A Distributed Adaptive Landmark Clustering Algorithm Based on mOverlay and Learning Automata for Topology Mismatch Problem in Unstructured Peer-to-Peer Networks

Journal:	*International Journal of Communication Systems*
Manuscript ID:	IJCS-14-0689.R1
Wiley - Manuscript type:	Research Article
Date Submitted by the Author:	05-Jan-2015
Complete List of Authors:	Saghiri, Ali Mohammad; Amirkabir University of Technology, Computer Engineering and Information Technology Meybodi, MohammadReza; Amirkabir University of Technology, Computer Engineering and Information Technology
Keywords:	Peer-to-Peer Network , Topology Mismatch Problem, Landmark Clustering Algorithm, Learning Automata

A Distributed Adaptive Landmark Clustering Algorithm Based on mOverlay and Learning Automata for Topology Mismatch Problem in Unstructured Peer-to-Peer Networks

Ali Mohammad Saghiri*†, Mohammad Reza Meybodi‡

Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology (Tehran Polytechnic), 424 Hafez Ave., Tehran, Iran

Abstract: Peer-to-peer networks are overlay networks that are built on top of communication networks that are called underlay networks. In these networks, peers are unaware of the underlying networks, so the peers choose their neighbors without considering the underlay positions, and therefore the resultant overlay network may have mismatches with its underlying network, causing redundant end-to-end delay. Landmark clustering algorithms, such as *mOverlay*, are used to solve topology mismatch problem. In the *mOverlay* algorithm, the overlay network is formed by clusters in which each cluster has a landmark peer. One of the drawbacks of *mOverlay* is that the selected landmark peer for each cluster is fixed during the operation of the network. Because of the dynamic nature of peer-to-peer networks, using a non-adaptive landmark selection algorithm may not be appropriate. In this paper, an adaptive landmark clustering algorithm obtained from the combination of *mOverlay* and *learning automata* is proposed. *Learning automata* are used to adaptively select appropriate landmark peers for the clusters in such a way that the total communication delay will be minimized. Simulation results have shown that the proposed algorithm outperforms the existing algorithms with respect to communication delay and average round-trip time between peers within clusters.

Keywords: Peer-to-Peer Network, Topology Mismatch Problem, Landmark Clustering Algorithm, Learning Automata

1. INTRODUCTION

Peer-to-peer networks are overlay networks that are constructed over underlay networks. In these networks, all peers communicate directly with each other and continually join and leave the network. These networks can be classified as either structured or unstructured networks. In structured peer-to-peer networks, distributed management algorithms, such as Chord[1],CAN[2] and P-Grid[3], are designated to manage the overlay topology to provide efficient resource locating algorithms. In unstructured peer-to-peer networks, there are some lightweight algorithms to manage the overlay topology. Due to simple design, unstructured peer-to-peer

* Corresponding author.
† Email address: a_m_saghiri@aut.ac.ir
‡ Email address: mmeybodi@aut.ac.ir

networks, such as Gnutella[4], have received much attention. In both structured and unstructured peer-to-peer networks, a major consequence of the network dynamicity caused by joining and leaving peers is that the overlay topology can change rapidly over time, and therefore the performance of the overlay topology in terms of traffic and end-to-end delay might be degraded. To prevent the degradation of performance, the topology management algorithms should be able to scale and adapt themselves to dynamic conditions in the network [5]. One of the problems in designing topology management algorithms in peer-to-peer networks is topology mismatch problem[6]–[9]. In peer-to-peer networks, peers are unaware of the underlying networks so the peers choose their neighbors without considering the underlay positions, and therefore the resultant overlay network may have mismatches with its underlying network, causing redundant end-to-end delay. The algorithms reported for solving the topology mismatch problems can be classified into three classes as given below.

- Algorithms in which each peer in the network uses some services, such as GPS[§] or Oracle, to gather information about the location of peers to solve the mismatch problem[10][11].
- Algorithms in which each peer in the network uses information about landmark peers to solve the mismatch problem [12]–[19]. This class of algorithms is known as landmark clustering.
- Algorithms in which each peer in the network uses information about its neighbors to solve the mismatch problem [7], [9], [20]–[22].

Landmark clustering algorithms are designated based on the fact that the peers close to each other in the underlay network are likely to have similar delays to a few landmark peers. Utilizing this fact, in the landmark clustering algorithm, peers arrange themselves into clusters such that peers that get in within a given cluster are relatively close to each other in terms of communication delay. Landmark clustering algorithms reported in the literature can be classified into two subclasses: algorithms that use information about the coordinates of peers [13], [18], [19] and algorithms that use information about delays between peers. The algorithms in the second subclass can be further classified into two subclasses: algorithms that use static landmark selection methods [12], [16], [17], [23] and algorithms that use dynamic landmark selection methods [14][15]. The algorithms that use static landmark selection methods suffer from several drawbacks, such as lack of scalability, a small set of fixed peers as landmark peers responsible for a large number of positions and also lack of adaptation to the changes that occur in peer-to-peer networks. To mitigate these problems, dynamic landmark selection methods are proposed. In dynamic landmark selection algorithms, such as *mOverlay*[14] and its extension reported in [15], the landmark selection algorithm dynamically creates new clusters when the existing clusters of the network are not appropriate for new peers.

mOverlay is a well known landmark clustering algorithm in unstructured peer-to-peer networks [14]. In *mOverlay,* the selected landmark peer for each cluster is fixed during the operation of the network. Because of the dynamic nature of peer-to-peer networks, the

[§]Global Positioning System

operational environment and the peers of each cluster may change over time (peers continually join and leave the cluster) and for this reason, using a non-adaptive algorithm for the selection of landmark peers during operation of the network may not be appropriate. Having an adaptive mechanism for selecting landmark peers as the network conditions (properties) change may have a strong impact on the efficiency of the clustering algorithm.

In this paper, we propose an adaptive landmark clustering algorithm for solving topology mismatch problem in unstructured peer-to-peer networks. This algorithm is obtained from the combination of *mOverlay* and *learning automata.* This algorithm uses the locating algorithm of *mOverlay* for dividing peers into clusters and uses *learning automata* to adaptively select an appropriate landmark peer in each cluster during the operation of the network with the aim of minimizing total communication delay. To show the superiority of the proposed algorithm, this algorithm is compared with *mOverlay*[14] and another algorithm reported in [15]. Simulation results have shown that the proposed algorithm outperforms the existing algorithms with respect to communication delay and average round-trip time between peers within clusters. The rest of the paper is organized as follows. The problem statement is given in section 2. In section 3, the theory of learning automata is described briefly. In section 4, we describe the proposed landmark clustering algorithm. Section 5 reports the results of experiments, and section 6 concludes the paper.

2. STATEMENT OF THE PROBLEM

Consider n peers that are connected to each other through an overlay network over an underlay network. The topology of the overlay network can be represented by a graph, such as $G = (V, E)$ in which $V = \{peer_1, peer_2, \dots, peer_n\}$ is a set of peers and $E \subseteq V \times V$ is a set of links connecting the peers in the overlay network. This graph is called as overlay graph. The topology of the underlay network can be represented by a graph $G' = (V', E')$ in which $V' = \{position_1, position_2, , \dots, position_n\}$ is a set of positions in the underlay network and $E' \subseteq V' \times V'$is a set of links connecting the positions in the underlay network. In peer-to-peer networks, each peer in V is mapped to a position in V' according to a one-to-one function $H: V \to V'$. In landmark clustering algorithms, the overlay graph is formed by clusters in which each cluster has a landmark peer. The peers which are not selected as landmark peers are called ordinary peers. The communication between ordinary peers should be handled by landmark peers. Let C_i(where $i \in \{1,2,3,4, \dots, p\}$) represents the set of p clusters in the network, and $L_i \in V$ denotes the landmark peer of cluster C_i. The topology of the landmark peers network can be represented by a graph $G^l = (V^l, E^l)$ in which $V^l \subseteq V$ is a set of landmark peers and $E^l \subseteq V \times V$ is a set of links connecting the landmark peers in the overlay network. This graph is called landmark graph. In the peer-to-peer network that is constructed based on the landmark clustering algorithms, each landmark peer in V^l is mapped to several ordinary peers in V according to a one-to-many function $H': V^l \to V$. Each landmark peer in V^l is mapped to several positions in V' using $H'$and H. Figure 1 shows an example of a peer-to-peer network that uses two clusters to manage the overlay topology. In this example, $peer_m$ and $peer_j$ are two landmark peers.

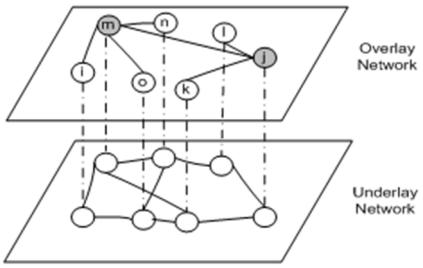


Figure 1. An overlay topology that uses a landmark clustering algorithm

A primary goal of landmark clustering algorithms is to improve the performance of the overlay network in terms of communication delay because the topology mismatch problem causes redundant communication delay. Landmark clustering algorithms consist of two parts: landmark selection to select some peers as landmark peers and topology optimization to organize the links between peers. The problem of finding the set of peers to be designated as landmark peers to minimize the total communication delay is similar to a version of super-peer selection in a super-peer network that is proven to be a NP-hard problem[16]. In [16], this problem was cast as a special case of the Hub Location Problem[24] that is a NP-Hard problem. Landmark clustering is more complex than the classic Hub Location problem, because landmark clustering must respond to dynamic joining and leaving of peers. For solving the landmark clustering problem, the following problem should be solved.

$$\text{Min } z = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{k=1}^{n} \sum_{m=1}^{n} (d_{ik} + d_{km} + d_{mj}) \times x_{ik} \times x_{jm} \quad (1)$$

$s.t,$

$$x_{ij} \leq x_{jj} \qquad i,j = 1,\dots,n \quad (2)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i,j = 1,\dots,n \quad (3)$$

$$\sum_{j=1}^{n} x_{jj} = \text{p} \quad (4)$$

$$x_{ij} \in \{0,1\} \qquad i,j = 1,\dots,n \quad (5)$$

In Equation (1), z is the total all-pairs end-to-end communication delay of the overlay network. In (1), x_{ij} indicates the existence or absence of a connection between $peer_i$ and $peer_j$. If $x_{ij} = 1$, then there is a connection between $peer_i$ and $peer_j$. If $x_{kk} = 1$, $peer_k$ is chosen as a landmark peer. d_{ij} *is* the end-to-end delay from $peer_i$ and $peer_j$ in the underlay network that is $d_{ij} = D(H(peer_i), H(peer_j))$ where function $D: V' \times V' \rightarrow \mathbb{R}$ gives the end-to-end delay between pairs of positions in the underlay network. Constraint (3) that is used in[14]indicates that each ordinary peer connects to exactly one landmark peer. Constraint (4) that is used in [12]indicates that there will be exactly *p* clusters, each with its own landmark peer.

3. LEARNING AUTOMATA

Learning automata are adaptive decision-making devices that operate in unknown random environments. A learning automaton has a finite set of actions, and each action has a certain probability (unknown to the automaton) for getting rewarded by its environment. The aim is to learn to choose the optimal action (i.e., the action with the highest probability of being rewarded) through repeated interactions with the system. If the learning algorithm is chosen properly, then the iterative process of interacting with the environment can be made to result in the selection of the optimal action. The interaction between the learning automaton and the environment is shown in Figure 2.

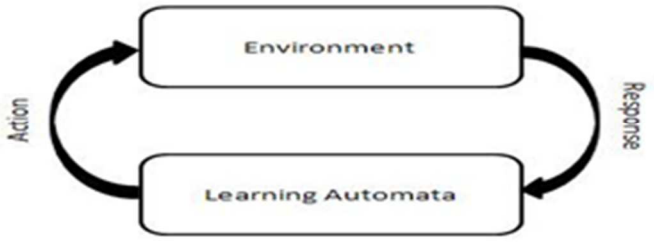


Figure 2. Learning automaton (LA)

Learning automata (LAs) can be classified into two main families, fixed and variable structure learning automata [25], [26]. Variable structure learning automaton that is used in this paper is represented by a sextuple $\langle \underline{\beta}, \underline{\phi}, \underline{\alpha}, \underline{P}, G, T \rangle$, where $\underline{\beta}$ a set of inputs actions, ϕ is a set of internal states, $\underline{\alpha}$ is a set of outputs, P denotes the state probability vector governing the choice of the state at each stage k, G is the output mapping, and T is the learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector.

It is evident that the crucial factor affecting the performance of the variable structure learning automata is learning algorithm for updating the action probabilities. Let α_i be the action chosen at time k as a sample realization from distribution p(k). Let a and b denote the reward and penalty parameters. The equations for updating the probability vector are defined by Equation 6 for favorable responses (β=1), and Equation 7 for unfavorable response (β=0).

$$p_i(k+1) = p_i(k) + a\big(1 - p_i(k)\big) \qquad (6)$$
$$p_j(k+1) = p_j(k) - ap_j(k), \qquad \forall j \neq i$$

$$p_i(k+1) = (1-b)p_i(k) \qquad (7)$$
$$p_j(k+1) = \frac{b}{r-1} + (1-b)p_j(k), \qquad \forall j \neq i$$

If $a = b$, the learning algorithm is called the linear reward penalty (L_{RP}) algorithm, if $b = \varepsilon a$ with $\varepsilon < 1$, then the learning algorithm is called the linear reward ε-penalty ($L_{R\varepsilon P}$) and finally, if $b = 0$, the learning algorithm is called the linear reward inaction (L_{RI}).

Learning automata have found applications in many areas, such as sensor networks[27]–[30], wireless data broadcasting systems[31]–[33], cognitive networks [34]–[36],mesh networks[37], peer-to-peer networks[38]–[42], channel assignment[43], image processing[44], neural networks engineering [45], [46] and evolutionary computing [47], [48], to mention a few.

4. THE PROPOSED ALGORITHM

In this section, an adaptive landmark clustering algorithm obtained from the combination of *mOverlay* and *learning automata* for solving the topology mismatch problem will be proposed. In this algorithm, each peer may play three roles: Unattached, Ordinary and Landmark. Each peer is equipped with a learning automaton that has two actions: "set the role to landmark peer" and "set the role to ordinary peer". Figure 3 shows the interaction of the learning automaton LA_i of $peer_i$ with its environment. The environment of learning automaton LA_i residing in peer $peer_i$ is composed of the local environment of $peer_i$ and the global environment that consists of local environments of the peers that are in the same cluster as $peer_i$, and the local environment of the landmark peers of clusters adjacent to the cluster of $peer_i$. Once a peer joins the network uses the *mOverlay* locating algorithm to find an appropriate cluster. If the peer finds an appropriate cluster C_x, the peer connects to the landmark peer of C_x and then activates the landmark peer to use a landmark selection algorithm for selecting a new landmark peer. The landmark selection algorithm requires several iterations to gradually find an appropriate landmark peer. Each iteration of the algorithm consists of five steps: 1. activating the learning automata, 2. gathering information about delays, 3.finding an appropriate landmark peer, 4. calculating a reinforcement signal and updating learning automata, and 5.declaring a new landmark peer for cluster C_x. These steps are described as follows. During step one, the landmark peer of cluster C_x activates a team of learning automata which consists of the learning automata of peers of cluster C_x to select their actions collectively and then goes to step two. The peers whose learning automata have selected "set the role to landmark peer" actions are called candidate peers. During step two, the peer tries to gather information about delays from candidate peers to ordinary peers of cluster C_x and landmark peers of clusters adjacent to cluster C_x. Then, the peer saves gathered information into a local database in the peer and goes to step three. During step three, the peer chooses an appropriate landmark peer from the local database and goes to step four. The appropriate landmark peer is a peer that has the lowest mean and variance of delays to ordinary peers of cluster C_x and the landmark peers of clusters adjacent to cluster C_x. During step four, learning automata of all peers in cluster C_x collectively update their action probability vectors using a reinforcement signal. If the chosen landmark peer is one of the candidate peers, the reinforcement signal is then set to 1, and otherwise the reinforcement signal is set to 0. At the end of step four, if the number of iterations of the algorithm is lower than a threshold called the *MAX_ITERATION_LEARING*, the peer goes to

step one. Otherwise, the peer goes to step five. During step five, the peer declares a new landmark peer for cluster C_x. For declaring this new landmark peer, the peer determines one of the peers of the cluster C_x whose probability of selecting the "set the role to landmark peer" action of its learning automaton is higher than the probabilities of selecting "set the role to landmark peer" action of the learning automata of other peers of cluster C_x to be the landmark peer. Other peers of the cluster that are not determined to be landmark peers set their role to ordinary. If a peer when joining the network cannot find an appropriate cluster (using *mOverlay*), the peer creates a new cluster and becomes the only member of that cluster. To compensate for the effect of leaving a landmark peer from a cluster, if each peer of a cluster cannot find the landmark peer of its cluster, then the peer activates the cluster members to select another landmark peer. In the rest of this section, the proposed algorithm is described in more detail.

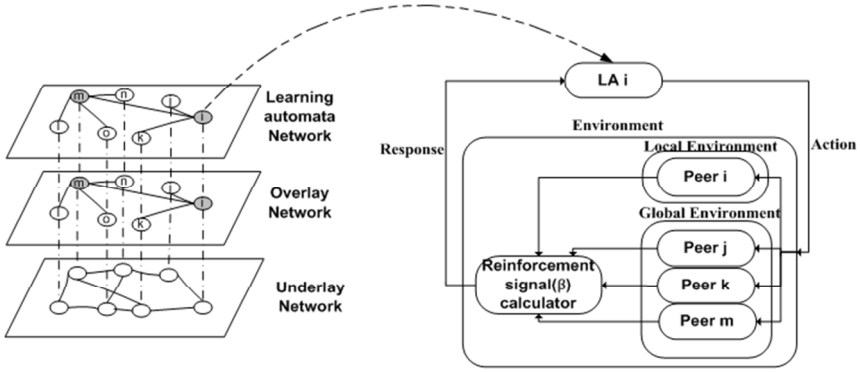


Figure 3. Learning automaton for $peer_i$ and its environment

In the proposed algorithm, the process executed by $peer_i$ when joining the network has three major phases: 1. Initialization phase, 2. Landmark selection phase, and 3. Maintenance phase. These phases are summarized in Figure 4. The detailed descriptions of these phases are given later.

```
Algorithm peer_management_skeleton()
    Inputs: MAX_ITERATION_LEARNING // maximum number of iterations
    Notations
    Let LA denotes the learning automaton of the peer.
    Variable ROLE at any time determines the role that the peer is playing which is either "ordinary peer" or
                                                                                "landmark peer" or
                                                                                "unattached peer".
    Variable C_x determines the identifier of the cluster that the peer belongs to at any time.
    Variable A determines the selected action of the learning automaton of the peer.
01  Begin
02  // Initialization phase
03  - ROLE ← "Unattached peer";
04  - Find an appropriate cluster C_x to which the peer belongs using mOverlay locating algorithm;
05  If (an appropriate cluster cannot be found) Then
06    - Create a new cluster C_x;
07    - Goto Maintenance phase;// goto line 37
08  Else
09    - ROLE ← "Ordinary peer";
10    - Connect to the landmark peer of cluster C_x;
11  EndIf
12  //Landmark selection phase
13  If (ROLE = "Ordinary peer") Then
14    - Activate the landmark peer of cluster C_x;// activate the landmark peer to select a new landmark for cluster C_x
15  EndIf
16  While (ROLE = "Unattached peer")
17    - Activate the peers of cluster C_x;// activate all peers of cluster Cx to collectively find the landmark peer
18    - s ← 1;// s is a counter defined to control the number of times that the learning automaton in the peer is updated;
19    While (s < Max_Iteratin_Learning)
20      - Find those peers of cluster C_x whose learning automata have selected action "landmark peer";
        // call these peers candidate peers
21      - Gather information about delays;
          // gather information about delays from each candidate peer peer_k to the landmark peers of clusters adjacent to
            cluster C_x and all ordinary peers of cluster C_x
22      - Save the gathered information about delays into a local database;
23      - LA selects an action and save the action in variable A;
24      - Choose one of the peers of cluster C_x using the local database of the peer ;
          // choose a peer which has the Lowest mean and variance of delays to ordinary peers of C_x and the landmark
            peers of clusters adjacent to cluster C_x
25      If (the chosen landmark peer is one of the candidate peers) Then
26        - β ← 1;
27      Else
28        - β ← 0;
29      EndIf
30      - Send the reinforcement signal(β) to all pees of cluster C_x;
31      - Update the action probability of LA based on β;
32      - s = s + 1;
33    EndWhile
34    - Gather information about learning automata of peers of cluster C_x;
35    - Declare the role of the peer and send messages containing the identifier of new landmark peer to the peers of
        cluster C_x ; // a peer is declared as new landmark peer if the probability of selecting "set the role to landmark
                        peer" action of its learning automaton is higher than the probabilities of selecting "set the role to
                        landmark peer" action of other learning automata of the peers in cluster C_x
36  End while
37  //Maintenance phase
38  - Wait until some event occur;
39  If (peer has been activated by a new ordinary peer peer_j in cluster C_x) Then
40    - Connect to peer_j;
41    - ROLE ← "Unattached peer";
42    - Goto Landmarselection phase;// goto line 12 to select a new landmark peer
43  EndIf
44  If (peer has been activated by an unattached peer peer_j in cluster C_x) Then
45    - ROLE ← "Unattached peer";
46    - LA selects an action and save the action in variable A;
47    If (A = "set the role to landmark peer") Then
48      - Gather information about delays;
        // Gather information about delays from the peer to the landmark peers of clusters adjacent to cluster C_x and
          delays from the peer to all ordinary peers of cluster C_x
49      - Send the gathered information and the selected action selected to peer_j;
50    EndIf
51    - Receive the reinforcement signal from peer_j;
52    - Update action probability of LA based on the reinforcement signal received from peer_j;
53    - Send the information about the probability vector of LA to peer_j;
54    If (peer_j has determined a new landmark peer) Then
55      - Receive the identifier of the new landmark peer from peer_j;
56      - Set the role of the peer using information received from peer_j;
          // if the peer has been selected as new landmark peer it sets its role to landmark peer otherwise it sets its role to
            ordinary peer
57    EndIf
58    - Goto Maintenance phase; // goto line 37
59  EndIf
60  If (peer cannot communicate with its landmark peer) Then
61    - ROLE ← "Unattached peer";
62    - Goto Landmark selection phase;// goto line 12 to select a new landmark peer
63  EndIf
64  End
```

Figure 4. Pseudo code of the proposed algorithm

Before we present the proposed algorithm in more detail, we need to define the following.

1. Rendezvous point (RP) is a global host cache in the overlay network that gives the new peer an entry point to the overlay. Rendezvous points will be implemented by a set of computers that are known to all new peers.

2. CandidateLandmarkSet of cluster C_i denoted by N_{C_i}contains all landmark peers residing in the clusters adjacent to cluster C_i

3. Cluster manager table of $peer_i$ denoted by CM_i saves information about the peers that have the same cluster identifier as $peer_i$ and landmark peers residing in the clusters adjacent to the cluster of $peer_i$. In each cluster, the ordinary peers must be registered in the cluster manager table of the landmark peer. CM_i has six columns "*Identifier*", "*Type*", "*Delay*", "*Probability*", "*Mean*" and "*Variance*". Column *Identifier* contains an identifier (such as IP address). Column *Type* can take either "Landmark peer" or "Ordinary peer". Each of the Columns *Delay*, *Mean or Variance* contains values from the interval [0,∞). Column *Probability* contains a value from the closed interval [0,1].

4. Eight types of messages (*ClusteringRequest*, *ClusteringReady*, *GetDelay*, *SetDelay*, *UpdateState*, *Status*, *UpdateRole* and *Alive*) are used in the proposed algorithm. Table 1 shows the fields of these messages. Field *Type* contains the type of the message. Field *TTL* contains the "time to live value" for the message to ensure that the message dies out. *Sender* and *Receiver* fields contain the sender or receiver identifier (such as IP addresses). Field *Delay* contains the delays to peers. Field *CandidateLandmarkSet* contains a set of identifiers of peers. Field *ClusterID* contains the cluster identifier. Field *ProbValue* contains a value from the closed interval [0,1]. Field *LandmarkPeer* contains a peer identifier. Field *IPeers* contains the identifiers of peers. Field *LastItr* contains either "True" or "False". Field *ReinforcementSignal* contains either "0" or "1".

Now we describe the three phases of the process that $peer_i$ executes in more detail.

A. *Initialization phase*

During the *Initialization* phase performed by $peer_i$, the peer tries to find an appropriate cluster. The *Initialization* phase consists of two sub-phases, the *location finding* phase and the *cluster formation* phase. In the *initialization* phase, $peer_i$ sets its variable *ROLE* to "Unattached peer" and then starts the *location finding* phase. The *location finding* phase is implemented by the *Find_Location* procedure that is shown in Figure 6. The *location finding* phase starts with the *preparation* phase and ends with the *search* phase.

During the *preparation* phase, $peer_i$ connects to the rendezvous point (RP) to fetch boot peers. Boot peers will be used by the peer to find an appropriate cluster. Then, $peer_i$ selects one of the boot peers randomly and finds cluster $C_x$of the selected boot peer. At the end of the *preparation* phase, the peer goes to the *search* phase.

During the *search* phase, $peer_i$ finds $peer_j$, the landmark peer of cluster C_x.$Peer_i$ then finds N_{C_x}.N_{C_x}contains landmark peers of clusters adjacent to the cluster C_x. Then, $peer_i$ sets its variable *ROLE* to "Ordinary Peer" and connects to the landmark peer of cluster C_x if d_{ki} is the same as d_{kj} for all $peer_k \in N_{C_x}$(In Figure 6, function *Meet_Cluster_Criterion* is used to check the criterion). After connecting to the cluster C_x, $peer_i$ goes to the *landmark selection* phase. If $peer_i$ could not connect to cluster C_x, $peer_i$ sets C_x to a cluster C_k such that $k = \arg min_k\{d_{ki} | \forall k \in N_{C_x}\}$ and $d_{ki} < D_{min}$, where D_{min} is the smallest delay measured so far between i and every visited cluster. The initial value of D_{min} is set to ∞. For each peer, a counter is defined to control the iterations of the search process of the peer. The *search* phase will be restarted if the counter of the peer is lower than a certain threshold *MAX_ITERATION*. If the peer could not identify an appropriate cluster, and the counter meets the threshold *MAX_ITERATION*, $peer_i$ goes to the *cluster formation* phase to create a new cluster interconnecting with cluster C_k.

During the *cluster formation* phase, the peer initiates a new cluster with a new *ClusterID* and at the same time, finds landmark peers of clusters adjacent to the cluster of $peer_i$ using the *MakeNeighbor* procedure. In this procedure, to find landmark peers, $peer_i$ repeats the *Find_ Location* procedure from all of the boot peers given by the rendezvous point (RP). $Peer_i$ saves the information of landmark peers in CM_i and then sets the variable *ROLE* to "Landmark Peer" and goes to the maintenance phase.

The *location finding* phase of the proposed algorithm is similar to the locating process of the *mOverlay* algorithm. In the *location finding* phase, for implementing the function *Meet_Cluster_Criterion*, we must specify exactly when two distances can be considered "the same." For a clustering threshold $t \in [0,1)$, we say distances m and n are the same if

$$(m \geq n \wedge (1-t).m \leq n) \vee (m < n \wedge (1-t).n \leq m) \qquad (8)$$

This test (that is also used in [15]) checks the relative difference between two distances. In Figure 6, the function *Meet_Cluster_Criterion* takes a clustering threshold *t*, a cluster identifier *ClusterID*, and a list of delays *DistanceList* and returns true if the delays between the peer and the neighbors of the selected cluster (denoted by *ClusterID*) are the same as the delays that are available in *DistanceList* and returns false otherwise.

```
Algorithm Initialize()
    Inputs: MAX_ITERATION // the stopping condition for the algorithm as a maximum number of iterations
            RP // the identifiers of rendezvous peers
            t// the clustering threshold
    Notations:
    Variable ClusterID determines the identifier of the selected cluster for peer.

    Begin
    - isNewCluster ← False;
    - ROLE ← "Unattached peer";
    //location finding phase
    - [isNewCluster, LandmarkPeer] ← Call Find_Location(MAX_ITERATION,RP,t);// the pseudo code of this procedure
                                                                              is given in Figure 6
    - ClusterID ← LandmarkPeer;// The identifier of landmark peer is used to determine the identifier of the cluster of
                                  the peer
    If (! isNewCluster) Then
      - Connect to LandmarkPeer ;
      - ROLE ← "Ordinary peer";
      - Call SelectLandmark(); //the pseudo code of this procedure is given in Figure 7
    Else
     // cluster formation phase
      - CM ← Call MakeNeighbors (RP);// function MakeNeighbores returns information about landmark
                                        peers of clusters adjacent to the cluster of the peer
      - ROLE ← "Landmark peer";
    EndIf
    - Call Maintain(); //the pseudo code of this procedure is given in Figure 8
    End
```

Figure 5. Pseudo code for initialize procedure

```
Algorithm Find_Location ()
    Inputs: MAX_ITERATION // the stopping condition for the algorithm as a maximum number of iterations
            RP // the identifiers of rendezvous peers
            t// the clustering threshold
    Output: isNewCluster // the flag which determine a new cluster is created or not
            LandmarkPeer // the peer chosen as Landmark peer
    Notations:
    Variable ClusterID determines the identifier of the selected cluster for the peer at any time.
    Variable CandidateLandmarkList determines a list contains landmark peers at any time.
    Variable DistanceList determines a list contains the landmark peers and distances to the peer at any time.
    Variable MinDis determines the distance to the selected landmark peer at any time.
    Variable MinCluster determines the identifier of a cluster which its distance to the peer is equal to MinDis at any time.
    Variable CurrentDis determines the distance to a landmark peer which has minimum delay to the peer at any time.
    Variable CurrentCluster determines a cluster which has minimum distance to the peer at any time.

    Begin
    //preparation phase
    - k ← 1;
    - isSelectionCompleted ← False;
    - isLocatingCompleted ← False;
    - Bootpeers ← Get_boot_peer(RP) ;
    - Bootpeer ← select(Bootpeers) ; // function select returns a random boot peer
    - ClusterID ← Get_Cluster(Bootpeer); // function Get_Cluster returns the ClusterID of the Bootpeer.
    //search phase
    Repeat
      - k ← k + 1;
      - LandmarkList ← Get_Neighbors(ClusterID);// function Get_Neighbors returns the identifiers of the
                                                   landmark peers of adjacent clusters to the cluster of the
                                                   peer.
      - DistanceList ← Measure_Distance(peer, LandmarkList);//function Measure_Distance returns a list
                                                              of Distances from the peer to each peer
                                                              which is available in the LandmarkList
      If (Meet_Cluster_Criterion (t, ClusterID, DistanceList)) Then
        - ROLE ← "Ordinary peer";
        - SelectedClusterID ← ClusterID;
        - isSelectionCompleted ← True;
        - isLocatingCompleted ← True;
      Else
        - [MinDis, MinCluster] ← Min(DistanceList);// function Min returns the information about a peer
                                                      which has minimum distance in the DistanceList
        If (MinDis < CurrentDis) Then
          - CurrentCluster ← MinCluster;
          - CurrentDis ← MinDis;
        EndIf
        If (k > MAX_ITERATION and isSelectionCompleted = False) Then
          - isLocatingCompleted ← True;
          - SelectedCluster ← "Null";
        EndIf
        - ClusterID ← MinCluster;
      EndIf
    Until (isLocatingCompleted);
    End
```

Figure 6. Pseudo code for Find Location procedure

B. *Landmark selection phase*

During the *Landmark Selection* phase performed by $peer_i$, the peer and its neighbors that have the same cluster as $peer_i$ try to find an appropriate landmark peer for the cluster using their *learning automata*. In this phase, a team of *learning automata* is used to adaptively find the landmark peer. In the *Landmark selection* phase, the peer performs a process that consists of five sub-phases: 1.*activation* phase, 2. *local search* phase, 3.*information exchange* phase, 4.*role selection* phase, and 5. *role declaration* phase. These phases are described below.

During the *activation* phase, if the value of variable *ROLE* is equal to "Ordinary peer", $peer_i$ sends a *ClusteringRequest* message to the landmark peer of its cluster to inform the landmark peer that a new peer is connected to the cluster. The field *CandidateLandmarkSet* of the generated *ClusteringRequest* message must be set to "Null". After sending *ClusteringRequest* message, $peer_i$ starts the *maintenance phase*. If the value of the variable *ROLE* is equal to "Unattached peer", $peer_i$ starts the *local search* phase.

During the *local search* phase, a *ClusteringRequest* message, which contains *CandidateLandmarkSet* and *ClusterID*, is initially sent by the peer within its cluster to inform the members of the cluster that the landmark selection phase is started. The values of variables *ClusterID* and *CandidateLandmarkSet* were determined in the *Initialization* phase. After sending the *ClusteringRequest* message, $peer_i$ waits for a certain duration (*LWAIT_DURATION*) to receive a *ClusteringReply* message from its neighbors to gather some information about the other peers of the cluster and the landmark peers of the adjacent clusters. *ClusteringReply* message received from $peer_j$ contains delays to landmark peers that are reported by *ClusteringRequest* message to $peer_j$. The $peer_i$ upon receiving a *ClusteringReply* message from $peer_j$, saves the information of $peer_j$ in CM_i and restarts the *local search* phase. The peers that respond to the *ClusteringRequest* message are called candidate peers. If $peer_i$ does not receive any message during the specified period of *LWAIT_DURATION*, then $peer_i$ goes to the *information exchange* phase.

During the *information exchange* phase, $peer_i$ generates the *GetDelay* messages containing the cluster members that are available in CM_i and then sends the *GetDelay* messages to the candidate peers. After sending the *GetDelay* messages, $peer_i$ waits for a certain duration (*NWAIT_DURATION*) to receive the *SetDelay* messages from those neighbors that had participated in the landmark selection phase. The $peer_i$ upon receiving a *SetDelay* message from $peer_j$, saves the delays that are available in the *SetDelay* message in the CM_i and restarts the *information exchange* phase. If $peer_i$ does not receive any message during the specified period of *NWAIT_DURATION*, then $peer_i$ goes to the *role selection* phase.

During the *role selection* phase, the learning automaton of $peer_i$ selects one of its actions " set the role to landmark peer" or "set the role to ordinary peer" according to its action probability vector, and then $peer_i$ selects an appropriate landmark peer from those peers that are available in CM_i. In the cluster of $peer_i$, for each candidate peer, the delays to landmark peers of

the adjacent clusters and ordinary peers in the cluster are computed during the *information exchange* and *local search* phases. For finding an appropriate landmark peer, for each candidate peer $peer_j$ that is available in CM_i, $peer_i$ computes the mean and variance of delays that are saved for $peer_j$ in CM_i. Using the computed information, $peer_i$ sorts the peers of the CM_i and selects one of the peers that it has the lowest mean and variance. In the pseudo code of the landmark selection algorithm shown in Figure 7, the function find_landmark() finds the landmark peer. After finding the landmark peer, $peer_i$ generates *UpdateState* message containing the selected landmark peer. Then, $peer_i$ computes the reinforcement signal and sends the *UpdateState* messages to all cluster members that are available in CM_i. The reinforcement signal (β) is 1 if the learning automaton of the selected landmark peer has selected "set the role to landmark peer" action and 0 otherwise. After sending the *UpdateState* message, the probability vector of the learning automaton is updated according to the computed reinforcement signal. In $peer_i$, a counter is defined to control the iterations of the learning process of the learning automaton. When the counter of the peer meets a certain threshold (*MAX_ITERATION_LEARNING*), the field *LastItr* of the generated *UpdateState* message must be set to "True", which means that the *role selection* phase of the peer is completed. $Peer_i$ goes to the *role declaration* phase if the counter of $peer_i$ meets the threshold, otherwise $peer_i$ goes to the *local search* phase.

During the *role declaration* phase, $peer_i$ waits for *SWAIT_DURATION* to receive *Status* messages. $Peer_i$, upon receiving a *Status* message from another peer $peer_j$, saves the probability value that is available in the status message in CM_i and then restarts the *role declaration* phase. If $peer_i$ does not receive any messages during the specified period of *SWAIT_DURATION*, then $peer_i$ selects one of candidate peers (including $peer_i$ if the learning automaton of $peer_i$ has selected the " set the role to landmark peer" action) which its probability of selection of "set the role to landmark peer" action is higher than other peers. $peer_i$ declares the selected candidate peer as new landmark peer. When the values of probabilities that are reported by status messages are similar, $peer_i$ selects a candidate peer that has the lowest value for mean and variance of delays that are computed during the *role selection* phase. In the pseudo code of the landmark selection algorithm that is shown in Figure 7, the function select_landmark() selects the landmark peer. After selecting a landmark peer, $peer_i$ generates *UpdateRole* messages containing the selected peer and the *ClusterID* of the peer. The value of variable *ClusterID* was determined in the *Initialization* phase. Then, $peer_i$ sends the *UpdateRole* messages to all members of the cluster and landmark peers of the adjacent clusters. After sending these messages, the peer sets its variable *ROLE*. If the selected landmark peer is $peer_i$, then $peer_i$ sets its variable *ROLE* to "Landmark peer". Otherwise, $peer_i$ sets its variable *ROLE* to "Ordinary peer". At the end of the *role declaration* phase, $peer_i$ goes to the *maintenance* phase.

```
Algorithm Select_Landmark ()
    Inputs: MAX_ITERATION_LEARNING // the stopping condition for the algorithm as a maximum number of iterations
    Notations:
    Let LA denotes the learning automaton of the peer.
    Variable A determines the selected action of the learning automaton of the peer.
    Variable NewLandmark determines the selected landmark peer at any time.
    Variable ROLE at any time determines the role of the peer.

    Begin
    - s ← 1;
    //Activation phase
    If (ROLE = "Ordinary peer") Then
      - Send a ClusteringRequest message to the LandmarkPeer;// the landmark peer is determined in
                                                              initialization phase
    EndIf
    While (ROLE = "Unattached peer") Do
      While (s ≤ MAX_ITERATION_LEARNING) Do
        // local search phase
        - Send ClusteringRequest messages to all members of the cluster;
        - iSLearningCompleted ← False;
        - iSLocalSearchCompleted ← False;
        Repeat
          -Wait for a certain duration LWait_DURATION for receiving a ClusteringReply message;
          If (no message has been received during LWait_DURATION) Then
            - iSLocalSearchCompleted ← True;
          Else
            If (a ClusteringReply message has been received from peer_j) Then
              - Insert the information reported by ClusteringReply message into CM;
            EndIf
          EndIf
        Until(iSLocalSearchCompleted)
        // Information exchange phase
        - Send GetDelay messages to candidate peers;// The peers which respond to the ClusteringRequest
                                                       messages are called candidate peers.
        - iSSetDelay ← False;
        Repeat
          - Wait for a certain duration NWait_DURATION for SetDelay message;
          If (no message has been received during NWait_DURATION) Then
            - iSSetDelay ← True;
          Else
            If (a SetDelay message has been received from peer_j) Then
              - Insert the information reported by SetDelay message into CM;
            EndIf
          EndIf
        Until (iSSetDelay)
        //role selection phase
        - LA selects an action and save the action in variable A;
        - NewLandmark ← find_landmark();
        If (action selected by the learning automaton of NewLandmark peer is "set the role to landmark peer") Then
          - β ← 1;
        Else
          - β ← 0;
        EndIf
        - Send the UpdateState messages to all cluster members;// UpdateState message contains β as reinforcement
                                                                 signal and NewLandmark as landmark peer.
        - LA updates its action probability vector using reinforcement signal β ;
        - s ← s + 1;
      EndWhile
      //role declaration phase
      - iSDeclarationCompleted ← False;
      Repeat
        - Wait for a certain duration SWait_DURATION for receiving Status message;
      If (no message has been received during SWeight_DURATION ) Then
        - iSDeclarationCompleted ← True;
      Else
        If (a Status message has been received during SWeight_DURATION ) Then
          - Insert the information reported by Status message into CM;
        EndIf
      EndIf
      Until (iSDeclarationCompleted)
      - NewLandmark ← select_Landmark();
      - Send UpdateRole messages to all peers which are available in CM;// UpdateRole message contains the
                                                                          NewLandmark and ClusterID of
                                                                          peer
      If (peer is NewLandmark) Then
        - ROLE ← "Landmark peer";
      Else
        - Connect to NewLandmark peer;
        - ROLE ← "Ordinary peer";
      EndIf
    EndWhile
    - Call Maintain();//the pseudo code of this procedure is given in Figure 8
    End
```

Figure 7. Pseudo code for Select_Landmark procedure

C. *Maintenance phase*

During the *maintenance* phase, $peer_i$ does some operation to compensate for the effect of a peer joining and leaving in its cluster. In this phase, $peer_i$ waits for a certain duration (*MWeight_DURATION*) to receive one of the *ClusteringRequest*, *GetDelay, UpdateState, Alive or UpdateRole* messages.

The $peer_i$ upon receiving a *ClusteringRequest* message from a new ordinary peer $peer_j$ saves the information reported by the *ClusteringRequest* message in CM_i, sets its variable *ROLE* to "Unattached peer", and then goes to the *landmark selection* phase. A *ClusteringRequest* message that its field *CandidateLandmarkSet* is equal to "Null" was produced by a new ordinary peer. $peer_i$, upon receiving a *ClusteringRequest* message that contains some peers as *CandidateLandmarkSet* from $peer_j$, saves the information reported by the *ClusteringRequest* message in CM_i and sets its variable *ROLE* to "Unattached peer", that means that the role of $peer_i$ is changed. After changing the role of $peer_i$, the learning automaton of the peer chooses an action. If the chosen action of the learning automaton is equal to "Landmark peer", $peer_i$ finds its delay to those landmark peers that are available in the *CandidateLandmarkSet* reported by the *ClusteringRequest* message and then sends a *ClusteringReply* to $peer_j$ that contains the computed delays. After sending the *ClusteringReply* message, $peer_i$ restarts the maintenance phase.

The $peer_i$ upon receiving a *GetDelay* message from $peer_j$ if its variable *ROLE* is equal to "Unattached peer"computes delays to cluster members reported by the *GetDelay* message as *IPeers*. $Peer_i$ then produces a *SetDelay* message containing the computed delays and then sends the *SetDelay* message to $peer_j$. After sending the *SetDelay* message, $peer_i$ restarts the *maintenance* phase.

The $peer_i$ upon receiving an *UpdateState* message from $peer_j$ if its variable *ROLE* is equal to "Unattached peer" updates its *learning automaton* using information about the selected landmark peer reported by *UpdateState* message. If field *LastItr* of the *UpdateState* message is equal to "true", then it produces a *Status* message containing the action probability of selecting the "set the role to landmark peer" action of the *learning automaton* and sends it to $peer_j$. After sending the *Status* message, $peer_i$ restarts the *maintenance* phase.

The $peer_i$ upon receiving a *UpdateRole* message from $peer_j$ residing in its cluster if its variable *ROLE* is equal to "Unattached peer"sets the value of *ClusterID* to the identifier of the landmark peer using information about the new landmark peer reported by the *UpdateRole* message and then sets its variable *ROLE*. If $peer_i$ has been selected as a new landmark peer, then $peer_i$ sets its variable *ROLE* to "Landmark peer", otherwise $peer_i$ sets its variable *ROLE* to "Ordinary peer". When the peer sets its *ROLE* to "Ordinary peer", the peer connects to the new landmark peer and restarts the *maintenance* phase. The $peer_i$, upon receiving an *UpdateRole* message from $peer_j$ residing in an adjacent cluster, updates the information of CM_i about adjacent clusters with information of the *UpdateRole* message and then restarts the *maintenance* phase.

The $peer_i$, upon receiving an *Alive* message from landmark peer $peer_j$ if its variable *ROLE* is equal to "Ordinary peer", connects to $peer_j$ and updates its cluster manager table (it is called CM_i) with the cluster manager table of $peer_j$ (it is called CM_j). After updating the cluster manager table, $peer_i$ restarts the *maintenance* phase.

If $peer_i$ does not receive any message during the specified period of *MWeight_DURATION*, $peer_i$ does some operations according to the value of the variable *ROLE*. If the value of the variable *ROLE* is equal to "Landmark peer", $peer_i$ sends *Alive* messages to all cluster members to inform them their landmark peer is still alive. If the value of variable *ROLE* is equal to "Ordinary peer", $peer_i$ waits for the certain duration of *MWeight_DURATION* to receive an *Alive* message from a landmark peer $peer_j$. If $peer_i$ does not receive any message during the period of *MWeight_DURATION*, $peer_i$ changes its role to unattached and then starts the landmark selection phase. When an ordinary peer cannot receive the *Alive* message from its landmark peer, this means that the landmark peer may not exist. If $peer_i$ receives an *Alive* message during the period, $peer_i$ updates its cluster manager table and restarts the *maintenance* phase.

```
Algorithm Maintain ()
    Notations:
    Let LA denotes the learning automaton of the peer.
    Let CM denotes the cluster manager table of the peer.
    Variable A determines the selected action of LA at any time.
    Variable ROLE at any time determines the role of the peer.

Begin
- maintainInterrupt ← "False";
While (maintainInterrupt = "False") Do
  - Wait for a certain duration MWait_DURATION for ClusteringeRequest, GetDelay, UpdateState ,Alive or
                                                                              UpdateRole messages;
 If (no message has been received during MWeight_DURATION)Then
   If (ROLE="Landmark peer") Then
     - Send Alive messages to all cluster members stored in CM;
   Else
     - Wait for a certain duration MWait_DURATION for receiving an Alive message;
     If (no message has been received during MWeight_DURATION ) Then
       - ROLE ← "Unattached peer";
       - maintainInterrupt ← "True";
     Else
       - Connect to peer_j;
       - Updates CM with CM_j;
     EndIf
   EndIf
 EndIf
 If (a ClusteringeRequest message has been received from peer_j) Then
   - Insert the information reported by ClusteringeRequest message into CM;
   If (CandidateLandmark field of ClusteringeRequest message is equal to "Null" ) Then
     - ROLE ← "Unattached peer";
     - maintainInterrupt ← "True";
   Else
     - ROLE ← "Unattached peer";
     - LA selects an action and save the action in variable A;
     If (A = "set the role to landmark peer") Then
       - Find delays to landmark peers; // landmark peers are reported by Clustering Request message as
                                                                          CandidateLandmarkSet
       - Send a ClusteringReply message to peer_j;// ClusteringReply message contains the computed delays
     EndIf
   EndIf
 EndIf
 If (a GetDelay message has been received from another peer_j and ROLE = "Unattached peer")Then
   - Find delays to cluster members; // cluster members are reported by GetDelay message as IPeers
   - Send a SetDelay message to peer_j;// SetDelay message contains the found delays
 EndIf
 If (an UpdateState message has been received from peer_j and ROLE = "Unattached peer") Then
   - Find β; // β is reported by UpdateState message as ReinforcementSignal
   - LA updates its action probability vector using reinforcement signal β ;
   If (LastItr = "True") Then // LastItr is a field of the UpdateState message
     - Send a Status message to peer_j;// Status message contains the probability of selecting
                                                  "set the role to landmark peer" action of LA
   EndIf
 EndIf
 If (an UpdateRole message has been received from peer_j)Then
   If (ClusterID of the peer is equal to ClusterID in UpdateRole message and ROLE = "Unattached peer") Then
     If (peer = Landmark peer) Then // landmark peer is reported by UpdateRole message
       - ROLE ← "Landmark peer";
     Else
       - ROLE ← "Ordinary peer";
     EndIf
   EndIf
   If (ClusterID of the peer is not equal to ClusterID in UpdateRole message) Then
     - Insert the information reported by UpdateRole message into CM;
   EndIf
 EndIf
 If (an Alive message has been received from peer_j and ROLE = "Ordinary peer") Then
   - Connect to peer_j;
   - Updates CM with CM_j;
 EndIf
EndWhile
- Call select_landmark();//the pseudo code of this procedure is given in Figure 7
End
```

Figure 8. Pseudo code for maintain procedure

5. EXPERIMENTAL RESULTS

All simulations have been implemented using the *OverSim* simulator[49]. This simulator supports different types of underlay network models: *Simple*, *SingleHost*, *INET,and ReaSE*. The *Simple model* is the most scalable model and the *ReaSE model* is able to generate different types of underlay networks. We used the *ReaSE model* to generate router-level topologies[50]. Experiments reported in this section are conducted on three different underlying networks: *Topology.1(k)* as an example of the *Simple model* where *k* determines the size of the network, *Topology.2* and *Topology.3* as examples of the *ReaSE model* as given in Table 2.

To evaluate the performance of the proposed algorithm that we call *lOverlay*, *lOverlay* is compared with the *mOverlay algorithm*[14] and two other algorithms, *uOverlay* and *bOverlay*, as described below.

- *uOverlay algorithm*: This algorithm is the proposed algorithm in which the *learning automaton* residing in each peer is replaced with a *pure chance automaton*. In a *pure chance automaton,* the actions of the automaton are always selected with equal probabilities[25]. In the *uOverlay* algorithm, the landmark peer of each cluster can be changed during the operation of the network based on information about delays between peers.
- *bOverlay algorithm:* This algorithm is an extension of the *mOverlay* algorithm[15]. This algorithm, such as the proposed algorithm, uses a threshold called the grouping threshold to specify exactly when two distances can be considered "the same". To study the effect of the grouping threshold on the performance of the *bOverlay,* in an experiment, the proposed algorithm is compared with the *bOverlay* for different values of the grouping threshold.

In both the *lOverlay* and the *uOverlay* algorithms, all waiting time durations are set to10*s*, and *MAX_ITERATION* is set to1000. For *lOverlay*, each peer is equipped with a variable structure learning automaton of type L_{RI} with the reward parameter a=0.1. In the grouping criterion of *mOverlay*, we used Equation 8 with t=0.3 to check the relative difference between two distances. The results reported are averages over 10 different runs. The algorithms are compared with respect to three metrics: Total communication delay (TCD), Mean round-trip time (MRT), and Mean cluster size (MCS). These metrics are briefly explained below.

- **Total communication delay (TCD)** is the total of all-pairs end-to-end communication delay of the overlay network. This metric is measured using Equation 1.
- **Mean round-trip time(MRT)**is the mean round-trip time between members of the same cluster. Clusters with only one peer are ignored in this case.
- **Mean cluster size(MCS)** is the average number of peers per cluster.

Experiment 1

This experiment is conducted to study the impact of the network size on the performance of the *lOverlay* and the *uOverlay* algorithms when the underlay network topology is generated by the simple model as described before. For this purpose, we generate five topologies from *Topology.1(k)* for k=10000, 20000, 30000, 40000 and 50000. For both the *lOverlay* and the *uOverlay* algorithms, the clustering threshold is set to 0.3 and *MAX_ITERATION_LEARNING* is set to *5*. The results obtained are compared with the results obtained for the *mOverlay* algorithm with respect to the criteria mentioned above. According to the results of this experiment that are shown in Figure 9 to Figure 11, one may conclude that the *lOverlay* algorithm performs better than the *uOverlay* and the *mOverlay* algorithms because in the *lOverlay* algorithm, the set of

learning automata used by the peers of each cluster in interacting with their environments gradually finds an appropriate landmark peer for that cluster. Finding an appropriate landmark peer results in both a lower *TCD* and a lower *MRT*. *uOverlay* also performs better than *mOverlay* in terms of *TCD* and *MRT*. This means that even selection of landmark peer using the *pure chance automaton* performs better than when the landmark peer of each cluster is fixed during the operation of the network because when *pure chance automata* are used, other peers in the cluster have a chance to become a landmark peer in case that they have a lower delay to cluster members and also landmark peers of the adjacent clusters.

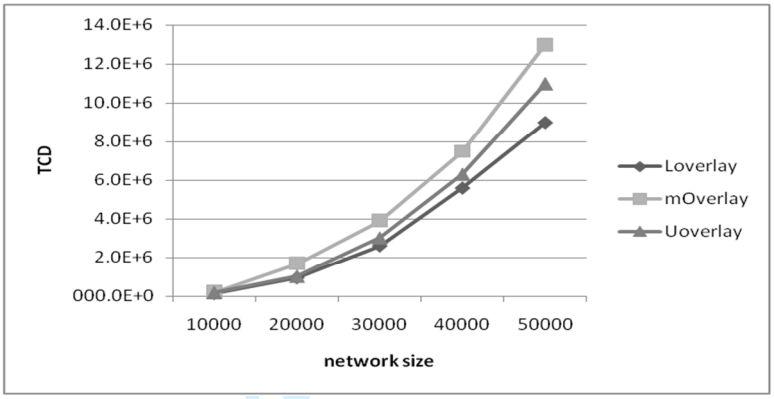


Figure 9. Comparison of *lOverlay* with *mOverlay* and *uOverlay* with respect to *TCD*

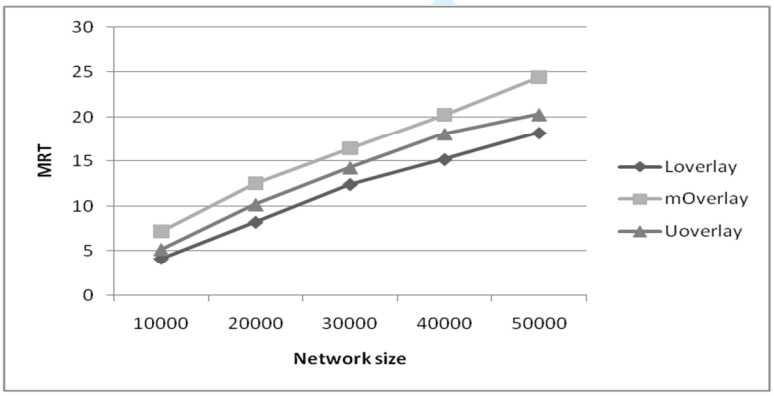


Figure 10. Comparison of *lOverlay* with *mOverlay* and *uOverlay* with respect to *MRT*

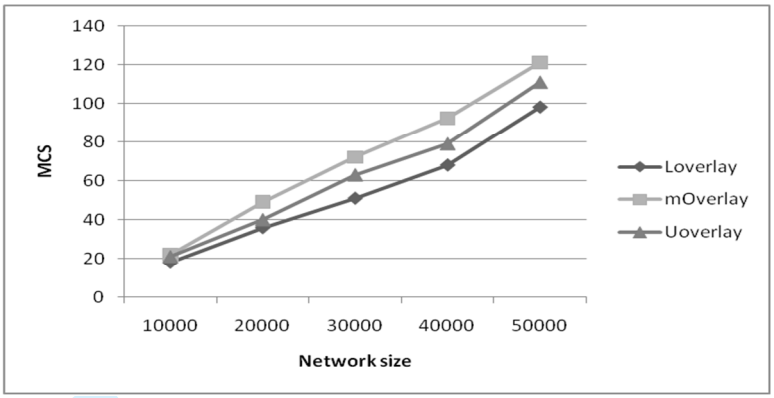


Figure 11. Comparison of *lOverlay* with mOverlay and *uOverlay* with respect to *MCS*

Experiment 2

This experiment is conducted to study the impact of the parameter *MAX_ITERATION_LEARNING* on the performance of the *lOverlay* and the *uOverlay* algorithms. For this purpose, *lOverlay* and *uOverlay* are tested for five values of parameter *MAX_ITERATION_LEARNING=1,5,10,15 and 20*, and parameter *t* is set to 0.3. In this experiment, *Topology1(10000)* is used as the underlay topology. The results obtained are compared with the results obtained for the *mOverlay* algorithm with respect to *MRT*, *TCD*, and *MCS*. It can be noted from the results given in Figure 12 to Figure 14, as the value of *MAX_ITERATION_LEARNING* increases, the performance of the *lOverlay* algorithm in terms of *TCD* and *MRT* improves because more iterations give the team of learning automata of each cluster more time to find a better landmark peer.

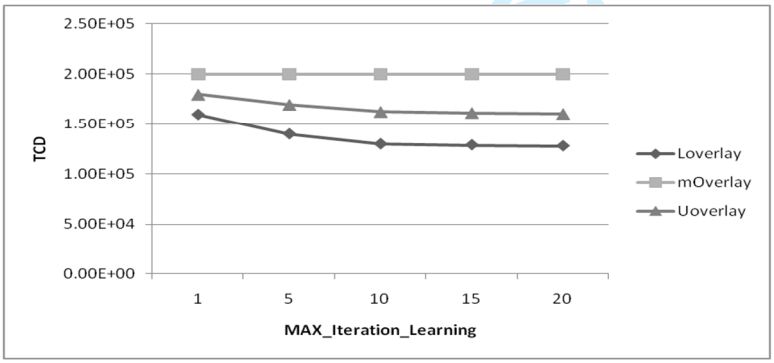


Figure 12. The impact of parameter *MAX_ITERATION_LEARNING* on the performance of the proposed algorithm with respect to *TCD*

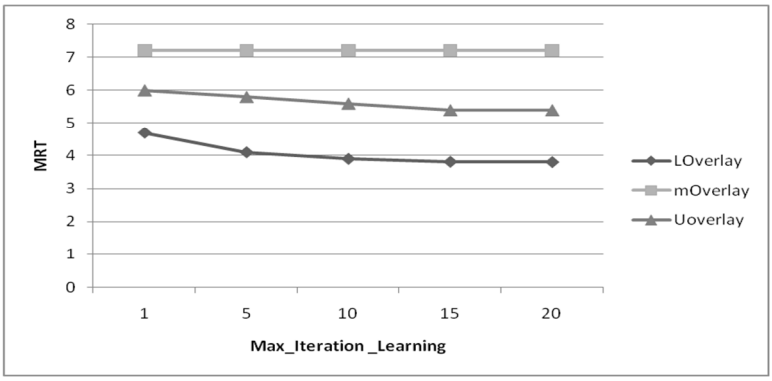


Figure 13. The impact of parameter *MAX_ITERATION_LEARNING* on the performance of the proposed algorithm with respect to *MRT*

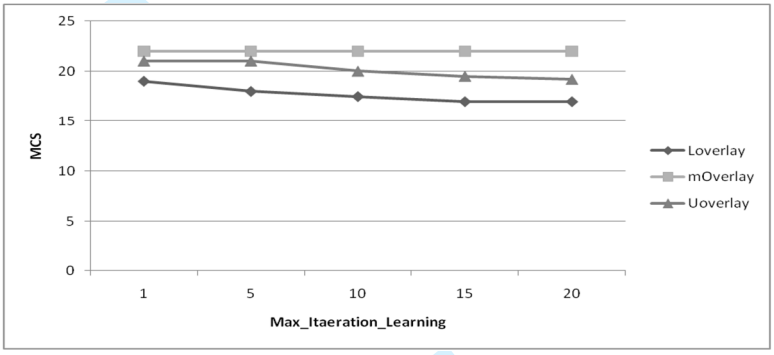


Figure 14. The impact of parameter *MAX_ITERATION_LEARNING* on the performance of the proposed algorithm with respect to *MCS*

Experiment 3

This experiment is conducted to study the impact of the parameter clustering threshold *t* on the performance of the *lOverlay* and the *uOverlay* algorithms. For this purpose, *lOverlay* and *uOverlay* are tested for four values for parameter *t*=0.2,0.3,0.4 and 0.5. *MAX_ITERATION_LEARNING* is set to 5. In this experiment, *Topology1(10000)* is used as the underlay topology. The results obtained are compared with the results obtained for the *mOverlay* algorithm in terms of *MRT*, *TCD*, and *MCS*. From the results given in Figure 15 to Figure 17, we may conclude the following:

- For both the *lOverlay* and the *uOverlay* algorithms, increasing the value of parameter *t* leads to increasing *MCS* and *MRT*.
- In terms of *TCD*, the *lOverlay* and the *uOverlay* algorithms perform better than the *mOverlay* algorithm for *t*=0.3 and 0.4. Low performance of the proposed algorithm for *t*=0.2 and *t*=0.5 as shown in the figures is caused by an inappropriate set of clusters created by the algorithm in the overlay network.

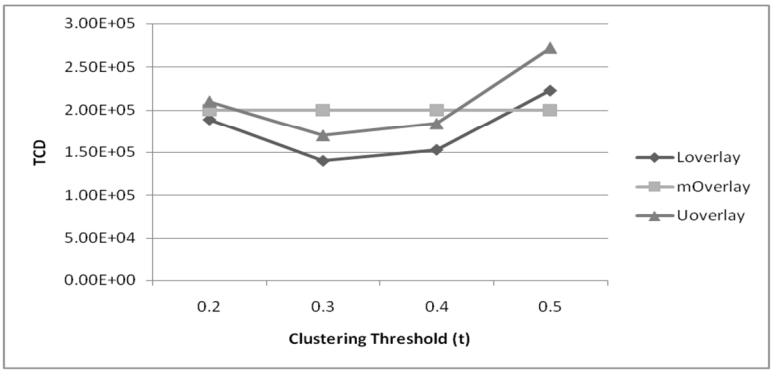


Figure 15. The impact of the parameter clustering threshold *t* on the performance of the proposed algorithm with respect to *TCD*

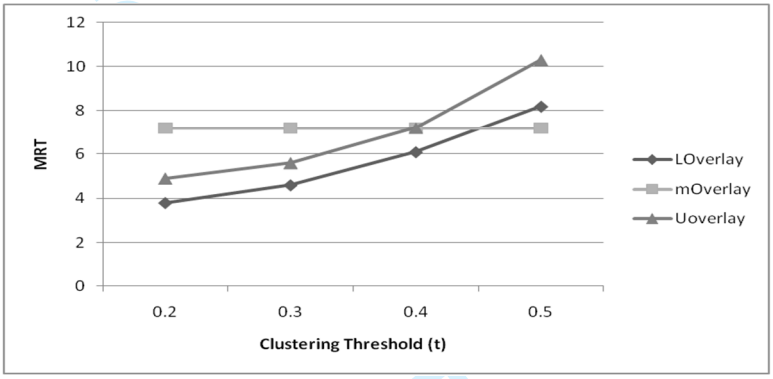


Figure 16. The impact of the parameter clustering threshold *t* on the performance of the proposed algorithm with respect to *MRT*

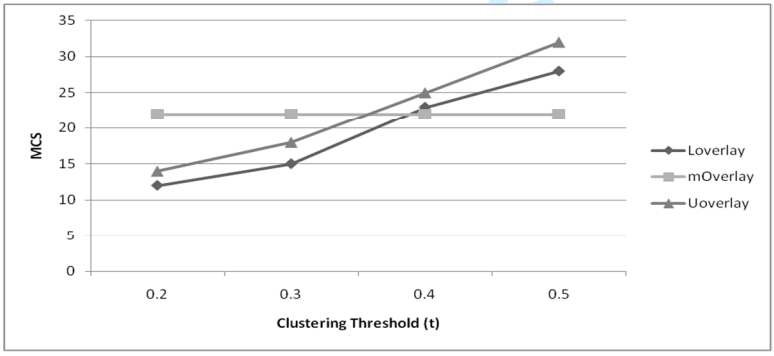


Figure 17. The impact of the parameter clustering threshold *t* on the performance of the proposed algorithm with respect to *MCS*

Experiment 4

In this experiment, we study the performance of the *lOverlay* and the *uOverlay* algorithms on topologies *Topology.2* and *Topology.3* that both belong to the class of router level topologies[50]. *Topology.2* contains few and populated groups, and *Topology.3* contains many

groups with low population in which the distance between the groups is far greater than the distance between the peers in each group. The results obtained from the *lOverlay* and the *uOverlay* algorithms are compared with the results obtained for the *mOverlay* algorithm with respect to *TCD*, *MRT*, and *MCS* when the clustering threshold t=0.3 and *MAX_ITERATION_LEARNING=5*. From the results of this experiment given in Figure 18 to Figure 20, we may conclude that for both topologies *Topology.2* and *Topology.3*, the *lOverlay* algorithm performs better than the *mOverlay* and the *uOverlay* algorithms in terms of *TCD*. This means that the *lOverlay* algorithm is efficient even for router level topologies.

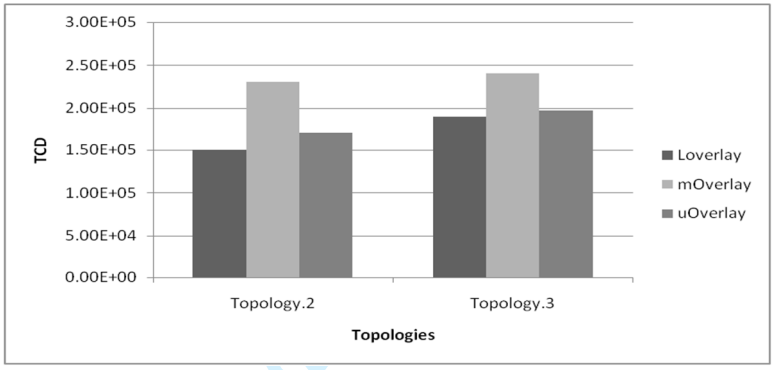


Figure 18. The impact of underlay topology on the performance of the proposed algorithm with respect to *TCD*

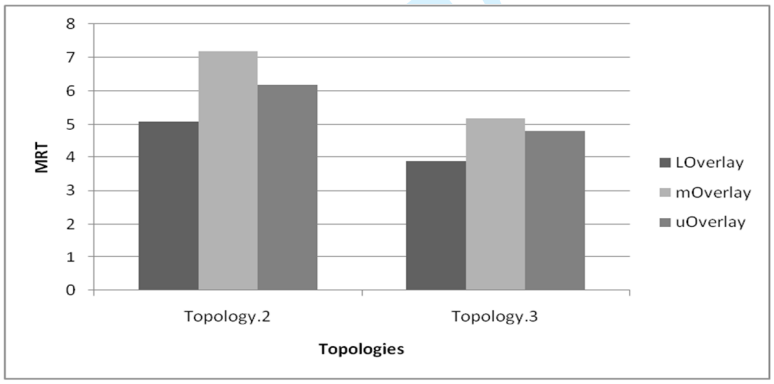


Figure 19. The impact of underlay topology on the performance of the proposed algorithm with respect to *MRT*

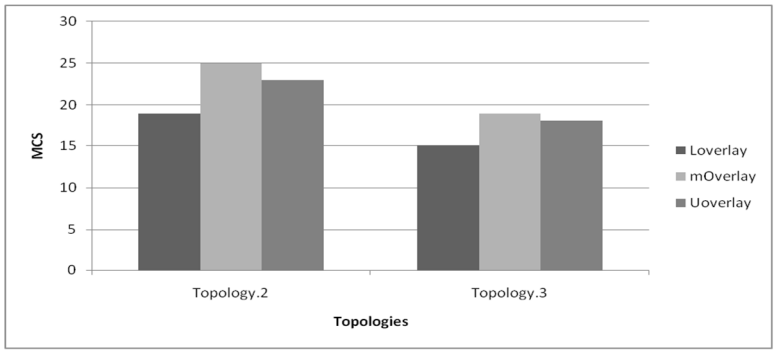


Figure 20. The impact of underlay topology on the performance of the proposed algorithm with respect to *MCS*

Experiment 5

In this experiment, we compare the performance of the *mOverlay*, the *lOverlay,* and the *uOverlay* algorithms to the performance of the *bOverlay* algorithm with respect to *TCD*, *MRT*, and *MCS*. For this purpose, the *bOverlay* algorithm is tested for four values for the parameter grouping threshold *g*=0.2, 0.3, 0.4 and 0.5, and *Topology1(10000)* as the underlay topology. For the *lOverlay* and the *uOverlay* algorithms, the clustering threshold and *MAX_ITERATION_LEARNING* are set to 0.3 and 10, respectively. The results of this experiment are given in Figure 21 to Figure 23. From the results of this experiment, we may conclude the following:

- In terms of *TCD*, the *lOverlay* algorithm performs better than the *bOverlay* algorithm for all the tested values of grouping threshold. Note that, unlike the *bOverlay* algorithm, the *lOverlay* algorithm uses an adaptive algorithm for landmark selection during the operation of the network.
- Increasing the value of the parameter grouping threshold results in increasing *MRT* and *MCS*.

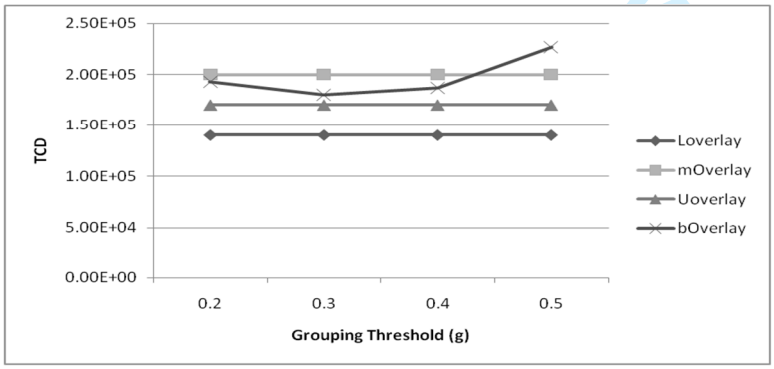


Figure 21. Comparison of different algorithms with *bOverlay* with respect to *TCD*

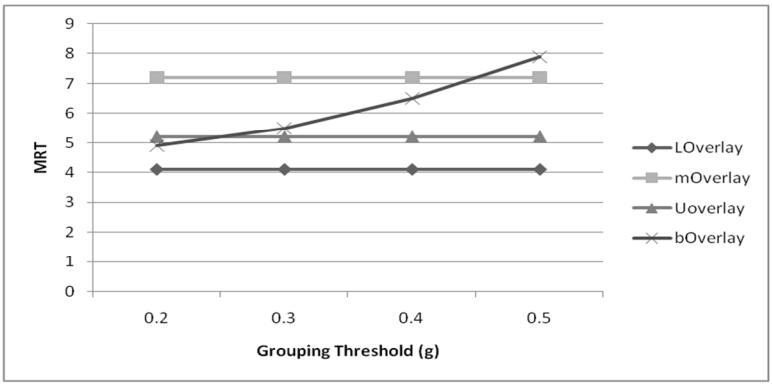


Figure 22. Comparison of different algorithms with *bOverlay* with respect to *MRT*

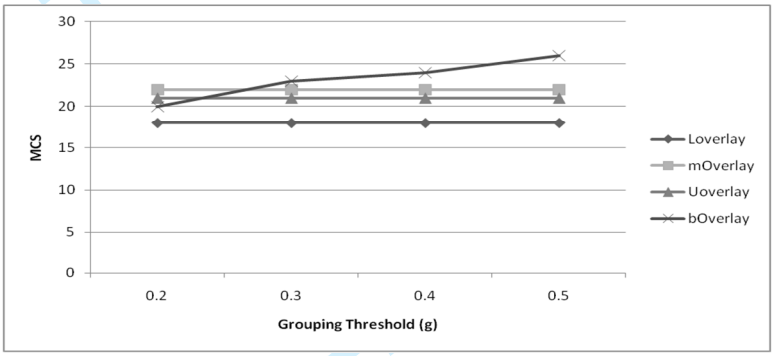


Figure 23. Comparison of different algorithms with *bOverlay* with respect to *MCS*

6. CONCLUSIONS

In this paper, a novel adaptive landmark clustering algorithm called the *lOverlay algorithm* for unstructured peer-to-peer networks was proposed. The proposed algorithm obtained from the combination of *mOverlay* and *learning automata* uses an adaptive algorithm based on *learning automata* for landmark selection during the operation of the network. Utilizing the proposed algorithm, the peers of each cluster in interaction with each other find an appropriate landmark peer in an adaptive manner. The results of the simulation show the superiority of the proposed algorithm over *mOverlay*[14] and the algorithm reported in [15] in terms of communication delay and average round-trip time between peers within clusters. In addition, to show the significance of learning in the proposed landmark selection algorithm, the proposed algorithm was compared with the *uOverlay* algorithm that is a version of the proposed algorithm in which the *learning automaton* in each peer is replaced with a *pure chance automaton*.

REFERENCES

1. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review 2001; **31**(4): 149–160.
2. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. ACM SIGCOMM Computer Communication Review 2001; **31**(4): 161–172.
3. Aberer K, Cudré-Mauroux P, Datta A, Despotovic Z, Hauswirth M, Punceva M, Schmidt R. P-Grid: a self-organizing structured P2P system. ACM SIGMOD Record 2003; **32**(3): 29–33.
4. Chawathe Y, Ratnasamy S, Breslau L, Lanham N, Shenker S. Making gnutella-like p2p systems scalable. Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications, Karlsruhe, Germany, 2003; 407–418.
5. Kwok YK. Peer-to-Peer Computing: Applications, Architecture, Protocols, and Challenges. CRC Press: Boca Raton, Florida, 2011.
6. Qiu T, Chan E, Ye M, Chen G, Zhao BY. Peer-exchange schemes to handle mismatch in peer-to-peer systems. The Journal of Supercomputing 2009; **48**(1): 15–42.
7. Liu Y. A two-hop solution to solving topology mismatch. IEEE Transactions on Parallel and Distributed Systems 2008; **19**: 1591–1600.
8. Liu Y, Zhuang Z, Xiao L, Ni LM. A distributed approach to solving overlay mismatching problem. Proceedings of the 24th International Conference on Distributed Computing Systems, Tokyo, Japan, 2004; 132–139.
9. Hsiao HC, Liao H, Yeh PS. A Near-Optimal Algorithm Attacking the Topology Mismatch Problem in Unstructured Peer-to-Peer Networks. IEEE Transactions on Parallel and Distributed Systems 2010; **21**(7): 983–997.
10. Lalitha B, Rao CDS. GPS Based Topology Matching Algorithm For P2P Systems. International Journal of Advanced Research in Computer Science and Software Engineering 2013; **3**(3): 145–154.
11. Aggarwal V, Feldmann A, Scheideler C. Can ISPs and P2P users cooperate for improved performance?. ACM SIGCOMM Computer Communication Review 2007: **37**(3); 29–40.
12. Ratnasamy S, Handley M, Karp R, Shenker S. Topologically-aware overlay construction and server selection. Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies, New York, USA, 2002; 1190–1199.
13. Tian R, Xiong Y, Zhang Q, Li B, Zhao BY, Li X. Hybrid overlay structure based on random walks. Proceedings of the 4th International workshop on Peer-To-Peer Systems. Ithaca, NY, USA, 2005; 152–162.
14. Zhang XY, Zhang Q, Zhang Z, Song G, Zhu W. A construction of locality-aware overlay network: mOverlay and its performance. IEEE Journal on Selected Areas in Communications 2004; **22**(1): 18–28.
15. Scheidegger M, Braun T. Improved locality-aware grouping in overlay networks. Proceedings of the Kommunikation in Verteilten Systemen, Bern, Switzerland, 2007; 27–38.
16. Wolf S, Merz P. Evolutionary local search for the super-peer selection problem and the p-hub median problem. Proceedings of the 4th international conference on Hybrid metaheuristics, Berlin, Heidelberg, 2007; 1–15.

17. Ju HJ, Du LJ. Nodes Clustering Method in Large-Scale Network. Proceedings of 8th International Conference on Wireless Communications, Networking and Mobile Computing, Shanghai, China, 2012; 1–4.
18. Li Y, Yu Z. An improved genetic algorithm for network nodes clustering. Proceedings of the Second international conference on Information Computing and Applications, Qinhuangdao, China, 2011; 399–406.
19. Jiang Y, You J, He X. A Particle Swarm Based Network Hosts Clustering Algorithm for Peer-to-Peer Networks. Proceedings of the International Conference on Computational Intelligence and Security, Guangzhou, China, 2006; 1176–1179.
20. Hsiao HC, Liao H, Huang CC. Resolving the topology mismatch problem in unstructured peer-to-peer networks. IEEE Transactions on Parallel and Distributed Systems 2009; **20**(11): 1668–1681.
21. Papadakis H, Fragopoulou P, Markatos E, Roussopoulos M. ITA: Innocuous Topology Awareness for Unstructured P2P Networks. IEEE Transactions on Parallel and Distributed Systems 2013; **24**(8): 1589–1601.
22. Xiao L, Liu Y, Ni LM. Improving unstructured peer-to-peer systems by adaptive connection establishment. IEEE Transactions on Computers 2005; **54**(9): 1091–1103.
23. Xu Z, Tang C, Zhang Z. Building topology-aware overlays using global soft-state. Proceedings of the 23rd International Conference on Distributed Computing Systems, Providence, RI, USA, 2003; 500–508.
24. O'kelly ME. A quadratic integer program for the location of interacting hub facilities. European Journal of Operational Research 1987; **32**(3): 393–404.
25. Narendra KS, Thathachar MAL. Learning Automata. Prentice-Hall: Englewood Cliffs, NJ, 1989.
26. Thathachar MAL, Sastry PS. Networks of Learning Automata: Techniques for online stochastic optimization. Kluwer Academic Publishers: Dordrecht, 2003.
27. Esnaashari M, Meybodi MR. Deployment of a Mobile Wireless Sensor Network with k-Coverage Constraint: A Cellular Learning Automata Approach. Wireless Networks 2013; **19**(5): 945-968.
28. Esnaashari M, Meybodi MR. A cellular learning automata-based deployment strategy for mobile wireless sensor networks. Journal of Parallel and Distributed Computing 2011; **71**(5): 988–1001.
29. Safavi SM, Meybodi MR, Esnaashari M. Learning Automata based Face-aware Mobicast. Wireless Personal Communications 2014; **77**(3)**:** 1923-1933.
30. Mostafaei H, Meybodi MR, Esnaashari M. A Learning Automata Based Area Coverage Algorithm for Wireless Sensor Networks. Journal of Electronic Science and Technology 2010; **8**(3)**:** 200-205.
31. Nicopolitidis P. Performance fairness across multiple applications in wireless push systems. International Journal of Communication Systems 2013. DOI:10.1002/dac.2648
32. Nicopolitidis P, Chrysostomou C, Papadimitriou GI, Pitsillides A, Pomportsis AS. On the efficient use of multiple channels by single-receiver clients in wireless data broadcasting. International Journal of Communication Systems 2014; **27**: 513–520. DOI: 10.1002/dac.2375
33. Polatoglou M, Nicopolitidis P, Papadimitriou GI. On low-complexity adaptive wireless push-based data broadcasting. International Journal of Communication Systems 2014; **27**(1): 194–200.
34. Misra S, Chatterjee SS, Guizani M. Stochastic learning automata-based channel selection in cognitive radio/dynamic spectrum access for WiMAX networks. International Journal of Communication Systems 2014. DOI: 10.1002/dac.270
35. Song Y, Zhang C, Fang Y. Stochastic traffic engineering in multi-hop cognitive wireless mesh networks. IEEE Transactions on Mobile Computing 2010; **9**(3): 305–316.

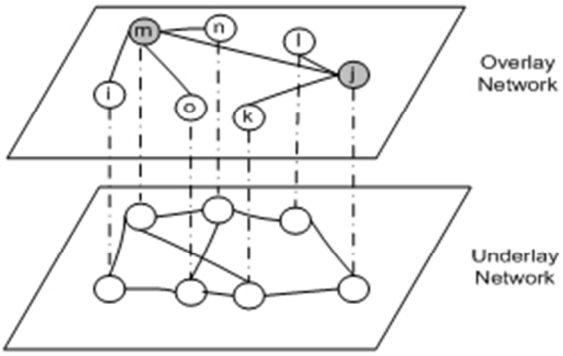
36. Akbari Torkestani J, Meybodi MR. A learning automata-based cognitive radio for clustered wireless ad-hoc networks. Journal of Network and Systems Management 2011; **19**(2): 278–297.
37. Jahanshahi M, Dehghan M, Meybodi MR. LAMR: Learning Automata based Multicast Routing Protocol for Multi-Channel Multi-Radio Wireless Mesh Networks. Applied Intelligence 2013; **38**(1): 58-77.
38. Gholami S, Meybodi MR, Saghiri AM. A Learning Automata-Based Version of SG-1 Protocol for Super-Peer Selection in Peer-to-Peer Networks. Proceedings of the 10th International Conference on Computing and Information Technology, Angsana Laguna, Phuket, Thailand, 2014; 189–201.
39. Ghorbani M, Meybodi MR, Saghiri AM. A new version of k-random walks algorithm in peer-to-peer networks utilizing learning automata. Proceedings of the 5th Conference on Information and Knowledge Technology, Shiraz, Iran, 2013; 1–6.
40. Ghorbani M, Meybodi MR, Saghiri AM. A novel self-adaptive search algorithm for unstructured peer-to-peer networks utilizing learning automata. Proceedings of the 3rd Joint Conference of AI & Robotics and 5th RoboCup Iran Open International Symposium, Qazvin, Iran, 2013; 1–6.
41. Akbari Torkestani J. A multi-attribute resource discovery algorithm for peer-to-peer grids. Applied Artificial Intelligence 2013; **27**(7): 575–598.
42. Ghorbani M, Saghiri AM, Meybodi MR. A Novel Learning based Search Algorithm for Unstructured Peer to Peer Networks. Technical Journal of Engineering and Applied Sciences 2013; **3**(2): 145–149.
43. Beigy H, Meybodi MR. A self-organizing channel assignment algorithm: A cellular learning automata approach. Intelligent Data Engineering and Automated Learning 2003; **14**: 119–126.
44. Meybodi MR, Kharazmi MR. Cellular learning automata and its application to image processing. Journal of Amirkabir 2004; **14**(56A): 1101–1126.
45. Beigy H, Meybodi MR. Adaptation of parameters of BP algorithm using learning automata. Proceedings of the Sixth Brazilian Symposium on Neural Networks, Rio de Janeiro, Brazil 2000; 24–31.
46. Meybodi MR, Beigy H. A note on learning automata-based schemes for adaptation of BP parameters. Neurocomputing 2002; **48**(1): 957–974.
47. Rezvanian A , Meybodi MR. Tracking extrema in dynamic environments using a learning automata-based immune algorithm. Grid and Distributed Computing Control and Automation, Springer, 2010; 216–225.
48. Hashemi AB, Meybodi MR. A note on the learning automata based algorithms for adaptive parameter selection in PSO. Applied Soft Computing 2011; **11**(1): 689–705.
49. Baumgart I, Heep B, Krause S. OverSim: A scalable and flexible overlay framework for simulation and real network applications. Proceedings of the Peer-to-Peer Computing, Seattle, Washington, USA, 2009; 87–88.
50. Li L, Alderson D, Willinger W, Doyle J. A first-principles approach to understanding the internet's router-level topology. ACM SIGCOMM Computer Communication Review 2004; **34**(4): 3–14.
51. Mahadevan P, Krioukov D, Fomenkov M, Huffaker B, Dimitropoulos X, Vahdat A. Lessons from three views of the Internet topology. University of California, San Diego, CA, USA, tr-2005-02, 2005.
52. Huffaker B, Plummer D, Moore D, Claffy KC. Topology discovery by active probing. Proceedings of Symposium on Applications and the Internet Workshops, Washington DC, USA, 2002; 90–96.

Table 1 the formats of the used messages

Message name	The fields of the message
ClusteringRequest	Type,Sender,TTL, CandidateLandmarkSet,ClusterID
ClusteringReady	Type, Sender, Reciver, TTL ,Delay
GetDelay	Type, Sender, Reciver, TTL, IPeers , Delay
SetDelay	Type, Sender, Reciver, TTL ,Delay
UpdateState	Type, Sender, Reciver, TTL , LandmarkPeer, LastItr, ReinforcementSignal
Status	Type, Sender, Reciver, TTL, ProbValue
UpdateRole	Type, Sender, Reciver, TTL,LandmarkPeer, ClusterID
Alive	Type, Sender, Reciver, TTL

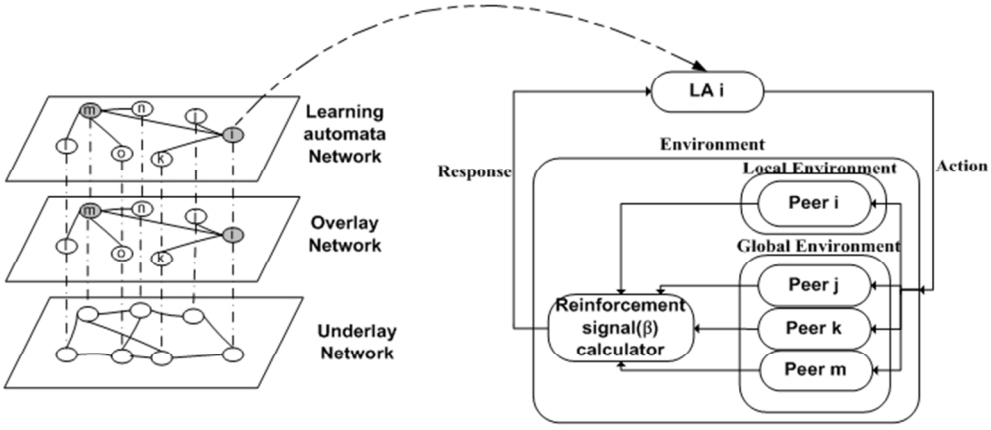
Table 2 Underlay topologies

Underlay topology	Descriptions
Topology. 1(k)	In this underlay topology, k peers are placed on a N-Dimensional Euclidean space and the Internet latencies are based on CAIDA/Skitter [51], [52] data.
Topology. 2	Consists of 10 autonomous systems, and about 1000 router -level peers. This topology contains few and populated groups
Topology. 3	Consists of 100 autonomous systems, and about 100 router -level peers. This topology contains many low populated groups in which the distance between the groups is far greater than the distance between the peers in each group.



83x55mm (96 x 96 DPI)

94x35mm (96 x 96 DPI)



192x88mm (96 x 96 DPI)

Algorithm peer_management_skeleton()

Inputs: *MAX_ITERATION_LEARNING* // maximum number of iterations
Notations
Let *LA* denotes the learning automaton of the *peer*.
Variable *ROLE* at any time determines the role that the *peer* is playing which is either "ordinary peer" or "landmark peer" or "unattached peer".
Variable C_x determines the identifier of the cluster that the *peer* belongs to at any time.
Variable *A* determines the selected action of the learning automaton of the *peer*.

01 **Begin**
02 **// Initialization phase**
03 - $ROLE \leftarrow$ "Unattached *peer*";
04 - Find an appropriate cluster C_x to which the *peer* belongs using *mOverlay* locating algorithm;
05 **If** (an appropriate cluster cannot be found) **Then**
06 - Create a new cluster C_x;
07 - Goto **Maintenance phase**;// goto line 37
08 **Else**
09 - $ROLE \leftarrow$ "Ordinary *peer*";
10 - Connect to the landmark peer of cluster C_x;
11 **EndIf**
12 **//Landmark selection phase**
13 **If** ($ROLE$ = "Ordinary *peer*") **Then**
14 - Activate the landmark peer of cluster C_x;// activate the landmark peer to select a new landmark for cluster C_x
15 **EndIf**
16 **While** ($ROLE$ = "Unattached *peer*")
17 - Activate the peers of cluster C_x;// activate all peers of cluster *Cx* to collectively find the landmark peer
18 - $s \leftarrow 1$;// s is a counter defined to control the number of times that the learning automaton in the *peer* is updated;
19 **While** ($s < Max\_Iteratin\_Learning$)
20 - Find those peers of cluster C_x whose learning automata have selected action "landmark peer";
// call these peers candidate peers
21 - Gather information about delays;
// gather information about delays from each candidate peer $peer_k$ to the landmark peers of clusters adjacent to cluster C_x and all ordinary peers of cluster C_x
22 - Save the gathered information about delays into a local database;
23 - LA selects an action and save the action in variable *A*;
24 - Choose one of the peers of cluster C_x using the local database of the *peer* ;
// choose a peer which has the Lowest mean and variance of delays to ordinary peers of C_x and the landmark peers of clusters adjacent to cluster C_x
25 **If** (the chosen landmark peer is one of the candidate peers) **Then**
26 - $\beta \leftarrow 1$;
27 **Else**
28 - $\beta \leftarrow 0$;
29 **EndIf**
30 - Send the reinforcement signal(β) to all pees of cluster C_x;
31 - Update the action probability of LA based on β;
32 - $s = s + 1$;
33 **EndWhile**
34 - Gather information about learning automata of peers of cluster C_x;
35 - Declare the role of the *peer* and send messages containing the identifier of new landmark peer to the peers of cluster C_x ; // a peer is declared as new landmark peer if the probability of selecting "set the role to landmark peer" action of its learning automaton is higher than the probabilities of selecting "set the role to landmark peer" action of other learning automata of the peers in cluster C_x
36 **End while**
37 **//Maintenance phase**
38 - Wait until some event occur;
39 **If** (*peer* has been activated by a new ordinary peer $peer_j$ in cluster C_x) **Then**
40 - Connect to $peer_j$;
41 - $ROLE \leftarrow$ "Unattached *peer*";
42 - Goto **Landmarselection phase**;// goto line 12 to select a new landmark peer
43 **EndIf**
44 **If** (*peer* has been activated by an unattached peer $peer_j$ in cluster C_x) **Then**
45 - $ROLE \leftarrow$ "Unattached *peer*";
46 - LA selects an action and save the action in variable *A*;
47 **If** (A = "set the role to landmark peer") **Then**
48 - Gather information about delays;
// Gather information about delays from the *peer* to the landmark peers of clusters adjacent to cluster C_x and delays from the peer to all ordinary peers of cluster C_x
49 - Send the gathered information and the selected action selected to $peer_j$;
50 **EndIf**
51 - Receive the reinforcement signal from $peer_j$;
52 - Update action probability of LA based on the reinforcement signal received from $peer_j$;
53 - Send the information about the probability vector of LA to $peer_j$;
54 **If** ($peer_j$ has determined a new landmark peer) **Then**
55 - Receive the identifier of the new landmark peer from $peer_j$;
56 - Set the role of the *peer* using information received from $peer_j$;
// if the *peer* has been selected as new landmark peer it sets its role to landmark peer otherwise it sets its role to ordinary peer
57 **EndIf**
58 - Goto **Maintenance phase**; // goto line 37
59 **EndIf**
60 **If** (*peer* cannot communicate with its landmark peer) **Then**
61 - $ROLE \leftarrow$ "Unattached *peer*";
62 - Goto **Landmark selection phase**;// goto line 12 to select a new landmark peer
63 **EndIf**
64 **End**

201x382mm (96 x 96 DPI)

Algorithm Initialize()

Inputs: *MAX_ITERATION* // the stopping condition for the algorithm as a maximum number of iterations
RP // the identifiers of rendezvous peers
t// the clustering threshold

Notations:
Variable *ClusterID* determines the identifier of the selected cluster for *peer*.

Begin
- *isNewCluster* ⟵ *False*;
- *ROLE* ⟵ "Unattached *peer*";
//location finding phase
- [*isNewCluster*, *LandmarkPeer*] ⟵ Call Find_Location(*MAX_ITERATION*,*RP*,*t*);// the pseudo code of this procedure is given in Figure 6
- *ClusterID* ⟵ *LandmarkPeer*;// The identifier of landmark peer is used to determine the identifier of the cluster of the *peer*

If (! *isNewCluster*) **Then**
- Connect to *LandmarkPeer* ;
- ROLE ⟵ "Ordinary *peer*";
- Call SelectLandmark(); //the pseudo code of this procedure is given in Figure 7

Else
// cluster formation phase
- *CM* ⟵ Call MakeNeighbors (*RP*);// function MakeNeighbores returns information about landmark peers of clusters adjacent to the cluster of the *peer*
- *ROLE* ⟵ "Landmark *peer*";

EndIf
- Call Maintain(); //the pseudo code of this procedure is given in Figure 8

End

202x126mm (96 x 96 DPI)

Algorithm Find_Location ()

Inputs: *MAX_ITERATION* // the stopping condition for the algorithm as a maximum number of iterations
RP // the identifiers of rendezvous peers
t// the clustering threshold
Output: *isNewCluster* // the flag which determine a new cluster is created or not
LandmarkPeer // the peer chosen as Landmark peer
Notations:
Variable *ClusterID* determines the identifier of the selected cluster for the *peer* at any time.
Variable *CandidateLandmarkList* determines a list contains landmark peers at any time.
Variable *DistanceList* determines a list contains the landmark peers and distances to the *peer* at any time.
Variable *MinDis* determines the distance to the selected landmark peer at any time.
Variable *MinCluster* determines the identifier of a cluster which its distance to the *peer* is equal to MinDis at any time.
Variable *CurrentDis* determines the distance to a landmark peer which has minimum delay to the *peer* at any time.
Variable *CurrentCluster* determines a cluster which has minimum distance to the *peer* at any time.

```
Begin
//preparation phase
- k ← 1;
- isSelectionCompleted ← False;
- isLocatingCompleted ← False;
- Bootpeers ← Get_boot_peer(RP) ;
- Bootpeer ← select(Bootpeers) ; // function select returns a random boot peer
- ClusterID ← Get_Cluster(Bootpeer); // function Get_Cluster returns the ClusterID of the Bootpeer.
//search phase
Repeat
  - k ← k + 1;
  - LandmarkList ← Get_Neighbors(ClusterID);// function Get_Neighbors returns the identifiers of the
                                              landmark peers of adjacent clusters to the cluster of the
                                              peer.
  - DistanceList ← Measure_Distance(peer, LandmarkList);//function Measure_Distance returns a list
                                                          of Distances from the peer to each peer
                                                          which is available in the LandmarkList
  If (Meet_Cluster_Criterion (t, ClusterID, DistanceList)) Then
    - ROLE ← "Ordinary peer";
    - SelectedClusterID ← ClusterID;
    - isSelectionCompleted ← True;
    - isLocatingCompleted ← True;
  Else
    - [MinDis, MinCluster] ← Min(DistanceList);// function Min returns the information about a peer
                                                 which has minimum distance in the DistanceList
    If (MinDis < CurrentDis) Then
      - CurrentCluster ← MinCluster;
      - CurrentDis ← MinDis;
    EndIf
    If (k > MAX_ITERATION and isSelectionCompleted = False) Then
      - isLocatingCompleted ← True;
      - SelectedCluster ← "Null";
    EndIf
    - ClusterID ← MinCluster;
  EndIf
Until (isLocatingCompleted);
End
```

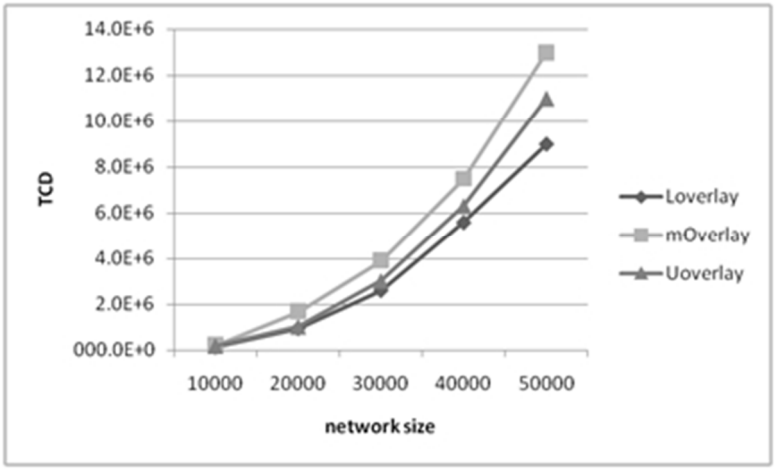
201x236mm (96 x 96 DPI)

```
Algorithm Select_Landmark ()
    Inputs: MAX_ITERATION_LEARNING // the stopping condition for the algorithm as a maximum number of iterations
    Notations:
    Let LA denotes the learning automaton of the peer.
    Variable A determines the selected action of the learning automaton of the peer.
    Variable NewLandmark determines the selected landmark peer at any time.
    Variable ROLE at any time determines the role of the peer.

    Begin
    - s ← 1;
    //Activation phase
    If (ROLE = "Ordinary peer") Then
      - Send a ClusteringRequest message to the LandmarkPeer;// the landmark peer is determined in
                                                              initialization phase
    EndIf
    While (ROLE = "Unattached peer") Do
      While (s ≤ MAX_ITERATION_LEARNING) Do
        // local search phase
        - Send ClusteringRequest messages to all members of the cluster;
        - iSLearningCompleted ← False;
        - iSLocalSearchCompleted ← False;
        Repeat
          -Wait for a certain duration LWait_DURATION for receiving a ClusteringReply message;
          If (no message has been received during LWait_DURATION) Then
            - iSLocalSearchCompleted ← True;
          Else
            If (a ClusteringReply message has been received from peer_i) Then
              - Insert the information reported by ClusteringReply message into CM_i;
            EndIf
          EndIf
        Until(iSLocalSearchCompleted)
        // Information exchange phase
        - Send GetDelay messages to candidate peers;// The peers which respond to the ClusteringRequest
                                                       messages are called candidate peers.
        - iSSetDelay ← False;
        Repeat
          - Wait for a certain duration NWait_DURATION for SetDelay message;
          If (no message has been received during NWait_DURATION) Then
            - iSSetDelay ← True;
          Else
            If (a SetDelay message has been received from peer_i) Then
              - Insert the information reported by SetDelay message into CM_i;
            EndIf
          EndIf
        Until (iSSetDelay)
        //role selection phase
        - LA selects an action and save the action in variable A;
        - NewLandmark ← find_landmark();
        If (action selected by the learning automaton of NewLandmark peer is "set the role to landmark peer") Then
          - β ← 1;
        Else
          - β ← 0;
        EndIf
        - Send the UpdateState messages to all cluster members;// UpdateState message contains β as reinforcement
                                                                  signal and NewLandmark as landmark peer.
        - LA updates its action probability vector using reinforcement signal β ;
        - s ← s + 1;
      EndWhile
      //role declaration phase
      - iSDeclarationCompleted ← False;
      Repeat
        - Wait for a certain duration SWait_DURATION for receiving Status message;
        If (no message has been received during SWeight_DURATION ) Then
          - iSDeclarationCompleted ← True;
        Else
          If (a Status message has been received during SWeight_DURATION ) Then
            - Insert the information reported by Status message into CM_i;
          EndIf
        EndIf
      Until (iSDeclarationCompleted)
      - NewLandmark ← select_Landmark();
      - Send UpdateRole messages to all peers which are available in CM_i;// UpdateRole message contains the
                                                                            NewLandmark and ClusterID of
                                                                            peer
      If (peer is NewLandmark) Then
        - ROLE ← "Landmark peer";
      Else
        - Connect to NewLandmark peer;
        - ROLE ← "Ordinary peer";
      EndIf
    EndWhile
    - Call Maintain();//the pseudo code of this procedure is given in Figure 8
    End
```

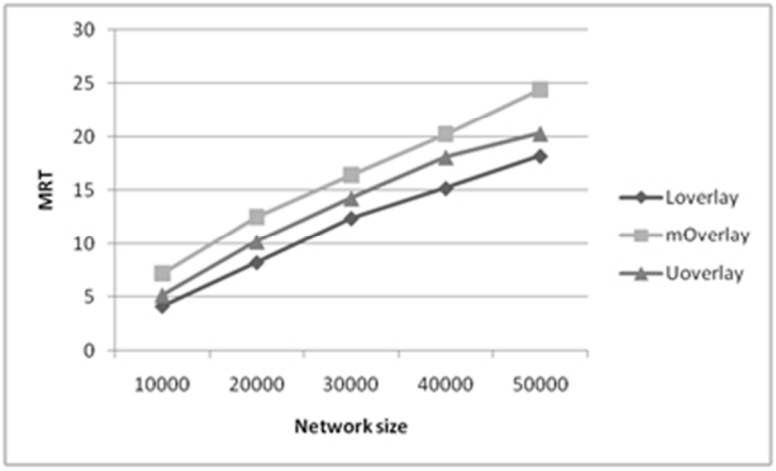
205x375mm (96 x 96 DPI)

```
Algorithm Maintain ()
    Notations:
    Let LA denotes the learning automaton of the peer.
    Let CM denotes the cluster manager table of the peer.
    Variable A determines the selected action of LA at any time.
    Variable ROLE at any time determines the role of the peer.

Begin
- maintainInterrupt ← "False";
While (maintainInterrupt = "False") Do
  - Wait for a certain duration MWait_DURATION for ClusteringRequest, GetDelay, UpdateState ,Alive or
                                                                    UpdateRole messages;
 If (no message has been received during MWeight_DURATION)Then
   If (ROLE="Landmark peer") Then
    - Send Alive messages to all cluster members stored in CM;
   Else
    - Wait for a certain duration MWait_DURATION for receiving an Alive message;
    If (no message has been received during MWeight_DURATION ) Then
      - ROLE ← "Unattached peer";
      - maintainInterrupt ← "True";
    Else
      - Connect to peer_i;
      - Updates CM with CM_i;
    EndIf
   EndIf
 EndIf
 If (a ClusteringRequest message has been received from peer_j) Then
   - Insert the information reported by ClusteringRequest message into CM;
   If (CandidateLandmark field of ClusteringRequest message is equal to "Null" ) Then
     - ROLE ← "Unattached peer";
     - maintainInterrupt ← "True";
   Else
     - ROLE ← "Unattached peer";
     - LA selects an action and save the action in variable A;
     If (A = "set the role to landmark peer") Then
       - Find delays to landmark peers; // landmark peers are reported by Clustering Request message as
                                          CandidateLandmarkSet
       - Send a ClusteringReply message to peer_j;// ClusteringReply message contains the computed delays
     EndIf
   EndIf
 EndIf
 If (a GetDelay message has been received from another peer_j and ROLE = "Unattached peer")Then
   - Find delays to cluster members; // cluster members are reported by GetDelay message as IPeers
   - Send a SetDelay message to peer_j;// SetDelay message contains the found delays
 EndIf
 If (an UpdateState message has been received from peer_j and ROLE = "Unattached peer") Then
   - Find β; // β is reported by UpdateState message as ReinforcementSignal
   - LA updates its action probability vector using reinforcement signal β ;
   If (LastItr = "True") Then // LastItr is a field of the UpdateState message
     - Send a Status message to peer_i;// Status message contains the probability of selecting
                                         "set the role to landmark peer" action of LA
   EndIf
 EndIf
 If (an UpdateRole message has been received from peer_j)Then
   If (ClusterID of the peer is equal to ClusterID in UpdateRole message and ROLE = "Unattached peer") Then
     If (peer = Landmark peer) Then // landmark peer is reported by UpdateRole message
       - ROLE ← "Landmark peer";
     Else
       - ROLE ← "Ordinary peer";
     EndIf
   EndIf
   If (ClusterID of the peer is not equal to ClusterID in UpdateRole message) Then
     - Insert the information reported by UpdateRole message into CM;
   EndIf
 EndIf
 If (an Alive message has been received from peer_j and ROLE = "Ordinary peer") Then
   - Connect to peer_i;
   - Updates CM with CM_i;
 EndIf
EndWhile
- Call select_landmark();//the pseudo code of this procedure is given in Figure 7
End
```

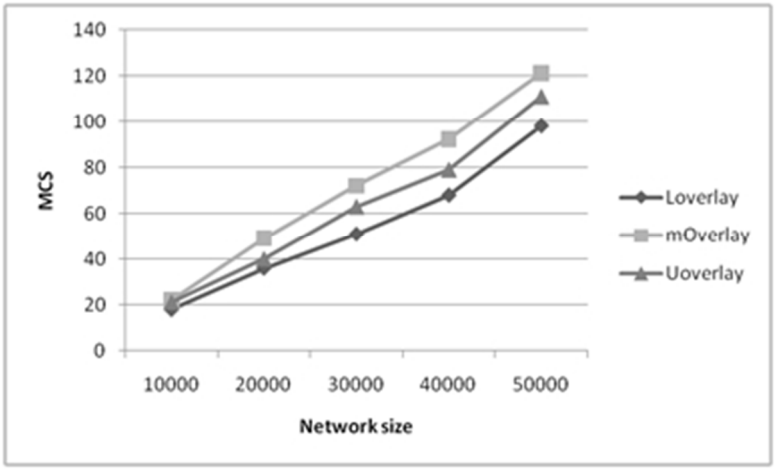
201x324mm (96 x 96 DPI)



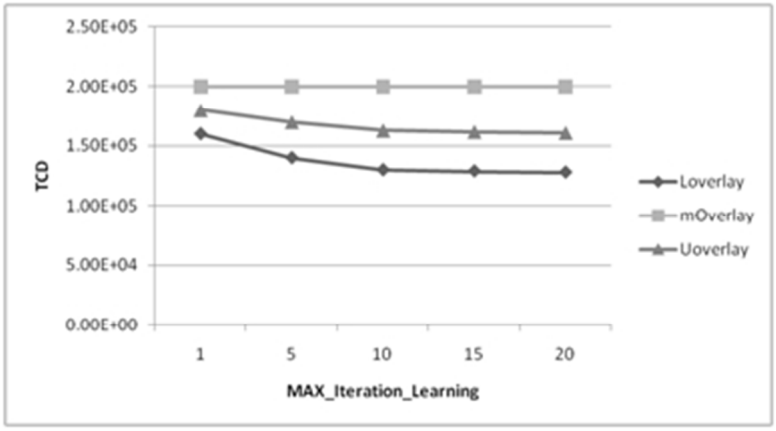
102x62mm (96 x 96 DPI)



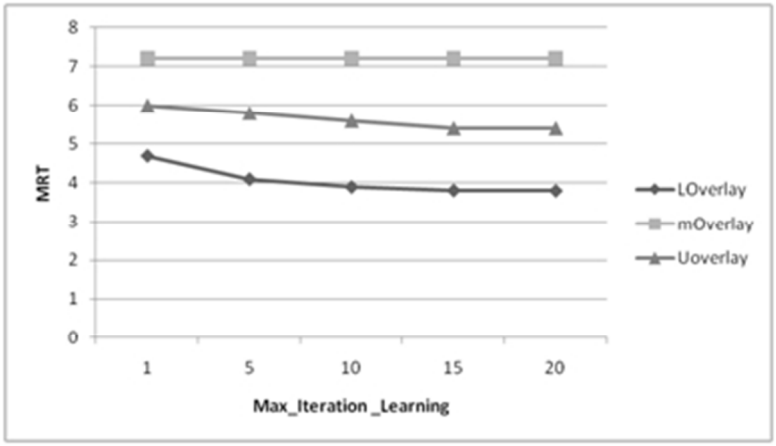
102x62mm (96 x 96 DPI)



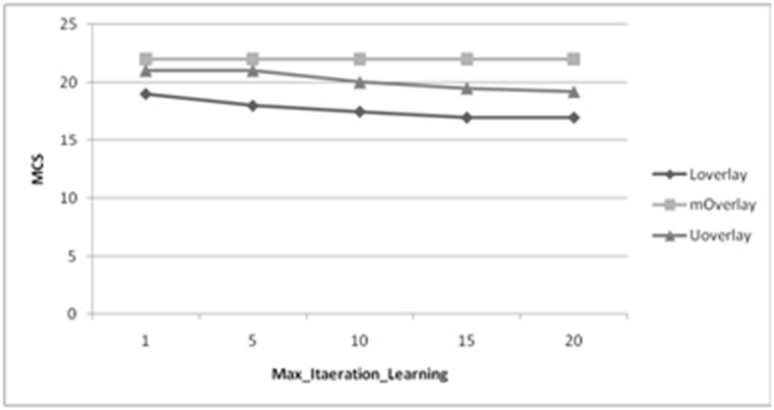
102x62mm (96 x 96 DPI)



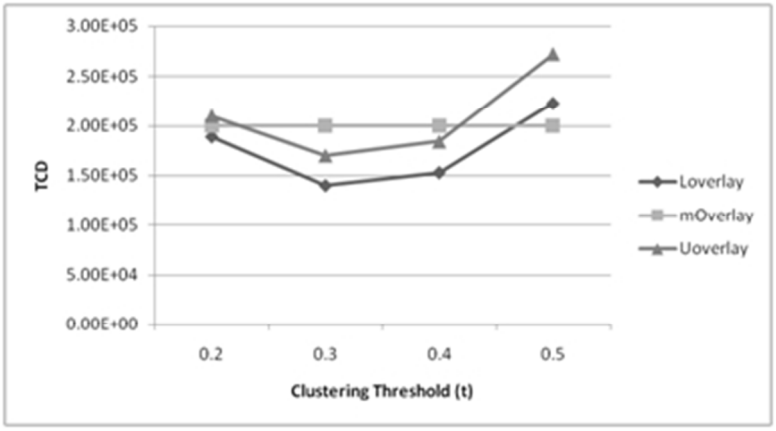
102x56mm (96 x 96 DPI)



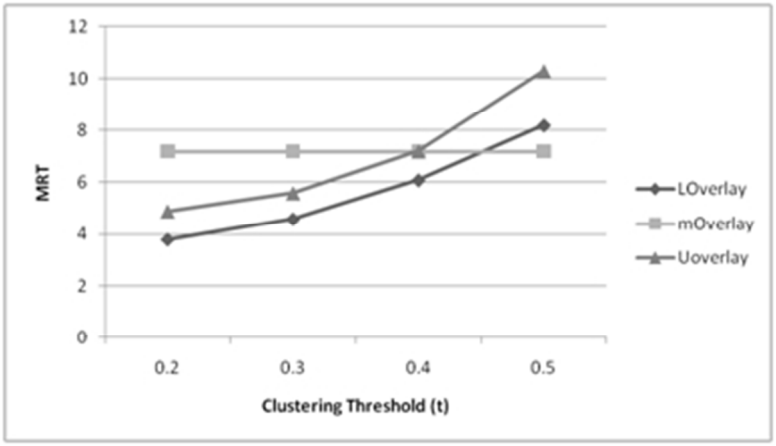
102x59mm (96 x 96 DPI)



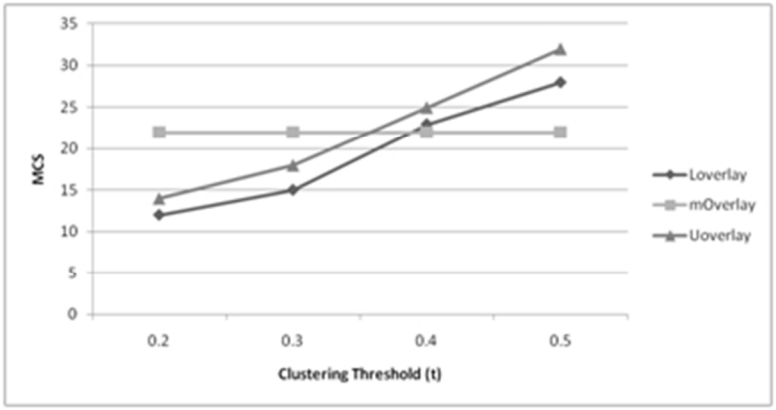
101x54mm (96 x 96 DPI)



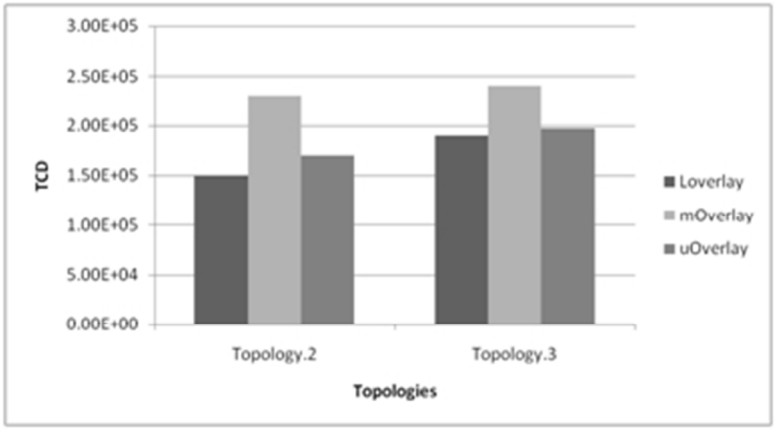
102x56mm (96 x 96 DPI)



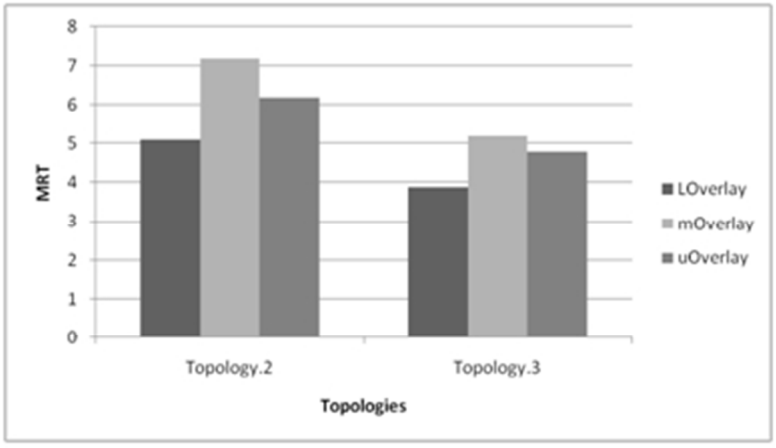
102x59mm (96 x 96 DPI)



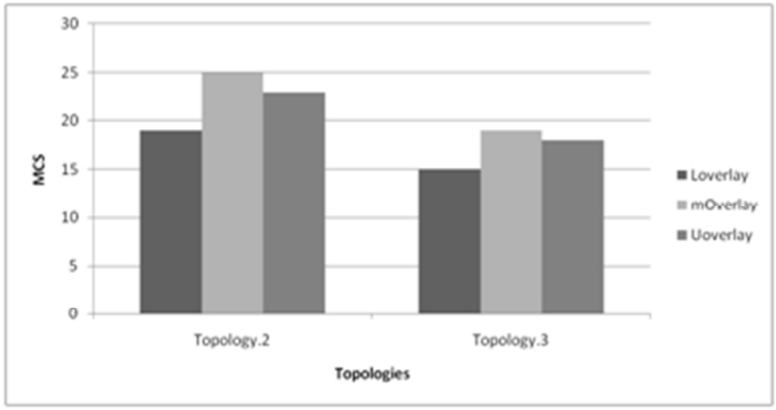
103x53mm (96 x 96 DPI)



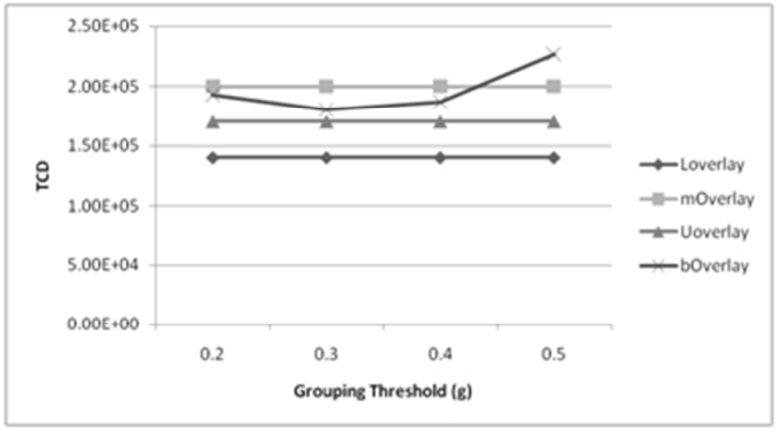
103x56mm (96 x 96 DPI)



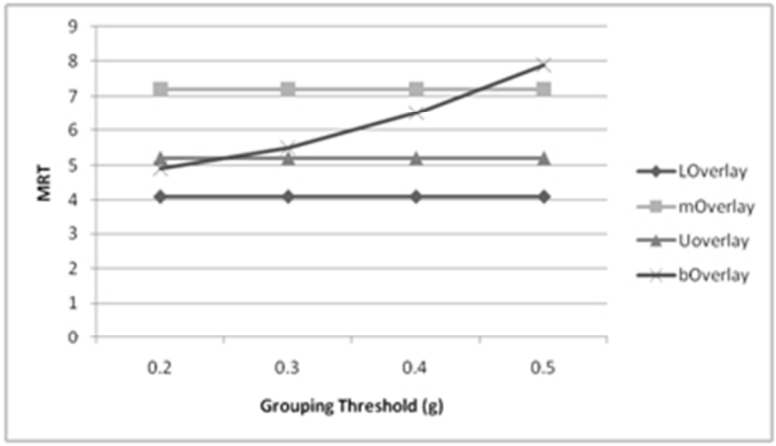
103x59mm (96 x 96 DPI)



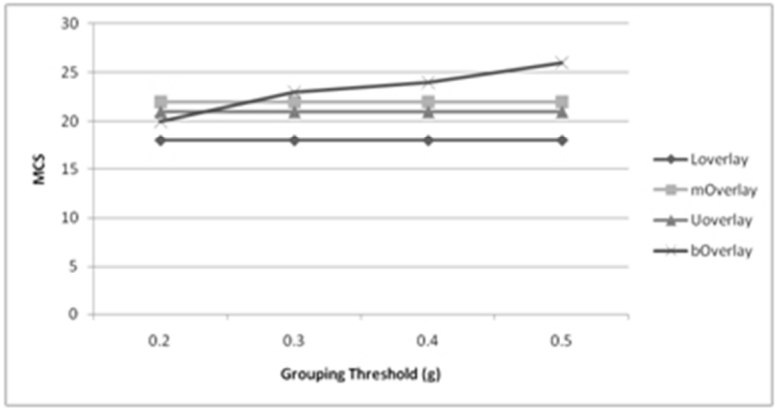
103x54mm (96 x 96 DPI)



103x56mm (96 x 96 DPI)



103x59mm (96 x 96 DPI)



103x53mm (96 x 96 DPI)