Summer 2011

MELOC - memory and location optimized caching for mobile Ad hoc networks

Lekshmi Manian Chidambaram

MELOC - MEMORY AND LOCATION OPTIMIZED CACHING FOR MOBILE AD HOC NETWORKS

by

LEKSHMI MANIAN CHIDAMBARAM

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2011

Approved by

Dr. Sanjay Kumar Madria, Advisor

Dr. Sriram Chellappan

Dr. Jagannathan Sarangapani

PUBLICATION THESIS OPTION

This thesis consists of the following article that has been submitted for publication as follows:

Pages 11-37 have been submitted to the 40th International Conference on Parallel Processing (IEEE ICPP 2011).

This thesis consists of the following article that will be submitted for publication as follows:

Pages 38-72 will be submitted to IEEE Transaction on Mobile Computing.

ABSTRACT

The advancement of Mobile ad hoc networks (MANET) is tremendous in the field of social and military applications. Caching and Replication are the two common techniques used to improve data access efficiency in Mobile Ad hoc networks. Caching favors data access efficiency by bringing data closer to the source. Existing caching approaches are deficient in reducing the number of cache locations, thus reducing the number of copies, which is needed for many mission critical applications considering safety and security. Conversely, reducing the number of caches should not affect the efficiency of data access. We design an efficient broker based caching model named "Memory and Location Optimized Caching (MELOC)", which reduces the number of cache locations, and at the same time preserves data access efficiency. Our caching model mostly chooses centrally located nodes as cache location. In addition, we cache only essential data closer to the source, saving memory. Hence our approach bears the name "Memory and Location Optimized caching (MELOC)". Our initial MELOC model suits only small MANET topology of 30 nodes. We further extend our initial caching model to suit large MANET topology of 100 nodes by overcoming certain disadvantages pertaining to large network topology.

ACKNOWLEDGMENTS

I am very grateful to my advisor, committee members and family on being part of this accomplishment. Primarily, I would like to thank my advisor Dr. Sanjay Kumar Madria, who gave me the opportunity to work in this research. His support and encouragement in this research is tremendous. His advice and guidance helped me in critical situations and gave me the passion to complete this thesis successfully. Further, I articulate my deep gratitude to Dr. Sriram Chellappan and Dr. Jagannathan Sarangapani for serving as my committee members and giving their valuable time to review this work.

I am grateful to all my colleagues and family members for their personal and professional help. This project is partially funded by the Air Force Research Laboratory (AFRL) in Rome, New York and I express my profound gratitude to them as well. Finally, I am glad to dedicate this work to the scientific community of Computer Science.

TABLE OF CONTENTS

SECTION

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

Mobile Ad hoc networks are self configuring wireless infrastructure, which favors communication when there are no access points. Such infrastructures are incredibly useful in military operations, where it is hard to construct fixed base stations. MANETs are rapidly deployable and self configuring, which is needed in rescue operations and battle field. Such infrastructures are even used in industries to provide communication between peers under drastic conditions, when the main infrastructure fails. Mobile Ad hoc networks can also connect to internet gateways to extract features from the internet. Such infrastructure bears the name MANET, and can be used by students on campus for file sharing and discussions. Mobile Ad hoc network has a self configuring nature, which is advantageous in critical situations. Conversely, MANET bear disadvantage due to mobility, which causes rapidly changing network topology and disconnections. Like wired networks, MANETs can have different network topologies. Some of the existing topologies of MANET are, pure hierarchical model, broker based model and hybrid model (hierarchical broker based). A sample mobile ad hoc network formed by conventional computer devices such as laptops and PDAs is shown in Figure 1.1.

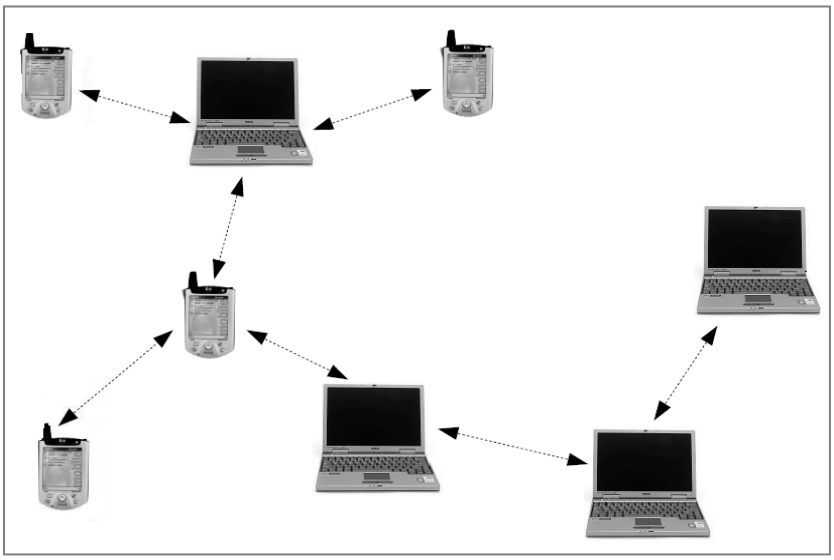

Figure 1.1. Sample Mobile Ad hoc Network Topology

1.1. BROKER BASED ARCHITECTURES IN MANETS

Some MANET infrastructure contains single broker which monitors the whole network and does essential operations, such as coordination. The broker is also mobile, but it showcases reduced mobility compared to other peers. Figure 1.2 showcases broker based architecture formed by UAVs (Unmanned Aerial Vehicles). For such small network topology; having a single broker is sufficient. Broker in such networks will be apparently controlled by ground stations. Still, efficient mobility handling mechanism has to be devised to handle disconnection of brokers.

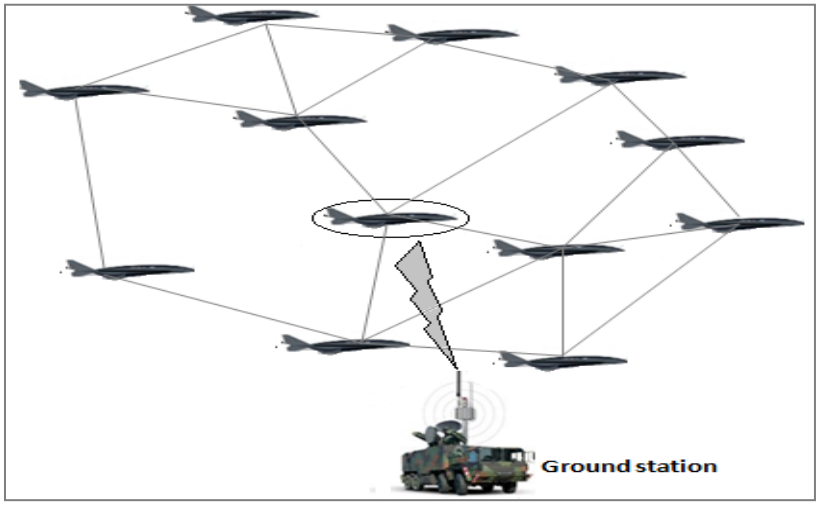

Figure 1.2. MANET Topology Formed by UAVs with a Single Broker

Some mobile ad hoc network applications uphold a hierarchical infrastructure with main broker at the first level and the sub brokers at the subsequent levels. Such hierarchical models are used in large MANET topologies to coordinate among peers. Figure 1.3 demonstrates a hierarchical broker based architecture for large MANET topology formed by armed forces. The main commander (main broker) located at the center coordinates the sub commander (sub brokers) which in turn coordinates the soldiers (peers) having handheld devices. Some topology of this kind might have sub broker elected by main broker based on factors such as mobility, connectivity etc.

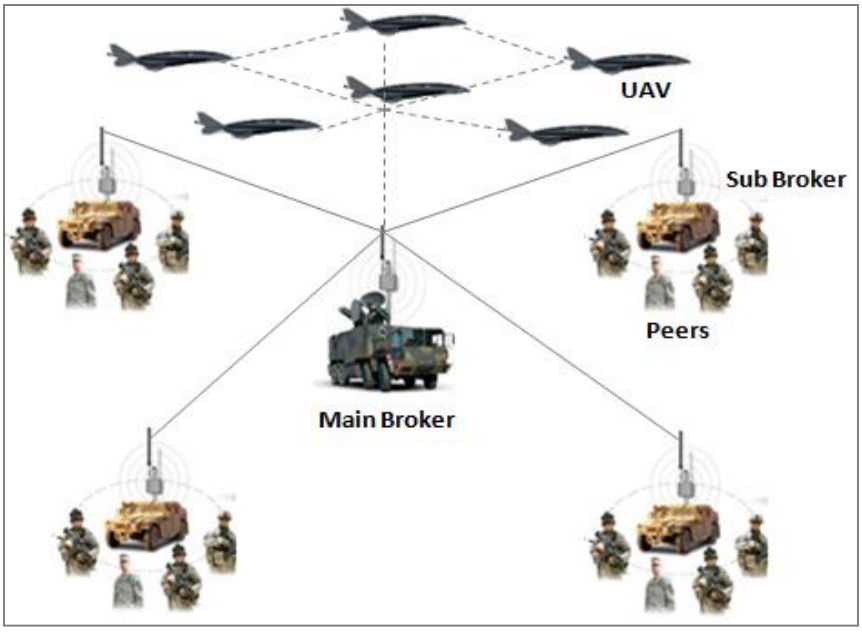

Figure 1.3. Hierarchical Broker based MANET Topology

1.2. DATA ACCESS EFFICIENCY IN MANETS

Caching and Replication are the two common techniques to improve efficiency of data access in Mobile Ad hoc networks. Caching is the process of pre-fetching the needed data and storing it closer to the source. Replication is the process of propagating changes through multiple copies. Replication refers to push model, whereas Caching refers to pull model. Caching however bears its own advantages such as memory usage and reduced network bandwidth as compared to replication. Moreover, Caching reduce resource usage through a reduction in round trips. The advantages of caching over replication showcase caching, a better technique for improving data access efficiency in MANETS. A caching model might focus on reducing the round trip time by saving memory or energy or both. The motivation of a caching model depends on the application. Some MANET application has very less memory; caching model for such application should efficiently utilize the available memory. Likewise, some MANET application has safety and security as the primary concern (Figure 1.2 and Figure 1.3). In general, every caching model aims to reduce the round trip time in retrieving the needed data. The performance

of any caching model with respect to round trip time varies based on their motivation. A good caching model should favor its motivation without affecting data access efficiency.

1.3.SAFETY IN MISSION CRITICAL APPLICATIONS

Safety and security are the primary concern in almost every mission critical applications. Mobile ad hoc networks formed in mission critical applications (Figures 1.1 and 1.2) require caching for faster access of data. At the same time, security in such applications needs to be preserved. One such way of providing good security is reducing the number of cache locations. In Figure 1.2, peers (soldiers) acting as cache locations containing critical data such as number of troops, location information, attack plan etc. Capturing a soldier can reveal this **secret** information. Hence, diminutive numbers of cache locations are required for such environments. Moreover, nodes located at centre have to be chosen as cache location, since it reduces the chance of being attacked. Though data are encrypted in such environment, capture of devices always imposes risk.

1.4.MOTIVATION

Though extensive research has been done on caching [3, 4, 5, 9, 10 and 12], most of the research does not focus on reducing the number of cache locations, which is needed for mission critical applications, and applications with limited memory and security constraints. Most of the researches focus on distributive caching model concerning disadvantages due to mobility in MANETS. There are certain MANET topologies bearing broker based architecture. In such environments, the availability of broker can be efficiently utilized to perform centralized approaches favoring data access efficiency. Moreover, nodes in such topologies have fixed trajectories with reduced disconnections. Existing caching models are deficient in identifying caches with respect to locations. It is obvious cache locations at the boundary of the network have higher chance of getting disconnected from the network, thus reducing availability. Considering all these disadvantages and facts, we designed and developed a broker based caching scheme for small MANET topology (Figure 1.2) named “MEMORY AND LOCATION OPTIMIZED CACHING (MELOC)” [paper I]. We extended our MELOC approach to large MANET topology (i.e. hierarchical broker based architectures) [paper II]. We

modified our basic approach incorporating certain features, to overthrow the disadvantages of MELOC in large networks

1.5.GENERAL METHODOLOGY

1.5.1. Reduce the Number of Cache Locations. Some mission critical applications require diminutive number of cache locations so that capture of nodes does not reveal secret information. Moreover, such applications have limited memory constraints, so that memory has to be efficiently utilized. Reducing the number of cache locations proportionally reduce the number of copies, thus saving memory and ensuring safety. One important challenge on reducing the number of cache locations is maintaining the data access efficiency.

1.5.2. Preserve Data Access Efficiency. Most of the existing caching approaches [3, 4, 5 and 10] improve data access efficiency by utilizing almost every node as cache location, all available memory are utilized over a period of time. Though Bin [10] improves data access efficiency with limited memory, it also utilizes every node as a cache location over a period of time. It is obvious as the number of cache location decreases, the data access efficiency decreases. Hence, nodes which favor high data access efficiency should be chosen as cache locations. The following metrics can be used to choose cache locations favoring data access efficiency

1) Shortest paths: Nodes occurring more number of times as intermediates in the shortest path are center to the network, which favors data access efficiency.

2) Connectivity: Highly connected nodes are reachable by more nodes in the network improving data access efficiency.

Moreover, only essential data has to be cached, since fewer cached locations bring lesser memory available. Caching data which are already available closer to the source does not gain a big advantage; existing approaches are caching such data if memory is available. Since, we have the objective of reducing the number of cache locations; we should utilize our memory efficiently. The following metrics can be used in choosing the essential data.

3) Caching data from very distant nodes: Through this metric, the chosen cache locations contain data whose original source is at a very high distance from the cache location. We

follow an idea of identifying groups based on cycles and caching data of every other group members into one or more members of the current group. Through, this only distant data will be cached by the cache locations.

1.5.3. Snapshot of Whole Network. Our broker based caching model needs to construct cycles for caching data from distant nodes. One good way of constructing cycles is to convert the network topology into graph. The algorithm we follow to construct cycles from graphs is extremely efficient and ground-breaking [paper 1]. This ideology requires the broker having the snapshot of the whole network. The underlying routing algorithm is an effective two layered graph based routing algorithm using store/forward concept [16]. As discussed through [16], the broker maintains the snapshot of the whole network through the construction of a connectivity graph.

1.5.4. Reduce Broadcasts. Every node should know the location of the needed data, so that request can be sent to appropriate source. A common way of exchanging such information is done through metadata. Existing approaches [4 and 10] performs a periodic broadcasts to inform nodes about data prevailing in the network. Such broadcast does not cause drastic effects for small networks, but for large networks it increases the update cost and message propagation delay. Since we are focusing on a hierarchical broker based architecture for large networks, the sub brokers can be efficiently used to avoid this hindrance. Every node updates its metadata to the sub broker assigned to them and the sub brokers exchange this Meta data information with other sub brokers. On need for a data item, the node forwards the request to the sub broker. Through this, broadcast across the whole network is reduced to a great extent. There are several approaches for electing sub brokers [18, 19 and 20]; we follow WCA [20] with little modifications on electing sub brokers.

1.5.5. Handling Disconnections due to Mobility. Since we are focusing on broker based caching model, disconnection of broker leads to serious hazards. Brokers can be selected or environment specific. Consider (Figure 1.2) the broker is always the UAV controlled by the ground station. In such cases, the system has to use excellent ideologies to handle disconnections. We use the virtual ID [6] to handle disconnection of brokers. In some cases, broker has to be selected based on the network topology. Consider (Figure 1.3) the commanding vehicle can be many; hence it is always good to

choose nodes which are very close to the center as main broker. This reduces the chance of main broker being attacked or captured easily. We have main broker election algorithms to handle such scenarios for large networks.

1.5.6. Cache Reallocation. The initial cache locations determined might become futile if the network topology changes. Hence, we have algorithms which decide up on cache reallocation, when there is a drastic topology change. In case of small MANET topology, the broker can periodically run the cache reallocation algorithm. Moreover, frequent cache reallocation will not cause drastic effects for small networks. In case of large MANET topology, frequent cache reallocation needs to be avoided due to wider placement of nodes and bandwidth constraints. To ensure this, partial reallocation can be performed based on cache location moving across zones in large networks. In our approach, zones are specific to the simulation area. Complete reallocation has to be done in large networks, only when there are severe topological changes.

2. RELATED WORK

There are number of approaches, which use caching to improve data access efficiency in Mobile Ad hoc networks. Almost every approach utilizes all available memory in the network, thus increasing the number of cache locations. Though some approaches propose caching model for networks under tight memory constraints, we are the first to propose a caching model, which reduces the number of cache locations. Some of the famous caching, replication and clustering approaches reviewed are

2.1. Cache Data and Cache Path

2.2. Benefit based data caching

2.3. Zone based cooperative caching scheme

2.4. Replica Allocation Methods

2.5. Weighted Clustering Algorithm

2.1. CACHE DATA AND CACHE PATH

Cao et al [4] proposed three caching algorithms, first is Cache Data, second is Cache Path and third a hybrid approach combining the above two. In Cache data algorithm, the nodes caches the data which passes through it based on its popularity. In Cache Path algorithm, a node caches the data path, when it is closer to the caching node compared to the original data center. The difference between these two paths is the path saved. Finally, a hybrid approach stipulate to use cache data, when the size of data is small and cache path, when path saved is large.

The disadvantage of Cache Data algorithm is that the forwarding clients consume a lot of caching space. Path could become obsolete in case of Cache path algorithm, which in turn causes extra processing overhead. Since mobility is an indigenous characteristic of mobile ad hoc networks, it might cause the forwarding client to move away; hence even data in Cache Data scheme might become obsolete. A common disadvantage of this approach is that if different nodes access different data items continuously over a certain period of time, the number of caches significantly increases.

2.2. BENEFIT BASED DATA CACHING

Bin [10] proposed a cache placement algorithm named Distributed Greedy Algorithm (DGA) for mobile network with tight memory constraints. Each node will maintain the nearest cache for all the data. In case of available memory, each node caches the passing data based on benefit score.

Benefit Score $B_{ij} = t_{ij} * \delta_j$

The benefit score for a data item D_j in node i is the product of access frequency of data item D_j in node i (t_{ij}) and least distance to the neighboring node containing the data item D_j (δ_j). When a node caches a data item it broadcasts information about the availability of the corresponding data item to the broker. Similarly, when a data item is deleted it broadcasts the non availability of the corresponding data item to the broker. The broker periodically broadcasts the Meta data of the cache updates to the whole network. In case of memory constraint data item with lowest benefit score is replaced. Though this approach focuses on improving the efficacy of data access with available memory, it does not intend to reduce the number of cache locations. The primary objective of Bin [10] approach aims at caching data with available memory from all nodes in the network; all the available memory are utilized over a period of time increasing the number of copies. But in our approach, we have the principal motivation of reducing the number of cache locations to attain the same data access efficiency, instead when all nodes are used as caches.

2.3. ZONE BASED COOPERATIVE CACHING SCHEME (ZC)

Narottam [12] proposes a zone based caching scheme for efficient data retrieval in MANETS. In ZC one-hop neighbors form a cooperation zone since the cost of communication with them is low both in terms of energy consumption and message. The mobile nodes share data with its neighbors lying in the zone.

When a client needs a data item it first looks at its local cache, if it is not present it floods the request to its zone members. If any of the zone members have the data item, it responds with an ACK. If the zone members do not contain the data item, the request is forwarded to the server. If any of the nodes in the routing path contains the data item, it satisfies the request.

Our MELOC approach for large networks varies slightly with ZC respect to data retrieval. In our approach the request is not broadcasted, it is forwarded only to the shortest sub broker as the sub broker contains the Meta data information. Moreover, in our approach the zones are based on the simulation area. We are avoiding multiple messages compared to Narottam [12], since we do not have ACK messages; in our approach the sub broker on getting the request forwards the request to the shortest source. Moreover, in case of ZC, every node will be sharing its original data with its neighbors utilizing all available memory, whereas in our approach we have reduced cache locations saving memory.

2.4. REPLICA ALLOCATION METHODS

Hara et al [2] provides three replication allocation methods SAF (Static access Frequency), DAFN (Dynamic access frequency and neighborhood) and DCG (Dynamic Connectivity based Grouping), assuming no data updates. In case of SAF, the access frequency to each data item from each host is taken into account for replica allocation. In case of DAFN, the access frequency to data item from each host and neighboring hosts is considered for replica allocation. Finally, in DCG the access frequency to each data item and the whole network is considered for replica allocation. In case of DCG, the network should be stable and should not suffer from single point of failure. Briefly, the core idea of these schemes replicates data periodically based on access frequency and network topology.

2.5. WEIGHTED CLUSTERING ALGORITHM (WCA)

Sajal [20] proposes a non periodic approach for choosing cluster heads. The metrics used for identifying cluster heads are degree, transmission power, mobility and battery power. The cluster head election procedure is delayed by identifying stable cluster heads, hence reducing computation cost. Each cluster head can support only δ nodes. The mobility factor is calculated with respect to a nodes current position and previous position. The final score is a weighted sum of all the metrics. The node with high score becomes a cluster head.

PAPER

I. MELOC: Memory and Location Optimized Caching Model for Small Mobile Ad hoc Networks

[1]Lekshmi Manian Chidambaram, [1]Sanjay Kumar Madria, [2]Mark Linderman and [3]Takahiro Hara

[1]Department of Computer Science, Missouri S & T, Rolla, MO

[2]Air Force Research Lab, Rome, NY and [3]Osaka University, Japan

ABSTRACT

Caching is a common technique to improve efficiency of data access in MANETs (Mobile Ad hoc Networks), where users communicate using small devices connected by resource constraint wireless networks. In some MANET applications, reduced numbers of cache locations are desirable due to security issues and higher consistency maintenance cost. However, reducing the number of caches by finding optimized cache locations (at highly connected and centrally positioned nodes) should not affect the performance efficacy of data access. Additionally, hops to data must also be minimized for better response time. Existing cooperative caching approaches are deficient in finding optimized cache locations, thus they do not focus on reducing the number of copies shared among nodes. In this paper, we design and evaluate a caching scheme within a broker-based architecture to improve data access in MANETs. Our scheme reduces the number of caches by efficiently bringing data closer to the source (minimizing hops). In addition, we identify centrally located and highly connected nodes as cache locations. The performance comparison of our scheme based on simulations with one such recent caching scheme; shows reduction in cache locations by 72%. We also improve the efficacy of data access by 30%. We evaluated data access efficiency using average hops and average roundtrip delay.

Index Terms: Caching, Cache Location, Cache Data, Ad hoc Networks, Broker-architecture

1. INTRODUCTION

Improving data accessibility using caching in mobile ad-hoc networks (MANETs) is an extensive area of research [3, 4, 5, 9, 10 and 12]. The topology changes play a vital role in determining the efficacy of any caching model designed for MANET. Caching can bring data closer to the source in multi-hop wireless networks, thus help in conserving overall energy, as wireless transmission consumes lot of battery power. Though, caching bears its own advantage, it has to overcome challenges such as security issues, mobility and load distribution. Furthermore, shared memory in MANET has to be efficiently utilized.

In many of the existing caching approaches [4, 5, 10 and 12], which duplicate data across multiple nodes in MANETs, the cost for maintaining cache increases in case of disconnections or frequent mobility. Some mission critical applications require diminutive number of duplications, to save energy and maintain data secrecy. Example of one such application favoring our motivation is given below:

Example: Consider Mobile Ad-hoc Networks formed by flying Unmanned Aerial Vehicles (UAVs) for air surveillance. There will be one leading regulator UAV controlled by a ground station, which in turn communicates with all other UAVs. Such application restricts data to be cached in multiple locations since capture of a UAV results in query/data plan leakage. In addition, their data being images/videos, available memory has to be efficiently utilized.

A caching approach overcoming above challenges, without affecting the performance efficacy of data access is needed. Our approach endeavors to improve data accessibility by choosing fewer centrally located cache locations (referred to as optimized only in a lose sense), thus, reducing the number of caches deployed. Our architecture has broker which run cache allocation algorithms. The core idea of our broker-based cache allocation is to segregate the network into groups and cache data of other group members at one or more members of the current group. Using this technique, we are bringing data closer to the sources, thus every node will be able to access all data in the network at shorter hops. In our approach, groups are formed through *"Identify Cycle"* algorithm, which does the job of identifying cycles by constructing a tree from the given network

topology. Caching data of other group members to one or more member in the current group is done through *"Simple Comparison"* algorithm, which determines initial cache locations and the corresponding data to be cached by them. We found that the results of our *"Simple Comparison"* algorithm produce an average of N/2 cache locations for a network of N nodes.

Cache locations favors data access efficiency if they occur at the center and are reachable by more number of nodes in the network. Hence, the results from *"Simple_ Comparison_Algorithm"* are further improved to determine the final cache locations and their corresponding cache data. The metrics used for improvement are shortest path from each peer to its most popular data items, degree of peers and available memory. The idea of using shortest path is to give preference to nodes that occur at the center of the network. Optimization is done by giving preference to nodes occurring more number of times as intermediate nodes in the shortest path. Incorporating "degree of peers" gives preference to nodes with increased number of connectivity; making them reachable from more number of nodes. It is obvious that available memory has to be taken into consideration while allocating data to appropriate cache locations. Once the broker determines the final cache locations, it instructs those cache locations to cache their corresponding data. Our approach might look like replication, but it is a special form of caching named **two-tier caching** (pre-fetching the data based on explicit instructions) as discussed in [9]. In our approach pre-fetching data to cache locations is based on explicit instruction from the broker.

Mobility at a greater extent increases the reallocation cost and endure adverse performance effects [3, 4, and 8]. With respect to our approach, initial number of cache locations determined by the broker might decrease the efficacy of data access. Such scrutiny renders the need for us to reallocate cache locations if there is a drastic topology change. The broker periodically runs a *"Decisive Algorithm"* to reallocate cache locations considering cache locations connectivity and change in their neighboring nodes. Mobility even causes disconnection of broker and this ideology motivated us to introduce the concept of virtual ID [6] for managing broker disconnections.

Since we maintain reduced number of cache locations, every cache service multiple requests, thus increasing its load. Henceforth, we propose an access model based

on a load balancing score function through which requesting node chooses the data source having reduced load.

Our approach is closely related to the Distributed Greedy Algorithm (DGA) discussed by Bin [10]. The DGA algorithm aims at efficient cache placement with limited memory capacity. The result of such a cache placement should significantly reduce the total access cost. The simulation experiments show that their approach when compared with Cao [4] performs better. The primary objective of Bin's approach aims at caching data with available memory from all nodes in the network, i.e., all the available memory are utilized over a period of time thus increasing the number of copies. But in our approach, we have the principal motivation of reducing the number of cache locations with improved data access efficiency. Our experimental evaluations show that our approach does reduce the cache locations by 72%, and also improves the efficacy of data access by 30% over [10].

2. RELATED WORKS

Cao et al [4] proposed three caching algorithms, Cache Data, Cache Path and a hybrid approach combining the above two. The disadvantage of Cache Data algorithm is that the forwarding clients consume a lot of caching space. Path could become obsolete in case of Cache path algorithm, which in turn causes extra processing overhead. Bin [10] proposed a cache placement algorithm named Distributed Greedy Algorithm (DGA) for mobile network with memory constraints. Each node maintains the nearest cache for all the data. In case of available memory, each node caches the passing data based on benefit score. The benefit score is the product of access frequency of data item and least distance to the neighboring node containing the data item. When a node caches (deleted) a data item it broadcasts information about the availability (non-availability) of the corresponding data item through add (delete) message. In case of memory constraint data item with lowest benefit score is replaced. A common disadvantage of approaches [4 and 10] is that they do not intend to reduce the number of cache locations where our approach does.

Cao et al [5] proposed a broadcast based searching and aggregate caching mechanism to improve information accessibility. In this approach, the decision of caching is based on neighboring nodes. As the topology frequently changes in MANETS; caching decision based on neighboring nodes is not effective.

Wang et al [3] focus on dynamic caching integrated with dynamic back up routing protocol. Dynamic backup routing protocol is an on-demand routing protocol where the intermediate nodes, which receives packets from source nodes gathers information to establish back up nodes. In this case the dynamic caching refers to caching, data and path as similar to Cao [4]. Hence the disadvantages discussed in Cao [4] are applicable here.

Hara et al [2] proposed schemes for replicating data periodically based on access frequency and network topology. However, this method does not take into account hop counts.

3. NETWORK AND SYSTEM MODEL

3.1. NETWORK MODEL

The network model is represented as a graph G (V, E), where the vertices "V" are the mobile hosts and the edges "E" are the links. An edge can exists between two mobile hosts if distance between them is less than the wireless transmission range. The underlying routing algorithm is Link State Routing [16 and 21] where every host will have the current snapshot of G as discussed through [16 and 21]. When there is a link breakage, the status will be broadcasted by the corresponding host such that G will be updated at all the hosts. Bandwidth is shared among the hosts within in the transmission range.

3.2. SYSTEM MODEL

The system environment is assumed to be a mobile ad hoc network environment with no fixed nodes. Data will be originated and shared between mobile hosts. The original data center for a data is the mobile host where it originated. For example, original data center for an image at air surveillance is the UAV that took the image initially. Each mobile host will be having a fixed memory available for sharing. Mobile hosts are identified as M_i, for $1<=i<=N$, where N is the density of the network. Data items are represented as D_{ij}; i refer to the mobile hosts, where the data item originated, and j represents the id of the data item. For example, the first data item originated at mobile host M_1 is represented as D_{11}. Each mobile host broadcasts its Meta data information, such that every other mobile host know the data available in the network. Each mobile host broadcast metadata only for the newly created data.

4. SYSTEM PROCESS

The choice of a broker is dependent on the application environment. For example, in case of air surveillance, the UAV controlled by ground station can be considered as a broker. Generalizing the scenario, the first occurring node is considered as a broker and we have only one broker node here. The process flow is shown in Figure 4.1.

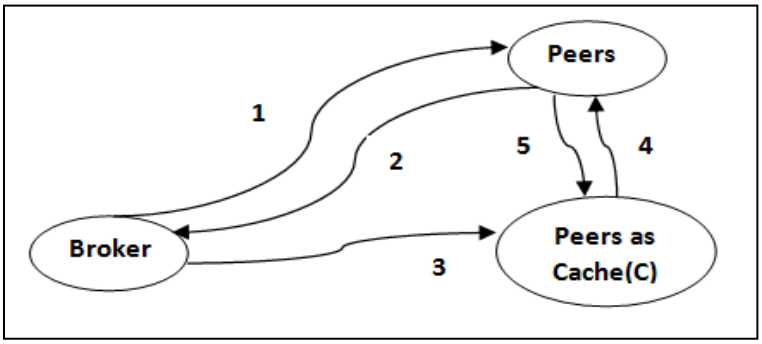

Figure 4.1. System Process

Step 1: The broker waits certain amount of time for significant occurrence of the nodes in a network. It then performs the **broker based operation** (section 5) to determine cache locations.

Step 2: The broker having the snapshot of the whole network applies "Identify Cycle" and "Simple Comparison" algorithms to determine the primary cache location **(C)** and their corresponding data to be cached (**CD)**.

Step 3: The broker improves the results of step 2 through "Cache Optimization" algorithm, which favors data access efficiency using factors such as shortest path and connectivity. The broker request these optimization factors from peers indicated as (1) in the above Figure, whereas (2) indicates the response

Step 4: It then instructs the corresponding peers chosen as cache location to cache their corresponding data indicated as (3) in Figure 4.1.

Step 5: Peers acting as cache location, contacts the original data center containing data, and caches data as instructed by the broker (*pre-fetching)*, shown through 4 & 5 in Figure 4.1.

Step 6: The broker periodically runs a "*Decisive"* algorithm to decide whether to re-determine cache locations. Cache reallocation happens only when topology changes are drastic.

5. BROKER BASED OPERATION

Our broker based cache determination algorithms identifies more distinct groups, and caches data items of every group into one or more nodes in other groups.

5.1. CACHE DETERMINATION

The work flow of our broker based cache determination process is depicted through Figure 5.1:

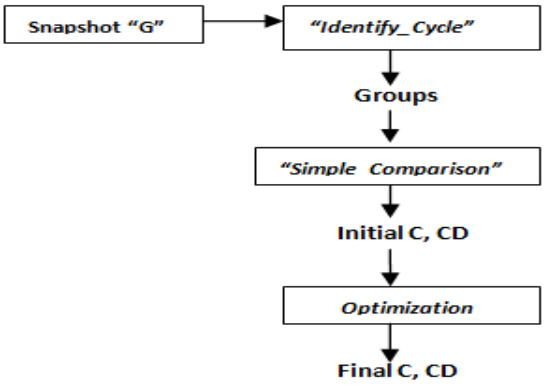

Figure 5.1.Broker Based Operation

5.1.1. Identify Cycle Algorithm. The broker has the snapshot of the whole network topology, i.e. G (V, E). The objective of the "Identify Cycle" algorithm is to construct cycles and form groups. It has two parts

a) Tree Construction Algorithm (TCA)

b) Group Formation Algorithm (GFA).

Tree Construction Algorithm (TCA) The Tree Construction Algorithm (Figure 5.2) converts the network topology into a tree of finite levels. The application of TCA for the sample network topology (Figure 5.3) yields a tree as shown in Figure 5.4. Our idea of TCA is based on Bread First Search (BFS) with some modifications. A node y visited by node x becomes a child of x in TCA, thus constructing a tree at run time. BFS does

not allow redundant nodes, whereas we bring restricted redundancy by applying the following rules on every node when they intend to visit other nodes

1. Never visit a node already visited by your sibling.
2. Never visit a node, if it lies in your ancestral path.
3. Stop visiting, if you have already visited.

With respect to Figure 5.4, node 4 at level 1, have neighbors [0, 7, 11, 3, 9, 2]. Since [7, 11, 9] are already visited by its sibling 2, it is not visited by 4 satisfying rule 1. [0] is not visited by 4 satisfying rule 2. [2, 3] dissatisfy all the rules hence visited by node 4. We bring this restricted redundancy for favoring Group Formation Algorithm.

Tree Construction Algorithm (TCA)

I/P: G (V, E), Broker=0

O/P: Tree with finite levels

Notation:

n	←Node, ∈ V
child[n]	←Children of node n
parent [n]	←Parent of node n
Ω [n]	←one hop neighbors of node n
μ	←Visited list
ℓ	←Level
£ [ℓ]	←Nodes at ℓ
δ [ℓ]	←Children of sibling nodes at £ [ℓ]

Trigger: *(Startup and Reallocation)*

1.root[tree]←broker

2.child[broker]←Ω[broker]

3.ℓ=1

4.Loop

5. δ[ℓ]←empty

Figure 5.2.Tree Construction Algorithm (TCA)

6. For all n: n ∈ £ [ℓ] ::!($n \in \mu$)
7. child[n] = Ω[n] *minus* δ[ℓ] *minus* parent[n]
8. μ.add(n)
9. δ[ℓ].add(n)
10. For all n: n∈ V::(n ∈ μ)
11. break Loop;
12. ℓ ++;
13. End Loop

Figure 5.2.Tree Construction Algorithm (TCA) (Continued)

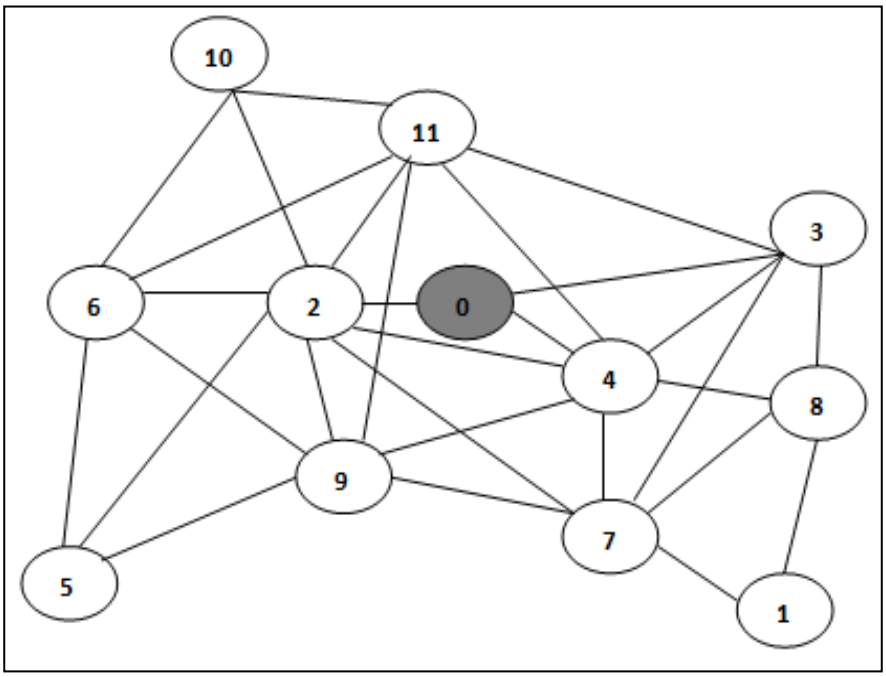

Figure 5.3. Sample MANET Network Topology

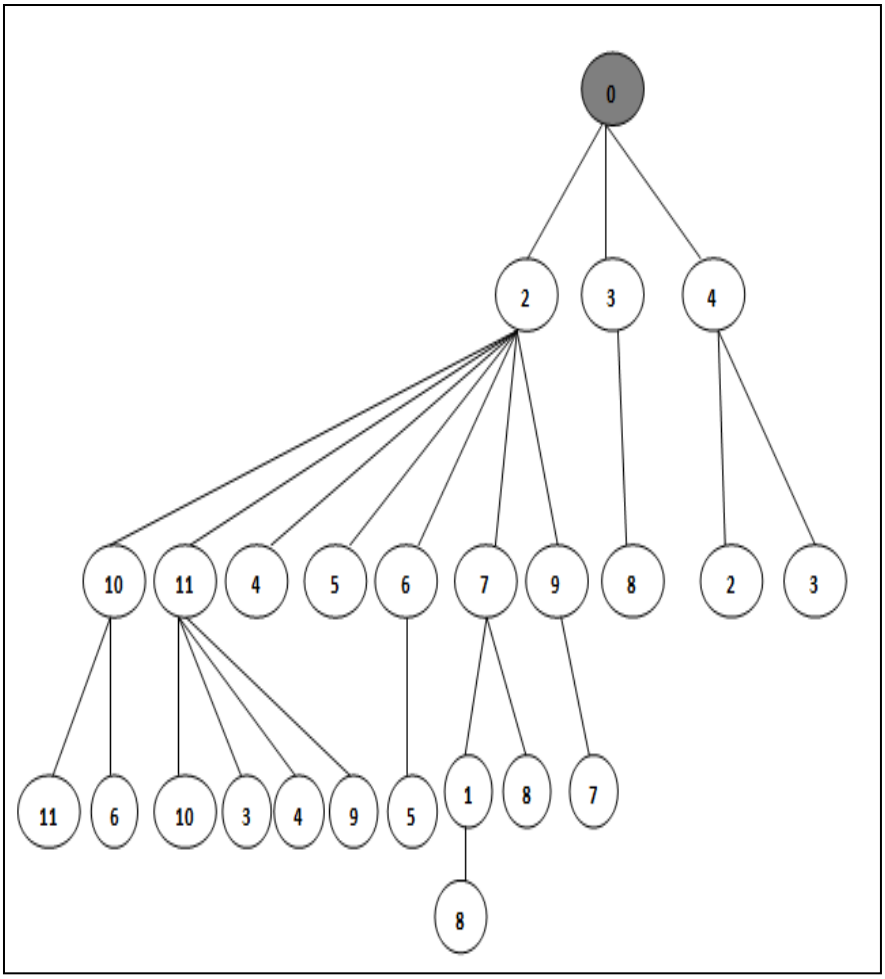

Figure 5.4.Constructed Tree by Applying TCA at Figure 5.3

Group Formation Algorithm (GFA) The Group formation algorithm is used for identifying cycles, which in turn are considered as groups in our approach. Cycles are formed by traversing tree obtained through TCA by matching redundant nodes. We identify groups as cycles, since it gives a greater chance of accumulating nodes having larger distance among each other. In addition, our idea of choosing distinct groups favors very few common nodes among groups and thus, will have a greater chance of caching data items from very distant nodes. The GFA algorithm is shown in Figure 5.5.

Group Formation Algorithm (GFA)

I/P: Tree from TCA and G (V, E)

O/P: Groups

Notation:

H [tree] ←Height of the input tree from TCA

T_N ←Group length Threshold

T_D ←Uncommonness Threshold

α ← List of leaf nodes

β ←List of groups, where β[i] represent the i^{th} group in β

nodes [G] ←Nodes in group G.

μ ←Occurrence list

M[n] ←Node at any level matching leaf node "n"

parent [n] ←Parent of any node n

LCA (m, n) ←Least Common ancestor of nodes m, n

Δ(m, n) ←Path from node m to node n

C ←Cycle

Ω ←List of Groups, where Ω [i] represent i^{th} group of Ω ***(o/p of GFA)***

N[n] ←Neighboring nodes of n

Trigger: *(Completion of TCA)*

1.if (H[tree]>3)

2. For all n: n ∈ α

3. For all s: s ∈ M [n]

4. C← Δ (LCA(s, n), n)+ Δ (parent [s], LCA (s, n))

5. Remove duplicate nodes from C

6. if (C.length> T_N)

7. β.add(C)

8. Sort (β) in increasing order of group size.

9. Ω.add(β[0])

10. μ.add(nodes[β[0]])

11. For all u: u∈ β ::! (u∈ β[0])

Figure 5.5. Group Formation Algorithm: (GFA)

12. if ((u minus μ) >= T_D)
13. Ω.add(u)
14. μ.add(nodes[u])
15. For all v: v ∈ V ::!(v ∈ μ)
16. For all w: w ∈ Ω
17. Assign v to w, which contains more N[v]

Figure 5.5. Group Formation Algorithm (GFA) (Continued)

The execution of GFA in Figure 5.4 with **T_N=4 & T_D=3** is shown in Figure 5.6. Line 1-8 of GFA constructs cycles (groups); **T_N (threshold)** in line 5 of GFA determines the length of the groups. **T_D (threshold)** in line 11 of GFA determines the distinct number of nodes between each group (uncommonness).

Steps 1-8 GFA:

No	Cycles
1	0,2,11,4
2	0,2,11,3
3	0,3,8,7,2
4	0,2,7,8,3
5	0,2,11,3,4
6	0,3,8,1,7,2
7	0,2,7,1,8,3

Steps 9-14 GFA:

No	Cycles
1	**0,2,11,4 ***
2	0,2,11,3
3	**0,3,8,7,2 ***
4	0,2,7,8,3
5	0,2,11,3,4
6	0,3,8,1,7,2
7	0,2,7,1,8,3

*←Selected groups

Steps 15-17 GFA

Groups*(O/P of Identify Cycle Algorithm)*
0, 2, 11, 4, **{5}, {6}, {9},{10}**
0, 3, 8, 7, 2, **{1}**

Figure 5.6. Execution of GFA for Tree at Figure 5.4

Due to this uncommonness threshold, some nodes (n) in the network might never become part of any group. We allocate such nodes (n) to the groups, which have more neighbors of n; line 15-17 of GFA does this operation. The nodes indicated by {} in Figure 5.6 are the newly joined nodes through line 15-17 of GFA. The completion of GFA brings the" Identify Cycle" Algorithm to an end.

5.1.2. Simple Comparison Algorithm (SCA). The key operation of our approach, i.e., caching data of other group members to one or more members in the current group is done through Simple Comparison Algorithm (Figure 5.7)**.** The broker will proceed to SCA only if the Simple Comparison Rule is satisfied.

Simple Comparison Rule: *If the output of "Identify Cycle" algorithm does not return more than one group, re-run the "Identify Cycle" algorithm by decreasing T_D of GFA by 1 till T_D becomes 1".*

Simple Comparison Algorithm (SCA)

I/P: List of Groups from "***Identify Cycle"***

O/P: <C, CD>

Notation:

Ω ←List of Groups from "***Identify Cycle"***.

ϵ[G] ←Elements of group G

μ ←List to store caches while comparing two groups.

₵(C, CD) ←O/P of SCA

Trigger: *(Non violation of Simple Comparison Rule)*

1.For all x: x ∈ Ω

2. μ←empty

3. For all y: y∈ Ω ::!($y = x$)

4. For all c: c ∈ ϵ[x] :: !(c ∈ y) &&!(c ∈ μ)

5. ₵.add(c,(y minus x))

6. μ.add(c)

Figure 5.7. Simple Comparison Algorithm (SCA)

Remember, line 7 in TCA does not follow any ordering for expanding the child. This idea of randomness favors the "*simple comparison rule*". Ordering in TCA might result same tree for every re-run which might cause the cache determination process to halt (i.e. having one group always). On the other end, randomness results in different trees for each re-run, obviating above such deadlock conditions, thus favoring "*simple comparison rule*".

In case of SCA, the groups obtained through "*Identify Cycle"* algorithm are compared with each other by applying certain rules for determining <C, CD> list where

C ←Indicates the cache location.

CD ←Indicates the mobile host id, whose original data has to be cached by C.

We have n (n-1) comparisons for n groups. Comparison between any two groups on determining C and CD is based on following rules. For e.g. Consider group named X is compared with group named Y.

While parsing every element E of X

Rule1: If Y contains E, don't choose E as Cache location "C" (*SCA: line 4).*

Rule2: If E has been already chosen as "C" for X, don't choose E as cache location "C". *(SCA: line 2 and 6)*

Rule3: If Y has elements of X, don't choose those elements of Y as "CD" for "C" *(SCA: line 5).*

Rule4: If elements of Y are already cached by "C" of X doesn't choose those items as "CD". *(SCA: line 5 minus operation).*

At all comparison, first C is identified and then the corresponding CD of C is determined following above rules. The same rule is applicable for comparison of Y with X also. The output of *"Identify Cycle"* algorithm shown in Figure 5.6 has two groups. Hence there are 2(2-1) = 2 comparisons as shown in Figure 5.8.

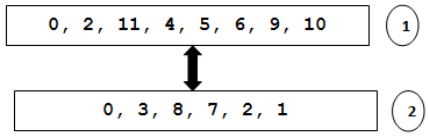

Figure 5.8. Comparison of Groups Obtained as O/P from Identify Cycle Algorithm

The <C, CD> list, obtained after the application of *"Simple Comparison"* algorithm is shown in Figure 5.9.

C	CD
11	[3, 8, 7, 1]
3	[11, 4, 5, 6, 9, 10]

Figure 5.9. Output of SCA.

Let us see, how comparison of Group 1 with Group 2 in Figure 5.8 yielded the first row in Figure 5.9? [0, 2] is present in group 2, hence ignore it based on **Rule 1.** "11" dissatisfy Rule 1 & 2. Hence choose "11" as C. Now we have to choose CD for "11". Ignore [0, 2] based on **Rule-3.** Take [3, 8, 7, 1] as CD for "11". Thus, we fill the first row of <C, CD> table. Likewise, comparison of Group 2 with Group 1 yields the second row in Figure 5.9. The results obtained through "*Simple_ Comparison_Algorithm"* should be optimized further.

5.1.3. Cache Optimization Algorithm (COA). We further strengthen *"Simple Comparison"* algorithm using additional factors to get better data access efficiency. Cache locations favors data access efficiency if it occurs at the center of the network. Hence, we use **shortest path** information from each node to its popular data items for identifying central nodes. The central nodes are those which occur more number of times as intermediate in these shortest paths. The cache locations from *"Simple Comparison"* algorithm are **clustered** with the central nodes of its own group (i.e. the group from

Identify Cycle algorithm). Cache locations improve data access efficiency if it is reachable by more number of nodes in the network. Hence, we order every cache location **cluster** based on its degree (**connectivity**). Then, the broker allocates data <CD> to every member (cache location) of the cluster based on its available **memory**. The cache optimization algorithm is shown in Figure 5.10.

Cache Optimization Algorithm (COA)

I/P: <C, CD> list from SCA, T_D from GFA, Groups *(O/P from GFA)*, Shortest path & memory from peers

O/P: Optimized <C, CD>

Notation:

₵<C, CD>	←<C, CD> list from "*Simple_ Comparison*" algorithm
SP	←List of shortest paths from each node.
I	←Node occurring as intermediate in SP
SL	←List of Nodes occurring as intermediate including duplicates
N [I]	←Number of occurrences of I
T_D	←Uncommonness Threshold (GFA)
T_D	←Uncommonness Threshold (GFA)
Ω	←List of Groups (O/P of GFA), where Ω [i] is the i^{th} group of Ω
α [G]	←Cache locations of the group G.
β	← List of C's in Cache location list ₵.
CP [C]	← Central nodes associated with C of ₵
CD[C]	← Cache data of C in ₵
€<C1, CD1>	← <CP[C], CD[C]> list
O<C, D>	← Final optimized <C, CD> list (O/P COA)

Trigger: *(Completion of SCA)*

1.Order I in SL with decreasing order of N[I], removing duplicates

2.For all I: I∈ SL ::!(N [I]>= T_D-1)

3. SL.remove(I)

Figure 5.10. Cache Optimization Algorithm

4.For all I: I∈ SL ::(I∈ β)

5. SL.remove(I)

6.For all z: z∈ β

7. CP [z].add(z)

8.For all x: x ∈ SL

9. For all y: y∈ Ω:: (x ∈ *y*)

10. For all z: z∈ β:: (z ∈ *y*)

11. CP [z].add(x)

12. For all z: z∈ β

13. Order each CP[z] in decreasing order of connectivity

14. For all z: z∈ β

15. €.add(CP[z], CD[z])

16. Iterate over each C1 of €, and fill it with CD1 based on available memory to generate O<C,D>

Figure 5.10. Cache Optimization Algorithm (Continued)

Step 1: The broker gets the shortest path from each host to it's ***most popular data item**. It identifies the intermediate nodes occurring in the shortest path as (I). For example, shortest path 1→2→3 indicates that M_1 most popular data item is D_3, whose original host is M_3 and the intermediate host is M_2.

Most popular data item: The term "most popular data item" for peers refers to the frequency in which it accesses the data items. The most popular data item of a node is the data getting accessed by it for most number of times. As the broker requests for the shortest path, each node will send the shortest path to the original node containing its most popular data item.

Step 2: The nodes occurring as intermediate (I) in the shortest paths are arranged in decreasing order in List (SL) based on their number of occurrences, provided it is greater than or equal to $\mathbf{T_D}$ **-1,** *(line 2-3 of COA).*

Step 3: Delete the nodes already chosen as cache location I from SL by comparing it with C's of SCA *(line 4-5 of COA).*

Step 4: Parse each I of SL and identify the corresponding groups (O/P of GFA) on which they occur. Identify C in the corresponding group and associate the parsed I of SL as pairs with C *(line 6-11 of COA).*

Step 5: Now we got the pairs associated with each C. Arrange C in decreasing order of their connectivity (degree) *(line 12-15 of COA).*The optimized <C, CD> table will be list of "C" with their corresponding "CD" as shown below :

C	**CD**
<List of C>	Data to cache
<List of C>	Data to cache

Step 6: In case of memory constraints, parse <List of C> by filling the corresponding data CD based on available memory, *(line 16-17 of COA).*

Thus the input <C, CD> table is optimized to get the final <C, CD> table.

5.2. CACHE REALLOCATION

The decisive algorithm shown in Figure 5.11 is run periodically. The cache reallocation happens on the successful execution of line 11 of the decisive algorithm. On reallocation, the cached data items in the previous cache locations are deleted.

Decisive Algorithm

I/P: Optimized <C, CD> list from COA algorithm

O/P: Decision to reallocate (true/false)

Notation:

T_c ← connectivity threshold.

T_{NS} ←neighbor set variation threshold.

T_R ←cache redetermination decision threshold.

D_{ct} ← Degree of cache location at time t.

N_{ct} ← Neighbor set of cache location at time t.

i ← Time at which cache locations are determined

j ←Time at which Decisive algorithm is run.

Figure 5.11. Decisive Algorithm

O ←Optimized <C, CD> list from COA

Trigger: (Periodic Timer)

1.var crc=0;

2. For all **c: c** ∈ C of O

3. If (D_{ci} - D_{cj}) > T_c then

4. crc++;

5. Else

6. If N_{ci} minus N_{cj} > T_{NS} then

7. crc++

8. End If

9. End If

10. If crc > T_R then

11. **return true;**

12. Else

13. r**eturn false;**

14. End IF.

Figure 5.11. Decisive Algorithm (Continued)

5.3. HANDLING BROKER DISCONNECTION

Since we use only one broker, thus disconnections might have adverse effects. The broker handles its own disconnection by using a virtual priority ID. An n-bit virtual ID is chosen. Let N be the maximum number of nodes that can occur in the network. Minimize n, such that 2^n >= N. For example, for a network of 16 nodes, minimizing n, we have 2^4 >= 16 resulting n= 4.

5.3.1. Distributing Virtual ID. The broker has the following metrics:

B-VID ←virtual ID of the broker.

D-VID ← virtual ID dispensed most recently.

TS←Recording the timestamp for each activity.

1.The first occurring node (broker) will have the lowest priority VID i.e. if n=4 the broker assigns 0000 to B-VID. It records the time stamp of its occurrence in TS and assigns B-VID to D-VID.

2.It checks the routing table periodically for new reachable nodes. If any new node is reachable, it compares the current time stamp with recorded time stamp. If the recorded time stamp is less than the current time stamp, it increments D-VID by 1 and dispenses it to the newly connected node. It then updates TS with the current time stamp.

3.If two or more nodes occur at the same timestamp, give the virtual ID in order of the host ID. For example, if D-VID at broker is 0001, and two hosts with ID 2 & 3 occur at the same time then give 0010 to M_2 and 0011 to M_3.

4.Every time a broker issues Virtual ID to a newly occurring node, it gives its identity as broker along with B-VID.

5.3.2. Handling Disconnections. If the broker has disconnected from the network, some other host would take the responsibility as a broker.

1.When a broker disconnects, every host detects the disconnection and adds 1 to its B-VID. The host whose virtual ID matches with B-VID+1 becomes the default broker.

2.The broker updates B-VID with its virtual ID.

3.The new broker broadcasts its identity and B-VID to all other hosts in the network.

4.Every host will update the new B-VID.

5.If the host does not get the new broker identity over a certain period of time, it means that the host with subsequent virtual ID has also been disconnected, hence it calculates B-VID+1+1. This process continues at every host until it gets the new broker identity or identifies itself as a broker.

6. DATA ACCESS MODEL

Our approach has limited number of caching nodes. A caching node might get multiple requests for its data, thus increasing the load. By performing proper load balancing, requests could be equally distributed to available resources which increase the overall efficacy. This section discusses the action taken by every node to access data based on load balancing score function.

DATA ACCESS BASED ON LOAD BALANCING SCORE

Though data are cached closer to the nodes, looking up always at the cache location increases its load. Hence a proper score function is required. Before requesting for data, the requesting node must send a "score_request" message. The "score_response" message from the destination should contain following factors to calculate the score.

1. Load Weight of the host.
2. Size of the data.
3. Bandwidth across the path.
4. Priority.

Load Weight of the host (LW_n) Each and Every host is assigned a random load weight based on their scheduled jobs. We use random load weights ranging from 1 to 3 for our simulation. We further, assign the delay due to single load as 10 milliseconds. **Size of the data** is represented using $Size_{Data}$

Bandwidth across Path A terminology BW_{min}, is used to represent the minimum bandwidth from source to destination.

Priority The priority of load weight (LW) of the intermediate nodes is less, since they just forward the request and response. On the other end, the priority of load weight of the destination node is more, since they are the actual cache locations.

$$P_N = \begin{cases} 4 & \text{N is the destination} \\ 1 & \text{N is Intermediate} \end{cases}$$

The requesting node calculates the load score function with the above metrics. The load score function ($Score_N$) obtained from all the above factors is as follows

$$Score_N = [(Size_{Data} / BW_{min}) + hopcount-1] + \sum_{i=0}^{n} LW_i * P_i$$

When multiple sources are available for a data item, the requesting node sends request to the source with lesser $Score_N$. As an example, referring Figure 6.1, consider node 1 can access a data item of size 300 KB from both nodes 3 and 6.

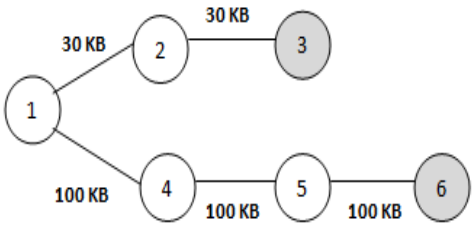

Figure 6.1. Data Access Using Load Balancing Score

Consider we have a constant LW of 3 across the path to node 3 and constant LW of 2 across the path to node 6. Calculating $Score_6$ for Node 6, we have $Score_6$=16. Calculating $Score_3$ for Node 3, we have $Score_3$=26. ($Score_6$ < $Score_3$), hence node 1 will access the data from node 6 even if the hop count is more. Therefore, this access model does not favor hop count.

7. PERFORMANCE ANALYSIS

Simulations are done in a network of 30 nodes in an area of 800 X 800 m^2. Our performance metrics include average roundtrip time, average hop count, number of cache locations and cache hit ratio. The evaluation of parameters with respect to above said performance metrics are as follows (a) Variations in a number of cache locations with respect to number of nodes (c) Effect of average roundtrip time and average hop count with increase in number of nodes (d) Effect of average roundtrip time and average hop count for different cache sizes (e) Behavior of cache hit ratio for different cache sizes

7.1. SIMULATION ENVIRONMENT

We have built a simulation environment in JAVA applet to analyze the behavior of MELOC. (Simulation parameters listed in Table 7.1).

Table 7.1. Simulation Parameters

Parameter	**Default Value**	**Range**
Simulation Area	800 X 800 m^2	
Database size n	60	20-80
Data size	1 MB	
Number of nodes	30	10-30
Bandwidth	1Mbps	
Transmission Range	250 meters	
Velocity	5 meters/second	
Client cache size	7 MB	3 – 15 MB
T_N	5	3-5
T_D	4	2-4
Query generation time	1 second	1 – 10 seconds

7.1.1. Average Roundtrip Time. $ART_i = 1/N \sum_{k=1}^{N}(T_f^i - T_s^i)_k$, where T_s is the time at which the request is send from the source and T_f is the time at which first acknowledgement is received by the source.

7.1.2. Environment Specifications. The nodes are added one by one by the user. The first occurring node has a node ID "0", the next occurring node has node Id "1", and likewise it is incremented for subsequent occurrence of nodes.

7.1.3. Node Movement Model. The number of nodes currently residing in the network move randomly based on a random path. Each node will move across subsequent locations in the random path. There is no range in velocity; nodes have a fixed velocity of 5 meters/ second.

7.1.4. Querying Model. Client requests are modeled using Zip-f like popularity distribution [11]. For our evaluation, we are generating queries based on Zip-f distribution with K =15 and α =0.8.

7.2. SIMULATION RESULTS

All experiments with number of nodes in X axis are conducted by increasing the number of nodes, data range and cache size relatively as shown in Table 7.2.

Table 7.2. Relative Parameter Range

No of nodes	Data range n	Cache size (MB)
10	15-25	3
15	25-35	4
20	35-45	5
25	45-55	6
30	55-65	7

We need to assign initial values of T_D and T_N for different network densities. We found that, the number of cache locations in our approach, increasing relatively with

increase in network density. This indicates that, the initial threshold T_N and T_D should be increased relatively with increase in number of nodes. Hence, we decided to increase T_N and T_D by 1 for every 20 nodes. Thus, we have $T_N = 3$ & $T_D=2$ for density up to 20 nodes, and $T_N=4$ and $T_D=3$ for density up to 30 nodes (Figure 7.1). The following experiments are done using this principle.

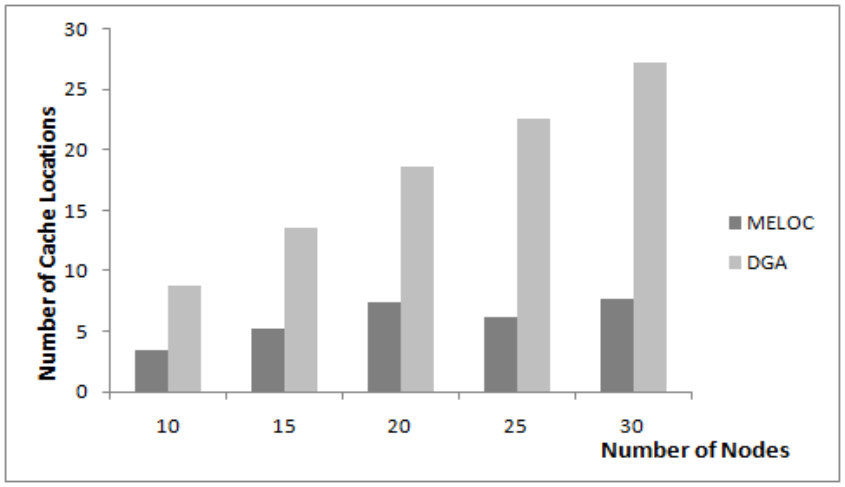

Figure 7.1. Number of Nodes VS Number of Cache Locations (MELOC VS DGA)

We evaluated the number of cache locations assigned by MELOC and DGA for different number of nodes (Figure 7.1) with respect to parameters discussed in Table 7.2. The number of cache locations allocated by MELOC is extremely less compared to DGA for all the cases. However, our approach is good, if we are able to attain same or higher data access efficiency as compared to DGA. Hence, we will be evaluating Average Roundtrip Time and Average Hop Count for both schemes.

Efficacy of data access compared to DGA

(i) We evaluated the average roundtrip time and average hop count with increase in cache size for 30 nodes. This is done to analyze the performance of MELOC with respect to memory constraints as compared to DGA. Figure 7.2 validates MELOC with

respect to average hop count. The average hop count of MELOC is better than DGA for all the cases. This proves that though DGA uses multiple copies of data, it fails to cache data closer to the needed source which is achieved by our approach. In case of DGA, though the number of data copies is large, only a fraction of copies is closer to the sources. This small fraction is mostly ruled out due to the random load weight in our access model causing an increase in hop counts for queries in DGA.

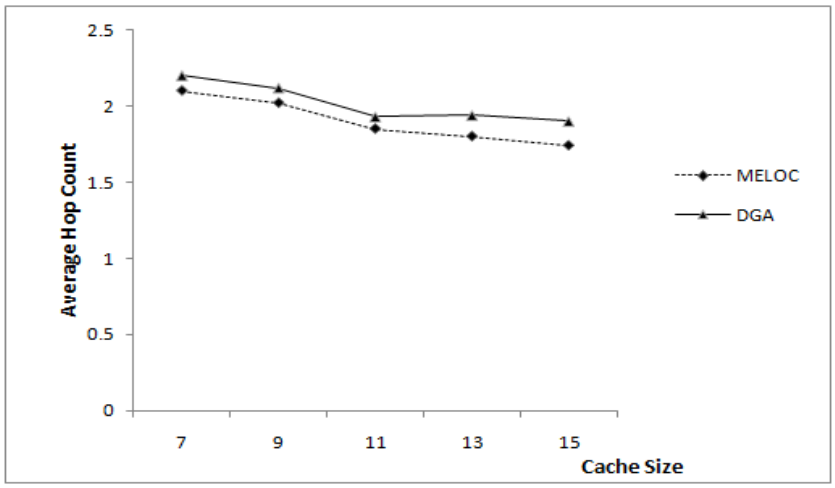

Figure 7.2. Varying Cache Size VS Average Hop Count- 30 Nodes (MELOC VS DGA)

Figure 7.3 shows the validation of MELOC with DGA for average roundtrip time. The average roundtrip time of MELOC is less than DGA for all cache sizes. This is due to the fact MELOC optimizes cache location by using 1) strongly connected nodes and 2) nodes which appear at the center of the network. Moreover, 2) chooses nodes, through which request for most popular data item passes. Caching at these cache locations has resulted in reduced roundtrip time for queries.

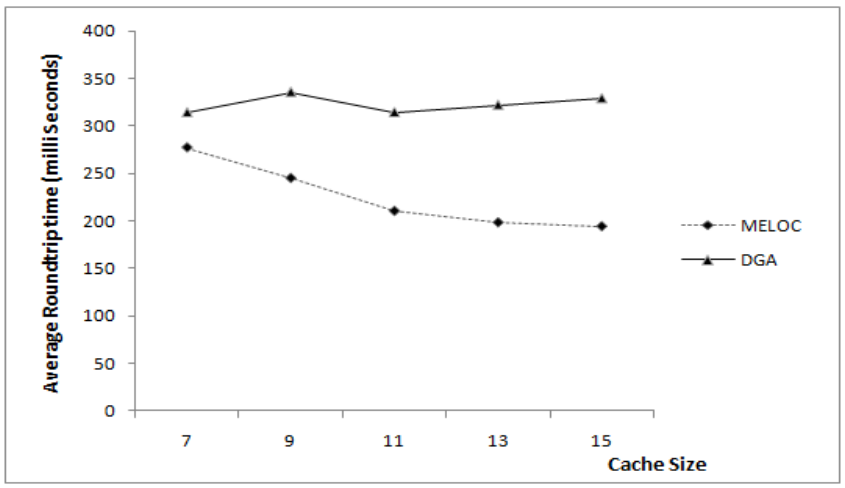

Figure 7.3. Varying Cache Size VS Average Roundtrip Time – 30 Nodes (MELOC VS DGA)

In our approach, updates are broadcasted only during cache reallocation by the broker whereas in case of DGA, for every request, nodes perform cache update operation if the benefit score of data item passing through it is greater than its cached items. Every node will broadcast this update to the broker. The broker in-turn broadcasts these updates periodically. Thus, DGA performs too many broadcasts dumping the message table, causing delay. We also observed significant delay due to updates. Moreover, the average hop count in DGA is higher than MELOC due to reasons discussed through Figure 7.1. Hence, the number of nodes performing update and broadcast operation is also more; thus increasing the roundtrip time for DGA. The number of updates and broadcasts are totally random; hence the roundtrip time for DGA is not constantly increasing or decreasing. Our approach does not have these additional delays, while processing queries; hence we have a constant decrease with increase in cache size.

Through (i), we showcase better data access efficiency compared to DGA with reduced number of cache locations.

(ii) We also evaluated average round trip time and average hop count with increase in number of nodes pertaining to parameters in Table 7.2. This is done to evaluate the performance of both approaches with respect to different network densities. Figure 7.4 shows the comparison of MELOC with DGA for different network densities

with respect to average round trip time. MELOC performs better than DGA for all network densities.

As discussed through Figure 7.3, DGA chooses every node as a cache, increasing the number of updates and broadcasts; thus causing the message table at each node to be overloaded. Therefore, the cost due to update and broadcasts increases proportionally with increase in number of nodes and cache sizes. Hence, in this case, for DGA, there is a constant increase in delay with respect to increase in number of nodes. In case of MELOC, we have limited the number of cache locations and the number of broadcasts is also less. Hence, the average response time is smaller compared to DGA. Moreover, the advantage of MELOC as discussed through Figures 7.2 and 7.3 are also applicable here.

Figure 7.5 shows the validation of MELOC with DGA for different network densities with respect to average hop count. MELOC performs better than DGA due to similar reasons as discussed through Figure 7.2. We also evaluated cache hit as shown in Figure 7.6.

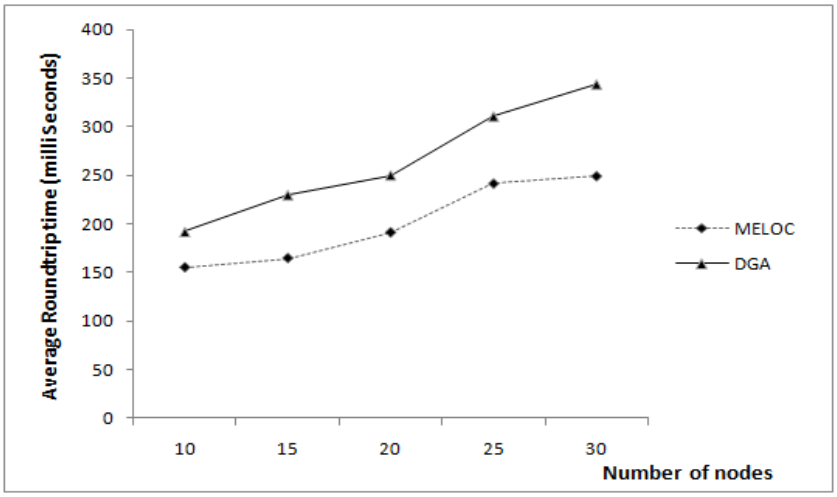

Figure 7.4. Number of Nodes VS Average Roundtrip (MELOC VS DGA)

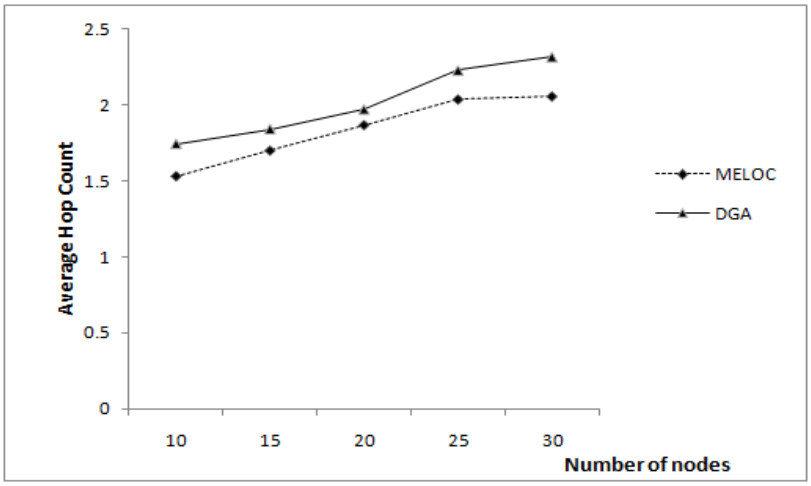

Figure 7.5. Number of Nodes VS Average Hop Count (MELOC VS DGA)

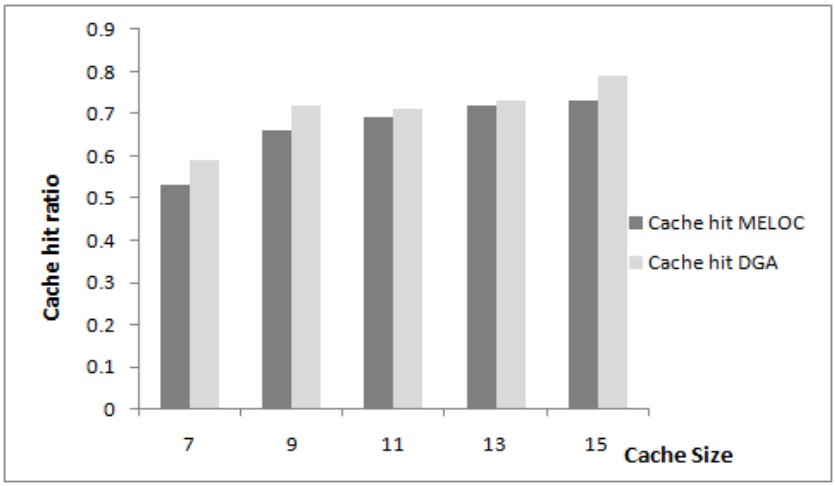

Figure 7.6. Cache Size VS Cache Hit Ratio (MELOC VS DGA)

(iii) The evaluation of cache hit ratio with increase in cache size is done for both MELOC and DGA to analyze the utilization of caches. The cache utilization is a combination of both local hit and remote hit. Figure 7.6 showcases the evaluation of cache hit ratio for MELOC and DGA. It is obvious as the size of the cache increases, availability increases. Moreover, we choose less number of nodes and utilize the

memory of those nodes alone. Hence, the cache utilization of our approach increases with increase in cache size. DGA uses memory of all the nodes in the network for caching; hence the cache utilization varies for different cache sizes. Thus, DGA has higher cache utilization as compared to MELOC.

Through experiments (i), (ii) and (iii), we showed that MELOC has better efficiency as compared to DGA requiring reduced number of cache locations. Considering Figures 7.1 & 7.4, the cache locations are reduced by 72% as compared to DGA and MELOC has achieved a performance improvement of 30.4% over DGA with respect to efficacy of data access.

8. CONCLUSION AND FUTURE WORK

A caching model to reduce the number of caches in mobile networks with improved data access efficiency is presented. Algorithms such as *"Identify Cycle", "Simple Comparison"* for choosing the reduced number of caches from optimized locations are presented. Simulation results showed significant improvement in the reduction in the number of cache locations and improvement in data accesses compared to DGA [10]. In future, we will extend this methodology as MELOC-X for large networks of 100+ nodes and analyze the performance compared to Distributed Greedy Algorithm [10]. For large networks, having only one broker increases the load, hence we will modify MELOC using sub-brokers.

II. MELOC-X: Extended Memory and Location Optimized Caching for Large Mobile Ad hoc Networks

[1]Lekshmi Manian Chidambaram and [1]Sanjay Kumar Madria

[1]Department of Computer Science, Missouri University of Science and Technology, Rolla, MO

ABSTRACT

Caching in Mobile Ad hoc network is an extensive area for research. Caching with reduced number of copies is needed for many military applications to address security concerns. In general, existing cooperative caching approaches are deficient in finding reduced number of cache locations. One such technique to reduce the number of cache locations without affecting the efficacy of data access for a small network topology is given in MELOC [1]. However, having a single broker and metadata broadcast across the whole network as MELOC [1] lead to severe performance hindrance in case of a large network topology. Moreover, frequent cache replacement due to changes in network topology does not favor cache hit and bandwidth conservation in case of large networks. In this paper, we design and evaluate an extended version of "memory and location optimized caching scheme (MELOC-X)", which suits large network topology, overcoming above challenges. Comparison with one such recent scheme Bin [10] showcased a significant improvement in performance. We also evaluate the impact of this scheme with respect to average hops and average roundtrip time through extensive simulations.

Index Terms: Caching, Cache Location, Ad hoc Networks, Broker-architecture

1. INTRODUCTION

The newer applications over Mobile Ad hoc Networks (MANET) are increasing dramatically day by day ranging from social networks to battlefield environment and therefore, faster access to data is needed in many such applications. One of the well known techniques to improve data accessibility in MANET is caching. Caching in MANET environment; is a way to make data readily available through reduction in hop counts. In many such techniques [3, 4, and 5] data is duplicated across many nodes in MANET to increase the availability of data. However, the additional cost of maintenance of caches occurs, which may be expensive as nodes are frequently moving. Moreover, shared memory in MANET is limited and therefore, it has to be efficiently utilized. Some mission critical applications require diminutive number of duplications, to save energy and maintain secrecy. One such example favoring our motivation is listed below:

Example 1: Consider Mobile Ad-hoc Networks formed by flying Unmanned Aerial Vehicles (UAV) for air surveillance. There will be one leading regulator UAV controlled by a ground station, which in turn communicates with all other UAV's. Such application restricts data to be cached in multiple locations, since capture of UAV results in query/data plan leakage. In addition, their data being images/videos mainly, cache memory is always limited. Example1 depicts a small network topology of 12 to 30 nodes, which we have solved through MELOC [1].

We summarize our MELOC [1] approach as follows, our MELOC [1] approach endeavors to improve data accessibility by choosing fewer centrally located cache locations (referred to as optimal only in a lose sense), thus, reducing the number of caches deployed. Our architecture has brokers which run cache allocation algorithms. The core idea of our broker-based cache allocation is to segregate the network into groups and cache data of other group members at one or more members of the current group. Using this technique, we are bringing data closer to the sources, thus every node will be able to access all data in the network at shorter hops. In our approach, groups are formed through *"Identify Cycle"* algorithm, which does the job of identifying cycles by constructing a tree from the given network topology. Caching data of other group members to one or more member in the current group is done through *"Simple*

Comparison" algorithm, which determines initial cache locations and the corresponding data to be cached by them. The results from *"Simple Comparison_Algorithm"* are further optimized to determine the final cache locations(C) and their corresponding cache data (CD). Optimization is done using factors such as 1) shortest path from each peer to its most popular data items, 2) degree of peers and 3) memory, which strengthens efficacy of data access. Thus, our MELOC [1] approach endeavors to improve data accessibility by choosing fewer cache locations.

Unlike example1, there might be large MANET topology requiring diminutive number of duplications shown in Example 2 and Example 3.

Example 2: Figure 1.1 showcases a MANET topology formed in armed forces environment. Caching secret data in multiple devices in such scenario is always risky in any situation. Consider the scenario, 100 corps having mobile devices are fighting in a territory. There might be a main commander, commanding instructions to sub commander. Every sub commander should share information with other commanders (sub, main) about territory locations, peers under them, their advancement etc. In such applications, multiple cache locations, favors the enemy.

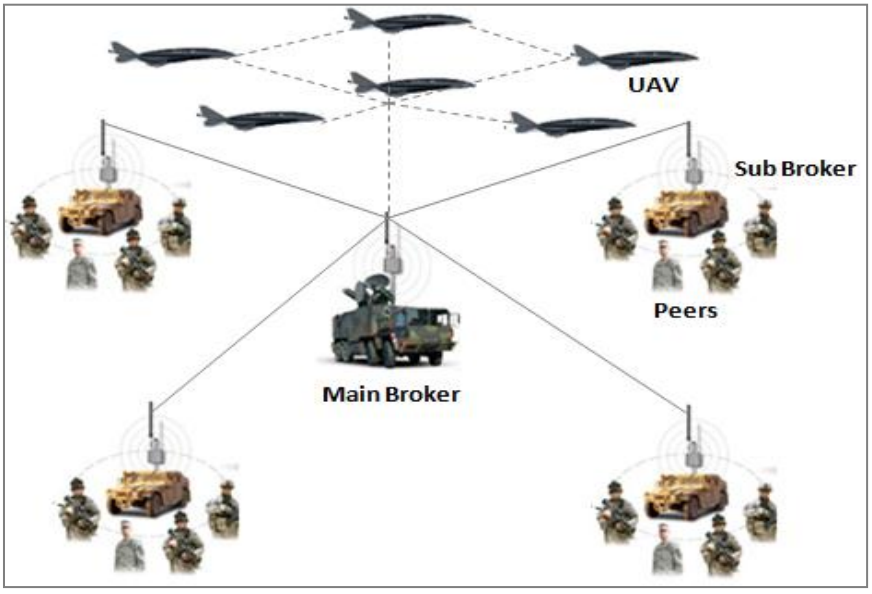

Figure 1.1. MANET Formed by Armed Forces

Example 3: Sharing of music and videos are famous among mobile users. Instead of downloading from the internet, users can communicate with their base station to look for similar files used by other users in their city or zone. Here the base station is the sub brokers and cache locations are the mobile users. Reducing cache locations in such environment save enormous amount of energy and cost.

Application of MELOC [1] to such large networks shown in example 2 and 3, does not favor many cases depicted below.

Case 1: In case of small networks (MELOC [1]), we allow metadata broadcasts across whole network, such that each node acquires information about data prevailing in the network. In case of large networks, such "meta-data broadcasts" across the whole network is always tedious. Hence some mechanism is needed to handle this hindrance.

Case 2: For small networks (MELOC [1]), main broker has greater visibility of every nodes and cache locations in the network. Every node can be reached at a shorter hop; hence frequent cache reallocation is not a problem. However, in large networks, reallocation should be done, only if there is a very drastic change. Since, frequent reallocation in large networks, causes bandwidth constraints, and increases number of messages. On the other end, reducing the frequency of cache reallocation in large networks should not affect the efficacy of data access.

Case 3: Previously we had a single broker to handle 12-30 nodes; applying the same terminology for large networks to handle 100 nodes increases the broker load. This too has to be brought into picture in our MELOCX design.

We extend MELOC [1] overcoming challenges in case 1, case2 and case 3 to suffice large network topology. We overcome the disadvantages discussed through case 1 by introducing sub brokers. We have sub brokers elected as cluster heads with little modification to the weighted clustering algorithm WCA [20]. We divide the simulation area into four zones, where every zone has an axial point at its center. The sub brokers are selected with respect to four axial points across the simulation area. The main broker will be closer to the central point of the simulation area. We have a reasonable assumption, that the mobility of the main broker is very slow compared to sub brokers and peers. The sub broker election methodology in our extended approach is run by the main broker. Every node, will broadcast its metadata information only to the sub broker

of its zone, the sub broker in turn broadcasts its metadata table to other sub brokers. Through this ideology we are avoiding the meta-data broadcast across whole network. Any node requesting data will send the request to its sub broker, which forwards the request to the destination by referring the Meta data table.

In case of MELOC [1], the broker periodically runs the cache redetermination algorithm (*decisive algorithm)* to decide reallocation of caches. On its decision to reallocate, it runs cache determination algorithm to identify new cache location, and the corresponding data to be cached by them. It deletes cached data from old cache locations and request new cache locations to cache data. This mechanism does not suit large networks due to disadvantages discussed in case 2. We overcome the disadvantage by using partial reallocation aided by main broker. The main broker will perform the complete reallocation only if there is a very drastic change.

The main broker does not monitor cache location movement, only the cache locations inform main broker about their mobility. The main broker just aid in electing sub brokers in the initial stage; performs partial reallocation on needed basis and re runs the cache determination algorithm on severe topological change. Thus the disadvantage of case3 is overcome.

In our approach pre-fetching data to cache locations is based on explicit instruction from the main broker. Hence, it is a special form of caching named two-tier caching as discussed in [9]. Our access model is different as compared to MELOC [1]. We have the request sent only to the shortest sub broker. If more than one source is available for the data, the sub broker forwards the request to the shortest source.

Our approach is closely related to the Distributed Greedy Algorithm (DGA) discussed in Bin [10]. The DGA algorithm aims at efficient cache placement with limited memory capacity. The result of such cache placement should significantly reduce the total access cost. The simulation experiments show, their approach when compared with Cao [3] performs better. The primary objective of Bin [10] approach aims at caching data with available memory from all nodes in the network; all the available memory are utilized over a period of time increasing the number of copies. But in our approach, we have the principal motivation of reducing the number of cache locations to attain the same data access efficiency, instead when all nodes are used as caches. Bin [10] approach

does not reduce the number of cache locations, whereas our approach does. The sub broker election in our approach is closely related to WCA [20]. We use two metrics mobility and degree to choose sub brokers. In our approach the mobility factor is calculated from axial points, whereas in WCA [20] mobility factor is calculated with respect to nodes previous positions. Our final score for choosing sub broker is a weighted factor of both mobility and degree. By calculating mobility factor with respect to axial points, we identify sub broker specific to zones.

The rest of our work is organized as follows Section 2 is related research, section 3 explains network & system model, section 4 explains the System Architecture, Section 5 elaborates the extended MELOC approach(MELOC-X), Section 6 explains methodologies for handling disconnections, Section 7 describes our access model, section 8 showcases our validation through experiments and section 9 concludes our approach.

2. RELATED RESEARCH

2.1. CACHE DATA, CACHE PATH

Cao et al [4] proposed three caching algorithms, first is Cache Data, second is Cache Path and third a hybrid approach combining the above two. In Cache data algorithm, the nodes caches the data which passes through it based on its popularity. In Cache Path algorithm, a node caches the data path, when it is closer to the caching node compared to the original data center. The difference between these two paths is the path saved. Finally, a hybrid approach stipulate to use cache data, when the size of data is small and cache path, when path saved is large. The disadvantage of Cache Data algorithm is that the forwarding clients consume a lot of caching space. Path could become obsolete in case of Cache path algorithm, which in turn causes extra processing overhead. Since mobility is an indigenous characteristic of mobile ad hoc networks, it might cause the forwarding client to move away; hence even data in Cache Data scheme might become obsolete. A common disadvantage of this approach is that if different nodes access different data items continuously over a certain period of time, the number of caches significantly increases.

2.2. BENEFIT BASED DATA CACHING

Bin [10] proposed a cache placement algorithm named Distributed Greedy Algorithm (DGA) for mobile network with memory constraints. Each node will maintain the nearest cache for all the data. In case of available memory, each node caches the passing data based on benefit score. The benefit score is the product of access frequency of data item and least distance to the neighboring node containing the data item. When a node caches a data item it sends information about the availability of the corresponding data item to the broker through an add message. Similarly, when a data item is deleted it sends the non availability of the corresponding data item through a delete message. In case of memory constraint data item with lowest benefit score is replaced. Though this approach focuses on improving the efficacy of data access with available memory, it does not intend to reduce the number of cache locations.

2.3. CACHING DECISION BASED ON NEIGHBORING NODES

Cao et al [5] proposed a broadcast based searching and aggregate caching mechanism to improve information accessibility in IMANETS i.e.) MANET's connected to Internet with access points (AP). In "Aggregate caching mechanism" discussed in this approach, the decision of caching is based on neighboring nodes. Since in mobile ad hoc networks, the topology frequently changes, the caching decision based on neighboring nodes is not effective. Moreover, though the broadcast based searching scheme is simple and efficient, the consumption of bandwidth and energy is high in case of mobile ad hoc networks.

2.4. CACHING USING DYNAMIC BACKUP ROUTING PROTOCOL

Wang et al [3] focus on dynamic caching integrated with dynamic back up routing protocol. Dynamic backup routing protocol is an on-demand routing protocol where the intermediate nodes, which receives packets from source nodes gathers information to establish back up nodes. In this case the dynamic caching refers to caching data and path as similar to Cao [3]. Hence the disadvantages discussed in Cao [3] are applicable here. Moreover, in case of compound caching scheme discussed in this paper, multiple nodes are recording the availability of data in a single node acting as a cache. Hence disconnection of this particular single node results in more false positive scenarios while searching for data.

2.5. REPLICA ALLOCATION METHODS (SAF,DAFN,DCG)

Hara et al [2] provides three replication allocation methods SAF (Static access Frequency), DAFN (Dynamic access frequency and neighborhood) and DCG (Dynamic Connectivity based Grouping), assuming no data updates. In case of SAF, the access frequency to each data item from each host is taken into account for replica allocation. In case of DAFN, the access frequency to data item from each host and neighboring hosts is considered for replica allocation. Finally, in DCG the access frequency to each data item and the whole network is considered for replica allocation. In case of DCG, the network should be stable and should not suffer from single point of failure. Briefly, the core idea of these schemes replicates data periodically based on access frequency and network

topology. Though this idea conserves bandwidth, the advantages of caching over replication are detrimental in this scheme.

2.6. TWO-TIER CACHING

Pitoura [9] illustrates the challenges of caching and replication in Mobile Ad-hoc networks such as low bandwidth, network disconnections and scarcity of resources. The two tier caching scheme discussed in this paper is acoustic. The characteristics of two-tier caching are as follows, the content of data to be pre-fetched by cache may be determined automatically by utilizing implicit information or through instructions given explicitly by the users. Furthermore, the propagation of updates performed at the mobile site follow lazy protocols i.e. one single transaction for each update. In our approach once the broker determines the appropriate cache location, it instructs the corresponding nodes to pre-fetch data, which is similar to pre-fetching based on explicit instructions. Hence, our approach bears the name caching in spite of replication.

2.7. REPLICATION APPROACH USING VIRTUAL BACKBONE

Emre et al [13] proposes a scalable data replication approach named SCALAR. SCALAR constructs a virtual backbone based on the approximation of minimum connected dominating set in graph theory. Data is managed in the cache location based on a cost function comprising access frequency and number of hops to the destination. Some of the disadvantages in this approach are, only the backbone nodes can participate in the replication decision. End nodes not becoming a part of backbone can send request only to the backbone nodes. Since replication decision and request processing are constrained only to backbone nodes, they consume more energy and power. The virtual backbone is constructed periodically when the topology changes, hence frequent change in topology results in more broadcasts.

2.8. STABILITY BASED MULTI OBJECTIVE CLUSTERING

Cheng [18] focuses on identifying relatively stable clusters based on relatively stable neighbors. Metrics chosen for clustering are optimized using multi objective evolutionary algorithm. The metrics used for identifying cluster heads are Number of

relatively stable neighbors, power consumption and node lifetime. Once cluster heads are formed, formation of clusters becomes easy. Though this approach aims on identifying stable cluster heads, the election of cluster heads are not location specific, it is based on stable neighbors. Hence, there is a chance of multiple cluster heads occurring in one area and no cluster heads in other area.

2.9. MOBILITY AND ENERGY AWARE CLUSTERING ALGORITHM: (MEACA)

Yi Xu [19] chooses cluster head with minimum mobility and maximum power. As the algorithm states, mobility and power consumption are the metrics used for choosing cluster heads. Every node is associated with only one cluster and every node is one hop away from cluster heads. Every node has to broadcast its mobility and power consumption metrics to its neighbors. The nodes order their metrics along with the neighbor's metrics. If they feel they are having a high score, they will broadcast themselves as cluster heads to their neighbors. Since every node should be one hop away from its cluster head, disconnected node from a cluster cannot be a part of a cluster, till it establishes a direct link with any cluster head. Moreover, the election of clusters is not location specific.

2.10. WEIGHTED CLUSTERING ALGORITHM (WCA)

Sajal [20] proposes a non periodic approach for choosing cluster heads. The metrics used for identifying cluster heads are degree, transmission power, mobility and battery power. The cluster head election procedure is delayed by identifying stable cluster heads, hence reducing computation cost. Each cluster head can support only δ nodes. The mobility factor is calculated with respect to a node current position and previous position. The final score is a weighted sum of all the metrics. The node with high score becomes a cluster head. Though this approach identifies stable cluster heads with good metrics, cluster head stable to a specific zone or location is not found.

3. NETWORK AND SYSTEM MODEL

3.1. NETWORK MODEL

The network topology is represented as a graph G (V, E), where the vertices "V" are the mobile hosts and the edges "E" are the links. An edge can exists between two mobile hosts if distance between them is less than the wireless transmission range. The underlying routing algorithm is Link State Routing [16]; hence every host will have the current snapshot of G. When there is a link breakage, the status will be broadcasted by the corresponding host using alive messages as discussed in [16] so that G will be updated at all the hosts. "Frequency Division multiple access" is considered based on current radio technology, so that the bandwidth is shared with respect to the number of hosts within in the transmission range.

3.2. SYSTEM ENVIRONMENT

The System environment is assumed to be a mobile ad hoc network environment with no fixed nodes. Data will be originated and shared between mobile hosts. The original data center for a data is the mobile host, where it originated. Each mobile host will be having a fixed memory available for sharing. Mobile hosts are identified as M_i, we have 1<=i<=N, and N is the density of the network. Data items are represented as D_{ij}; i refer to the mobile hosts, where the data item originated, and j represents the id of the data item. For example, the first data item originated at mobile host M_1 is represented as D_{11}.Each mobile host periodically broadcast metadata, for the newly created data, only to its sub brokers. The sub brokers maintain Meta data table. Every sub broker shares its Meta data table with other sub brokers periodically. A node requesting data will always send the request to its shortest sub broker, which in turn forwards the request to the shortest source.

4. SYSTEM ARCHITECTURE

We elaborate our system architecture (Figure 4.1) as follows

i. Our system architecture has a central point (X_c,Y_c), which is center to the simulation area. In a real environment, this central point may be based on latitude and longitude.

ii. We have four zones divided across the central point ($>X_c$,$>Y_c$), ($<X_c$, $<Y_c$), ($>X_c$, $<Y_c$) and ($<X_c$,$>Y_c$).

iii. The main broker is chosen as the most stable node closest to the central point (X_c, Y_c). For e.g.) consider Figure 1.1, it is not advisable to have the commanding vehicle at the boundaries.

a) We have four axial points (AP1, AP2, AP3 & AP4) centre to each zone .Let d be the distance from central point to the boundary of the simulation area.

b) The four axial points are (X_c+d/2 ,Y_c+d/2), (X_c+d/2 ,Y_c-d/2), (X_c-d/2 ,Y_c+d/2) and (X_c-d/2 ,Y_c-d/2)

c) In general, we represent these axial points as (X_a, Y_a).

iv. We have sub brokers and cache locations at every zone.

The metrics discussed above are showcased in Figure 4.1

Assumption: Every node knows the central point and axial points of the simulation area.

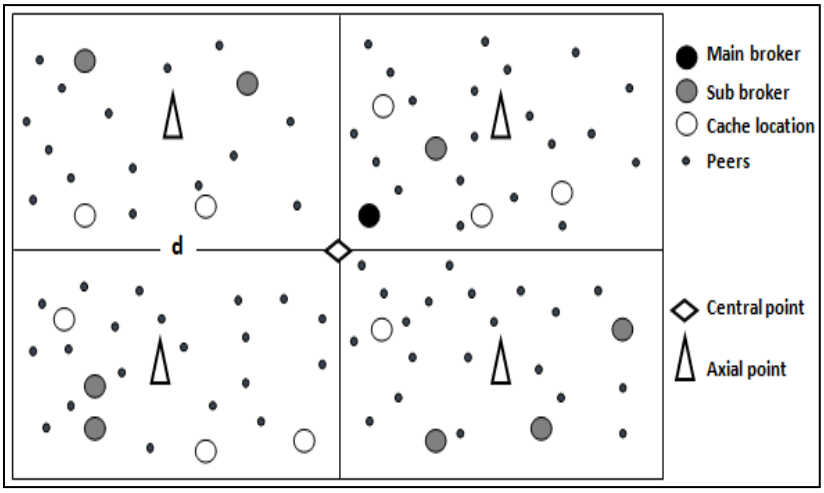

Figure 4.1.System Architecture

5. EXTENDED MELOC APPROACH

In our extended MELOC approach, we try to distribute the load across sub brokers to handle efficient Meta data broadcasts; such that the data access efficiency through MELOC [1] is even preserved in large networks. We adopt partial reallocation, so that data efficiency is preserved without having complete reallocation. In this section, we discuss the design of MELOC-X favoring above ideologies. We have the following sub sections a) main broker election b) sub broker election c) Cache determination and d) Cache reallocation under MELOC-X.

5.1. SYSTEM PROCESS

Our System process is illustrated as follows

i. Every node within a threshold distance limit from central point performs a distributive operation in electing the main broker.
ii. Once the node identifies itself as main broker it broadcasts its identity to every other node in the network.
iii. The main broker then elects sub brokers and broadcasts sub broker identity to every node in the network.
iv. The nodes up on generating data; broadcast the meta-data information to its corresponding sub brokers.
v. The sub broker maintains the metadata table of its group. The sub broker also exchanges its metadata table with other sub brokers.
vi. The main broker runs the cache determination algorithms (MELOC [1]) and instructs the corresponding cache location to cache data.
vii. The caching node informs the metadata information of its cached data to its sub brokers.
viii. The main broker runs the partial reallocations if needed. On worst case (i.e. on drastic topology changes), it performs complete reallocation by moving to step (iii).

The system process is depicted through Figure 5.1, where MELOC [1] cache determination process is shaded; the remaining techniques are added to suffice large network design. We proceed further starting from Main Broker election (Figure 5.2).

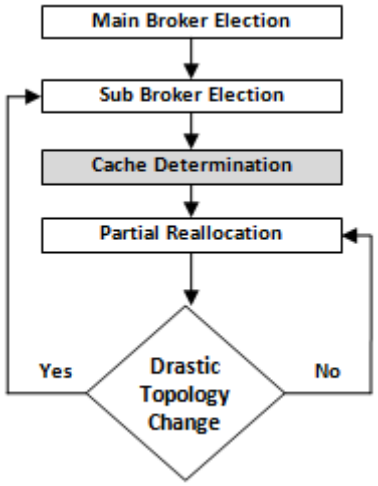

Figure 5.1. Extended MELOC Approach Process Flow

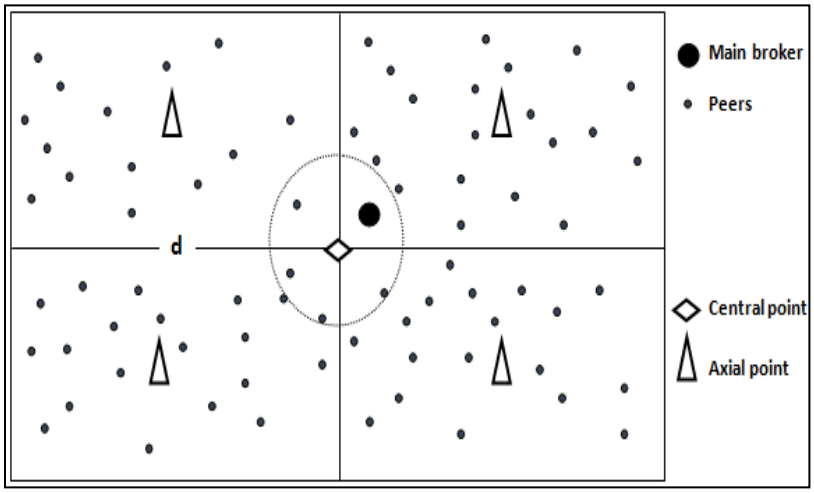

Figure 5.2. Depicting Main Broker Election

5.2. MAIN BROKER ELECTION

We elect node which is close to the central point (X_c, Y_c) as main broker. It is obvious; having the main broker at the center reduces disconnections and chance of getting captured easily. Every node with in a threshold limit (T_{Dist}) from central point, broadcasts its "Distance to the Central Point (DCP)". Every other node within this T_{Dist}

limit, sorts their DCP along with the DCP's of the other nodes. The node with minimum DCP becomes the main broker, which in turn broadcasts its identity. The algorithm for main broker election is shown in Figure 5.3.

Main Broker Election Algorithm

O/P: Main broker identification

Assumption:

Every node in the network knows the central point (CP) and axial points (AP).

Notations:

T_{Dist}	←Distance Threshold
N[V]	←Set of nodes within the T_{Dist} limit from CP.
$Dist_n$	←Distance of node n from CP.
M (n, Distn)	←Message from n containing $Dist_n$.
M (n,MB)	←Identity Message from main broker
Q_n	←Local Queue of node n.
MB	←Main Broker.

Trigger :*(Startup or Disconnection of previous main broker)*

1. For all n ∈ V of G (V, E):: $Dist_n < T_{Dist}$

   Broadcast M (n,$Dist_n$) to all n ∈ V

2. For all n ∈ N[V]::Receive M(k,$Dist_k$)&& !(k ∈ n)

   Place k in Q_n , in increasing order of $Dist_k$.

3. For all n ∈ N[V]

   MB←Head of Q_n.

4. For all n ∈ V of G (V, E):: n=MB.

   Broadcast M (n, MB) to all n ∈ V

Figure 5.3. Main Broker Election Algorithm

The peers enclosed within the dotted circle in Figure 5.2, are the nodes within the distance threshold limit (T_{Dist}). Once the main broker is elected, it initiates the sub broker election mechanism. The sub broker election is illustrated in next section.

5.3. SUB BROKER ELECTION

This section elaborates election of sub brokers (Figure 5.4b), dissemination of sub broker's identity, and limited metadata broadcasts through sub brokers. Once the main broker broadcasts its identity, every other node performs a distributive operation by calculating a weighted score with respect to its closest axial points. The main broker decides the sub broker and corresponding nodes under them, using these weighted score and axial points. Mobility and degree of nodes are the two metrics to calculate the weighted score. Nodes in the network know their sub broker through the main broker. The sub broker election process is shown in Figure 5.4a).

5.3.1. Sub Broker Election Process. *Sub broker election metrics:* Every node calculates mobility factor with respect to its closest axial point (X_a, Y_a). The mobility factor is calculated for sample time T (e.g. 10 intervals) with respect to current location of mobile hosts (X_t,Y_t) at time t.

$M_v=1/T \sum_{t=1}^{T} \sqrt{(X_t - X_a)^2 + (Y_t - Y_a)^2}$, where v ∈ V of G (V,E)

Every node calculates its average degree represented as D_v for sample time T. The final score is the weighted score of both mobility factor and degree.

$S_v= W1 * M_v+W2*D_v$, where v ∈ V of G (V, E)

If W1>W2, then more stable nodes closer to axial points are given preference to be elected as sub brokers. Our motivation is to identify more stable brokers, hence we have W1>W2.

Every node sends the weighted score and its shortest axial point (taken to calculate M_v) to the main broker, represented as (1) in Figure 5.4a. The main broker group nodes with respect to their shortest axial point, and sort members of axial groups based on their S_v in decreasing order. The main broker chooses the first β nodes (user defined) as sub brokers for every group as shown through (3) of Figure 5.4a. Nodes might be present in zone, which do not have any sub brokers; such nodes will be assigned to sub brokers in its clock wise zone.

Sub broker identity Dissemination: The main broker sends the sub broker IDs of a group with a flag to members of the group ((2) of Figure 5.4a). A true flag indicates that the receiving node is a sub broker and a false flag indicates the receiving node is not a sub broker. If it is a sub broker the main broker sends the identity of other sub brokers also. Through this every ordinary node knows the identity of its own sub brokers and every sub broker knows the identity of other sub brokers in the network.

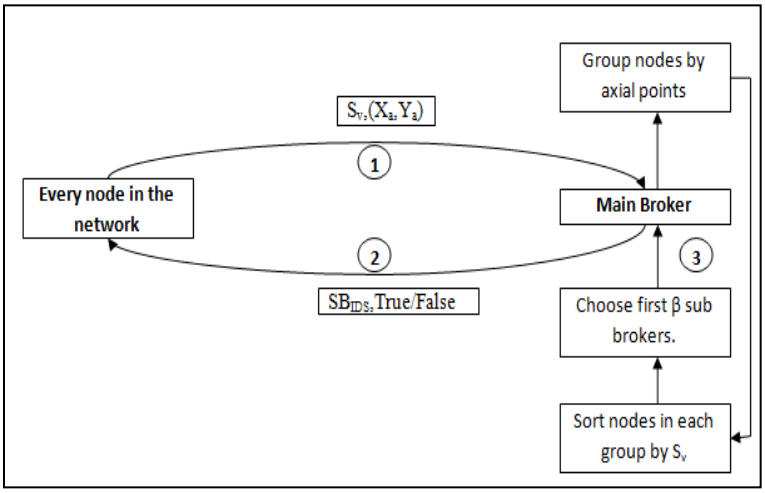

Figure 5.4a. Sub Broker Election Process

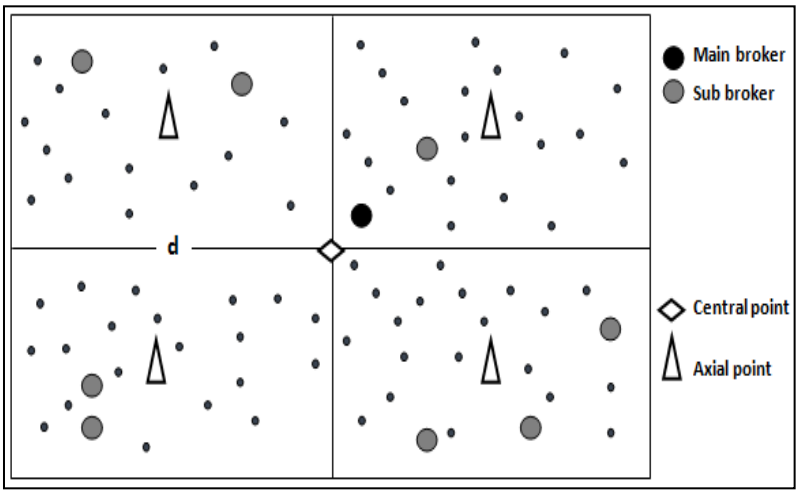

Figure 5.4b. Depicting Sub Broker Election.

5.3.2. Metadata Broadcasts. Every node creates metadata information when a new data is generated or cached. This metadata is sent only to its sub brokers. The sub brokers in turn exchange the metadata information of their group with other sub brokers. Through this, every sub broker in the network will know about the Meta data information of data prevailing in the network.

5.4. CACHE DETERMINATION

Once sub brokers are elected, the main broker runs the cache determination algorithm of MELOC [1]. Our MELOC [1] scheme reduces the number of caches by efficiently bringing data closer to the source. In addition, we identify centrally located and highly connected nodes as cache locations. In our MELOC [1] cache determination algorithm, we have two metrics

C ←Indicates the cache location.

CD ←Indicates the mobile host id, whose original data has to be cached by C.

The output of our cache determination algorithm is the <C, CD> table. The system flow of our MELOC [1] approach is shown in Figure 5.5.

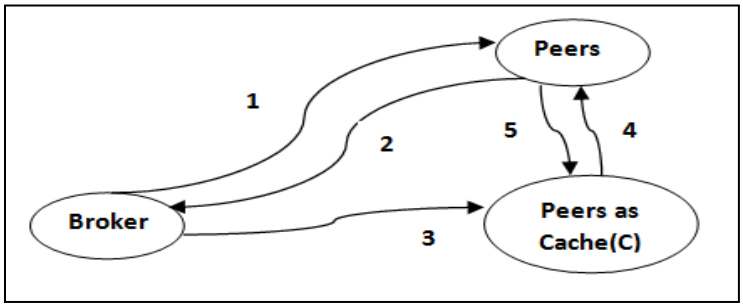

Figure 5.5. MELOC [1] System Process

Step 1: The main broker having the snapshot of the whole network applies "*Identify Cycle" & "Simple_ Comparison"* algorithms to determine the initial cache location and their corresponding data to be cached (Figure 5.6).

Step 2: The main broker optimizes the results of step 2 through "*Cache_ Optimization"* Algorithm, which favors data access efficiency using factors such as shortest path and connectivity. The broker request these optimization factors from peers indicated as (1) in the above Figure, whereas (2) indicates the response

Step 3: It then instructs the corresponding peers chosen as cache location **(C)** to cache their corresponding data **(CD)**, indicated as (3) in Figure 5.5.

Step 4: Peers acting as cache location, contacts the original data center containing data and caches data as instructed by the broker (*pre-fetching)*, shown through 4 & 5 in Figure 5.5.

Step 5: The broker periodically runs a "*Decisive"* algorithm to decide whether to re-determine cache locations. Cache reallocation happens, only when topology changes are drastic.

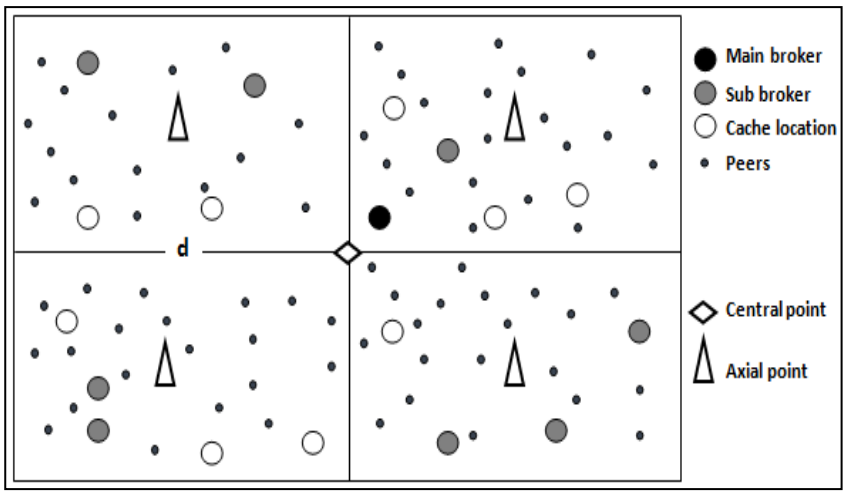

Figure 5.6. Depicting Cache Determination

The cache determination in MELOC-X is same as MELOC [1]. However, MELOC-X differs with MELOC [1] with respect to the following facts

i. In MELOC [1], we assume the first occurring node as main broker, whereas in MELOC-X, we have a *"Main_Broker_Election"* algorithm for electing main broker.
ii. In MELOC [1], cache locations broadcasts metadata of cached data to every node in the network, while in MELOC-X, they send their metadata information only to sub brokers.
iii. For MELOC [1], we have complete reallocation based on *"Decisive_Algorithm"*, but in MELOC-X we have partial reallocation to reduce the frequency of cache reallocation (as explained in next section).

5.5. CACHE REALLOCATION

5.5.1. Partial Reallocation. Large network bears some disadvantages with respect to cache reallocation. The same principle of reallocating the entire cache locations as MELOC [1] is not applicable to large networks due to limited bandwidth constraints and wider placement of cache locations. Every time during reallocation, the main broker has to identify new set of cache locations. Moreover, the old and new cache locations may be placed at a wider distance from the broker, which makes the update process intricate. Furthermore, frequent cache re-allocation increases the number of messages. Hence complete reallocation should be done in large networks, unless there is a need.

Since we intend to avoid frequent reallocation, some mechanism has to be handled to maintain the efficacy of data access. Hence we go for Partial Reallocation. Partial reallocation happens when cache locations move across their zone. The main broker provides the zone limit for every cache locations during initial cache allocation. When a cache location C, crosses its zone, it informs the main broker with a *"Cache_Invalidation"* message containing the following metrics

a) Previous Zone identity (PZ_{ID}).
b) Current Zone identity (CZ_{ID}).
c) Cached data of C <CD>.

The main broker runs the "*Partial_Reallocation"* algorithm for both the previous and current zones. The partial reallocation algorithm is shown in Figure 5.7.

Partial Reallocation Algorithm

I/P: Metrics from cache locations PZ_{ID}, CZ_{ID}, <CD> list

O/P: Cache Reallocation Operation

Notations:

PZ ←Previous Zone

CZ ←Current Zone

N_{CD} ←List of nodes whose original data is in <CD> list.

N_{PZ} ←List of nodes in C's previous zone.

N_{CZ} ←List of nodes in C's current zone.

H [PZ] ←Highly connected node in previous zone

C ←Cache location crossing the zone

CD[C] ←Cached data of C

Trigger :*(Cache Location moving from PZ to CZ)*

if ((N_{CD} *minus* N_{PZ})> N_{CD}/2) then

    Insert CD[C] into H [PZ]

Else

    No operation

if ((N_{CD} *minus* N_{CZ})> N_{CD}/2) then

    No operation

Else

    Delete CD[C] from C.

Figure 5.7. Partial Reallocation Algorithm

When the cache location(C) moves away from the zone, the previous zone may or may not require the cached data of C. Similarly the current zone may or may not require the cached data of C. The condition for determining, whether the zones require C's cached data is as follows

i. N_{CD} be the list of nodes whose original data item is in <CD>

ii. N_{PZ} be the list of nodes in cache location (C)'s previous zone.

iii. N_{CZ} be the list of nodes in cache location (C)'s current zone.

iv. The main broker will maintain C's data in current zone only if (N_{CD} minus N_{CZ})>N_{CD}/2, otherwise it deletes the cached data of C. Likewise if previous zone requires the cache data i.e. (N_{CD} minus N_{PZ})> N_{CD}/2, the main broker will insert C's cached data into a highly connected node in the previous zone.

The movement of cache location C between any two zones leads to any of the four cases listed in Table 5.1

Table 5.1. Possible Cases for Partial Reallocation

Previous Zone (N_{CD}-N_{PZ})> N_{CD}/2	**Current Zone** (N_{CD}-N_{CZ})>N_{CD}/2	**Main Broker Operation**
True	True	**Insert** cached data of C into a highly connected node in previous zone **No operation** with respect to current zone.
True	False	**Insert** cached data of C into a highly connected node in previous zone **Delete** cached data in C
False	True	**No operation** with respect to previous zone and current zone
False	False	**No operation** with respect to previous zone **Delete** cached data in C

In case of partial reallocation, having a single line for zone separation increases the invalidation messages due to mobility. Hence we use a virtual wall of certain width as boundary, represented as dotted line in Figure 5.8.

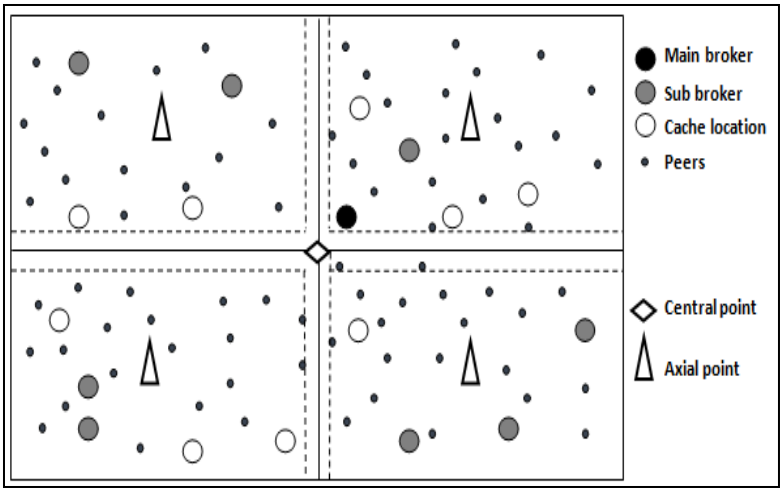

Figure 5.8. Partial Reallocation Virtual Wall

Cache locations crossing the wall and moving to the other zone are considered for partial reallocation. At the same time, this virtual wall is only used for tracking mobility. Other information pertaining to cache location such as sub broker, current zone and previous zone are calculated with respect to single line.

5.5.2. Complete Reallocation. Complete cache reallocation happens only when there is a severe network topological change; each *"Cache_Invalidation"* message has an invalidation count x (IC). We also have sub broker sending "*Subbroker_ Invalidation"* message, when crossing their zone limits; each "*Subbroker_Invalidation"* message has an invalidation count 2x. The main broker will be adding up these invalidation counts (IC). The main broker periodically checks for disconnection of cache locations or sub broker. Every disconnection has an invalidation count 4x. Complete reallocation happens, when the invalidation count exceeds threshold (T_{CR}). At the time of reallocation, the main broker repeats the entire process starting from sub broker election (as shown in Figure 5.1); all previous cache memory is erased. The invalidation count x is user defined and can be assigned with respect to the environment. The algorithm for complete reallocation is shown in Figure 5.9.

Complete Reallocation Algorithm

I/P: Invalidation messages from sub brokers and cache locations.

O/P: Complete Reallocation true or false.

Notations:

IC ←Invalidation count, initially IC=0.

D ←Boolean, true when cache or sub broker is disconnected

x ←User defined count variable

IM[C] ←Invalidation message from cache location

IM[S] ←Invalidation message from sub broker

T_{CR} ←Complete Reallocation threshold

CR ←Boolean variable, initially false

Trigger :(Cache Location or sub broker movement)

```
If (IM[C])
    If (IC > T_CR)
        CR=true
    Else
        Call Partial_Reallocation algorithm
        IC=IC+x;
If (IM[S])
    If (IC > T_CR)
        CR=true
    Else
        IC=IC + 2x;
```

Trigger :(Periodic)

```
If (D)
   IC=IC + 4x;
If (CR)
    Perform complete reallocation
```

Figure 5.9. Complete Reallocation Algorithm

6. HANDLING DISCONNECTIONS

The sub broker disconnections and cache disconnections are handled by main broker using invalidation counts (IC). The main broker will be informing its current invalidation count value to the sub brokers, every time it is updated. The main broker are closer to the central point, hence the chance of main broker, completely getting disconnected from the network is less. However, node failure can happen, so what if the main broker node fails? As discussed in section 5.2, every node within the T_{Dist} limit; realizes the main broker disconnection and participate in the main broker election algorithm, the new main broker now broadcasts its identity to the whole network. The main broker will be retrieving the Invalidation count (IC) value from any of the sub brokers, such that partial or complete reallocation is not affected.

7. DATA ACCESS MODEL

Our access model is based on reduced hops. Requesting node (R) will get the data item from target node (T), which is at a shorter distance from R. In case of DGA, the broker periodically broadcasts the update to all the nodes in the network; hence every node having the meta-data can request the data item from the shortest source, whereas in our approach only sub brokers(S) maintains the meta-data table. This is one important difference between MELOC-X and DGA. Hence, access based on reduced hops with respect to our approach (MELOC-X) needs some modification.

The sub broker requesting data can directly get the source information from its meta-data table and forward the request directly. However, every other node will check its local memory first, if data is not available; they forward the request to the sub-broker (S). The sub broker on receiving the request identifies the sources containing requested data item from its meta-data table. If the sub broker contains the data, it suffices (R). Otherwise, it forwards the request to the shortest source (T) along with the identity of the requesting node(R). The source (T) responds the requesting node(R) with the data item. In MELOC-X, the target source (T) can be a cache location or ordinary node. Similarly, the sub broker (S) too can be a cache location. Thus we have a two way communication compared to one way communication of DGA. Client requests are modeled using Zip-f like popularity distribution [10]. For our evaluation, we are generating queries based on Zip-f distribution with K =15 and α =0.8. We have two prime performance metrics utilizing this data access model in both MELOC-X and DGA as discussed below.

7.1 AVERAGE ROUNDTRIP TIME (ART)

The formula for Average roundtrip in our approach is given as

$$\mathbf{ART_i} = 1/N \sum_{k=1}^{N} (T_f^i - T_s^i)_k + \sum_{k=1}^{N} M_{D_k}$$

, where, N is the number of queries, T_s is the time at which the request is send from the source and T_f is the time at which first acknowledgement is received by the source. M_D represents the maintenance delay due to cache updates through partial reallocation in MELOC-X and through broker broadcasts in DGA. M_D also encounters message processing delays, which increases proportionally with size of the message table.

7.2 AVERAGE HOP COUNT (AHC)

In case of MELOC-X, the communication is two way through the sub broker. Consider node R, getting the request satisfied through sub broker S from target node T. For average hop count, we consider only the hop count between node R and T, the sub broker S is never taken into account. In case of ART, the delay due to every node, including the sub broker is taken into account. Hence, average hop count in our evaluation is not analogous with average round trip time for MELOC-X.

8. PERFORMANCE ANALYSIS

Simulations are done on a network of 100 nodes same as [4] and [12] in an area of 1200 X 1000 m^2. Our performance metrics include Average roundtrip time, Average hop count, Cache count; Cache hit ratio and Query Success Ratio. The evaluation of parameters with respect to above said performance metrics are as follows (a) Variations in number of cache locations with respect to number of nodes (c) Effect of average roundtrip time, average hop count and cache hit ratio with increase in number of nodes. (d) Effect of average roundtrip time and average hop count with increase in number of data items. (e) Effect of average roundtrip time and average hop count for different cache sizes (f) Behavior of Query success ratio with respect to mobility (g) Disparity in number of updates with increase in number of nodes. This section describes our simulation environment and validations through experiments.

8.1. SIMULATION ENVIRONMENT

We have built a simulation environment in JAVA applet to analyze the behavior of MELOC. (Simulation parameters are listed in Table 8.1).

Table 8.1. Simulation Parameters

Parameter	**Default Value**	**Range**
Simulation Area	1200 X 1000 m^2	
Database size n	1000	100-1000
Sub broker size β	4	2-4
Data size	1 MB	
Number of nodes	100	30-100
Bandwidth	1Mbps	
Transmission Range	200 meters	
Velocity	5 meters/sec	5-15 meters/sec
Client cache size	20 MB	2-20 MB
T_N	5	4-7
T_D	4	3-6
Query generation time	1 second	1 – 10 seconds

8.1.1. Environment Specifications. The nodes are added one by one. The first occurring node has a node ID "0", the next occurring node has node Id "1", and likewise it is incremented for subsequent occurrence of nodes.

8.1.2. Node Movement Model. The number of nodes currently residing in the network move randomly based on a random path. Each node will move across subsequent locations in the random path.

8.2. SIMULATION RESULTS

Experiments with number of nodes in X axis are conducted by increasing the number of nodes (N), Number of Data items (D) and cache size relatively. We assume that, every node is capable of generating 5 data items. Hence the maximum number of data items available before caching is (N*5). Having small cache size for larger number of nodes causes starvation of data; hence we increase the cache size as a unit of (N/5). The relative parameter range is shown in Table 8.2.

Table 8.2. Relative Parameter Range

No of nodes (N)	**Number of data (D)**	**Cache size (MB)**
30	150	6
40	200	8
50	250	10
60	300	12
70	350	14
80	400	16
90	450	18
100	500	20

i) We need to determine initial values of T_D and T_N for different network densities. We varied T_D and T_N for different network densities. The number of cache locations increases relatively with decrease in T_N and T_D values for MELOC-X as shown in Figure 8.1. This is due to the fact that the number of groups from *"Identify Cycle"* algorithm increases with decrease in T_N, leading the *"Simple Comparison _Algorithm"* to yield more cache locations. For network densities having more than 30 nodes, we are able to reduce the number of cache locations nearly by 50% compared to DGA, with T_N= 5 and T_D = 4.

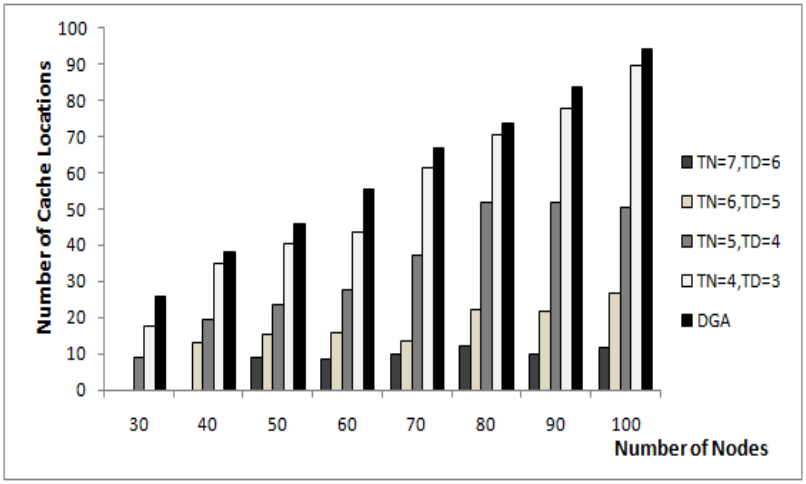

Figure 8.1. Number of Nodes VS Number of Cache Locations

Comparing MELOC-X with DGA in Figure 8.1, the number of cache locations allocated by MELOC is less compared to DGA for all the cases. Further, DGA almost use every node as a cache location. Our approach is good, if we are able to attain same or higher data access efficiency as compared to DGA. Hence, we will be evaluating with respect to Average Roundtrip time and Average hop count

ii) First, we evaluate roundtrip time and hop count increasing number of nodes. We initialize T_N and T_D values for different network densities (N), such that closer to N/2 cache locations are retrieved for N nodes. Hence, with respect to Figure 8.1, we assign T_N=5 and T_D=4 for network density greater than 30 and for 30 nodes we assign T_N=4 and T_D=3. The results obtained are shown through Figure 8.2 and 8.3.

As of Figure 8.2, in case of DGA, every node send its update to the broker and broker broadcasts the update to the whole network periodically. Whereas, in our approach, there is no broadcasts across the whole network. The nodes send their updates only to the sub broker and the sub broker exchanges meta data table with each other. Hence the number of broadcasts and updates (Figure 8.4a and 8.4b) is extremely less compared to DGA favoring round trip time for MELOC-X. As the network size increases the number of nodes performing update and broadcast is more, causing wide difference in round trip time between DGA and MELOC-X.

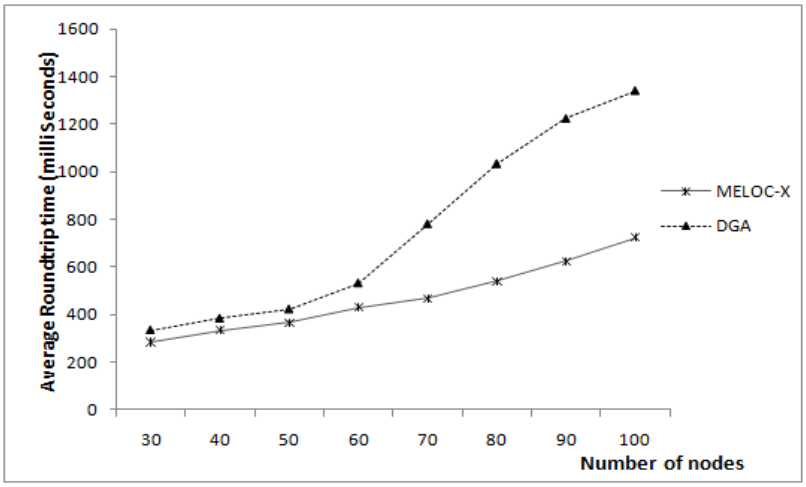

Figure 8.2. Number of Nodes VS Average Roundtrip Time (MELOC-X VS DGA)

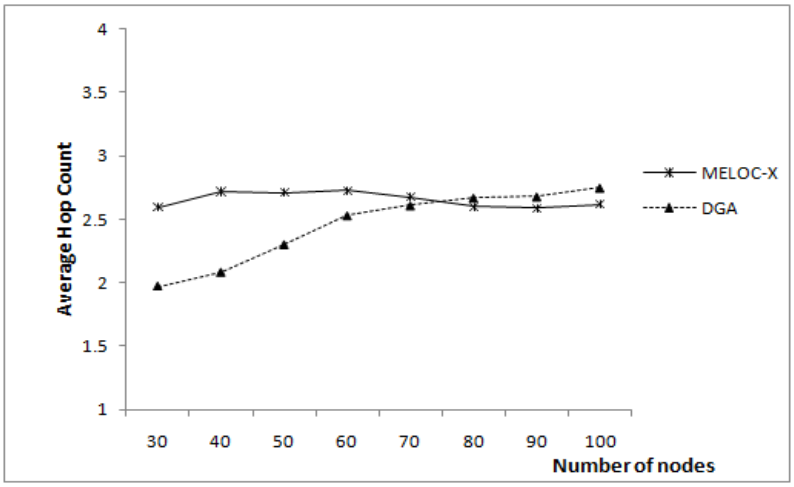

Figure 8.3. Number of Nodes VS Average Hop Count (MELOC-X VS DGA)

Regarding Figure 8.3, MELOC-X have very high hop count for network density less than 70, this is due to reduced number of cache locations and nodes getting placed at a wider distance from each other , the number of centrally located cache locations are also less. When the number of nodes are more, the effectiveness of *"Cache Optimization Algorithm"* in MELOC-X, brings more cache locations centre to the network. Hence, the

average hop count is good for large networks.Relating this ideology to Figure 8.2, this inefficiency causes our approach performing closer to DGA up to 60 nodes, after 60 nodes our performance is extremely better. We also compared the cache hit ratio of MELCO-X and DGA through Figure 8.5a) and 8.5b).

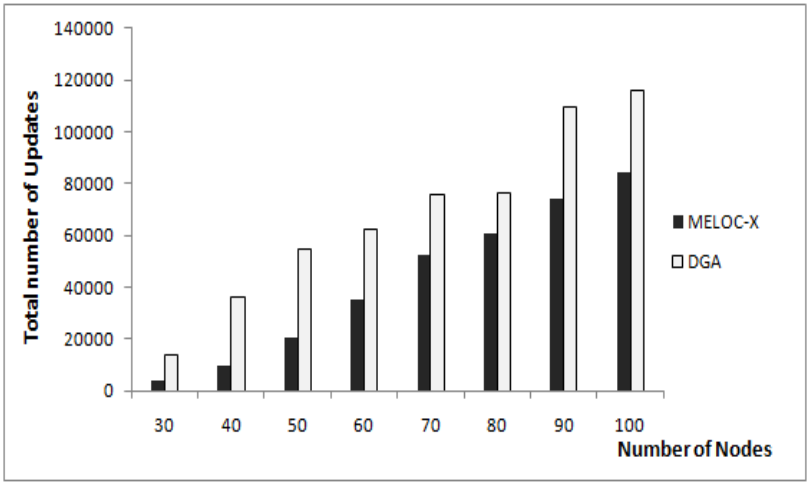

Figure 8.4a. Number of Nodes VS Total Number of Updates (MELOC-X VS DGA)

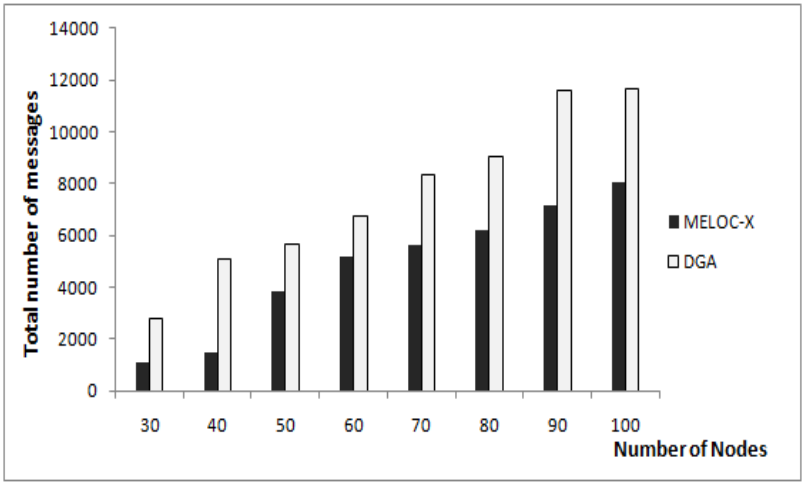

Figure 8.4b. Number of Nodes VS Total Number of Messages (MELOC-X VS DGA)

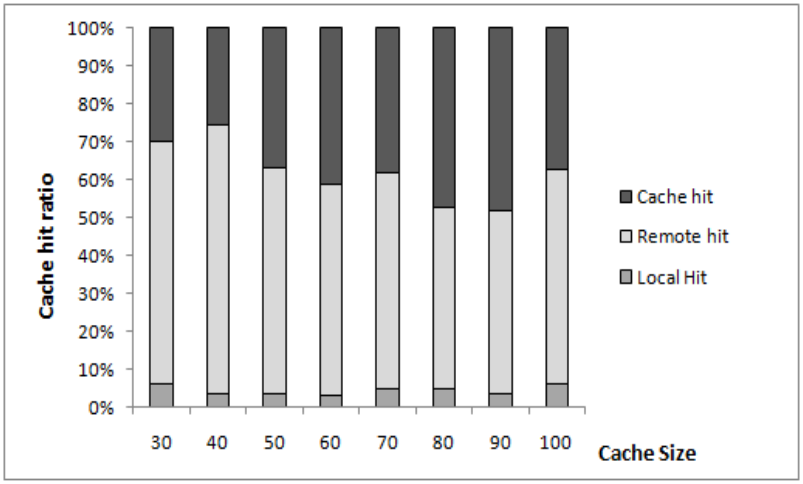

Figure 8.5a. Number of Nodes VS Cache Hit Ratio % (MELOC-X)

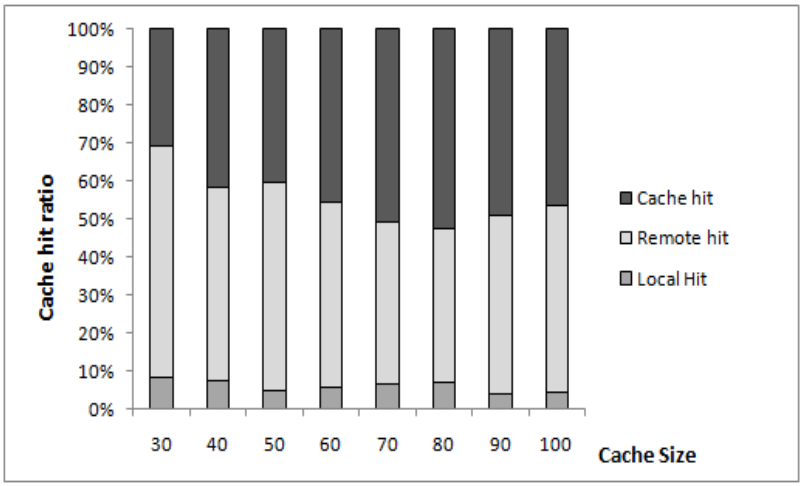

Figure 8.5b. Number of Nodes VS Cache Hit Ratio % (DGA)

DGA has low remote hit compared to our approach, since they maintain more copies. Since we achieve a better roundtrip time compared to DGA (Figure 8.2), this reveals the fact; our approach brings only essential data (distant data) closer to the source. We avoid stuffing unnecessary data even if memory is available, thus showcasing efficient utilization of memory.

iii) We evaluated average hop count and average roundtrip time by increasing the number of data items. We evaluated the number of cache locations for MELOC-X and DGA, varying number of data items for 100 nodes with a constant cache size of 10 MB as shown in Figure 8.6. It is obvious that, DGA almost use every node as cache location. For MELOC-X, the number of cache location occurs between 50-52 (for T_N=5 and T_D=4) irrespective of the amount of data prevailing in the network, thus, showcasing consistent performance. However, the number of cache locations in our approach is inversely proportional to cache size shown through subsequent experiments **(iv-Figure 8.9)**.

With constant cache size and number of nodes, the availability of data decreases with increase in number of data items, hence the average round trip time and average hop count increases with increase in number of records for both MELOC-X and DGA as shown through Figure 8.7 and 8.8. However, we are favoring better roundtrip time compared to DGA due to the disadvantages discussed through Figure 8.2.

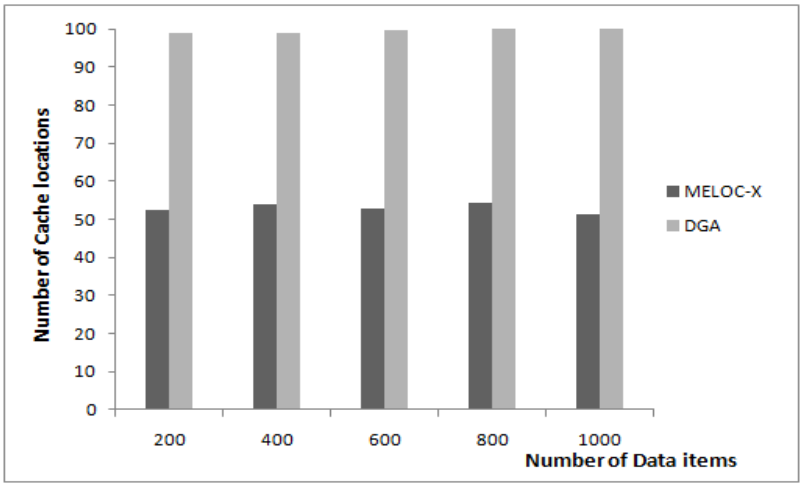

Figure 8.6. Number of Data VS Number of Cache Locations (MELOC-X VS DGA)

DGA utilizes almost every node as cache location (Figure 8.6), thus increasing number of copies compared to MELOC-X. Hence, DGA performs better compared to MELOC-X with respect to average hop count (Figure 8.7). Our two way communication

through sub broker too resulted in increased hop count. However, this two way communication has to be performed in MELOC-X to reduce broadcast, thus favoring roundtrip time (Figure 8.8).

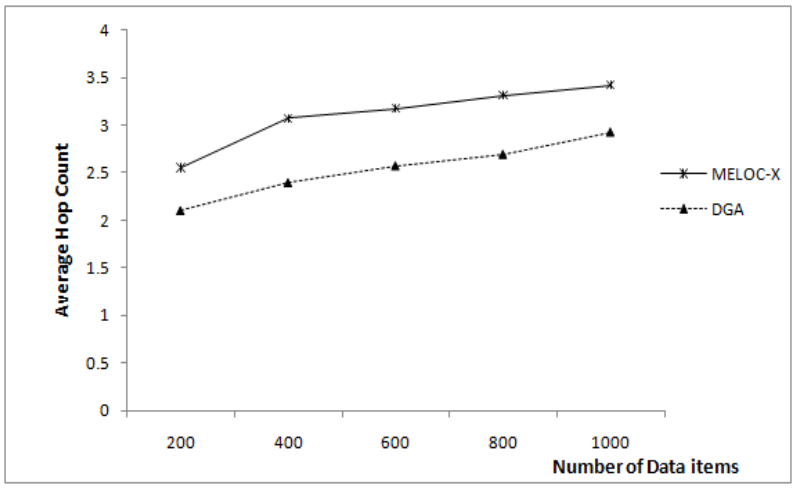

Figure 8.7. Number of Data VS Average Hop Count (MELOC-X VS DGA)

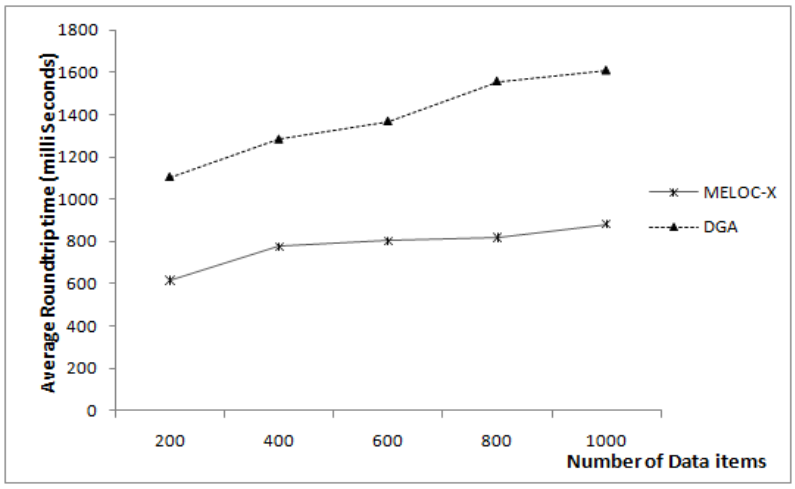

Figure 8.8. Number of Data VS Average Roundtrip Time (MELOC-X VS DGA)

iv) We also evaluated average round trip (Figure 8.10), number of cache locations (Figure 8.9) and average hop count (Figure 8.11) for varying cache sizes with 100 nodes and 200 data items as done in Bin [10]. DGA almost use every node as cache location irrespective of cache size. In case of MELOC-X, the number of cache locations increases with decrease in cache size; this is due to the *"Cache Optimization"* algorithm of MELOC [1]. The plotted results are shown through Figure 8.9.

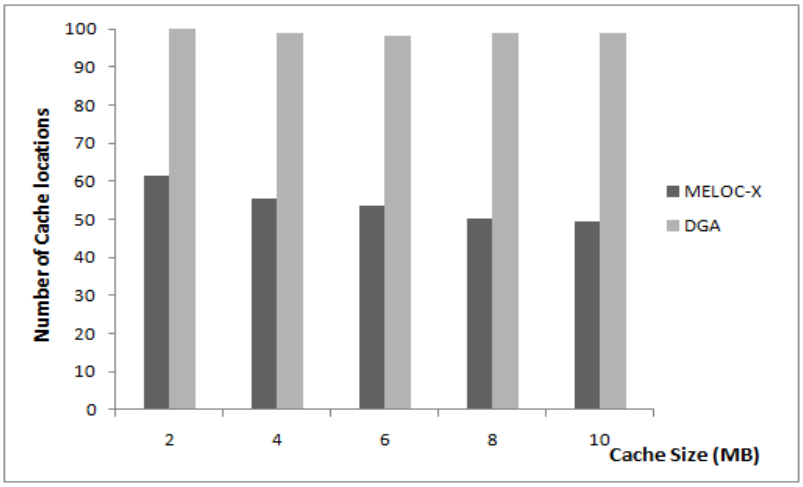

Figure 8.9. Cache Size VS Number of Cache Locations (MELOC-X VS DGA)

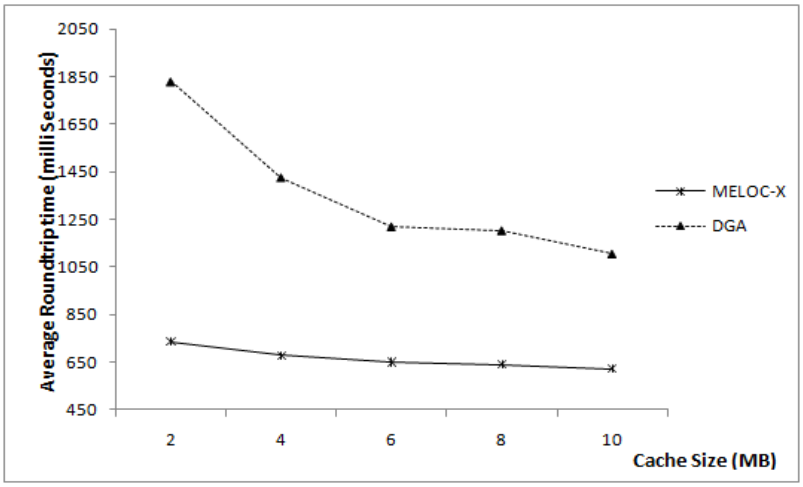

Figure 8.10. Cache Size VS Average Roundtrip Time (MELOC-X VS DGA)

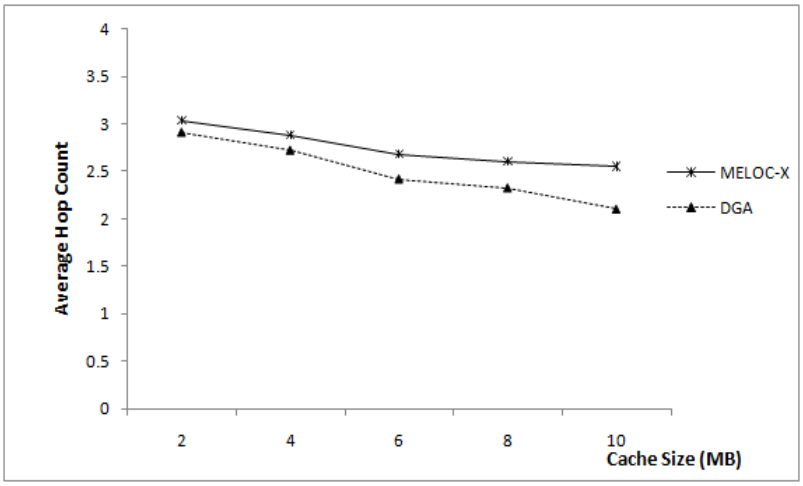

Figure 8.11. Cache Size VS Average Hop Count (MELOC-X VS DGA)

As the cache size increases, the number of cache copies increases, thus constantly decreasing the round trip delay and hop count for both MELOC and DGA as shown through Figure 8.10 and 8.11. With respect to average round trip time (Figure 8.10); MELOC-X performs better than DGA due to the disadvantages of DGA discussed through Figure 8.2, reduced cache size brings reduced numbers of copies for DGA resulting in wide variation compared to MELOC-X. Regarding average hop count (Figure 8.11), the number of copies in DGA is more, favoring hop count compared to MELOC-X. However, the difference is not that huge; remember we are achieving this performance with almost N/2 cache locations (Figure 8.9). Furthermore, as the cache size decreases DGA is performing closer to MELOC-X, showcasing our effective memory utilization.

We also evaluated the cache hit ratio for both MELOC-X and DGA. Referring Figure 8.12a) the number of cache copies decreases with decrease in cache size, hence the cache hit decreases with decrease in cache size, the same occurs for DGA too (Figure 8.12b). However, since DGA uses every node as cache location, the cache hit will be less only at very less cache size.

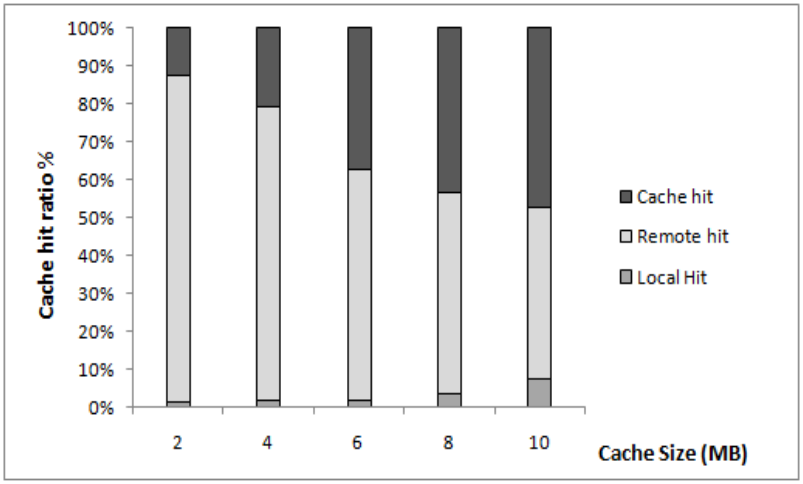

Figure 8.12a. Cache Size VS Cache Hit Ratio %(MELOC-X)

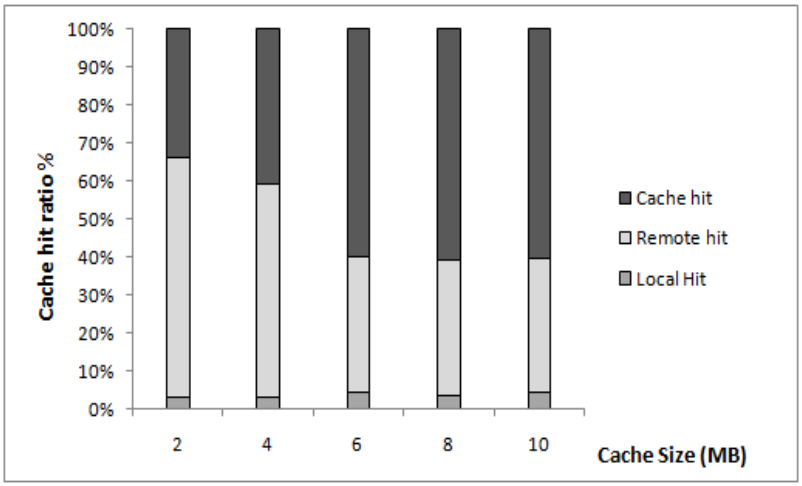

Figure 8.12b. Cache Size VS Cache Hit Ratio %(DGA)

iii) We also evaluated Query success ratio with respect to mobility for a network density of 50 nodes (Figure 8.13), where too many disconnections may prevail. Our MELOC-X approach performs much better than DGA at low order mobility; this is due to the ideology of our approach choosing centrally located nodes as cache locations. These nodes have very less chance of moving away from the network, thus favoring query success. As the mobility increases, even centrally located nodes move away from the network, causing disconnections. Hence, our approach performs closer to DGA at higher order mobility.

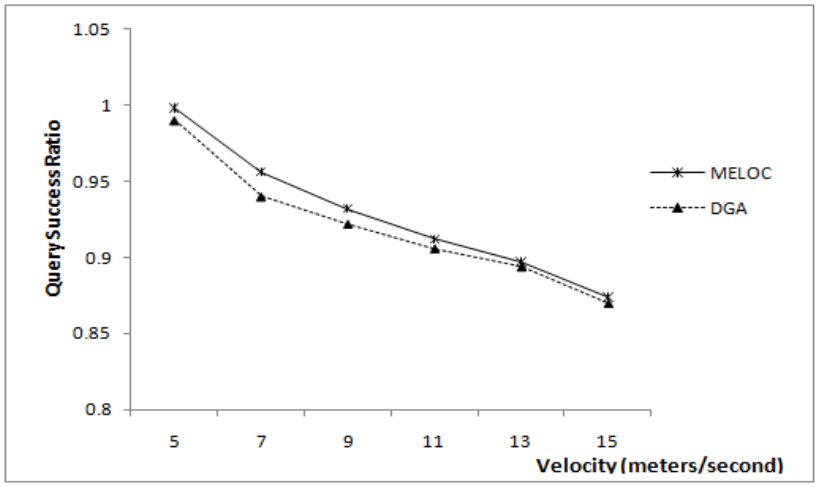

Figure 8.13. Mobility VS Query Success Ratio % (MELOC-X VS DGA)

Figure 8.10 showcased improvement of MELOC-X up to 51% more than DGA, with just N/2 cache locations. Though, we have higher hop count compared to DGA, the difference is very less (Figure 8.7 and 8.11). In addition Figure 8.4a and 8.4b, showcases excellent utilization of energy in our approach compared to Bin [10]. Through all these, we are able to prove, that our MELOC-X preserves data access efficiency by reducing the number of cache locations.

9. CONCLUSION

A caching model for large mobile ad hoc networks to reduce number of caches, without affecting efficacy of data access is presented. The basic Algorithm MELOC [1] is extended to suit large network topology by overcoming disadvantages such as frequent reallocation, increased broadcasts and increased broker load. Simulation results showed significant reduction in number of cache locations, and improvement in data accesses compared to DGA [10].

REFERENCES

[1] Lekshmi Manian Chidambaram, Sanjay K. Madria," MELOC- Memory and location optimized caching for Small Mobile Ad hoc Networks," Proceedings of ICPP 2011, Taiwan.

[2] Takahiro Hara, Sanjay K. Madria, "Data Replication for Improving Data Accessibility in Ad Hoc Networks," IEEE Transactions on Mobile Computing , Nov 2006, vol. 5, no. 11, pp. 1515-1532, Piscataway, NJ, USA

[3] Ying-Hong Wang, Jenhui Chen, Chih-Feng Chao and Tai-Hong Yueh, A Dynamic Caching Mechanism for Mobile Ad Hoc Networks, ICPADS, 2005, pp 605-609, doi: 10.1109/ICPADS.2005.20, Washington DC, USA

[4] Liangzhong Yin, Guohong Cao, Supporting Cooperative Caching in Ad Hoc Networks IEEE Transactions on Mobile Computing, Jan 2006, vol. 5, no. 1, pp. 77-89, doi:10.1109/TMC.2006.15, Piscataway, NJ, U.S.A.

[5] Sunho Lim, Wang-Chien Lee, Guohong Cao, and Chita R. Das, A Novel Caching Scheme for Improving Internet-based Mobile Ad Hoc Networks Performance, Ad Hoc Networks Journal, Mar 2006 , Vol. 4, Issue 2, pp. 225-239, doi: 10.1016/j.adhoc.2004.04.013, Amsterdam, Netherlands.

[6] Guor-Huar Lu, Sourabh Jain , Shanzhen Chen , Zhi-Li Zhang, Virtual id routing: a scalable routing framework with support for mobility and routing efficiency, ACM SIGCOMM, 2008, doi:10.1145/1403007.1403025, pp 79-84, NY, U.S.A

[7] Mayank Pandey, BanshiDharChaudry, A Reconfigurable Distributed Broker Infrastructure for Publish Subscribe Based MANET, SUTC, 2008, pp 361-366, doi: 10.1109/SUTC.2008.30, Washington, U.S.A.

[8] Vipin M, Sankar K, Sarad A V, Building Reliable and Fault Resilient Mobile Ad Hoc Networks, ICSCN, 2008, pp 264-268, doi: 10.1109/ICSCN.2008.4447201 Philadelphia, U.S.A.

[9] Evaggelia Pitoura, Panos K. Chrysanthis,"Caching and Replication in Mobile Data Management", ICDE 2007, pp 846-855, doi:10.1109/ICDE.2007.367930,Istanbul, Turkey

[10] Bin Tang, "Benefit- based Data Caching in Ad hoc Networks", IEEE transactions on Mobile Computing, March 2008, Vol.7, Issue. 3, pp 289-304, doi: 10.1109/TMC.2007.70770 , Washington DC, U.S.A

[11] Lee Breslau, Pei Cao, "Web caching and Zipf- like distributions: Evidence and implications", Mar 1999, Vol.1, pp 126-134, doi: 10.1109/INFCOM.1999.749260, NY, U.S.A.

[12] Narottam Chand, "Cooperative caching in mobile ad hoc networks based on data utility", ACM, Jan 2007, Vol. 3, Issue. 1, pp 19-37, ISSN: 1574-017X, Amsterdam, Netherlands.

[13] Emre Atsan, O znur O zkasap, "SCALAR: Scalable Data Lookup and Replication Framework for Mobile Ad-hoc Networks," ICDCS, June 2008, pp 327, doi:10.1109, Beijing,China

[14] Takiro Hara, "Cooperative Caching by Mobile Clients in Push-based Information Systems", ACM CIKM 2002, pp 186-193, ISBN: 1-58113-492-4, NY, U.S.A

[15] Abhinay Rahore, Sanjay K Madria **"**Adaptive searching and replication of images in mobile hierarchical peer-to-peer network's", Data and Knowledge Engineering, 2007, Elsevier

[16] Hemanth Meka, Sanjay K Madria, "Efficient Simulation Architecture for Routing and Replication in Mobile Peer to Peer Network of UAVs", IEEE MDM 2010, pp 281, doi:10.1109/MDM.2010.92, Kansas City, U.S.A.

[17] Willis Lang, Jignesh M. Patel, "On Energy Management, Load Balancing and Replication", ACM SIGMOD 2010, pp 35-42, ISSN: 0163-5808, New York, U.S.A

[18] Hui Cheng, Sajal K. Dhas, "Stability-based multi-objective clustering in mobile ad hoc networks", , ACM SIGMOBILE 2006, doi : 10.1145/1185373.1185408, NY, U.S.A.

[19] Yi Xu, Wenye Wang, "MEACA: Mobility and Energy Aware Clustering Algorithm for Constructing Stable MANETs," MILCOM 2006, pp.1-7, Washington DC, U.S.A.

[20] Mainak Chaterjee, Sajal K. Dhas, "WCA: A Weighted Clustering Algorithm for Mobile Ad Hoc Networks", Doi: 10.1023/A: 1013941929408, Volume 1 / 1998 - Volume 13 / 2010, Springer, Netherlands.

[21] Hemanth Meka, Lekshmi Manian, Sanjay K Madria, "ROMAN- Routing and Opportunistic Management in Airborne networks", proceeding for CTS 2011, Pennsylvania, U.S.A

[22] Mershad, Artail, "Semantic Caching for Mobile Ad hoc Networks", IEEE MSN 09, pp 25-32, doi: 10.1109/MSN.2009.33, Fujian

VITA

Lekshmi Manian Chidambaram was born on December 9, 1985 in Nagercoil, India. He received his Bachelor of Engineering in Computer Science from Anna University, India in May 2007. He worked as a Software Engineer in Cognizant Technology Solutions, India from July 2007 to June 2009. Since then, he has been a graduate student in the Department of Computer Science at Missouri University of Science and Technology. He worked as a Graduate Research Assistant under Dr. Sanjay Kumar Madria from August 2009 to May 2011. He received his Masters in Computer Science at Missouri University of Science and Technology in May 2011.