Title: Technical Report	
Project Title: Connectivity and coverage of underwater sensor networks	
Name: Md Asadul Islam	
Date: April 2, 2014	

Abstract

1 Introduction

Interest of monitoring underwater environment is growing day by day. Sensor network is a promising tool for underwater environment monitoring because of its remote monitoring and control technology. Underwater Sensor networking is using for military surveillance[8], disaster prevention, assisted navigation, offshore exploration, tsunami monitoring and oceanographic data collection. Coverage and connectivity are two important aspect of underwater sensor network. One can determine the quality of surveillance of a underwater wireless sensor network from network coverage. Many to one data flow from a set of sources to a common sink over a tree based routing topology, is a fundamental communication architecture of a underwater sensor networks. Connectivity is an

Table 1: Physical Layer Parameters

Parameter Name	Value	Description
Standard	IEEE 802.11	Can be used as medium access control protocol (MAC) for underwater acoustic communication with modification[5].
Range	1m-10km [6]	Short range modem provides very high bandwidth (typically MHz or more) where long range modem provides low bandwidth(several bps)
Data Rates	bps - MBps	Varies from standard to standard from 5 bps to 19200 bps.
Energy Source	battery, external sources	Primarily battery is used as a power source but for most of the standard external power supply can be used as a power supply.
Well-known models	LinQuest Inc, Hydro International	UWM1000,UWM2000,S1510 Radio Modem,Digital Underwater Modem UM 30
Path loss model		Path loss of underwater acoustic communication channel depends on the distance between the transmitter and receiver and signal frequency [**?**].

important issue for UWSN in-order to perform localization [16], [17],[4], routing [12],[15],[9]. In this task we are interested to measure the connectivity of UWSN. We are considering a network consists of anchored sensor nodes or free moving sensor nodes. The connectivity we are interested, can be

- All node connectivity: every node is connected with sink.
- 2-terminal connectivity: is the connectivity that two given vertices, called the source and the sink, can communicate.

- k-terminal connectivity: compute the probability that every operational pair of sites in k-can communicate with each other.

- In [1] authors propose a distributed node deployment scheme which can increase the initial network coverage in an iterative basis. They assuming that the nodes are initially deployed at the bottom of the water and can only move in vertical direction in 3-D space. The idea is to relocate the nodes at different depths based on a local agreement in order to reduce the sensing overlaps among the neighbouring nodes. Redundancy is observe by one of the node called Leader. It utilizes vertex colouring problem formulation in-order to determine coverage overlap. The nodes continue to adjust their depths until there is no room for improving their coverage. They consider both tether and and untethered architecture for node deployment.

- The work of [17] highlighted the localization of a sensor node. According to their model, the moving speed of underwater node changes continuously and shows semi-periodic properties. They predict the future mobility pattern from the past mobility information. The nodes maintain connectivity in-order to provide accurate localization information.

- In [3] authors proposes Meandering Current Mobility Model (MCM) which is able to capture the physical movement of the sensor nodes with ocean currents. In MCM, nodes are moving by the effect of meandering sub-surface currents and vortices. According to MCM, there is a strong correlations between nearby sensors. Vertical movements in ocean are negligible with respect to horizontal ones. Thus, in their model they neglect vertical displacement which makes mobility in 2D. This model is more realistic than other mobility models [7][17] for UWSNs since nodes are drifted according to the movement of the ocean. They studied dynamic coverage and connectivity as a function of time under the MCM model. They also consider the effect of different deployment strategies on network coverage and connectivity.

- Connectivity and coverage are important issue for UWSN. In [2] authors tackling the problem of determining how to deploy minimum number of sensor nodes so that all points inside the network is within the sensing range of at least one sensor and all sensor nodes can communicate with each other, possibly over a multi-hop path. They used sphere-based communication and sensing model. They place a node at the center of each virtual cell created by truncated octahedron-based tessellation. They provide solutions for both limited and full communication redundancy requirements.

- The work of [10], [11] extends [3] and proposes a ring like motion to capture the basic of sea flow. According to this model, there are two types of mobility, uncontrollable mobility which breaks the coverage of sensor network and controllable mobility which restore the coverage of sensor network. In the ring-like model [10], they consider local variety and main circulation in-order to capture some characteristics of water bodies. They use probability in-order to determine the next position of a node.

- In [13], [14] author proposed a polynomial algorithm to solve Steiner tree problem. The Steiner tree problem in an undirected graph is the problem of finding a tree spanning a pre-specified set of vertices at minimum cost where the cost of a tree is equal to the sum of the cost of its edges.

We device a scheme where we approximate the UWSN by using a special types of graph which is called partial k-tree. According to our scheme each node can be located into a set of regions with certain probabilities which is known as probabilistic locality set. The locality set of a node at a certain time instant is known from the initial location of the node and underwater current.

We introduce a problem called the Connectivity in Underwater Sensor Network (CUSN) problem for Underwater Wireless Sensor Network (UWSN). We have a set of nodes and we have a probabilistic locality set of each node. We would like to compute the probability that the network is connected.

2 System Model and Problem Formulation

In the section we are going to describe notation, connectivity and coverage model and node connectivity model. Finally we present problem formulation for $Conn(G, \mathbf{R})$.

2.1 Notation

In this section we are going to describe some of the notations, we used throughout this paper.

- $G = (V, E)$: the underlying connectivity graph of a given UWSN.
- $R_{tr}(v)$: the transmission range for node v.
- $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$: the locality set for node v.
- $\mathbf{R} = \{R_v : v \in V\}$: the set of all the locality sets of all node in G.
- $p(r_{(v,i)})$: the probability that node v is in region $r_{(v,i)}$.
- PES : an ordering $(v_1, v_2, ..., v_{n-k})$ of the nodes of G for every $i \in \{1, 2, .., n-k\}$:, the node v_i is simplicial in the subgraph of G.
- $\{K_{(v,1)}, K_{(v,2)}, ..., K_{(v,k)}\}$: are k-cliques associated with vertex v in a k-tree, where v is a simplicial vertex.
- $K_{(v,base)}$: the base clique to which v is attached and it is formed by all adjacent vertices of v.
- $\{T_{(v,1)}, T_{(v,2)}, ..., T_{(v,k)}\}$: tables associated with clique $\{K_{(v,1)}, K_{(v,2)}, ..., K_{(v,k)}\}$ respectively.
- $V(G')$: the nodes of a subgraph G'.
- $Reach(r_{(x,i)}, r_{(y,j)}) = 1$ if node x in region $r_{(x,i)}$ can reach node y in region $r_{(y,j)}$ else 0.
- $Conn(G, \mathbf{R})$: The probability that the nodes of G are in a state where all nodes are connected $= \sum Pr[S : S$ is a connected state of nodes in $G]$.

2.2 Connectivity and Coverage Model

Underwater Wireless Sensor Network (UWSN) is modelled by a graph G of (V, E) where $V = (v_1, v_2, v_3, .., v_n)$ is a set of vertices and E is a set of edges. The transmission range for node $v \in V$ is $R_{tr}(v)$. One node can transmit data to other node if they are within transmission range of each other.

2.3 Location Probability Model

We are considering each sensor node $v \in V$ can be located into a set of regions, $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with certain probabilities, $p(r_{(v,1)}), p(r_{(v,2)}), ...$ which are called location probabilities. Mathematical sum of a node's location probabilities are 1 but in some cases we are considering a set of regions with high probable values where sum of locations probabilities can be less than 1. Location probability of a node in a specific region, is independent of location probability of that node located into other regions. Also location probabilities of one node is independent of the location probabilities of other nodes. Also we are considering the probable locations of a node can be contiguous as well as non-contiguous.

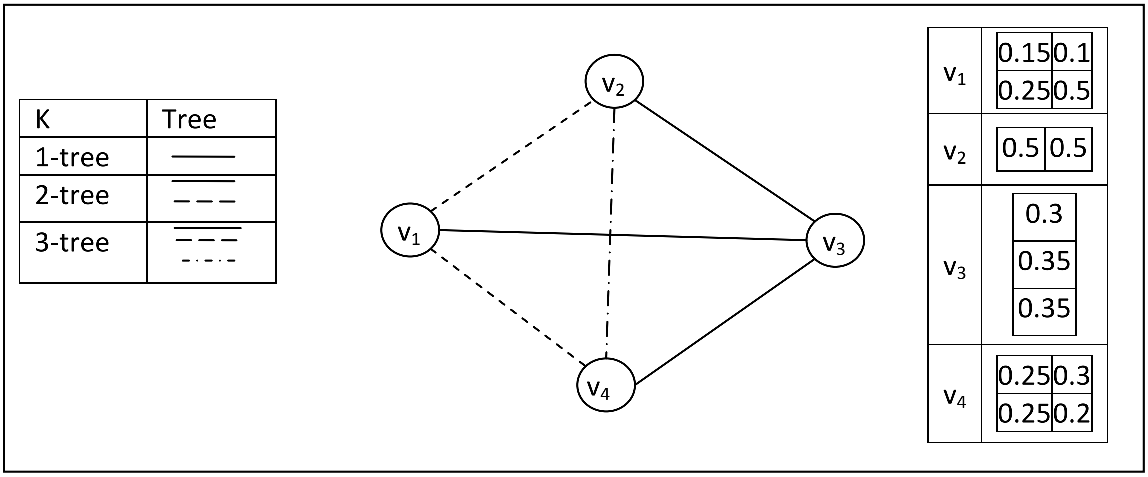

Figure 1: A partial 3-tree with 4 nodes.

Example 2.1. Fig. 1 illustrates a network of 4 nodes. Locality set of node v_1, v_2, v_3 and v_4 has $4, 2, 3$ and 4 regions. We are also considering v_4 as a sink node. Transmission range $R_{tr} = 8.5$ unit. By using this transmission range node v_1 from any of its probable regions can communicate with all probable regions of node v_2 and v_3 but not with node v_4. Similarly node v_2 from all probable regions can communicate with all probable regions of node v_4 and node v_1 but for node v_3 only the region with probability value 0.3. Node v_3 can communicate with node v_4 when v_4 is located in the regions with probability values 0.25 and 0.2. ■

2.4 Node Connectivity Model

If one or more regions in the locality set of node $x \in V$ is within the transmission radius $R_{tr}(y)$ of one or more regions in the locality set of node $y \in V$ and vice-versa then node x is reachable from node y.

A network state, S arises when each node is assigned to a specific region from it's locality set. A connected state, S_{conn} is a network state where each node can communicate with others.

2.5 Problem Statement

Now we formally define the problem.

Definition (The $Conn(G, \mathbf{R})$ Problem)**.** Let G is a UWSN where each node $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)}$ where $i = 1, 2, ...$ Also $R_{tr}(v)$ is the transmission radius for node $v \in V$. we would like to find the probability $Conn(G, \mathbf{R})$ that each node is connected. ■

An exhaustive approach to compute $Conn(G, \mathbf{R})$ can be

- compute $Reach(r_{(x,i)}, r_{(y,j)})$ for every pair of region $r_{(x,i)}, r_{(y,j)}$ where $(x, y) \in V$
- find all network states $\mathbf{S}$ for the given graph G.
- find all connected states $\mathbf{S}_{conn} \subset \mathbf{S}$

- calculate $Conn(G, \mathbf{R}) = \sum Pr[S : S \in \mathbf{S}_{conn}]$ where $Pr[S]$ can be computed by multiplying all nodes specific regional probability in state S.

Running time of the exhaustive algorithm can be calculated as follows

- let $f(n, \mathbf{R})$ is the time to find all network states $\mathbf{S}$ for the given graph G.
- let $g(n, \mathbf{R})$ the time to find all connected states $S \subset \mathbf{S}$
- in-order to calculate $Conn(G, \mathbf{R})$ the upper bound can be $|R_1| \times |R_2| \times .. \times |R_n|$. Also let v is the node located maximum number of regions then we can calculate it $|R_1| \times |R_2| \times .. \times |R_n| = \mathcal{O}(|R_v|^n)$
- so the total running time will be $\mathcal{O}(|R_v|^n)$ as $f(n, \mathbf{R})$ and $g(n, \mathbf{R})$ minor in comparison.

3 k-trees

In this section, we will define clique, k-tree and perfect elimination sequence which is used in the main algorithm.
A clique is a set of vertices that induce a complete subgraph of a graph G. A clique with k vertices is considered to be a k-tree.

Definition. The class of k-tree can be defined recursively as follows:

- The complete graph on k vertices is a k-tree.
- Lets G_n is a k-tree with n vertices where $n \geq k$. Then we can construct a k-tree G_{n+1} of $n + 1$ vertices by adding a vertex adjacent to exactly k vertices, namely all vertices of a k clique of G_n. ■

A partial k-tree is any subgraph of a k-tree . partial k-trees are rich class of graph. Forest is an example of partial 1-tree. Series-parallel graphs and chordal graphs are subfamily of partial 2-trees. Also Halin graphs, Nested SAT and IO-graphs are subclasses of partial 3-trees.

Example 3.1. Fig. 1 depict a partial 2-tree. The complete graph of 2 vertices namely v_1 and v_2 is a 2-tree. Then we added vertex V_3 which is adjacent to both v_1 and v_2 is a 2-tree of 3 vertices. Finally a vertices s is added to the clique $v_1 v_2$ to form the 2-tree with 4 vertices.■

3.1 Perfect Elimination Sequence

A perfect elimination sequence (PES) in a graph is an ordering of the vertices of the graph such that, for each vertex v, v and the neighbors of v that occur after v in the order form a clique. In-order to find PES we need simplicial vertex.

Definition (Simplicial Vertex)**.** A simplicial vertex of a graph G is a vertex v such that the neighbours of v form a clique in G. Clearly, if G has a PES, then the last vertex in it is simplicial in G.■

Example 3.2. In fig. 1 we show that there are two simplicial vertices, v_1 and s. But s is our sink node so we are not eliminating s. So After elimination of v_1, we will be able to eliminate either v_2 or v_3. So the PES will be either v_1, v_2 or v_1, v_3.■

4 Main Algorithm

In this section we present an algorithm to compute the exact solution for $Conn(G, \mathbf{R})$ problem. More specially, the algorithm takes as input a UWSN network G where each node has a probabilistic locality set, a PES and compute Prob, a solution to the input $Conn(G, \mathbf{R})$ instance. The function uses a dynamic programming framework to solve $Conn(G, \mathbf{R})$ problem.

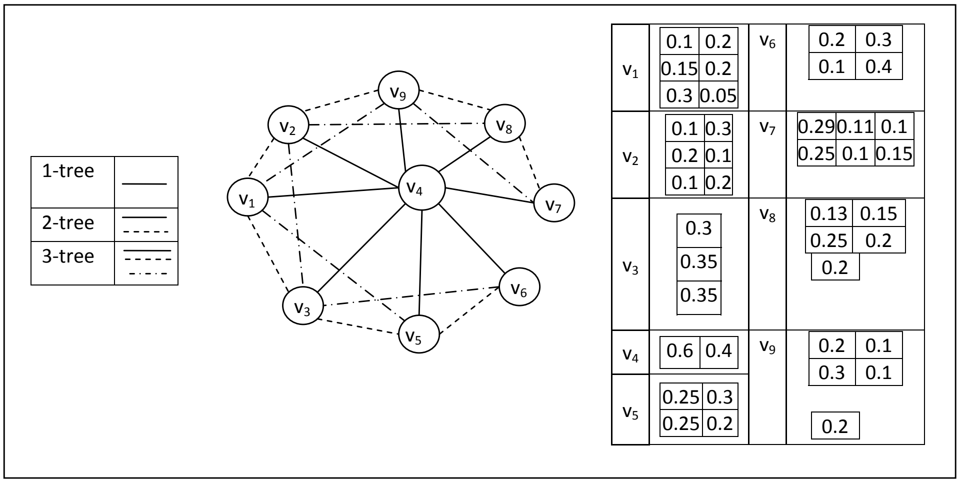

Figure 2: A UWSN modelled by a 3-tree.

4.1 Key Data-structures

Each clique associated with a table. Each row in the table defines key-value mapping.

- A key is a set of sets namely partition(s) of nodes along with corresponding regions in that particular clique.
- A value is a probability which is the multiplication of regional probability of nodes for the clique.

Example 4.1. Fig. 2 illustrates a graph G which is a 3-tree with 9 nodes and their corresponding probabilistic locality set. There are 19 triangles so there are 19 cliques associated with this 3-tree. For each clique the algorithm maintain a table. For every table there are some rows as key-value mapping. For example the clique $< v_1, v_2, v_3 >$ can be partitioned 5 different ways including $\{v_1, v_2, v_3\}, \{v_1, v_2\}\{v_3\}, \{v_1, v_3\}\{v_2\}, \{v_1\}\{v_2, v_3\}$ and $\{v_1\}\{v_2\}\{v_3\}$. For each partition there are $6 \times 6 \times 4 = 144$ rows as key-value mappings because the locality set of node v_1, v_2 and v_3 are $6, 6$ and 4 respectively. One of the row is $\{v_1, v_2, v_3\}^{(1,1,1)}0.004$ where v_1, v_2 and v_3 are in same partition with region $1, 1$ and 1 respectively and the probability 0.004 is multiplication of corresponding regional probabilities.■

When the algorithm starts elimination node by the order of PES there will be table merging and details is presented in section 4.2.

4.2 Algorithms Description

We now explain the main steps of function main, merge and merge partitions.

4.2.1 Function Main

Main function eliminates every node according to the PES by merging all cliques associated with that node and updating the result to base clique. The last remaining clique associates with the sink

Algorithm 1: Function Main(G, $\mathbf{R}$, $p(r_{(v,i)})$, PES)

Input: a UWSN $G = (V, E)$ is a partial k-tree where each node, $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}...\}$ with probability, $\{p(r_{(v,1)}), p(r_{(v,2)})...\}$ and $(x, y) \in E$ if $x \in V$ can be located one of it's locality set and $y \in V$ can be located one of it's locality set, so that they reach each other. PES is a perfect elimination sequence $(v_1, v_2, ..., v_{n-k})$ of G.

Output: Prob, a solution to the input instance.

Notation: $Temp$ is a map from keys to probabilities.

1 **Initialize** every clique by a table.
2 **for** $i = 1, 2, ..., n - k$ **do**
3   node v_i is associated with k-cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$
    $//T_{(v_i,1)}, T_{(v_i,2)}, .., T_{(v_i,k)}$ are the tables associated with cliques $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$ respectively
4     $Temp = T_{(v_i,1)}$
5   **for** $j = 2, 3.., k$ **do**
6         $Temp = merge(Temp, T_{(v_i,j)})$
    end
    // clique $K_{(v_i,base)}$ is the base clique and table $T_{(v_i,base)}$ is the base table of node v_i
7     $Temp = merge(Temp, T_{(v_i,base)})$
8   remove node v_i from $Temp$ and assign the result to $T_{(v_i,base)}$
end
9 **return** *Prob=*$\sum$*(All probability for single partition in the remaining table)*

node. Finally main function calculates the connectivity from the last clique by adding probabilities for those row which is associate with single partition.

In more details,

- Step 1 : initialize each clique by a table describe in subsection 4.1.
- Step 2 : processes each node v_i in PES. Every processing node, v_i is associated with k cliques.
- Step 3 : finds all those cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$.
- Step 4 : assign table $Temp$ with table $T_{(v_i,1)}$ associated with clique $K_{(v_i,1)}$.
- Steps 5-6 : iteratively merge all associated tables of node v_i and assign the result to table $Temp$ by using $merge$ function.
- Step 7 : merge $Temp$ table with base table of node v_i $T_{(v_i,base)}$ by using $merge$ function and assign the result to $Temp$.
- Step 8 : remove the vertex v_i and it's locality set from $Temp$ and assign the result to $T_{(v_i,base)}$.
- Step 9 : calculate and return connectivity for the network.

Example 4.2. One of the PES of the 3-tree depict in fig. 2 is $< v_1, v_6, v_5, v_7, v_8, v_9 >$. In-order to eliminate v_1, the algorithm merge three cliques $< v_1, v_3, v_4 >$, $< v_1, v_2, v_3 >$ and $< v_1, v_2, v_4 >$ to $< v_1, v_2, v_3, v_4 >$. Next step the algorithm find the base clique $< v_2, v_3, v_4 >$ and merge with $< v_1, v_2, v_3, v_4 >$. Finally it deletes v_1 from merged clique and update the result to $< v_2, v_3, v_4 >$ ■

4.2.2 Function merge

The primary task of merge function is to merge two table T_1, T_2, update keys and values of the newly created table T and return the table T.

$T_1(v_1, v_2, v_3)$

.	.
$\{v_1, v_2\}^{(1,1)}\{v_3\}^{(2)}$	0.002
.	.
.	.
.	.

$\times$

$T_2(v_1, v_3, v_4)$

.	.
$\{v_1\}^{(1)}\{v_3, v_4\}^{(2,1)}$	0.008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}\{v_3, v_4\}^{(2,1)}$	0.0008
.	.
.	.
.	.

Figure 3 a). Merging two table into $Temp$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}\{v_3, v_4\}^{(2,1)}$	0.0008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_2, v_3, v_4)$

.	.
$\{v_2\}^{(1)}\{v_3, v_4\}^{(2,1)}$	0.0008
.	.
.	.
.	.

Figure 3 b). Deleting node v_1 from $Temp$

In more details,

- Step 1 : finds the common vertices between two tables and assign the common nodes probability 1.
- Step 2 : check if whether or not the common vertex set C is empty. If it is empty then the algorithm returns otherwise go to next step.
- Steps 3-5 : performs the row-wise merging by using function *mpar* and create another row.
- Steps 6-7 : updates the locality set for every vertex v of newly created row.
- Steps 8-9 : calculates the regional probability of common nodes.
- Step 10 : updates the newly created row probability by multiplying row-wise probability and dividing by the sum of common nodes regional probabilities.
- Step 11 : insert the row into table T and
- Step 12 : return table T.

Example 4.3. Fig 3 a). illustrates merging row $\{v_1, v_2\}^{\{1,1\}}\{v_3\}^{\{2\}}$ 0.002 of table T_1 with row $\{v_1\}^{\{1\}}\{v_3, v_4\}^{\{2,1\}}$ 0.008 of table T_2 which is fundamental operation for merging two tables. The function merge uses $pMerge$ function which takes two partitions $\{v_1, v_2\}\{v_3\}$ and $\{v_1\}\{v_3, v_4\}$ and returned $\{v_1, v_2\}\{v_3, v_4\}$ because $\{v_1, v_2\} \cup \{v_1\} = \{v_1, v_2\}$ and $\{v_3\} \cup \{v_3, v_4\} = \{v_3, v_4\}$. The merge function updates the locality set of each node of the merged partition $\{v_1, v_2\}^{\{1,1\}}\{v_3, v_4\}^{\{2,1\}}$ from the locality set of nodes in merging partitions. There are two common nodes v_1 and v_3 located in region 1 and 2 respectively with probability 0.1 and 0.2 respectively in the merging partitions. It update the probability of newly created partition 0.0008 by multiplying row-wise probabilities 0.002×0.008 and dividing the probability by product of the common nodes corresponding regional probabilities 0.1×0.2.

Fig 3 b). shows the deletion of node v_1 from $Temp$. The function main simply look for the node and remove it and it's locality set from key.■

Algorithm 2: Function merge(T_1, T_2)

Input: Two tables T_1 and T_2 that share at least one common vertex
Output: A table T
Notation C is a set of vertices and Obj is a row of table T and $Prob\_C$ is a double variable

set C = the set of common vertices between T_1 and T_2 , set $Prob\_C = 1$
if $C \neq \emptyset$ **then**
    foreach *row* r *in* T_1 **do**
        foreach *row* s *in* T_2 **do**
            $Obj.par$ =pMerge($r.par$,$s.par$)
            foreach *vertex* v_i *in* $Obj.par$ *where* $i = 1, 2, .., k + 1$ **do**
                $Obj.loc[v_i] = r.loc[v_i]||s.loc[v_i]$
            end
            foreach *vertex* $v \in C$ **do**
                $Prob\_C = Prob\_C * s.loc[v]$
            end
            $Obj < Obj.par : Obj.loc >= \frac{Prob[r] \times Prob[s]}{Prob\_C}$
            Insert Obj in T as a row.
        end
    end
end
return *Table* T

4.2.3 Function Merge Partitions

The merging partitions is mainly done using the union operation by mapr function. The mpar function takes as input two partitions P_1, P_2 and merge them into one partition P.

In more details,

- Steps 1-2 : adds all the sets of P_2 to P_1.
- Steps 3-4 : selects two sets s^* and t^* from partition P_1
- Step 5: checks whether or not they are disjoint . If they are not disjoint then it will go to step 6 otherwise it will return to step 4.
- Step 6 : delete s^* from P_1.
- Step 7 : computes the union of s^* and t^* and insert it at the beginning of partition P_1.
- Step 8 : delete t^* from P_1.
- Steps 9-10 : set the iterator to the beginning of P_1 and return to step 5.
- Step 11 : assign P_1 to P and finally the algorithm return P.

Algorithm 3: function pMerge(P_1, P_2)

Input: Two partitions P_1 and P_2

Output: A partition P

Notation: s and t are two set iterators and their corresponding set are indicated by s^* and t^*.

foreach *set* s^* *in* P_2 **do**
    $P_1.push\_back(s^*)$
end
for $(s = P_1.begin();\ s \neq P_1.end();\ ++s)$ **do**
    for $(t = s.next();\ t \neq P_1.end();\ ++t)$ **do**
        if $s^* \cap t^* \neq \emptyset$ **then**
            $P_1.delete(s^*)$
            $P_1.push\_front(s^* \cup t^*)$
            $P_1.delete(t^*)$
            $s = P_1.begin()$
            break
        end
    end
end
set $P = P_1$
return P

4.3 Verification Cases

In this subsection we are going to present some cases by using these one can verify the correctness of our algorithm.

- For a partial k-tree, there should be more than one PES. In our algorithm, we tried all elimination sequence and we got same result.

4.4 Correctness

In this section we prove the correctness of our algorithm. It is an exhaustive algorithm which takes care all possible configuration. Our algorithm consists of two fundamental operation, partition merge and table merge. First we prove that partition merge and table merge are correct then we prove that our algorithm is correct.

The partition merge perform core operations of the algorithm. The partition merge is correct because it is merging two partitions into one given that they share at least one node. The table merge is correct because of the following reason. The table merge is merging each row of one table with each row of another table by holding the condition that they share atleast one node with same regional position. It is using partition merge in-order to merge two rows which is correct. The main algorithm is elimination a node by merging all clique associated with that node and updating the merged clique with base clique of that node. Each clique is associated with table so cliques are merged by using table merge operation which is correct. That concludes that the algorithm is correct.

4.5 Running time

5 Simulation Results

In this section we present simulation results that aim to investigate the following aspect

(a) complexity of our algorithm increases with increasing the value of k for partial k-tree.

(b) influence of k on accuracy where $k = 1, 2, 3, ...$

(c) effect of choosing different k-trees.

Table 2: Running time (RT) with respect to k

k	Network I	Network II	Network III
1	30	30	20
2	290	380	380
3	85000	875000	940000

Table 3: Accuracy with respect to k

k	Network I	Network II	Network III
1	62	32.96	81.8
2	71.2	36.15	98.95
3	75	37.31	100

We used three networks in-order represent running time and accuracy. Network I consists of 7 nodes with locality set of each node range from 4 to 8. Network II consists of 9 nodes where each node can be located from 3 to 6 locations.Network III consists of 12 nodes with locality set of each node range from 2 to 8.

a) k **versus running time** table 1 shows three scenarios running time with respect to k of. In every scenario the running time of the algorithm increases with k. We also observe that difference between the running time 1-tree and 2-tree are 10 to 15 times where the running time between 2-tree and 3-tree are more than 1000 times. This is because of complexity of 3-tree is much more than 2-tree.

b) k **versus accuracy** table 2 illustrates accuracy increases with k. The increase of accuracy is large for 1-tree to 2-tree than for 2-tree to 3-tree.

c1) **Effect of random edge selection.** Fig 6 illustrates connectivity versus transmission range when the algorithm selects randomly. We have the following observation from fig 6.

- with increase of transmission range the connectivity is also increasing. This is due to,increase of transmission range the probability of an edge between two nodes also increasing.
- connectivity for 2-tree is always greater than or equal to the connectivity of 1-tree for same transmission range. This is because in 2-tree there are more edges in comparison with 1-tree.
- The above is also true for 2-tree to 3-tree.

c2) **Effect of greedy edge selection.** Fig 7 illustrates connectivity versus transmission range when we are selecting edges by greedy techniques. In greedy techniques those edges are selected which has a high probable values. It is observable from figure that the network is fully connected with a smaller transmission range for all tree in comparison with random edge selection strategy.

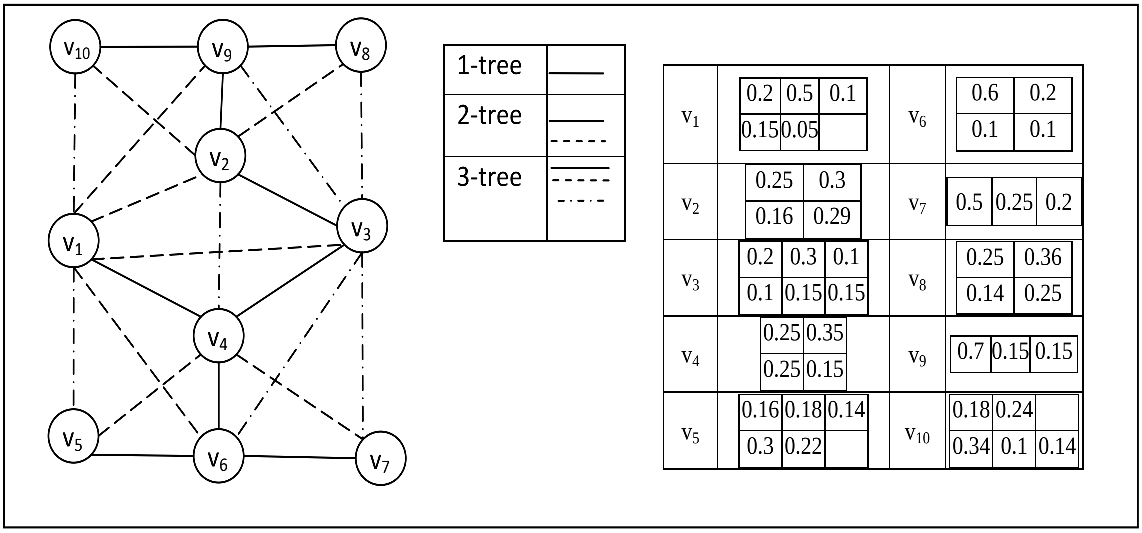

Figure 3: NetworkI

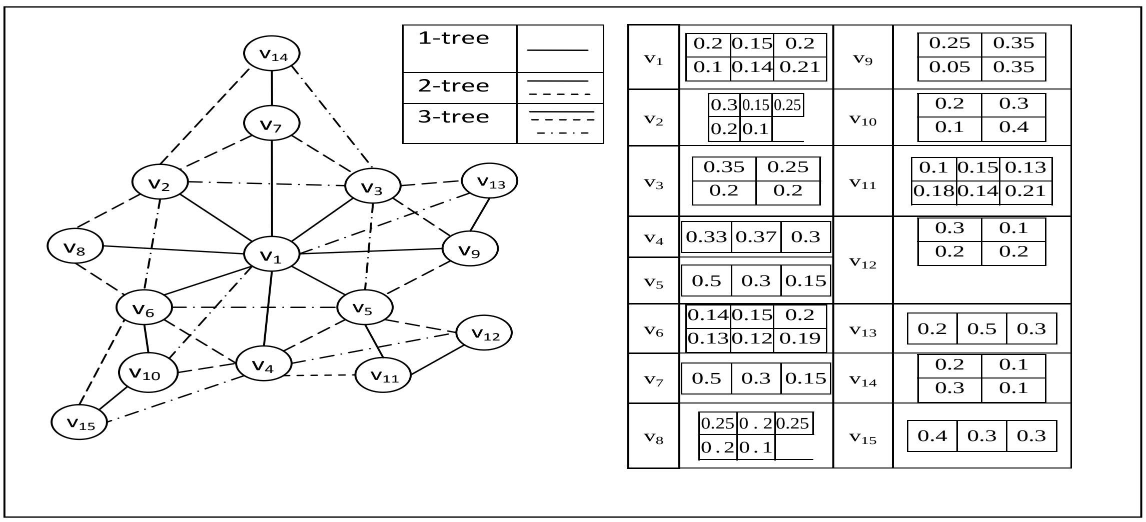

Figure 4: NetworkII

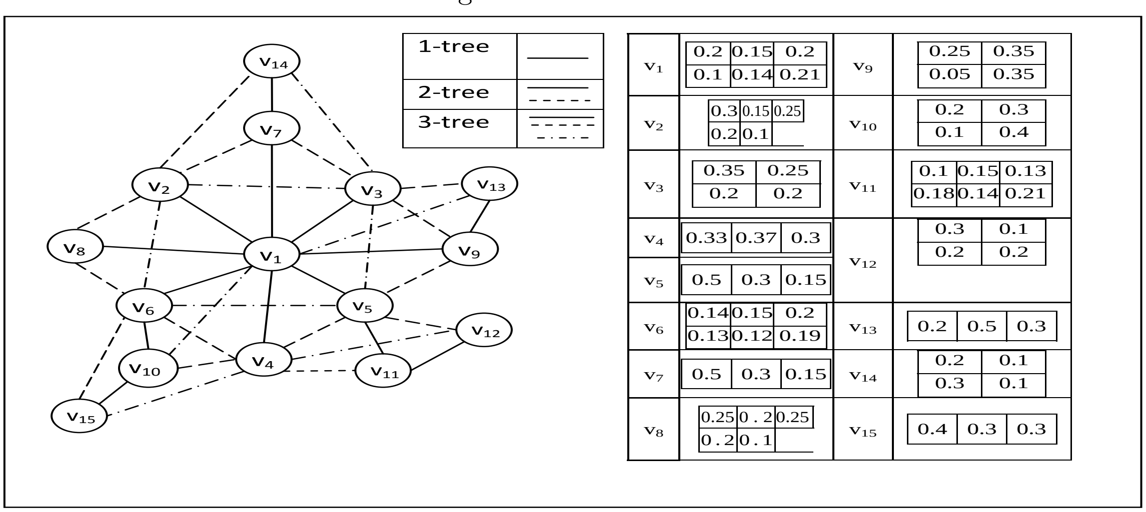

Figure 5: NetworkIII

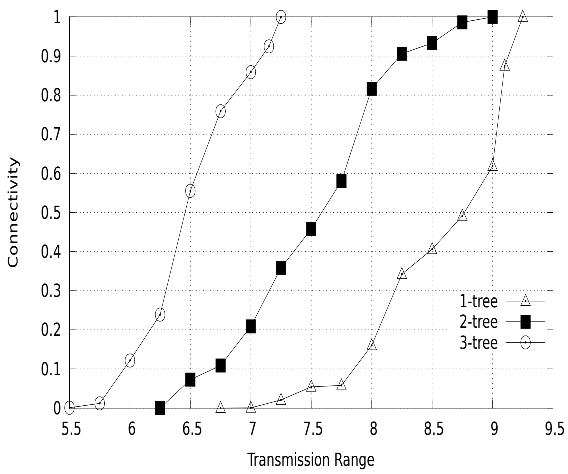

Figure 6: Connectivity versus transmission range with random edge selection.

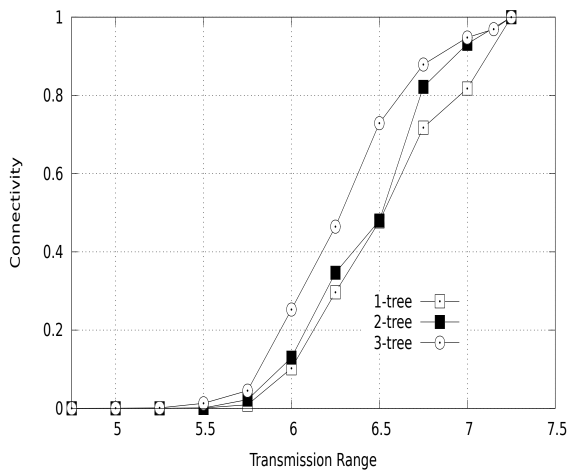

Figure 7: Connectivity versus transmission range with greedy edge selection.

6 Network with Relays

In this section we add relay nodes to our network. So our data structures changes as a result main algorithm and merge function also changes in some aspect. We describe the updates and show the results after adding relays.
We added relays, $Rel \subset V$ to the network in addition to sensor nodes which is referring target $Tar \subset V$ node in this section.

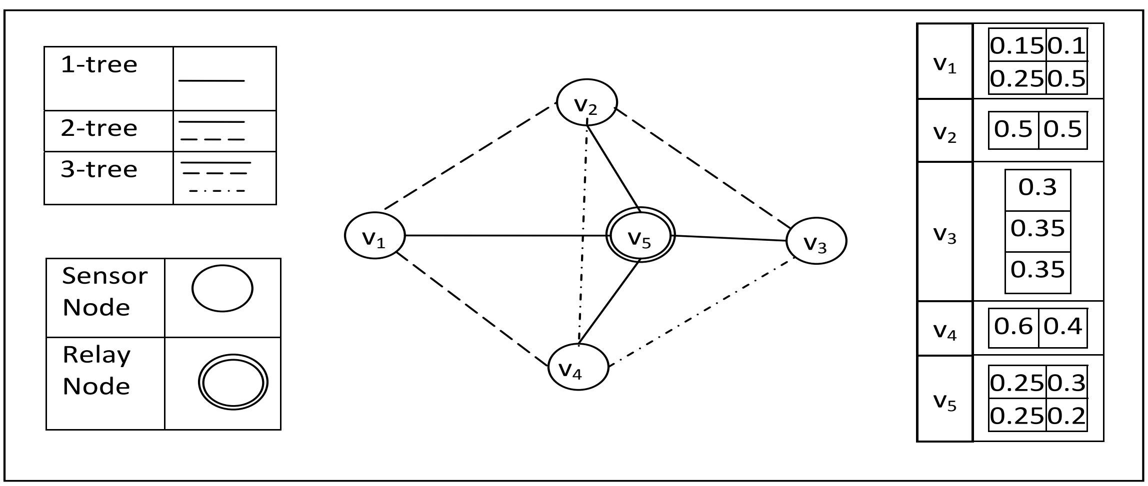

Figure 8: A partial 2-tree with 6 nodes.

Example 6.1. Fig **??** illustrates a network of 6 nodes where v_1, v_2, v_3 and v_4 are target nodes and v_5, v_6 are relays. v_5 is enhancing the connectivity of the network but v_6 is not increasing the connectivity. So we can simply ignore v_6 and measure the connectivity from v_1 to v_5

6.1 Problem Statement

In this section we define the problem.

Definition (The $Conn(G, Tar)$ Problem)**.** We are given, V the set of all nodes where each node $v \in V$ is located in a set of regions, $R_v = \{r_{(v,1)}, r_{(v,2)}, ...\}$ with probability $p(r_{(v,i)})$ where $i = 1, 2, ...$ $Rel \in V$ is the set of relay nodes and $Tar \in V$ is the set of target nodes where $V = Rel \cup Tar$. Also, if $x \in V$ and $y \in V$ are two nodes where x can be located into one of it's locality set and y can be located into one of it's locality set that they can communicate with each other then there is a link between x and y, denoted by $(x, y) \in E$. Here we use relay nodes to enhance the performance of the network but we are interested to find out the probability $Conn(G, Tar)$ that all target nodes Tar are connected.

6.2 Key Data Structures

We already know from section 4.1 that a typical row in a table is key-value mapping. A key consists of i). partitions ii). regional set iii). target node attach. We explain them this section.

i) partitions: There can be more than one partition associated with each row. Each partition consists with one or more node. We use braces to distinguish each partition. Two or more nodes are in the same partition means they can communicate each other when they are in regions indicated by regional set. For example in the key $\{v_1, v_2\}^{(1,1)}_{(1)} v_3{}^{(2)}_{(0)}$, there are two partitions including $\{v_1, v_2\}$ and $\{v_3\}$. Also node v_1 can reach node 2 when they are both in region 1 of their corresponding locality set but neither node v_1 nor node v_2 from region 1 can reach node 3 when node v_3 is in region 2 of it's locality set.

ii). regional set: The regional set of a partition consists of the position of corresponding node in the locality set. The regional set is indicated as a superscript of partition and is surrounded by parentheses. For example in the key $\{v_1, v_2\}^{(1,1)}_{(1)} v_3{}^{(2)}_{(0)}$, there are two regional sets $(1, 1)$ and (2) associated with two partitions $\{v_1, v_2\}$ and $\{v_3\}$ respectively. More specifically the partition-regional set pair $\{v_1, v_2\}^{(1,1)}$ indicates that node v_1 and v_2 are both located in region 1 of their corresponding locality set.

iii). Target node attach: The target node attach associates with every partition. If there is one or more target node in a partition then the value of target node attach is 1 otherwise it is 0. In the above example we indicate the target node attach as a subscript of the partition. First partition in the above key v_1 and v_2 are target nodes so as a subscript we put 1 and for second partition we simply put 0 to indicate that v_3 is a relay node.

6.3 Main function

- Step 8: check whether or not the node v_i, we are going to eliminate is a relay node. If v_i is relay node, It goes to next step otherwise it goes to step 10.
- Step 9: search every partition in every row of the table $Temp$ for node v_i. It deletes node v_i from every partition that contains node v_i. It also updates the regional sets of corresponding partitions from which v_i was deleted by removing the corresponding regions of v_i.
- Step 10: If v_i is sensor node then it goes to next step.
- Step 11: search every partition in every row of table $Temp$ for v_i. If the partitions that contains v_i consists of more than one node then remove v_i from that partitions. It also removes the location information of v_i from the regional sets corresponding to those partitions. If a

$T_1(v_1, v_2, v_3)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_3\}^{(2)}_{(0)}$	0.002
.	.
.	.
.	.

$\times$

$T_2(v_1, v_3, v_4)$

.	.
$\{v_1\}^{(1)}_{(1)}\{v_3, v_4\}^{(2,1)}_{(1)}$	0.008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_3, v_4\}^{\{2,1\}}_{(1)}$	0.0008
.	.
.	.
.	.

Figure 3 a). Merging two table into $Temp$

$Temp(v_1, v_2, v_3, v_4)$

.	.
$\{v_1, v_2\}^{(1,1)}_{(1)}\{v_3, v_4\}^{(2,1)}_{(1)}$	0.0008
.	.
.	.
.	.

$\Rightarrow$

$Temp(v_2, v_3, v_4)$

.	.
$\{v_2\}^{(1)}_{(1)}\{v_3, v_4\}^{(2,1)}_{(1)}$	0.0008
.	.
.	.
.	.

Figure 3 b). Deleting node v_1 from $Temp$

partition contains v_i itself this kind of partition is called bad partition. The algorithm simply ignore bad partition.

Algorithm 4: Function Main(G, $\mathbf{R}$, Tar, Rel, $p(r_{(v,i)})$, PES)

Input: a UWSN $G = (V, E)$ is a partial k-tree where each node, $v \in V$ can be located into a set of regions $R_v = \{r_{(v,1)}, r_{(v,2)}...\}$ with probability, $\{p(r_{(v,1)}), p(r_{(v,2)})...\}$ and $(x, y) \in E$ if $x \in V$ can be located one of it's locality set and $y \in V$ can be located one of it's locality set, so that they reach each other. A set of target nodes $Tar = \{v_1, v_2, ...\}$ where $Tar \subset V$. A set of relay nodes $Rel = \{v_1, v_2, ...\}$ where $Rel \subset V$. PES is a perfect elimination sequence $(v_1, v_2, ..., v_{n-k})$ of G.

Output: Prob, a solution to the input instance.

Notation: $Temp$ is a map from keys to probabilities.

Initialize every clique by a table.
for $i = 1, 2, ..., n - k$ **do**
- node v_i is associated with k-cliques, $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$
  $//T_{(v_i,1)}, T_{(v_i,2)}, .., T_{(v_i,k)}$ are the tables associated with cliques $K_{(v_i,1)}, K_{(v_i,2)}, .., K_{(v_i,k)}$ respectively
- $Temp = T_{(v_i,1)}$
- **for** $j = 2, 3.., k$ **do**
  - $Temp = merge(Temp, T_{(v_i,j)})$
- **end**
- // clique $K_{(v_i,base)}$ is the base clique and table $T_{(v_i,base)}$ is the base table of node v_i
- $Temp = merge(Temp, T_{(v_i,base)})$
- **if** *v_i is a relay node* **then**
  - remove node v_i from $Temp$ and assign the result to $T_{(v_i,base)}$ after updating the regional set.
- **end**
- **else**
  - Remove v_i from all the partitions of $Temp$ except the partition with v_i itself and assign the result to $T_{(v_i,base)}$ after updating the regional set.
    //We simply ignore the row where there is a partition of v_i itself.
- **end**

end
return *Prob*=$\sum$*(All probability for single partition in the remaining table)*

6.4 Merge Function

Algorithm 5: Function merge(T_1, T_2)

Input: Two tables T_1 and T_2 that share at least one common vertex
Output: A table T
Notation C is a set of vertices and Obj is a row of table T and $Prob\_C$ is a double variable

```
1  set C = the set of common vertices between T1 and T2 , set Prob_C = 1
2  if C ≠ ∅ then
3      foreach row r in T1 do
4          foreach row s in T2 do
5              Obj.par = pMerge(r.par, s.par)
6              foreach vertex v_i in Obj.par where i = 1, 2, .., k + 1 do
7                  Obj.locmap[v_i] = r.locmap[v_i] || s.locmap[v_i]
               end
8              foreach node v in every partition P in Obj.par do
9                  foreach node u in every partition Q in s.par do
10                     if v == u then
11                         Obj.tAttach[P] = max(Obj.tAttach[P], s.tAttach[Q])
                       end
                   end
12                 foreach node w in every partition T in r.par do
13                     if v == w then
14                         Obj.tAttach[P] = max(Obj.tAttach[P], r.tAttach[T])
                       end
                   end
               end
15             foreach vertex v ∈ C do
16                 Prob_C = Prob_C * s.loc[v]
               end
17             Obj < Obj.par : Obj.loc >= (Prob[r] × Prob[s]) / Prob_C
18             Insert Obj in T as a row.
           end
       end
   end
19 return Table T
```

- Step 8: selects every node v of every partition P in the newly created row Obj.
- Step 9: iterates every node u of every partition Q in the row s.
- Step 10: check whether u and v are the same same node or not. If they are same then it goes to next step otherwise it goes to step 9.
- Step 11: updates the target attach entry of partition P by taking the max between the target attach entry of partition P and target attach entry of partition Q.
- Step 12: iterates every node w of every partition T in the row r.

Table 4: Accuracy with respect to k

k	Network I	Network IR	Network II	NeworkIIR	NetworkIII	Network IIIR
1	30.23	86.96	5.23	22.27	15.99	28.44
2	53.72	96.72	17.55	59.43	80.23	89.3
3	60.20	97.60	20.96	60.5	83.3	92.3

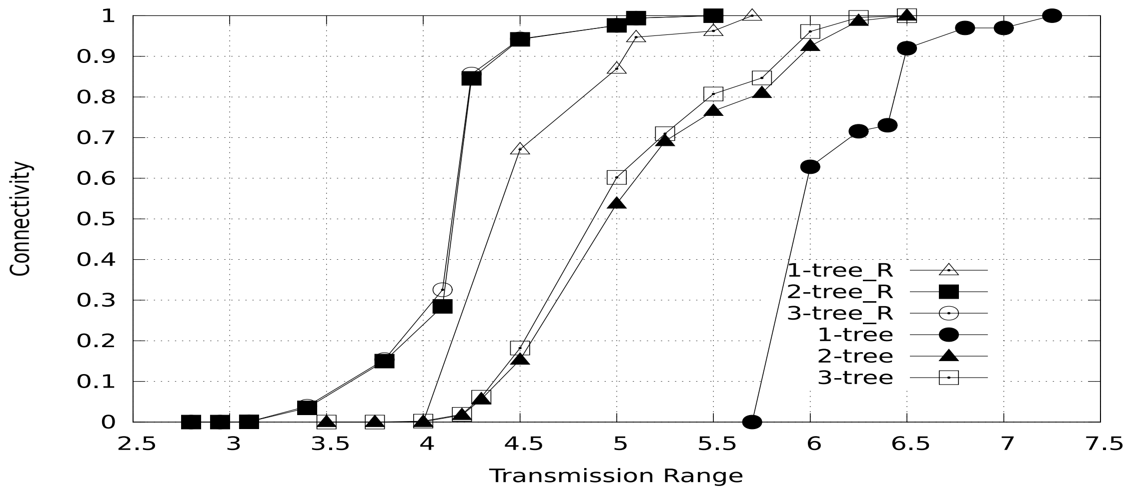

Figure 9: NetworkIII

- Step 13: check whether w and v are the same same node or not. If they are same then it goes to next step otherwise it goes to step 9.
- Step 11: updates the target attach entry of partition P by taking the max between the target attach entry of partition P and target attach entry of partition T.

6.5 Simulation Results

References

[1] Kemal Akkaya and Andrew Newell. Self-deployment of sensors for maximized coverage in underwater acoustic sensor networks. *Computer Communications*, 32(7):1233–1244, 2009.

[2] S. M. Nazrul Alam and Zygmunt J. Haas. Coverage and connectivity in three-dimensional underwater sensor networks. *Wireless Communication and Mobile Computing (WCMC)*, 8(8):995–1009, 2008.

[3] Antonio Caruso, Francesco Paparella, Luiz Filipe M Vieira, Melike Erol, and Mario Gerla. The meandering current mobility model and its impact on underwater mobile sensor networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 221–225. IEEE, 2008.

[4] Melike Erol-Kantarci, Sema Oktug, Luiz Vieira, and Mario Gerla. Performance evaluation of distributed localization techniques for mobile underwater acoustic sensor networks. *Ad Hoc Networks*, 9(1):61–72, 2011.

[5] Alan C. Farrell and Jun Peng. Performance of {IEEE} 802.11 {MAC} in underwater wireless channels. *Procedia Computer Science*, 10(0):62 – 69, 2012. {ANT} 2012 and MobiWIS 2012.

[6] John Heidemann, Milica Stojanovic, and Michele Zorzi. Underwater sensor networks: applications, advances and challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 370(1958):158–175, 2012.

[7] Jules Jaffe and Curt Schurgers. Sensor networks of freely drifting autonomous underwater explorers. In *Proceedings of the 1st ACM international workshop on Underwater networks*, pages 93–96. ACM, 2006.

[8] Jiejun Kong, Jun-hong Cui, Dapeng Wu, and Mario Gerla. Building underwater ad-hoc networks and sensor networks for large scale real-time aquatic applications. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 1535–1541. IEEE, 2005.

[9] Uichin Lee, Paul Wang, Youngtae Noh, FLM Vieira, Mario Gerla, and Jun-Hong Cui. Pressure routing for underwater sensor networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[10] Ji Luo, Dan Wang, and Qian Zhang. Double mobility: coverage of the sea surface with mobile sensor networks. In *INFOCOM 2009, IEEE*, pages 118–126. IEEE, 2009.

[11] Ji Luo, Dan Wang, and Qian Zhang. On the double mobility problem for water surface coverage with mobile sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 23(1):146–159, 2012.

[12] Youngtae Noh, Uichin Lee, Paul Wang, Brian Sung Chul Choi, and Mario Gerla. Vapr: Void-aware pressure routing for underwater sensor networks. *Mobile Computing, IEEE Transactions on*, 12(5):895–908, 2013.

[13] Joseph A Wald and Charles J Colbourn. Steiner trees in probabilistic networks. *Microelectronics Reliability*, 23(5):837–840, 1983.

[14] Joseph A Wald and Charles J Colbourn. Steiner trees, partial 2–trees, and minimum ifi networks. *Networks*, 13(2):159–167, 1983.

[15] Lei Ying, Sanjay Shakkottai, Aneesh Reddy, and Shihuan Liu. On combining shortest-path and back-pressure routing over multihop wireless networks. *IEEE/ACM Transactions on Networking (TON)*, 19(3):841–854, 2011.

[16] Zhong Zhou, Jun-Hong Cui, and Shengli Zhou. Efficient localization for large-scale underwater sensor networks. *Ad Hoc Networks*, 8(3):267–279, 2010.

[17] Zhong Zhou, Zheng Peng, Jun-Hong Cui, Zhijie Shi, and Amvrossios C Bagtzoglou. Scalable localization with mobility prediction for underwater sensor networks. *Mobile Computing, IEEE Transactions on*, 10(3):335–348, 2011.