure communication channel. Further, code and itinerary are also protected for agents. The idea is to take advantage of distributed sealed object and re-router node(BN) in the agent itinerary to make the agent’s code secure and its itinerary secure and robust. Thus, only a portion of code is available to BS/BN for execution. The following is structure of the agent when using agent container, agent code, and itinerary protection.

\[ MA_i = \text{Control Code, Main Code, Agent Container, Static Data, Results, } S_o (H(\text{Control Code, Agent Container})), \text{Itinerary}_i \]
Itinerary_i = E_i(Agent Container, PE_i, PE_RHi) \mid NULL

where \( E_i(\cdot) \) is an encryption function using public key of the PE_i.

It should be noted that Agent container contains the list of objects, each of which is executed at particular node. This container also contains a special object that is used to re-inject the agent in the itinerary, when the next node in the itinerary is unreachable.

### 6.3 Agent and SN Namespace

This system maintains a local ID at each layer. These ID’s are down-streamed at the boot up time, i.e., the ID of system high up in the hierarchy will be sent to all the lower layers. The layer low in this hierarchy will prefix the parent layer ID to its own local ID and this combination forms the new ID of the layer. The lowest layer (layer 7th) will be at the node/PE level. Each agent/SN will have a 12-digit Hexadecimal ID. Format of ID is shown in Table 6.1. In this format we have 1-digits to represent a country code, 1-digit for state provincial, 1-digit for sub-state provincial, 1 Digit for next sub-provincial, 1 digit for next sub-region, 2 digits for next sub-region, 2 digit for smallest region beyond that area will not be divided and 3 digits for representing the agent/SN identification number (ID). Out of 3 bytes- 1 byte is used by representing BNs and remaining 2 Bytes for SNs, i.e., 64 K SNs/BN.

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Layer 4</th>
<th>Layer 5</th>
<th>Layer 6</th>
<th>Layer 7</th>
<th>Source of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country Code</td>
<td>state provincial</td>
<td>sub-state provincial</td>
<td>sub-provincial</td>
<td>sub-region</td>
<td>sub-region</td>
<td>smallest region</td>
<td>Agent ID</td>
</tr>
<tr>
<td>1 - 256</td>
<td>1 - 256</td>
<td>1 - 256</td>
<td>1 - 256</td>
<td>64 K</td>
<td>64 K</td>
<td>64K</td>
<td>Maximum range</td>
</tr>
<tr>
<td>1 BYTE</td>
<td>1 BYTE</td>
<td>1 BYTE</td>
<td>1 BYTE</td>
<td>1 BYTE</td>
<td>2 BYTE</td>
<td>2 BYTE</td>
<td>3 BYTE</td>
</tr>
</tbody>
</table>

**Table 6.1:** Format of 12 digit hexadecimal id
This format enables a total population of SNs 16M nodes in a smallest area where deployment of agent may be done. Thus, by using this assumption \(2^{88}\) agents/SNs can be uniquely identified across a country and \(2^{96}\) SNs/agents across the world which is quite a good assumption to manage a whole world using agents and SNs. The first time registration will be only through the layer 7. This process will append the agent/SN ID with the system ID thus giving a unique ID to each SN/agent. For example, see Table 6.1 where we are getting “111111111111” for SN X belonging to country India (Layer 1), Haryana State(Layer 2), Ambala District(ayer 3), Barara tehsil(Layer 4), Narayangarh block(Layer 5), Panjlasa panchayat(Layer 6) and Seembla Village (Layer 7). Here “1” is a code of India and personal ID of SN X is “1111”. This SN will be under BN 1 in the said area.

This ID will be up-streamed towards the root system via the branch of the hierarchical tree, thus making identity available to each of its parental ancestor layer. This way every SN/agent has a unique ID and will communicate to any ancestor layer via its local layer. Any information may be collected or disseminated across the whole network or particular region, etc. In case of certain failure, maliciousness the node/agent can directly communicate with its next ancestor higher in the branch because of P2P nature of this system. This system also prevents any unauthorized access by a SN/agent to any other system to which it dose not belong. This is because the ID of that SN/agent will be supported only by the branch to which it belongs. In case of roaming of mobile BS, the system may be enhanced to provide only limited privileges to the agents like query, etc.

6.4 System Model And Assumptions

We will consider MA in multihop environments with the absence of a centralized control. Without region head, we have to answer the following questions. (1) How is an MA routed from BS to source of information (SN), from SN-to-SN, BN-to-BN, and from SN/BN to BS in an efficient way? (2) How does an MA decide a sequence to visit multiple source nodes (SN/BN)? (3) If the sensory data of all the SNs cannot be fused into a single data packet with a fixed size, will the MACP still perform more efficiently than the CSCP? How about in the environments where the source nodes (SN/BN) are not close to one another, and the sensory data do not have enough redundancy?
A MA is defined as an entity of four attributes: identification, itinerary, data container, and method. More details about MA architecture are given in Section 6.2}. Brief illustration of these attributes is as follows:

1. **Identification**: is in the format of 2-tuple \((O_i, A_i)\), where \(O_i\) indicates the identification number of its owner and \(A_i\) the agent identification assigned by its owner. \(A_i\) is assigned as per rules defined in Section 6.3}. Each MA can be uniquely identified by this identification. We have used \(MA_{O_i,A_i}\) to indicate different MAs.

2. **Itinerary**: includes itinerary information assigned by its associated processing element (PE) when dispatched. This policy is defined as per selection of agent architecture and mobility management policies.

3. **Data Container**: agent's private data buffer which carries integration results and itinerary information.

4. **Method**: the implementation of the data fusion algorithms.

Let \(PE_i\) represent a certain PE with an identification \(i\) that is in charge of the surveillance of a certain area. Let \(\{MA_{O_{1,i}},...,MA_{O_{m,i}}\}\) represent a group of \(m\) MAs dispatched by \(PE_i\). Without loss of generality, it is assumed that each \(MA_{O_{i},A_i}\) visits the same number of SNs, denoted by \(n_s\). In this scenario, the benefit introduced by the use of MAs largely depends on the planning of the agent architecture and agent itinerary. We choose an agent architecture and agent itinerary that consumes the least amount of resources such as time and energy in order to finish the fusion task.

6.5 System Architecture
A general Adaptive and Hierarchical Distributed System (AHDS) consists of a set of SNs, a set of PEs, and a communication network interconnecting the various PEs {Figure 6.1}. This model is adaptive in nature and hierarchical in architecture. One or more SNs are associated with each PE. One SN can report to more than one PE, if required. A PE and its associated SNs are referred to as a cluster. We have assumed that the WSN environment is divided into cluster of clusters (we
refer it as network domain), clusters (subnetworks) and SNs (local sites). Figure 6.1 shows a logical view of an agent based WSNs. There is a CPE known as domain management server (DMS) in each network domain which has information about all other DMSs in the network. It also has information about all the cluster in the network domain. It is responsible for maintaining uniqueness of names of clusters, which are part of that network and helps to identify the cluster in which an agent is present. Each DMS maintains a Domain Agent Database (DAD), for information about the current location of all agents which were created in that domain or transited through it.

Data are transferred from SNs to their associated PE(s) where the data integration takes place. PEs can also coordinate with each other to achieve a better estimation of the environment and report to higher level PEs. Communication between clusters of clusters is done by central PE. Thus, from the architecture shown in Figure 6.1 it is clear that far SNs can also become of the part of the network through their central PE (CPE) and Cluster head PE (CHPE).

Each cluster has a CPE known as BS it is similar to the mobile service station (MSS). The BSs are connected to each other by wireless network. A BS can be in wireless communication with the MAs and SNs in its cluster means an agent executing SN is similar to wireless mobile device. Every cluster maintains information about all SNs which are part of that cluster. A SN can be a member of an existing cluster or can start in a new cluster. In each cluster, a Cluster Sensor Database (CSD) is present at a CHPE which runs at the BN of a subnetwork. It also contains location information about each agent which was created for that cluster or transited through it. The CHPE acts as the Name Server (NS), which manages the CSD. NS is responsible for maintaining uniqueness of names of all SNs and MAs running/created for that cluster. When a new agent is created, the user assigns a name to it by registering in the DAD of its birth CPE(BS).

The location of an agent can change with time. It may move from its present cluster to a neighboring cluster while participating in a communication session or it may stop communicating with all agents for a period of time and then pop-up in another part of the network. An MA can communicate with other agents which are running on other SN, only through the CHPE of the cluster in which it is present. If an agent (which is running on a SN) wishes to communicate with an agent, first it has to determine the location of the MA (the cluster in which the MA is
currently residing). This location information is stored at CHPE/CPEs. Depending on the frequency of location updates, this location information may be current, or out-of-date. Once the location of the MA has been determined, the information is routed through the CHPE of the cluster in which the MA is present. Then the CHPE relays the information to the destination MA over a wireless channel via CPE/CHPE. We assumed that CHPEs act as location servers (LSs). Hence, all the CHPE collectively maintain the location directory. Each CHPE runs SAP Directory Agent which maintains necessary information in the form of database.

Agent migration from one network domain to another is always accomplished through the CPE. During inter domain migration the agent has to update location information in the DAD of the present domain and register in the DAD of the target network domain.

![Figure 6.1: Adaptive and Hierarchical Distributed System (AHDS) for WSNs](image)

For intra cluster migration, it has to update its location information in the CSD of that cluster. This is an Intra Cluster Location Update. During inter cluster migration, the agent has to update the location information in the CSD of present cluster and register in the CSD of the target cluster, specifying the SN in that cluster to which it is migrating.
6.6 Mobility Management

MA itinerary can be determined either statically or dynamically, i.e., it can be calculated either before the agent is dispatched or while the agent is migrating. Refer Section 6.1 for details for types of itinerary. Dynamic itinerary planning is more flexible, and can adapt to environmental changing (SN ups and downs) in real-time. However, since the itinerary is calculated on the fly, it also consumes more computation time and more power of the local SN. On the other hand, although static itinerary cannot adapt to the network change, it is able to save both computation and power since the itinerary only needs to be calculated once. Computation-efficiency, power-efficiency, and flexibility are three parameters that cannot be satisfied at the same time. In AHDS CHPE (BN) maintains information about SNs in their cluster (region) as well as MAs running in their control. Using these parameters two vibrant improvised route preparation strategies are developed for agents.

6.7 Improvised Vibrant Itinerary Scheduling Algorithm

The itinerary attribute in $MA_{O,A}$ is in the format of 4-tuple $(B_{O,A}, r_{O,A}, L_{O,A}, U_{O,A})$:

- $B_{O,A}$: The center coordinator of a region/cluster area to be traveled by $MA_{O,A}$;
- $r_{O,A}$: Approximate the radius of the region/cluster area;
- $L_{O,A}$: The list of BNs $MA_{O,A}$ needs to visit in one trip. BN maintains local itinerary information as per requirement of the applications. As soon as agent reaches at BN it updates its itinerary. Noted that $L_{O,A}$ is an unsorted list when $MA_{O,A}$ is dispatched the first time. A sorted $L_{O,A}$, however, can be reused after its first trip. But not always possible due to energy misuse of BNs.
- $U_{O,A}$: the list of BNs $MA_{O,A}$ deferred for future consideration. This list is generated by using unreachable destination. It is assumed that in future unvisited destination may be part of the network.

In the model we have considered that set of SNs/BNs will work on the same task and only one node will be active and rest will be in slip mode. It is assumed...
that $B_k$ BN. It is also assumed that per BN there are $m$ SNs of interest. The following algorithm is used compute the itinerary.

**Improvised_Vibrant Itinerary_Scheduling** ($B_{O_0, A_0}, r_{O_0, A_0}, L_{O_0, A_0}, U_{O_0, A_0}, B_k$) 

{  
/* where $B_k$ - current location, where $k \in [0,n_B]$, $k = 0$ means $MA_{O_0, A_0}$ is at $PE_i$. $n_B$ number of BNs in the Network */

while (true) do{ 
  switch the value of $k$ do{
    case $k = 0$
    {
      find BN $B$ with the smallest distance $d(S, B_{O_0, A_0})$
      from $L_{O_0, A_0}$;
      $D = B$;
    }
    case $k = n$
    {
      $D = PE_i$;
    }
    otherwise
    {
      find BN $B$ with the smallest distance $d(B_k, B)$ from
      the rest of $L_{O_0, A_0}$;
      $D = B$;
    }
  }
  if $D$ is active then{
    migrate to $D$;
    $k = k + 1$;
    For (i=1, $m$) {
      Find SN $S_i$ with smallest distance $d(D, S_i)$
    }
  }
}

}
If $S_i$ is of the interest then record the information.

break;

} else {
    delete $D$ from $L_{O_i,A_i}$ and put into $U_{O_i,A_i}$;
    $k = k + 1$;
}

If $L_{O_i,A_i}$ is empty then repeat the above process for the unvisited list of BNs available in $U_{O_i,A_i}$ without maintaining unvisited list of SNs.

6.8 Improvised Vibrant Optimal Itinerary Scheduling Algorithm

The problem of optimally planning the route for MAs from probability point of view, where $n_s$ sites and these sites are distributed into $n_b$ groups at which a certain task might be successfully performed. The probability of success at each site(SN) is $p_i$. We study data fusion in WSN by designing MA itinerary, where computation time and power consumption are the two major concerns. On the other hand, as large amount of SNs are deployed, redundancy in the SN readouts are used to provide error tolerance. Multi-resolution techniques are popularly used such that when the accuracy requirement is not high, processing can be carried out at a coarser resolution to save both the computation time and the energy consumed. Furthermore, when a MA is migrating around the WSN and accumulating SN readouts, if the accuracy of the result has reached the requirement of a certain task, the agent can return to the processing center (BN-currently where an agent is working, from there it will return back to BS) directly without finishing the entire trip. Due to the complexity of this approach, it is better applied before the agent is dispatched.

Let $\Delta_{O_i,A_i}$ be the accuracy requirement for a specific task (e.g. accuracy of range of detected target) that agent $MA_{O_i,A_i}$ carries out. Each agent plans to visit $k = 1, \ldots, n_b$ BNs and $i = 1, \ldots, n_s$ SNs per BN. It assumed that each SN measures the same set of parameter(s), and the readout of each parameter is a range of real
numbers \([a_i, b_i]\). Let \(p_i\) be the percentage of SN readout that includes true values of the environment, \(q_i\) be the percentage of power remaining on the SN. Let \(t_{pr}\) be the processing time spent at each SN and \(e_i\) be the power consumed at each SN which are the same for all the SNs. Let us also assume that the processing time is much longer than the agent traveling time, so that it can be ignored. \(L_{O_A\cdot A}\) represents the list of BNs that \(MA_{O_A\cdot A}\) should visit and \(U_{O_A\cdot A}\) is the list of deferred BNs. As the migration goes on, the width of parameter estimation \([a_i, b_i]\) should be getting narrower and narrower, while the accuracy approaching \(\Delta_{O_A\cdot A}\).

We have used the change made between adjacent readout ranges \(|(b_i - a_i) - (b_{i-1} - a_{i-1})|/|(b_{i-1} - a_{i-1})|\) to represent the accuracy. If the accuracy has reached \(\Delta_{O_A\cdot A}\), then the MA does not need to go through the rest of the BNs, instead, it can return to the PE (BN/BS) directly which saves both migration time and network bandwidth. An optimal list of BNs \(L_{O_A\cdot A}\) and \(U_{O_A\cdot A}\) is searched such that the cost of computation time and the relative energy consumption with respect to each node (BN/SN) itself reaches the minimum. An objective function as Eq. 6.1 is derived, where \(H_T\) is the time consumed and \(H_p\) is the relative energy consumed, \(\alpha\) is a positive real number that is less than 1. It indicates the tradeoff between \(H_T\) and \(H_p\).

\[
H(L_{O_A\cdot A}) = \alpha H_T + (1 - \alpha) H_p
\]

where
\[
H_T = \sum_{k=1}^{n_k} \sum_{i=1}^{m_i} t_{pr} p_i \quad \text{and} \quad H_p = \sum_{k=1}^{n_k} \sum_{i=1}^{m_i} e_i q_i
\]

This optimization problem can be easily solved by genetic algorithm. With the optimal itinerary \(L_{O_A\cdot A}\) obtained, Improvised Vibrant Optimal Itinerary Scheduling algorithm describes how it is used subject to the requirement of accuracy.

\[
\text{Improvised\_Vibrant\_Optimal\_Itinerary\_Scheduling} \left(L_{O_A\cdot A}, \Delta_{O_A\cdot A}, a_i, b_i, \omega_{old}\right)
\]
/* Where \( L_{O_i,A_k} \) (optimal itinerary for agents to reach to BNs, now from a BN agent will reach to individual SN belonging to a BN), \( \Lambda_{O_i,A_k} \) (accuracy requirement), \([a_i, b_i]\) (SN readout of a specific parameter), where \( k = 1, \ldots, n_k \), \( i = 1, \ldots, m_i \), \( \omega_{old} \) (the width of the previous SN readout). */

\[
\omega_{old} = \infty ;
\]

\( k = 1 ; \)

while \( k \leq n_k \) do{
    \[
    k = k + 1 ;
    \]
    \( i = 1 ; \)

while \( i \leq m_i \) do{
    \[
    i = i + 1 ;
    \]
    migrate to \( L_{O_i,A_k} \);

obtain the SN readout as \([a_i, b_i]\);

if \( ((b_i - a_i) - \omega_{old})/\omega_{old} < \Lambda_{O_i,A_k} \) then
    break;
else{
    \[
    \omega_{old} = b_i - a_i ;
    \]
    \( i = i + 1 ; \)
}

Migrate to \( L_{O_i,A_k} \)

\( k = k + 1 ; \)

return to the processing center (BN/BS);
6.9 Improvised Vibrant Itinerary Scheduling Genetic Algorithm for BS and BN

The amount of time an agent spends processing data at each node/BN and the time it takes to migrate is predictable. In practice, such information may be estimated based on empirical measurements, where the measured time per node includes the processing, migration, and sleeping time. This knowledge is necessary to ensure that the network is explored within a certain time.

Given a set of BNs $B$ and set of SNs $s \in S$ to monitor a field $F$, BNs and SNs form a WSN in which SNs cooperate to send a message from any node to BS via BN and vice-versa. A BN $b \in B$ may act as router and a SN $s \in S$, router or both. In addition SNs are assumed to be heterogeneous in terms of energy, communication range and sensing range. Some of the BNs might be mobile in which agents can move from one place to another while others are stationary. BNs are high energy & computing power nodes. The BNs are also called smart nodes. BNs can measure the required parameters such as channel capacity and reliability and current residual energy. This data is used to select next best hop. A threshold value is set for each parameter where a node $s \in S$ reports the changed parameter to BN if the parameter value is increased more than the threshold (s). SNs report their sense data based on the monitored field activities as well as according to the BS requirements. The BS is responsible of setting the reporting rate during the configuration phase. Nevertheless BNs assumed to have enough storage and capacities to discover their neighbors. A single BS collect information of the entire network by using MA which are visiting every region/cluster with through BN within bounded time while minimizing energy consumption. This can be done using clones of a MA operating autonomously and in parallel following unique routes. These routes start and end at the node (BS/BN/SN).

Many applications do not need every node to be visited, especially if the nodes are physically close and at a certain distance. For this purpose a density variable $d$ which specifies how many of the nodes within the network need to be visited for the network to be considered explored. Specifically, a network is considered discovered if all nodes (BNs/SNs) $n_b \in B$ and per BN $m_s \in S$ ($n_b \cdot m_s = n_s \in S$) have either been visited, or have a neighbor within $d$ hops that has been visited. If $d = 0$, the examination problem reduces to
one where every node needs to be visited.

The time constraint $t$ is specified by the maximum length of a path, expressed as the number of nodes. We do not use an actual time since the amount of time an agent spends on each node, and the amount of time it takes an agent to migrate, is predictable.

The cost of agent $m_a$, visiting $(n_b \cdot m_z = n_s \in S)$ nodes, following a certain path $p$, is denoted $c_{m_a}(n_b, m_z, p)$. It is defined in terms of the weight of the agent, $w_{m_a}(n_b, m_z, p)$, and the amount of energy remaining on the BN $e_{n_b}$ and SNs, $e_{n_s}$, as follows:

$$c_{m_a}(n_b, m_z, p) = \frac{w_{m_a}(n_b, m_z, p)}{e_{n_s}} \quad \ldots \quad (6.2)$$

Note that as the amount of energy available on a node increases, the cost decreases and vice-versa. This is necessary to balance the residual energy of the nodes and thereby extend the lifetime of the network.

The weight of agent $m_a$, $w_{m_a}(n_b, m_z, p)$, is a function that returns the weight, in bytes, of agent $m_a$, when visiting node $(n_b \cdot m_z = n_s \in S)$, following path $p$. The weight of an agent depends on the size of its code and the amount of data that it carries. The size of its code is fixed, whereas the size of the data varies every hop and depends on what the application does upon visiting each node. Finally, let $P$ be the set of all paths and $P_{n_b}$ be the set of paths to reach BNs, $P_{n_s}$ be the set of paths to reach SNs, that include BNs $n_b$, and SN $n_s$ where $P_{n_b} \subseteq P$, $P_{n_s} \subseteq P$, we can formulate the following minimization problem:

$$\min \left( \sum_{n_b \in S} \sum_{p \in P_{n_b}} \frac{w_{m_a}(n_b, p)}{e_{n_b}} \right) \quad \ldots \quad (6.3)$$

$$\min \left( \sum_{n_s \in S} \sum_{p \in P_{n_s}} \frac{w_{m_a}(n_s, m_z, p)}{e_{n_s}} \right) \quad \ldots \quad (6.4)$$

if we merge equations (6.3 and 6.4) we get the following
Such that \( \forall p \in P : |p| \leq t \) and the WSN is covered with density \( d \).

The following pseudo-code of the genetic algorithm (GA) runs by every BN and the BS in a network. This algorithm takes as input the definition of a SN network \( S \) (i.e., the topology), the time constraint \( t \) (i.e., the maximum number of nodes of the longest route), the density \( d \), and the specification of the varying size of the agents \( w_{ma}(n_b, m_s, p) \). The outcome is the number of agents and their routes, such that the information collection is achieved in an energy-aware manner and within a time deadline.

\[
\begin{align*}
&\left( \sum_{n_b \in B} \sum_{m_s \in S_b} \sum_{p \in P_{n_b}} w_{ma}(n_b, m_s, p) \right) / e_{n_b} \quad \ldots \quad (6.5)
\end{align*}
\]

IVIS_Genetic_Algorithm_GA \( \left( B, m_s, d, t, w_{ma}(n_b, m_s, p) \right) \)

CreateInitialPopulation\((n_b, m_s)\);

For gen=1 to MaxGen

Cross();
Mutarate();
NaturalSelectmon();
AddNewIndividuals\((n_i)\);
endFor

return Agent Routes

endGA

An initial set \( P \), of solutions is generated. These solutions are a collection of random paths that cover the network, and are generated by repeatedly calculating paths until the whole network \( S \), have been covered with a density of \( d \). Each path is selected by starting at the BS/BN and iteratively selecting a random neighbor until the collection point is reached again. Only those neighboring nodes whose minimum distances (in number of hops) to the collection point are not greater than the remaining time will be selected. Since the time constraint is expressed as the maximum number of nodes per path, comparing a nodes distance to the collection point with the deadline is trivial. All feasible neighbors are equally likely to be selected, and the initial size
of $P$, $|p|$, is a design parameter.

At each iteration a new generation is added to $P$ by selecting $x$ individuals to be crossed. Two individuals are crossed by taking a portion of each individual’s math and combining them to form a new individual. Depending of how those individuals are selected, and which portion of their paths are crossed, new more efficient paths may be formed. Mutation is done by crossing an individual with itself.

All solutions are evaluated by the function being minimized in 2. Upon evaluation, $k_{n_0}$ % of the worst individuals are randomly selected and removed from the population. Note that to avoid losing the best solutions generated, the five best solutions are never removed, irrespective of $k_{n_0}$.

### 6.10 Implementation and Performance Study

We have performed different types of experiments for the evaluation of AHDS. In the first evaluation we have tested AHDS on a real network. We deployed it on a network consisting of 25 machines running SAP. Having a real implementation allowed us to define realistic $w_{ms}(n_i,m_s,p)$ functions and empirically determine a translation from the deadline expressed in seconds to the maximum number of hops per path. In the second experiments we have discovered the network using AHDS for different size of network. In the next three experiments we have evaluated the performance of AHDS, Non Agent Based Multipath Routing [202], and Agent Based Multipath Routing [203]. We have used MATLAB for the simulation and parameters are adjusted to use linear battery and IEEE 802.11 as MAC layer with only one BS and several BNs per network. Simulation is done for heterogeneous set of nodes with different sensing and communication ranges and the initial energy. The running time for any experiment is assumed open till the network is disconnected. Therefore, they will have equal effect on the routing conditions. These values are application specific and might vary accordingly.

#### 6.10.1 Event notification When Size of Network Grows

Figure 6.2 shows that AHDS outperforms in comparison to Master/Slave [9] and message passing techniques. AHDS uses BN (high energy node) which is directly
govern by BS and maintains all the current information about all the events happening in the network. Frequent program update of the nodes can also be done with the help of BN. Agent migrates from one region to another through the BN only. SAP uses 7 layer hierarchical model for managing WSN. A BN makes communication with sensing nodes over one hop distance only. For multi-hop communication it uses only BN-to-BN or BN-BS communication which having large resource of energy.

![Success rate of event notification](image.png)

**Figure 6.2:** Success rate of event notification

### 6.10.2 Network Discovery

We ran four different experiments with \( t = 4, 7, 12, \) and 15 to measure actual amount of time an agent takes in migration from one node another and from one part of the network another. The IVIS_Genetic_Algorithm_GA generated thirty two initial solutions, executed for eight iterations, and added four new solutions every iteration thereafter. For each experiment, we averaged the four runs and plotted the percentage of the network explored versus time. Figure 6.3 shows that the IVIS_Genetic_Algorithm_GA produces faster results with less variance when the maximum path length is more restrictive (shorter). With a time constraint of \( t = 4, 7, 12, \) and 15, MAs spent an average of 6sec, 7.25sec, 11.5sec and 14.5sec in the WSN, respectively. The average time it takes to move an agent one hop is 1.0328 sec.
6.10.3 Effect of Number of Messages on the Average Dissipated Energy

We study the effect of number of message on the average dissipated energy for NABMR, ABMR and AHDS. 1000 nodes are uniformly distributed in an area of 1000 X 1000m². The number of messages changes from 100 to 600 messages, and the number of paths generated is changed from single path to 10 paths with each experiment for NABMR, AMBR and AHDS as shown in Figure 6.4. The average dissipated energy for NABMR is more from AMBR and AHDS. The AMBR and AHDS show less energy dissipation rate due to less computation in taking the routing decision. Both ABMR and AHDS have taken less energy from the SNs due to the huge reduction in the parameters update messages send from each node to its neighbors. This value is more reduced in case of AHDS because most of the decision is taken by BN. For instance, in case of 600 messages that are sent from different source nodes to the BS, ABMR consumes on average 23.33%, NABMR consumes 29.83%, and AHDS consumes 20.25% of the SNs energy.
6.10.4 Message Delivery Rate in the Network

We conducted experiment to compare NABMR, ABMR, and AHDS in terms of message delivery rate in the network as shown in Figure 6.5. The delivery rate of AHDS and AMBR on average is almost the same. However, at some points such as 450, 500, 550, 600, AHDS outperformed in comparison to NAMBR due to selecting large number of paths condition based on the information importance. On the other hand, ABMR seems to over perform in comparison to NABMR in terms of the message delivery rate. It is observed that in NABMR some of nodes might die because of energy depletion due to large number of messages required to be sent. The delivery rate of NAMBR is reduced sharply due to high message loss. In case of NAMBR on average almost 12% of the messages are dropped. However, ABMR and AHDS are well performed in such cases where the delivery rate is almost 96.5% and 97.3%, respectively. There is 3.5% and 2.7% loss in ABMR and AHDS respectively. And it is noticed that it is due to congestion in the network for some periods of time and prevents ABMR and AHDS from finding any alternative paths.

Figure 6.4: Average dissipation energy percentage per node
6.10.5 Effect of Number of Nodes on the Message Delay

We study the effect of increasing the number of nodes on the average message transmission delay. As shown in Figure 6.6, we measured the delay in message delivery observed in the network. AHDS seems to slightly outperform in most of the cases because BN helps to BS in collecting the route information. On the other hand, AMBR and NABMR are performing almost the same with a little delay in NABMR due to the initial step in collecting the neighbor’s information. For instance, NABMR has a delay of 4.1 seconds when 600 nodes are used while AMBR needs 3.7 seconds average to deliver same number of messages. AHDS has a delay of 3.3 seconds average to deliver same number of messages. In addition, as mentioned, some messages will never be delivered due the unavailability of path. ABMR and AHDS are performing almost the same with few milliseconds more delay in ABMR. Overall ABMR and AHDS are adapting to the network conditions.
6.11 Results and Discussion

In MADFWs agent has liability to invite some nearby slave SNs to cooperatively position the object and inhibit other irrelevant SNs from tracking the object. As a result, the communication and sensing overheads are greatly reduced. We compared the performance of NABMR, ABMR and AHDS algorithms through a set of experiments. Results show the effectiveness of NABMR and ABMR in terms of energy dissipation, delivery rate, and delay. We compared the performance of NABMR, ABMR and AHDS through a set of experiments. Our experiments show the effectiveness of NABMR, ABMR and AHDS in terms of energy dissipation, delivery rate, and delay. This happens because BN maintains required itinerary information for the agent and no need to do this job at BS level.

6.12 Summary

In this chapter, we have presented the environment architecture, agent architecture and agent/SN namespace of SAP. One of its novel features of the presented model is its flexibility and extendibility. Model allows the agent programmer to design different agent architectures based on different agent mobility patterns. Agent/SN namespace and presented model is useful to monitor the whole world using the WSNs. Agent/SN namespace divides every country into the 7 layers in
hierarchical manner (7 types of regions). In the next chapter fault tolerant model of SAP will be discussed.