SERVICE DISCOVERY ORIENTED CLUSTERING FOR MOBILE ADHOC NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜLŞAH BULUT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

MAY 2010

Approval of the thesis

SERVICE DISCOVERY ORIENTED CLUSTERING FOR ADHOC NETWORKS

submitted by **GÜLŞAH BULUT** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering, Middle East Technical University** by,

Prof. Dr. Canan ÖZGEN _____
Dean, Graduate School of, **Natural and Applied Sciences**

Prof. Dr. Müslim BOZYİĞİT _____
Head of Department, **Computer Engineering**

Dr. Cevat ŞENER _____
Supervisor, **Computer Engineering Dept., METU**

Examining Commitee Members:

Prof. Dr. Müslim Bozyiğit _____
Computer Engineering Dept., METU

Dr. Cevat ŞENER _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. İbrahim Körpeoğlu _____
Computer Engineering Dept., Bilkent University

Dr. Attila Özgit _____
Computer Engineering Dept., METU

Dr. Sinan Kalkan _____
Computer Engineering Dept., METU

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Lastname : Gülşah BULUT

Signature :

ABSTRACT

SERVICE DISCOVERY ORIENTED CLUSTERING FOR MOBILE ADHOC NETWORKS

Bulut, Gülşah
M.S., Department of Computer Engineering
Supervisor: Dr. Cevat Şener

May 2010, 64 pages

Adhoc networks do not depend on any fixed infrastructure. The most outstanding features of adhoc networks are non-centralized structure and dynamic topology change due to high mobility. Since mentioned dynamics of mobile adhoc networks complicate reaching the resources in the network, service discovery is significantly an important part of constructing stand-alone and self-configurable mobile adhoc networks. The heterogeneity of the devices and limited resources such as battery are also load up more difficulty to service discovery.

Due to the volatile nature of the adhoc networks, service discovery algorithms proposed for mobile and adhoc networks suffer from some problems. Scalability becomes a problem when the service discovery is based on flooding messages over the network. Furthermore, the high traffic which occurs due to the message exchange between network nodes makes the communication almost impossible. Partitioning a network into sub-networks is an efficient way of handling scalability problem.

In this thesis, a mobility based service discovery algorithm for clustered MANET is presented. The algorithm has two main parts. First one is for

partitioning the MANET into sub-networks, named “clustering”. Second part is composed of an efficient discovery of services on overall network. Clustering algorithm used in this study is enhanced version of DMAC (Distributed Mobility Adaptive Clustering, which is one of the golden algorithms of the wireless network clustering area). To be fast and flexible in service discovery layer, a simple and fast-responding algorithm is implemented. Integration of two algorithms enables devices to be mobile in the network

Keywords: Adhoc Network, MANET, Clustering Algorithms, Service Discovery, Distributed Computing.

ÖZ

ANLIK KABLOSUZ AĞLARDA SERVİS KEŞFİ TEMELLİ KÜMELEME

Bulut, Gülşah
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Cevat Şener

Mayıs 2010, 64 sayfa

Anlık kablosuz ağlar, sabit bir altyapı üzerine kurulmazlar. En göze çarpan özellikleri ise merkezi olmayan yapıları ve yüksek devinime bağlı dinamik topolojileridir.Anlık kablosuz ağların bahsedilen dinamikleri, ağ içindeki kaynaklara erişimi karmaşık hale getirir. Buna bağlı olarak da, servis keşfi; anlık kablosuz ağların tek başına ayakta durabilmesi ve kendi kendini yapılandırabilmeleri açısından yapının son derece önemli bir parçasıdır. Ağdaki aygıt çeşitliliği ve batarya gibi kısıtlı kaynaklar da servis keşfine daha fazla zorluk yükler.

Anlık ağların değişken doğalarına bağlı olarak, sözü edilen ağlar için önerilen algoritmalar bazı problemlerden muzdariptir. Örneğin servis keşfi, mesajların tüm ağa yayılması üzerine kurulmuşsa, ölçeklenebilirlik büyük bir problem haline gelir. Dahası, ağ düğümleri arasında mesaj alışverişine bağlı oluşan yüksek trafik, ağ içinde iletişimi neredeyse imkansız hale getirir. Sonuç olarak, ölçeklenebilirlik problemini aşmak için ağı daha küçük alt ağlara bölmek iyi bir yöntem halini alır.
Bu tezde, kümelenmiş anlık ağlar üzerinde devinim temelli servis keşfi algoritması sunulmuştur. Algoritma iki ana parçadan oluşur. Bunlardan ilki, ağı daha küçük alt ağlara, yani kümelere ayırır. İkinci parça ise, tüm ağ üzerinde etkili ve verimli bir servis keşfi yapmak üzere tasarlanmıştır. Kümeleme algoritması olarak, kablosuz ağ

kümeleme alanında altın bir yöntem olan DMAC (Dağıtık ve Devinim Uyarlı Kümeleme) algoritmasının geliştirilmiş hali kullanılmıştır. Servis keşfi katmanında hızlı ve esnek olabilmek adına, basit ve sorgulara hızlı cevap veren bir algoritması gerçekleştirilmiştir. Bu iki algoritmanın entegrasyonu ile ağdaki aygıtlara hareket özgürlüğü sağlanmaya çalışılmıştır.

Anahtar Kelimeler: Anlık Kablosuz Ağlar, Devinimli Ağlar, Kümeleme Algoritmaları, Servis keşfi

Çocuk zamanlarımın hayal kahramanı, kızkardeşime...

ACKNOWLEDGMENTS

I would like to first of all thank my supervisor, Dr. Cevat Şener for his warm encouragement and guidance throughout my research work. He was the one who supported me when I felt to be unsuccessful. I would like to express my most heartfelt gratitude to my family, who put up with all my complaints and rants about everything in these three years of higher education. Mum and dad - this would not have happened without you.

I have been partially funded by the Turkish Scientific and Technical Council (TÜBİTAK) BİDEB 2210 National Graduate Scholarship Programme. Thanks goes to many educational programs I have had chance to attend to provided by TÜBİTAK and Computer Engineering Department (CENG) at the Middle East Technical University (METU).

I am grateful to WIMAX workgroup at METU for their seminars which gave me a good know-how about wireless networks. As a side note, this thesis has been implemented in OMNET++ with MIXIM plug in and typed in Open Office. Thanks to MIXIM developers for their fast response on fixing plug in core and best regards to open source software community!

TABLE OF CONTENTS

LIST OF FIGURES

FIGURES

LIST OF TABLES

TABLES

LIST OF ABBREVATIONS

MANET	Mobile and Ad hoc Network
DMAC	Distributed Mobility Adaptive Clustering
DHT	Distributed Hash Table
OLSR	Optimized Link State Routing
SLE	Service Location Extension
SSD	Scalable Service Discovery
AODV	Ad hoc On Demand Distance Vector
DSR	Dynamic Source Routing
DMAC	Distributed Mobility Adaptive Clustering
WIMAX	Worldwide Interoperability for Microwave Access
P2P	Peer to Peer
JXTA	Juxtapose (Sun Microsystems)
CAN	Coordinated Assistance Network
PDA	Personel Digital Assitant
UPnP	Universal Plug and Play
SDP	Service Discovery Protocol
IETF	Internet Engineering Task Force
DHCP	Dynamic Host Configuration Protocol
ONC	Open Network Computing
RPC	Remote Procedure Call
IRDA	Infrared Data Association
OLSR	Optimized Link State Routing

CHAPTER 1

INTRODUCTION

In this chapter, a brief introduction is provided and the reason and rationale behind service discovery and why clustering of ad hoc network has been added will be discussed as the first topic. In the following section, overall view of scalability, clustering and service discovery are presented. The approach taken in our research will be explored in the third section. Some of the design decisions that had to be taken will be given briefly and their details are left to the following chapters.

1.1 Reason and Rationale

Widespread availability of wireless communication has rapidly increased the usage of hand-held devices in the recent years. Such communication environments, named *ad hoc networks* which do not require any fixed infrastructure, have triggered researches on this area. It is obvious that the researches on wireless computing will make more sense in the future as their usage increases.

Ad hoc networks are described as self-configuring networks that are formed and deformed on the fly by a collection of nodes without the help of any prior infrastructure or centralized management in [1]. The definition clearly states that the nodes of an ad hoc network are mobile. Furthermore, they are free to join / leave the network at any time. Thanks to mobility for enabling nodes movement-free but, flexibility and mobility introduce new problems to the communication environment. Scalability and service discovery are the most challenging issues caused by the mobility and infra-structreless architecture. Furthermore, limited resources such as battery, low memory and diversity of these resources among nodes restrict the solution domain and forces algorithms to consider many other network parameters.

The phrase *Service discovery* is used in a wide meaning in the literature. From any text file to any executable, any class method which is called remotely, well defined web servises and sometimes accessing a database may also included as *service.* The discovery of any resource in the network is named as *service discovery*. Another widely used term during the thesis is word *clustering.* In many fields of computer science, clustering is used to group or segment a set of values. In the concept of mobile and adhoc networks, clustering is used to define segmenting the whole network in to smaller inter-connected groups. The meaning of *clustering* is also used in mentioned context during this thesis. The combination of the two definiton, totally describe " providing a service discovery routine over a clustered network".

To use the network efficiently, scalability and service discovery problems have to be solved. How will a node reach a service or a resource? When and on what route? Furthermore, assuming the node finds the correct service, should it store the way to the service provider? In another words, should the node "cache" the route to the provider? At first glance, caching seems a bright idea. However, mobility factor influences caching. What if the provider moves and the route to the provider changes? More severely, what if it disconnects from network? If a node can not find a needed service, then it is not logical to be included in that network.

Furthermore, finding a service is not enough; the node must be able to communicate to the provider node without any interrupt due to high message traffic or high cost of message overhead. Think of a node, trying to find what it needs by flooding its request to every node in the network. This may not seem as a significant problem, even not a problem at all. But the traffic will rapidly increase if all nodes try to find a service by message flooding and in some time none of the nodes will be able to discover services. Additionally, any two nodes, communicating on that time will be affected from high traffic. Neither requester nor provider will be able to send messages to the destination. If we discard flooding and focus on locality, how many hops should be taken account to search for a service? And if the node can not find the service in that particular area, how should it search the rest of the network

without affecting the overall communication? As easily suggested by the questions above, service discovery is indeed a challenging task.

Although being a challenging task, the service discovery and service usage is an vital need in SAR groups, in sensor network application and in other groups or applications where a stable and predefined networking infrastructure is not available. Not only mentioned groups, also a group of people visiting an exhibiton area or in a conference may need such service discovery and service usage availability.

Apart from the studies conducted in related field, this thesis provides a solution where mobility is highly supported while discovering the requested services over clustered network structure. In other words, the point of view in the thesis is enabling the devices to be movement-free while discovering the services. As a rule of thumb, assumption or prediction about mobility is avoided for the sake of the algorithm. The reason behind the idea is keeping the service discovery and clustering algorithms uneffected from the mobility parameters such as speed, direction and acceleration. Since the change in mentioned parameters are undeterministic, assumption on mobility is avoided.

Another distinct property of the proposed solution in this thesis is mobility parameters of the nodes are different from each other. In other words, the simulation tests are run in different node speeds in order to approximate the real world scenarios. Not only approximating real world, also a stress test is conducted to check stability of the proposed solution.
In this thesis, the proposed solution is splited in to two main sections. In the first section, a clustering algorithm, named DMAC is applied and the network is partitioned in to clusters. After core implementation of DMAC, the service discovery algorithm is implemented over clustered structure. In the second section, the DMAC algorithm is improved in order to form more stable clusters. Service popularity is also added as another parameter to the cluster head election criterion.

By the help of mentioned parameter, the clustering algorithm is improved to be adaptive to the needs of service discovery algorithm. The combination of both algorithms perform better traversals while a service is searched among clusters. As time passes and a provided service of node becomes popular, the provider itself or the neighbor of the provider that is on the path of the service is promoted to head role in order to shorten traversal and hit paths. Caching is applied on clusterheads in order to aid shortening the mentioned paths. In other words, the head nodes stores the services provided by their cluster members.

The next chapter presents a wide summary of solutions previously studied in the related field. The flow of the whole thesis is provided in section 1.2

1.2 Rest of the Thesis

In the next chapter, a background information is represented in order to examine solutions that are studied in the field previously. In the third chapter, the solution proposed for the given problem set is provided in details. After discussing implementation and design details in third chapter; the simulation environment, experiments that are executed on the solution and the results are discussed in chapter four. Finally, a conclusion is represented in chapter five. Possible improvements and ideas that could be studied in the thesis are also provided in final chapter.

CHAPTER 2

LITERATURE SURVEY

In this chapter, a detailed background information which is reviewed from the studies done in this is represented. The chapter is divided in to four sub-sections. The first section is a review about scalability while the second section presents a summary on clustering. The chapter continues with the review about service discovery and cluster aided service discovery sections.

2.1 Scalability

For scalability problem of ad hoc networks, DHT based P2P solutions such as Pastry [2], JXTA [4], CAN [3], Chord [5], Tapestry[6] are proposed. Pastry is a generic peer-to-peer object location and routing scheme, based on a self-organizing overlay network of nodes connected to the Internet [2]. In the design of CAN, a virtual d-dimensional Cartesian coordinate space on a d-torus is used. But, the given coordinate space is completely logical and does not depend on the physical structure of the network. At any time, the whole network is dynamically partitioned among all the nodes in which every node owns its individual, distinct zone within the overall space [3]. Every node in the network stores the information of the nodes which lies in its zone in a hash table. As a result, whole system constructs a DHT architecture since every node has its own hash table. The state maintained by a CAN node does not depend on the network size N, but the lookup cost increases faster than logN [5].

JXTA is P2P solution of SUN, and JXTA technology is a set of open protocols where cell phones, wireless PDAs, PCs and servers on the network are enabled to communicate and interact in a peer-to-peer standarts [4]. JXTA members constitute a virtual network and in the network; members can communicate to other peers

directly, even when some of the members may be behind firewalls and network address translations (NATs) or on different network transports [4]. Chord, which was proposed in 2001, argues that the scalability of consistent hashing is improved by avoiding the need of knowledge of all other network members. A member of Chord system needs particular routing information about the network. Since aforementioned routing information is distributed, a network member resolves the hash function by communicating with other network members. In a network with N-nodes, each member maintains information about O(logN) other nodes and a routing table lookup needs O(logN) messages [5]. Tapestry mainly focuses on the locality information of the nodes in order to minimize message latency and maximize message throughput. As CAN maps the physical network layer onto a logical virtual d-dimension space, it does not take network distance to account when constructing the overlay. Therefore, a given overlay hop may span the diameter of the network. Similar to CAN, Chord approach is also not interested in distances. So, it faces the mentioned case of CAN.

There have been so many solutions proposed for the scalability problem which we will not mention here. But roughly, these solutions are composition of DHT and clustering, adjusting some other network parameters such as hop count of clustering neighborhood or number of layers in the cluster. We will continue our survey with the clustering approaches in ad hoc networks.

2.2 Clustering

The concept of dividing the geographical region to be covered into small zones has been presented implicitly as clustering in the literature. Choosing cluster head optimally is an NP-hard problem and existing solutions to this problem are based on heuristic approaches [22]. The heuristic approaches used in clustering are highest degree heuristic, lowest-id heuristic and node weight heuristic. In the first one, the degree of a node is computed according to the distance from others. The latter method assigns a unique id to each node and the node with minimum id is chosen as

cluster head. The last one uses a set of parameters to compose the total weight. Total weight is used to choose the cluster head. Since simulation experiments show that the first and second approaches do not perform well on ad hoc networks, we will not examine them deeply.

Hence, for the efficient management and avoiding effects of flooding, clustering approach is proposed in some papers. Many techniques are used in clustering and every study takes account different parameters. The important questions that should be asked while clustering are;

- How should the cluster head be elected?
- How large will be the cluster size?
- How will the nodes detect the cluster head failure?
- How many clusters should be formed?

As an answer to the first question, several parameters such as battery power, mobility ratio, processing power, distance of nodes to each other are evaluated. By weighting those parameters, a final value is assigned to each node. The node with minimum or maximum criterion value becomes the cluster head. For the second question, number of hop count between nodes is the main criterion. N-hop neighborhood information is used for cluster construction where $N>0$. The final question is solved by periodically checking whether the head is alive or not. This might be done by non-head nodes or the head itself may advertise its existence periodically.

The clustering study in [21] is a gold algorithm which is referenced in many service discovery and clustering proposals. First of all, the algorithm permits nodes to move even in cluster set up phase. It is fully distributed, every node decides its own role according to the information of one hop neighbors. If a node is not a cluster head, it should be only one hop away from its cluster head. Therefore, the algorithm uses one hop neighborhood information to form the clusters. Cluster head is elected according to weight based criterion including node speed, transmission power.

Once the clusters are formed, the rest of the algorithm operates in message driven format. The system has 3 types of messages used in communication between nodes: *CH (node1)*, which is invoked by a node that will advertises itself as the cluster head. *JOIN(node1, node2)* is the message where *node1* uses when it is going to be part of the cluster whose cluster head is *node2*. RESIGN(w) is invoked only by cluster heads who are going to switch to ordinary node role since its weight is smaller than the threshold *w*. The simulation results of the algorithm shows a good performance when compared to "lowest id first" algorithm

The second study on clustering is [22], in which for an N node complete network, it proposes 2 stage distributed O(N) randomized algorithm that always finds the minimum number of star-shaped clusters having maximum size. Actually, the algorithm is aimed to apply to Bluetooth based application. The algorithm has some interesting assumptions. For instance, every node knows total number of nodes in the entire network, the maximum number of nodes that a single cluster can handle. Furthermore, the architecture forces clusters to be star shaped and at the end of the cluster organization process, a single node to have complete information about all clusters. The details of the algorithm can be obtained in [22]. For large scale ad hoc networks, it is obvious that the proposal by [22] will suffer from the assumption that every node have to know total node number. Furthermore, the shape of the clusters is not a matter at all; any shape functioning efficiently is enough for clustering. Thus, cluster shape is not a parameter for clustering process.

In [23], the study takes into account the number of nodes that a cluster head can optimally handle, transmission power, mobility and battery power. For load balance among cluster heads, a predefined threshold value is used. More precisely, the node count a cluster head can manage is a system constant. The solution forces network to invoke the cluster head election procedure as rare as possible. The reason behind the idea is minimizing communication and computation cost. The algorithm also claims that if a cluster head is forced to serve more than the cluster node threshold,

the system will suffer in the sense that the nodes incur more delay because they have to wait longer than usual.

Battery power is an important issue since cluster heads have to consume more battery due to the fact they have to do some extra computation. To help minimizing the cluster head election procedure, relatively less mobile nodes are considered good candidates. When the cluster head moves fast, the nodes may be detached from the cluster head and as a result, re-affiliation occurs. It takes place when one of the ordinary nodes moves out of a cluster and joins another existing cluster. Hence, the amount of information exchange between the node and the corresponding cluster head is local and relatively small. On the other hand, the information update in the event of a change in the dominant set is much more than a re-affiliation. The election procedure is invoked when system initially set activated or the current cluster head nodes can not cover all of the network nodes. Mentioned is a good example of node weight based clustering.

The study in [24] gives a good overview for cluster sizing and hierarchical clustering. K-hop clustering is favored to one hop clustering for efficient scalability. Furthermore, [24] claims that maintenance multilevel clustering requires heavy communication overheads due to the random change of multilevel topology. On the other hand, as we agree with this study, the maintenance of single level clustering is simple since it only manages local topology changes caused by node mobility. But the algorithm implements an interesting idea, which should be questioned: When a new cluster is created, the head broadcasts a cluster state packet to all other cluster heads in the network. We are in doubt whether the head state message should traverse all network. It may not be a vital need for the network. Furthermore, this packet is periodically traversed if;

- members leaves or joins the cluster.
- new neighboring cluster is connected or a current neighboring cluster is disconnected.

- cluster topology changes as a result of cluster creation, removal or head election.

The second and third events are rational reasons to flood the cluster head state message but first reason might be eliminated since it is an intra-cluster event. The study also supports our idea about proactive routing by the claim that packets of proactive routing protocols must carry complete routing list from source to destination which causes high overhead in long multi-hop routes. In contrast to its proposal of k-hop neighborhood clustering, we believe that one-hop neighborhood information is more adequate for ad hoc networks in order to manage high mobility.

Another study worth mentioning in this thesis is [25], which responds efficiently against mobility issue. The most outstanding feature of the proposal is that it does not trigger cluster reorganization procedure if the relative mobility is zero or approximately zero. To achieve such property, mobility patterns of the groups are extracted and applied as parameter in clustering procedure. As being a very recent paper, it also points that lowest id algorithm can not perform expected result in mobility aspect.

Furthermore, the study targets some real time scenarios for the proposal it claims. For instance, a group of rescuers who occasionally concentrate their searching in a specific area, a group of people staying together during a networking session in a conference or engaging in a discussion while visiting in an exhibition hall. In fact, for the success of the proposed method, the nodes in the same group, in other words the cluster members, should be moving in same direction with same speed. Any nodes that do not move in group mobility design have dramatic increase on the variance. Thus, mobility pattern should be used carefully. Nevertheless, taking relative mobility in to account while developing a mobility based clustering should be kept in mind and can be tried. Among all clsuering proposals, dealing with relative speed is distinct idea. For the details of the study, [25] can be revived.

As a final sentence about clustering, it should be noted that there have been many papers on clustering for ad hoc networks but some of them focus on wireless sensor networks or peer to peer structures. Those studies are out of the scope of this thesis but to have a general perspective; researchers interested on clustering can examine them. Next section presents the survey on service discovery proposals.

2.3 Service Discovery

Service discovery problem is firstly addressed in SLP [7], Jini [10], UPnP [8], Salutation [10] and SDP [11]. SLP is an IETF standard and works in centralized manner. It has User Agent (UA), Service Agent (SA) and Directory Agent (DA). UA performs service discovery, on behalf of the client (user or application). SA advertise the location and characteristics of services on behalf of services and DA collects service addresses and information received from SAs in their database and respond to service requests from UAs. When a new service connects to the network, the SA contacts the DA in order to advertise its existence [10]. This is called service registration. When a node needs a service, the UA asks the available services in the network from DA. This process is called service request [10]. But before requesting or advertising a service, UA or SA should discover DA. SLP uses three types of DA discovery: Static, dynamic and passive discovery.

In the first one, the SLP agents get the address of DA from DHCP. In active discovery, UA and SA sends requests to the SLP multicast group address. DA, listening to this address, gets the requests and responds via unicast. In passive discovery, DA periodically send multicast advertisements of their services. UAs and SAs learn the address of DA from advertisements and connects to DA via unicast. DA is usually required in large networks but not a must in overall design. Therefore, SLP operates in two modes: Centralized and decentralized modes. Needless to say, centralized version of SLP is not adequate for ad hoc networks since they do not have fixed infrastructure. Furthermore, non-centralized version of SLP suffers from high message traffic since UAs periodically send out their service request to the

SLP multicast address. All of the SAs listen to multicast address and if they have any service requested from any UA, they respond to the corresponding node via unicast. As a result, non-centralized version of SLP is a typical example of flooding based discovery.

Jini is developed by Sun and describes the problem of how devices connect with each other in order to form a network, called “jini community”. Jini consists of an architecture and programming model and describes how the devices provide services to other devices in the network.

Each device in Jini network is assumed to have a Java Virtual Machine running on it. The basis of Jini architecture is similar to basis of SLP. To join a Jini network, a device or application registers itself to the *Lookup Table* on a lookup server. A lookup table is a database for all services on the network. Devices and applications register with a Jini network using a process called *Discovery and Join*. Not only pointers to services, the Lookup Table may keep Java-based program code. Hence, services may upload device drivers, interfaces and other programs to enable the client to use the service. The code mobility replaces the requirement of pre-installing drivers on the requester [10]. When a requester needs to take advantage of the service, the related code is downloaded from the requester's Lookup Table. In contrast to SLP, the Jini object code introduces direct access to the service using an interface known to the client. (In SLP, service request returns a Service URL instead of direct access.)

Salutation is developed by the Salutation Consortium, which is an open industry consortium. The architecture includes Salutation Manager (SLM) and Transport Layer as two major components. SLM is the basis of the architecture. It serves as a service broker. Any device advertising its services registers to the SLM. When a client ask its local SLM for a service search, the search is performed by coordinating among SLMs [12]. The SLMs operate on the Transport Manager which gives a way

of communication facility independent from underlying network transport layer. It functions as reliable communication layer.

The SLM interfaces contain service discovery, service registration and service access facilities. In service discovery procedure, the SLM discovers other SLMs and the services contained in that SLM. The communication protocol between the SLMs is done using Sun's ONC RPC. This feature is called capability exchange. Its main duty is connecting the SLM for service exchange which is a must in the overall architecture. The SLM holds a registry in order to keep service informations. A client registers its services to SLM registry by itself. A client application can ask the local SLM to periodically check the availability of services. This checking is done between the local manager and the corresponding manager.[12] The communication protocol independence of Salutation architecture is achieved by the interface (SLM-TI) between Transport Manager and SLM [12]. Transport Manager itself is dependent to the network transport it supports. Therefore, a SLM may have more than one Transport Manager, in case it is attached to multiple, physically different networks. But the SLM sees its underlying transport through the transport-independent interface (SLM-TI) [12].

UpnP (Universal Plag and Play) is an architecture for peer-to-peer communication which is introduced as an extension to plag and play peripheral model, it is more than that. In UpnP, a device can join to the network dynamically, get an IP address and discover other devices and services and send out its services upon request. Any device can disconnect the network silently without any inconsistent state. UPnP leverages TCP/IP and the Web technologies, including IP, TCP, UDP, HTTP and XML, to enable seamless proximity networking in addition to control and data transfer among networked devices in the home and office [12].

For the service discovery issue, UPnP uses Simple Service Discovery Protocol (SSDP). This protocol uses HTTp over multicast and unicast UDP which are referred to as HTTPMU and HTTPU, respectively [12]. When a device wants to join

the network, it advertises (sddp:alive) multicast message to advertise its services to control points. In contrast to other protocols, UPnP does not have a central service repository. To seek for a desired service, a device sends out sddp:discover multicast message when a new device is added to the network. Any device that hears multicast message should respond to it with a unicast response message [12].

UPnP uses XML to describe device features. For instance, the advertisement message includes a URL which addresses a XML file in the network that explains the device resources and services. Since XML is a simple and powerful way of constructing text-based resources, it gives the opportinity of being extensibility and flexibility properties to UPnP. The operational steps of UPnP can be listed as discovery, description, control, eventing and presentation. Briefly, in discovery step, a newly joining device multicasts its services to the control points. If a control point is added, it can search for desired services. In description process, the service requester gets the details of the service from the XML file which is pointed by the URL given in the multicast message. The control step is the duration of interaction of requester and provider. Messages are constructed in SOAP format when a service is called or result of a service is returned to the requester. This is very likely to web service method calls. The eventing step is designed for notification of variable service parameters to the potential clients of the services.

Up to this point, brief discussion of first generation service discovery protocols are presented. Before starting to examine combined solutions, comparisons and vulnerable points of protocols will be discussed.

First generation algorithms were mainly constructed on IP protocol and assumed that devices are communicating over Internet. Some of them have central directory as a service registry which is almost impossible in ad hoc networks. They usually assume peer to peer communication either via sending service request service broker, service registry, lookup directory or via flooding service request message to overall network Not only central service directory mechanism, but also flooding is

not adequate for service discovery in ad hoc network. Basically, all of these solutions address similar aspects but in different perspectives and weights. Therefore, an absolute comparison is not rational.

Although, a rough summary is presented on these algorithms, they have significant differences from each other. For instance, Jini is tightly coupled with Java programming language and requires JVM for each device to run needed code and uses RMI mechanism to invoke services on remote devices. This property of Jini makes it independent from operating system and hardware but forces every device to run JVM on it. In contrast to Jini, Salutation and SLP protocols are not dependent on a programming language. Furthermore, it is defined as an upper layer on transport layer and independent from any transport protocol. It is not limited to HTTP over UDP over IP unlike UPnP and SLP. One drawback of UPnP and SLP is being designed on TCP/IP protocol. Jini and UPnP envision pervasive computing environments being enabled by their solutions, whereas Salutation and SLP are primarily dealing with the service discovery problem [12]. UPnP is the only solution using XML based communication which is an important point for being flexible and extensible. Because XML allows for powerful description of device capability, control command issued to the device, event from it.

For second generation service discovery algorithms, we will first look at Konark [13], which was developed in 2004. Although it is a relatively old study, it gives the outline of the ad hoc service discovery in optimal borders. Konark defines itself as service discovery and delivery protocol designed for specifically for ad hoc, peer-to peer networks, and targeted towards device independent services in general and m-commerce oriented software services in particular [13].

For service discovery part, Konark uses fully distributed, peer to peer approach to enable each device to publish and discover services in the network. To be flexible and extensible, service description and meta data description of services are XML based. Since the architecture is peer to peer based, each device in the network acts

as both micro HTTP server and client. To support device and operating system diversity, messaging between nodes are SOAP-based. Konark is neither network layer solution nor application layer solution. Hence, it is a middle-ware solution which designed to close the gap between the two layers. We believe that, such hybrid solutions perform better than solutions which are focused on one layer. The solution assumes IP level connectivity and nothing more on it. Any protocol implemented on IP level, such as 802.11, IrDA, Bluetooth can operate under Konark. Thus, solution is network layer independent which we believe, a positive property for service discovery solutions.

As each device is a local server, it has a service registry which is constructed in a tree structure. The deeper the tree is Last accessed, the more specific services are achieved. In another words, services are grouped according to their properties. For instance, entertainment node in the tree branches in to two deeper branches such as “music” and “game”. Game node may include final leaves such as chess, snake etc. This mechanism resembles an ontology like structure where services are grouped in semantic manner.

A second outstanding study done in 2003 is [14] by U. C. Kozat and L. Tassiulas. The proposed solution is a distributed service discovery architecture that relies on a virtual backbone for locating and registering available services. The proposal consists of two independent components. The first component is formation of a virtual backbone and the second one is distribution of service registrations, requests and replies. In their network model, partitioning is not allowed for the sake of generality; since each partition may be treated as an independent network. The proposal supports directory architecture and for this purpose, the network level solution is built up in two parts. First one is the BBM (Backbone Management) phase where a subset of relatively stable network nodes are selected as the dominating set and adapting this dominating set to the topology changes by adding or removing nodes into / from the set. BBM uses only 1-hop local broadcast control message which are called “hello beacons” in order to from backbone set, create

virtual link between backbone nodes and maintain the backbone. When the BBM phase is successfully completed, the network will have a mesh structured virtual backbone which are named as “black nodes”.

As expected result, every node in the network can not be a member of the dominating set. The nodes only 1-hop away from the a dominating set member is called as “green nodes”. The backbone neighbor of any green node is called Virtual Access Point (VAP) of that node. Hence, if a green node is acting as a server in the network, then it should register all of its services to the directory agent residing on VAP. The rest of the nodes which are neither in dominating set nor 1-hop neighbor of any dominating set member are called as “white nodes”. White nodes should communicate to the backbone member over green nodes.

To elect the dominating set, initially, every node is assumed as white nodes and starts to propagate hello beacons. Any node receiving a hello beacon starts to construct its own NIT (neighborhood information table) and routing table using beacons. Nodes also calculates its total number of neighbors, total number of white neighbors and NNLF (normalized link failure frequency). At the end of waiting period, any white node k,
which complies with the stability constraint, which is highly dependent on NNLF, joins
the virtual backbone and becomes a black node [14].

To maintain the dominating set effectively, three main issues are argued to be solved in the proposal. These issues can be listed as; a green node losing its VAP node, a black node deserted by their green nodes and a black node which is overpopulated. For the first problem, new nodes are forced to join to dominating set. Hence, some of the green nodes with highest stability are selected as blacknodes such that no nodes remain without a VAP node. In the second situation, deserted black nodes leave the dominating set by themselves. Desertion may happen because of a black node may migrate to a location where none of green nodes have this node as their

VAP node (or equivalently all green nodes associated with the same black node may move out of range or have failed to communicate). Therefore, when a black node figures out to be deserted, it should turn itself to a green node. At that time, the issue degrades first situation where a green node loses its VAP node. The solution to the last issue is grouping some of the black nodes in same location in order to balance the load on the overpopulated black node.

After the BBM phase, relatively stable backbone is constructed but there is no mechanism for service discovery to let servers register their services with one or more directory agents and clients request for services. When a server registers with a directory agent, then its registry information should be distributed to the other directory agents located on other VAPs. To do this, a multicast or broadcast messaging is needed. Additionally, every server should keep registration information scope locally by bounding the number of black nodes that the registration message could traverse [14]. So when a node request for a service, the request is forwarded to VAP node of that node. If the request has a valid service entry in directory agent of VAP, then related information is sent back to the requester. But if a valid entry could not be found, then the request is forwarded to other directory agents by multicasting or broadcasting. To avoid too much broadcasting and multicasting in a wireless environment, the proposal uses source based multicast tree algorithm. Shortly, the algorithm avoids the duplicate request messages caused by flooding. Briefly we give the overall perspective of the proposal and for further details of the study, [14] can be examined.

We will continue to examine previous studies with a proposal based on caching which is done by S. Motegi et al [17]. The study is an enhancement on the proposal of “Arguments for cross layer optimization in Bluetooth scatternets” by S. Sesha et al. Simply, the solution uses broadcasting for service discovery messages and response of the service request is turned to the client by unicast. The intermediate nodes which the request and response messages pass cache, the service replies. Additionally, the requester node also caches its own service requests. Furthermore,

if an intermediate node does not have a definition for the requested service in its cache, it simply rebroadcasts the request message. This broadcasting of the node that provides the service makes it possible to find services which are provided behind the node. On the implementation phase, comparison of the enhanced version of algorithm to original one is done. In our opinion, results are not as expected from the enhancement. We will briefly look at the caching issue and why it does not work properly on ad hoc networks in discussion section. For the original and the enhanced version of the study can be found at [17].

We will continue our survey with a study that uses proactive routing principle while helping service discovery. The solution proposed in [20] operates both in peer to peer mode and directory based mode. To remember, in directory based architectures, a client registers its services to the directory and queries the directory in need of service request. If the system is working on directory based mode, the directories should periodically advertises itself. If a service node can not hear from directory, then it explicitly broadcasts a query requesting for the directory information. If there is no directory in the system, nodes simply reply to the queries matching their services which corresponds to peer to peer communication. Hence, the service requester nodes have to flood their request queries to the entire network. The switching between two modes are provided by marking a request with a flag if it is not replied by a directory and resending it to the entire network. By this action, system switches from directory mode to peer –to- peer mode. Actually, paper claims that property of supporting both modes and switching between them is done in order to adapt the dynamics of the ad hoc networks. We will leave this claim in discussion part in more detail.

In [18], IETF standard OLSR protocol is embedded to the service discovery algorithm. Shortly in OLSR, hello messages are periodically sent to sense and establish the sender's neighbors. Topology control messages are flooded periodically via the multipoint relay nodes throughout the entire network to

advertise the topology information of each node. Importance of topology messages arises in building up the routing table locally at each node.

To manage the service discovery issue, the solution adds a new message type which is names as SLE. Mentioned messages perform service advertising, query, response and register functions and they are flooded to the entire network by topology control messages (multipoint relay forwarding). When the location of the service directory is known, the query SLE is unicasted to the service directory. A response SLE may be sent using unicast, although it may be sent using multipoint relay so that other nodes interested in that service may cache the information.

For the scalability problem, proactive routing mechanism uses clustering in order to limit the message traffic which arises due to SLE and multipoint relay forwarding messages. Briefly, cluster heads receives the multipoint relay forwarding messages and summarizes them on behalf of the cluster it is elected by. However in general, SLE messages can not be summarized by cluster heads and have to flooded with full content
to entire network. The study chooses and recommends to work without a service
directory under clustered architecture.
In contrast to [18], [19] favors reactive routing and it is inspired from on demand routing protocols such as AODV and DSR. For service discovery issue, the study extends the routing message format. So it attaches service discovery header to control packets used by routing. In a reactive routing protocol, a route request packet containing the sought destination is broadcast and is replied by a node knowing a route to destination. The service request packet contains a route request packet and an additional header describing the needed service, which is named as service request extension. Briefly, the extension part contains some information such as service selector or any semantics used for service discovery. As a reply to the service request packet, a service reply packet is prepared including service rely

extension and standard routing packet. Similar to service request extension, it contains information about the service such as lifetime and other parameters.

When a client requests a service, it creates a service request packet and sets destination address to zero. The packet is propagated through the network as an ordinary route request packet. Any node receiving this packet fills the address if it knows a matching provider or if it is a provider for that service. Hence, each node stores service bindings which contains an association of a service type to addresses of provider nodes. To keep the system up to date, providers update their bindings locally and other nodes will cache services with a lifetime indicator from service reply packets which pass over them. If there are multiple services bound under same type, the closest provider to the requester is chosen and its address is sent back to the requester. To avoid incorrect replies due to cache inconsistency, the service with the largest lifetime is selected to sent back.

Using a reactive routing protocol in basis, however, the study also adds some proactive enhancements to its solution. This is done for the sake of caching consistency. Generally, pro-activity is activated by a provider becoming just available or vice verse. Hence, caching mechanism explicitly removes cached entries by negative announcements for maintain caching benefits. However, the result of this decision is not reflected as a parameter the simulation results. This is just an assumption in the study.

As a final study in this category, we will examine [29]. Briefly, the study is based on the integration of service discovery with network layer. The motivation behind the idea is efficiency of cross layer service discovery methods in energy constraint mobile devices. Because putting service discovery at the network layer reduces the control messages overhead and as a result, less number of messages are exchanged among devices. The study uses AODV routing protocol and strictly dependent to it. In other words, the method can be used only with AODV since the request and reply messages of AODV is extended for service discovery purpose. Furthermore, the

study believes that in a cross layer service discovery architecture, periodic advertisement of services does not cause to much traffic overhead since the service discovery parameters are embedded in routing protocol messages.

Since it is based on routing protocol, ever node in the network maintains a routing table and a service table. Each node advertises its services periodically and nodes receiving the advertisement updates its service table. The node is also free to select interested services among received service advertisement. If a node hears an advertisement, it reschedules its broadcasts at a later time to avoid network congestion. This is an interesting feature of the study. In the simulation studies, 50 nodes with 25 services are tested against 174 requests. In our opinion, large scale network concept must have more than 50 nodes. Furthermore, there is nothing special to mobility issue.

Before introducing the world of clustering and service discovery, we will briefly discuss the pros and cons of the aforementioned techniques. The study done in Konark [13] is peer to peer and inadequate for large scale networks. It is independent from network layer routing protocol and assumes IP level connectivity. Furthermore every node in the network have to maintain a micro-HTTP server. These properties of Konark forces clients to obtain an IP and set up HTTP server. On the other hand, for any place peer to peer communication is available, Konark is suitable.

Proposal in [14] uses virtual backbone mechanism and the stability ratio of virtual backbone nodes are not questioned for real life scenarios. The idea behind the proposal seems like adapting backbone structures of fixed networks in to ad hoc networks. Since the proposal is mainly a network layer solution, broadcasting and multicasting features are used for query and response issues. To decrease the effect of high message overhead, the solution uses source based multicast tree algorithm but the efficiency of this algorithm for reducing message overhead is not clear. The algorithm itself also burdens extra overhead to the system. But the overall system

has the advantage of little extra service discovery effort since the service discovery messages are embedded in network layer routing messages.
The study [17] uses improved version of caching and the simulation results seems to perform better than standard caching used in the study it is compared to. In general, the bare caching fails in mobile and ad hoc environments since intermediate nodes respond with configuration information on services from the cache and do not rebroadcast the request message. Hence, this method has the drawback of lowering the number of discoverable services since the clients can not discover other services whose configuration is not cached at the responding nodes [17].

Proposal in [20] favors DHT against directory based architecture depending on the simulation results they obtain. But the directory based architecture used in the simulations periodically advertises itself and this means extra message overhead. In architectures where such messaging is tuned, the results may change.

In [18], network layer support is used for service discovery. Routing messages are extended for this purpose. But all messaging depends on broadcasting, which simply means flooding To control and tune messaging overhead, a cluster based routing is added to the solution. However, clustering can limit and reduce the multipoint relay message but SRE messages ca not be summarized by cluster heads and have to be flooded.

Study of [19] consolidates our idea on proactive routing and caching. The simulation results show that caching with reactive routing performs poorly against reactive routing without caching on invoking valid services. Furthermore, the study states that proactive routing with full caching fails since nodes will maintain a route to a matching provider whenever possible, nodes do not incorporate new information. If a new, relatively remote provider becomes available and a route to another provider exists, the node will never learn the new provider, as no state change of the node is required. Therefore, in our idea, if caching will be used, it should not be in entire network layer and should not depend on flooding messages

over the nodes. Instead, a relatively local and highly communicated hosts may be cached.

2.4 Cluster-Aided Service Discovery

The first study we will mention in this category is multi-layer cluster based service discovery [16]. As the name explains, the solution consists of two parts. In first part, the nodes of the network are partitioned into clusters. Furthermore, the selected cluster heads are again subject to head election procedure. Election process continues recursively until a fully hierarchical structure is obtained. The second part of the solution constructs service descriptions by an ontology which are also in tree structure. Mainly, the solution is based on mapping these two trees on each other. One important issue that should be pointed out is that the proposal assumes that services are described by an ontology common to all nodes. To combine the two layers, a set of devices forms the first level clusters if and only if all offered services are described by the same leaf concept and every two devices from this set are mutually reachable via that set of devices. The clustering policy does not restrict the schema by any parameter such as cluster size or cluster shape. On the next layer, clusters that offer services described by the same layer in the ontology are combined and a connected graph from a cluster view is formed. The architecture consists of n +1 layers where n corresponds ontology tree depth and 1 corresponds to device level which is named as leaves. On the upper layer of the ontology, service descriptions become more general. If desired service can not be found in layer m, then it is forwarded to the layer m+1. On the upper layers, as the service description becomes more general, the search scope gets larger.

The proposal claims that it allows to find services without relying on a centralized directory server, minimizes message overhead and welcomes the high dynamics and ever changing topology of ad hoc networks gracefully. We will briefly discuss this claim at the end of this section. For the lemmas and theoretical proofs, [16] can be consulted.

Another study worth mentioning is “Lanes” [15] by M. Klein et al. The basic idea of Lanes is to define a two dimensional overlay structure, called *lanes*, which is similar to, but less strict than the one implemented in the approach of the CAN that is mentioned in section 1.2. One dimension of the overlay is used to propagate service advertisements, the other one to distribute service requests. The authors claim that the lane architecture is fault-resilient, efficient structure that can be used in semantic based service discovery.

The proposal claims that strict grid architecture of CAN should be weakened in order to satisfy the bottleneck of the ad hoc network such as limited sources etc. The Lane design uses the special case of CAN where d=2. In standard CAN, every node have to maintain 4 neighbors where dimension is 2. In Lane proposal, nodes are loosely coupled such that Y interval is not important in two dimensional overlay. Hence; only X interval is a matter, now the grid structure degrades to lane structure. Within a lane, nodes are fixedly ordered and each of them knows its predecessor and its successor [15]. Nodes in the same lane share the same anycast addresses which help to use anycast routing for sending messages from lane to lane [15].

Lanes solution builds up service discovery and service announcement functions over lightweight CAN architecture. Actually, the service discovery mechanism is similar to original CAN’s. But CAN’s hashing mechanism could not be directly used in Lanes since simple hashing can not handle rich semantic service description. Hence, service discovery is highly separated from overlay structure of the network. It is built up by exclusively using the fundamental semantic of service trading. Since there are two orthogonal dimensions, one of them is used for announcing offered services and the other is used for searching suitable services. The solution uses combination of a proactive structure within one lane and a reactive structure between the lanes which leads to anycasts to reach an arbitrary node in neighboring lanes. To give an idea about lanes, we summarized general view. Broken connection handling, node addition / removal issues of Lanes is not mentioned here. For further information, [15] can be reviewed.

Multi layer clustering approach of [16] has two main drawbacks. First one is, the idea of hierarchical clustering in a highly dynamic environment. Since the nodes are mobile, it may not be possible to maintain such an architecture with little effort, messaging and computational overhead. Another drawback is mapping a semantic structure on to this architecture for the mobility reason. In our idea, using semantics in such an environment is also not very efficient since every node in the network have to know the semantics of how organizing its services. A newly joining node can not advertise its services in such an ad hoc network if it has no idea of the of the service description semantics.

Lanes architecture also suffers from semantic idea. Furthermore, it uses a light version
of CAN, which constructs a virtual overlay over the physical network structure. Thus, location information can not be used efficiently in such architectures since it sees the network in virtual manner. Therefore, we believe that using both semantics and virtual network overlay might add some extra computation burden to the nodes.

In [26], cluster-based peer to peer service discovery is proposed. The study aims low message overhead, scalability and robustness. Nodes in a MANET that run this algorithm will form dynamic clusters and service information is cached in clusters by DHTs. For the clustering process, every node is assumed to have a unique identifier. At the initialization phase, all nodes are in unknown phase and broadcasts hello messages periodically. The periods of broadcasting is a system parameter and can be adaptively changed according to the mobility patterns of the nodes.

In a hello message, the unique id and state of the sender node, neighborhood and cluster information that sender knows are set. The cluster has a radius, which is measured by hop counts, called *csize*. To adjust the communication, the radius of cluster heads are set to (2*csize+1). If a node with an unknown state hears a hello message from a head, it changes ist state to member mode. If the node can not hear

any hello message in a time interval, it switches to head mode since it has locally minimum unique id.

For the service discovery and caching, service information is subject to a hashing function. The extracted hash value is used by consistent hashing to distribute the it among the unique ids of cluster members. When a node wants to find a specific service, it puts the service description hash value and its own unique id in a packet and sends it to its cluster head. cluster head first checks intra-cluster sources by mapping the hash value of the desired service to the node ids. If a matching node is found, the information is returned to sender in a service reply packet. In case cluster head can not find it in the cluster, it sends out the request to other cluster heads in the network.

The algorithm [26] has a very simple clustering algorithm but it uses lowest id heuristic unfortunately. It does not take into account node capabilities. It should be improved by node weight heuristic. Furthermore, the radius between two cluster is (2*csize+1). However, the motivation behind this assignment is not very clear. Brief explanation claims that cluster heads can know how and where to contact their neighbor clusters but pseudo-code of clustering algorithm does not use the mentioned value. Such hop count limit can be used but (2*csize+1) is fixed value and the motivation behind the decision is not described. Furthermore, this value is not changed in the simulation experiments and its default value is not given in the system parameters table. Thus, it should be a system parameter and its effect to simulation result should be traced.

There are also studies which utilize the advantages of directory based mechanisms and P2P technologies. A good example is [27]. The architecture basically relies on P2P overlay network which is constructed by couples of directories. Similar to the previous study, it also stores service information in DHTs. In the architecture, the directory nodes construct a cluster layer and serve ordinary nodes. An ordinary node is only one hop away from any directory node in the cluster. In the cluster layer,

each member node owns a database for cache of the service information. The idea behind this approach is to avoid flooded advertisement or request within the network.

The cluster formation is based on node weight heuristic. As similar to other studies; transmission power, mobility factor, energy factor and processing capability are the parameters used for assigning a node's weight. When every node calculates its own weight, neighbor nodes compare their values with each other. The nodes with minimum weight are elected as directory of the cluster. The processing power is a distinguishing factor if any equality occurs between nodes. The elected directory multicasts a piece of cluster information within the cluster announcing its existence. Then the node begins to listen to the messages from the member nodes. The neighbor nodes hearing this message changes its state to member mode and send its existence to the cluster periodically and observe the message density from the directory to estimate the degree of separation. If the message density is high, the member nodes sends a message to the cluster stating that it is going to leave. Similarly, if directory node can not hear any heartbeat from member node for a time interval, that node is removed from the list.For the service discovery, a typical DHT implementation is used. Since we have mentioned it in the previous studies, we will not repeat it here.

To compare the results, [27] chooses SSD by F. Sailhan et al. The simulation results show that they have better results in message overhead issue, service discovery success rate and average delay rate. But it is questionable whether 80 nodes are enough to prove this algorithm works in large scale networks. For 80 nodes, simple flooding may perform similar results and it has not been discussed in paper.

In [28], DMAC [21] based service discovery is proposed. Although the study focuses on wireless sensor networks, it takes mobility into account and it is in the scope of this thesis. As mentioned in the previous section, DMAC uses one-hop neighborhood information to form clusters. The study of [28] loosens this rule and

allows an ordinary node be two or three hops away from cluster head. For the intermediate hops, “parent” concept is introduced. In the clustering formation phase, nodes with the highest capability grades among their neighbors announce themselves cluster heads. The remaining nodes choose as parent the neighbor with the highest capability grade. Hence, the ordinary nodes hear cluster head announcement message from their parents.

The clustering structure proposed in this study resembles a forest composed of a set of trees (or clusters). The height of the cluster is the longest path from the root node to leaf. Two clusters are said to be adjacent if they have a common node connected through a link. This common node can be considered as “gateway” node between the clusters. If these gateway nodes are leaf nodes, they rebroadcast the cluster head information. Hence adjacent clusters are able to obtain cluster head information of the neighboring clusters.

If a node discovers a new neighbor with a higher capability grade than its current parent, it selects the new one as its parent. Failure of parent link also triggers an ordinary node to search a new parent. Newly discovered parent can be in an adjacent cluster; as a result, cluster head information is propagated to the leaf. Different from this algorithm, DMAC imposes maximum cluster height of one; resulting in direct access to the cluster head instead of parent node. In DMAC, role decision of a node is dependent on the decision of the neighbors with higher weights. The study claims that single topology change in DMAC may trigger re-clustering of a whole chain of dependent nodes.

The service discovery protocol has two main parts: service registration and service discovery. In the first part, leaf nodes send their services to their parent. Any node which is not a root also sends its own requests and requests of children to an upper layer. Hence, every node knows the services of lower layers in the hierarchy. Since the registration process requires unicast messages to be transmitted from children to its parents, it can be integrated with the transfer of knowledge on adjacent clusters

Thus information update message is used for both service registration and transferring the knowledge on adjacent clusters.

In the service discovery part, the distributed service directory which is constructed in registration phase is used. The node sends out its request to its parent. If first level parent can not find it, the forwarding iterates to upper levels in the hierarchy. If message reaches the root node and it can not find it in the local directories, the message is send out the other cluster heads. The next hope on the path leading to the adjacent cluster is decided by every node that acts as forwarder of the message. Although the response to the request can be returned by the first node that finds a match in its registry, it is issued by the providing node itself. Motivation behind the idea is to avoid matching any non-up to date service descriptions to the requester. Since some applications may change description or even simpler, some parameters, it is a good idea.

To improve hit ratio of service discovery, only root nodes caches the requested services for a limited period of time. To achieve satisfying results, three parts of work is done. At the first part, original DMAC algorithm is implemented. On the second part, a simple service discovery algorithm is implemented and original DMAC algorithm is tested with service discovery algorithm. On the third part, DMAC algorithm is improved in some manners in order to aid service discovery which will lead it to cause less message traffic and improved service discovery hit ratios. The next chapter will start with the presentation of formal definition of wireless networks and the assumptions done for the sake of the simulation code. Before the implementation, as a final step and deep understanding of the simulation, the design philosophy and the architectural key points are given.

CHAPTER 3

SERVICE DISCOVERY ORIENTED CLUSTERING

In this chapter, the design and implementation of the algorithms used in thesis is represented. Before the essentials of the algorithms, it is suitable to give the definitions, preliminaries.

3.1 Problem Definition and Preliminaries

An ad hoc network is modeled by an undirected graph G=(V,E) where V is the set of wireless nodes forming wireless network. If u and $v \in V$ can mutually receive signal from each other, it denotes an *edge* and presented as $\{u, v\} \in E$ where E is the set of all edges in the wireless network. For a wireless node, to communicate each other, a set of nodes which this node receives signal is called set of neighbors and denoted as $\lambda(u)$ where $u \in V$. Since nodes might be mobile in a wireless network, the graph G and $\lambda(u)$ can change. Generally, a node belonging to V will be denoted as *u*.

A node in the network has a unique identifier called *nodeId* in the rest of the thesis. Furthermore, every node has a *weight* which indicates its appropriateness to be a clusterhead. The weight of *u* is denoted as w_u. It is the sum of some attributes of *u* such as remaining battery power, mobility ratio, CPU and memory capacity. For simplicity in simulation, it is a real number assigned randomly when *u* powers up.

The word *clustering* in this context means partitioning the graph G into sub-graphs such as C_i where $i>0$. The node set of C_i is represented as V_{C_i}. C_i has a clusterhead *c* satisfying $w_c > w_u$ where $u \in V_{C_i}$ and $c \in V_{C_i}$. The nodes except *c* are called

ordinary nodes in cluster *C*. There are some assumptions used in simulation implementation:

- If *u* can hear *v*, then *v* can hear *u*.
- The packet queue of *u* is large enough to handle messages.
- The messages sent by *u* is received correctly by $\lambda(u)$

3.2 Design Philosophy

To improve service discovery in ad hoc networks, either service discovery itself should be improved or the underlying structure should be improved in order to aid and support service discovery algorithm. An important point that should be kept in mind is not to separate both of two from each other. This will result in more communication cost between the two. For this purpose, some studies mentioned in previous chapters decided to embed service request and service response information into routing messages. In this thesis, the second method is chosen to improve the overall performance and service discovery is embedded in to clustering algorithm.

To discover services, an adhoc network can choose two different ways as a networking infrastructure. The first way is flooding as mentioned in previous chapters. As it causes too much messaging overhead in some cases, this problem resulted in sub-partitioning the network to decrease the messaging overhead. Since simulation results of clustered networks perform better against flooding-type networks, this thesis chooses to focus on clustering as a networking infrastructure.

Flooding-type networks usually best perform when number of nodes in the network is relatively small and almost all of the nodes can hear each other. For instance, a group of students doing team work in a laboratory is a good candidate for flooding implementation. Furthermore, since the number of nodes in the network is relatively acceptable to manage in a routing or service table. This fact results in fast response to any service request and fast sense of service break hosted in any node. But when

the network size gets larger, it is very difficult for flooding topology to handle traffic. In such situation, partitioning the network in to sub-networks, namely *clustering*, is a good idea to control the traffic via clusterheads. If each node had complete view of entire network topology as in flooding, routing table would explode to an immense size. Clustering is therefore a vital for efficient resource utilization and load balancing in large-scale dynamic networks.[25]

However, solely clustering can not meet the service discovery requirements of dynamic ad hoc networks. To get the maximum performance for clustering, there are some architectural decisions to be highlighted;

- The number of layers that the clustering will be performed.
- The number of hops used for constructing neighborhood.
- The number of directly connected clusterheads in neighborhood of a clusterhead.

The "layer" problem of clustering is mentioned in details in previous chapters. The decision for layer problem used in this thesis' clustering algorithm is one-layer clustering. The fact that any node failure in lower layer causing extra reorganization on higher layers plays an important role in "one-layer clustering" decision. For constructing neighborhood list, one hop neighborhood information is used. The motivation behind this decision is two or three hop neighborhood information results in slow response to topology changes although the nodes have better knowledge of surrounding neighbors. The situation where two or three hop neighborhood best performs is scalability issues of non-mobile nodes.

The discovery phase has also two important design issues to be considered. The most outstanding and performance critical issue is caching. Unfortunately, caching mechanism does not have an absolute impact on service discovery. The impact varies as mobility of the nodes vary. Hence, the caching mechanism must be a property of the service discovery that is triggered according to the state of the ad hoc network, mainly depending on the mobility of the nodes. As a supporting property

to the caching mechanism, either on demand routing or proactive routing should be implemented. The natural behavior a caching mechanism shows is proactive routing.

In proactive routing, the path between two node is determined before it is needed. Hence, a detailed management of routing table is required. For the sake of service discovery, proactive routing caches the routes used in previous messaging sessions. On the other hand, on demand routing does not cache any routes belonging to previous transmissions. The idea behind on demand routing is that caching routes in ad hoc network is usually results in false paths since nodes move frequently. Not only failing paths occur, but also it causes extra message traffic and needs one more path request among nodes to find an alive path to the destination node. Hence, a second try for a path to the destination will also delay the getting service request result.

When compared to each other, proactive routing seems to supports caching mechanism better than on demand routing. On the other hand, it should be kept in mind that mobility may force ad hoc network clusters to use on demand routing instead of proactive routing. As in caching mechanism, it should be decided according to the state of the clusters of network.

Although the service discovery algorithm itself is not the main subject examined in this thesis, the algorithms supporting service discovery have to be tested against a service discovery for the sake of performance and reliability. Hence, it should be a consistent algorithm running in every node to welcome any service request and it should be able to map any key word to corresponding / related service that the node hosts. A service discovery entry must store a keyword set each service. Furthermore, it must enable requesting node to be aware of the required parameters to trigger the service. Briefly, the two core functionality a service discovery algorithm must include are ability to map a request extracting it from keyword set and to have adequate information to be served to the requester in order to start service. In previous chapters, many service discovery algorithms are discussed in a large range

from embedding little information in to routing protocols to highly complex semantic based algorithms. The simulation results of semantic networks are satisfactory when the service discovery domain is known to every node. However, it requires complex algorithms to execute and may result in slow response to the service requests. On the other hand, embedding discovery on routing, service discovery is strictly bounded to the routing layer capabilities. The best method is decoupling service discovery and routing layer without causing extra messaging overhead between two. In other words, the two components should be in the same layer with loose coupling.

3.3 DMAC Algorithm

DMAC algorithm is a well-known algorithm for clustering ad hoc networks. It is simple to implement and it has flexibility property to adapt new situations. Furthermore, it does not assume stability on the cluster set up. Non-stability assumption is an important feature for mobile nodes since initialization phase affect the whole cluster structures, cluster numbers and further communication. It has six important procedures triggered in particular situations. Pseudo-codes of the methods will be explained in this thesis. Further information about the algorithm can be found in [21]. Before explaining the methods, auxiliary methods and predicates should be presented as a preliminary information.

- *Clusterhead*: variable where each node stores cluster information it belongs to.
- *Ch(u)*: method returning true if *u* is a clusterhead, false if *u* is an ordinary node.
- *Cluster(u)*: the set of ordinary nodes whose clusterhead is *u*

Every node acts as an ordinary node when it powers up. Hence, *Clusterhead* variable is null, *Ch(u)* is false, *Cluster(u)* is empty.

3.3.1 Initialization Method

Since simulation environment does not contain physical radio signals, the nodes periodically send hello messages to inform its neighbors that it is alive and active in the network. When a node newly joins the network, it does not know its own role. To decide it, it has to get hello messages from its neighbors. A hello message contains the weight of the sender node. If node receives at least one hello message from any neighbor, it immediately executes initialization method to decide its own role. It may not sound logical to execute cluster initialization on the first hello message. However, original DMAC implementation executes it in this way. In fact, the way it executes cluster initialization should be enhanced.

```
PROCEDURE Init;
begin
   if {z ∈ Γ(v) : Ch(z)} ≠ ∅
      then begin
              x := max_{w_z > w_v} {z : Ch(z)};
              send JOIN(v,x);
              Clusterhead := x
           end
      else begin
              send CH(v);
              Ch(v) := true;
              Clusterhead := v;
              Cluster(v) := {v};
              if |{z ∈ Γ(v) : Ch(z)}| > k then send RESIGN(min_k{w_z : z ∈ Γ(v) ∧ Ch(z)})
           end
end;
```

Figure 1 - Initialization Method

Briefly, the node checks the neighbor list to see if there exists any head node. If such a node exists, the node becomes an ordinary node and sends its join request to related head node. Otherwise, the node announces itself as head and checks the count of the cluster head neighbors in its neighbor list. If the count exceeds a predefined threshold, the head with minimum weight exceeding the threshold is found. A resign message is send to the network to force head nodes to give up head role and become an ordinary node. Since the mentioned action will also force the

ordinary nodes belonging to resigning head, they will have to find a new head. Hence, resign messages may invalidate current cluster structures and cause many ordinary nodes to take additional action.

3.3.2 Link Failure Method

In the link failure situation, there might be two different cases:

- Executing node is a head and failing node belongs to executor's cluster.
- Executing node is an ordinary node and failing node is executor's head.

```
PROCEDURE Link_failure(u);
begin
  if Ch(v) and (u ∈ Cluster(v))
     then Cluster(v) := Cluster(v) \ {u}
    else if Clusterhead = u then
           if {z ∈ Γ(v) : Ch(z)} ≠ ∅
              then begin
                     x := max_{w_z > w_v}{z : Ch(z)};
                     send JOIN(v,x);
                     Clusterhead := x
                   end
              else begin
                     send CH(v);
                     Ch(v) := true;
                     Clusterhead := v;
                     Cluster(v) := {v};
                     if |{z ∈ Γ(v) : Ch(z)}| > k then
                        send RESIGN(min_k{w_z : z ∈ Γ(v) ∧ Ch(z)})
                     end
end;
```

Figure 2 - Link Failure Method

In the first case, the head node immediately removes failing node from the cluster. In the second case, the ordinary node has to find a new head to join its cluster. Mentioned method may result in invalidating the current cluster architecture. If the failing node is a head node and executor is an ordinary node, it has to find a new head or announce itself as a clusterhead. An immediate check is executed to figure out if number of total heads exceed the predefined threshold value.

3.3.3 Add New Link Method

```
PROCEDURE New_link(u);
begin
   if Ch(u) then
      if (w_u > w_Clusterhead + h)
         then begin
                 send JOIN(v,u);
                 Clusterhead := u;
                 if Ch(v) then Ch(v) := false
              end
         else if Ch(v) and |{z ∈ Γ(v) : Ch(z)}| > k then
                 begin
                    w := min_k{w_z : z ∈ Γ(v) ∧ Ch(z)};
                    if w_v > w then send RESIGN(w)
                               else begin
                                       x := max_{w_z > w_v}{z : Ch(z)};
                                       send JOIN(v,x);
                                       Clusterhead := x;
                                       Ch(v) := false
                                    end
                 end
end;
```

Figure 3 - Add New Link Method

The new link method, as clear as its name, executed whenever the executor node receives hello messages from neighbor nodes. If this new neighbor is a head node, executor node checks whether its weight is greater than the current head to which this executor belongs. If this is the case, the executor node joins to the cluster of new head and moves into an ordinary role.

If the executor is a head node and it has more than k head node neighbors, it decides the node weight to resign. The node to resign might be in two cases:

- The executor head node might have to resign.
- Another head will have to resign.

If the first one is the case, the head node have to be an ordinary node and find a new head for it self to join its cluster. In the case, an ordinary resign is executed in the nodes having weight less than the threshold value.

3.3.4 Clusterhead Receive Method

The cluster head receive method is triggered when a node gets a clusterhead message from a head node when that node announces itself as a head node. Regardless of being a head node, the executor node checks if the weight of sender head is greater than the weight of head node that the executor currently belongs to. If this is the case, the executor node immediately leaves the current head and joins to new head.

If the executor node is a head node; the set of clusterheads is checked to figure out if the number of clusterheads exceed the predefined value. In this case, the head node with minimum weight is found and a resign message is generated including the weight of the mentioned head node. Any head node having weight smaller than the minimum weight extracted in previous step has to give up clusterhead node when resign message is received.

```
On receiving CH(u);
begin
   if (w_u > w_Clusterhead + h) then begin
                send JOIN(v,u);
                Clusterhead := u;
                if Ch(v) then Ch(v) := false
             end
      else if Ch(v) and |{z ∈ Γ(v) : Ch(z)}| > k then
                begin
                   w := min_k{w_z : z ∈ Γ(v) ∧ Ch(z)};
                   if w_v > w then send RESIGN(w)
                                 else begin
                                        x := max_{w_z > w_v}{z : Ch(z)};
                                        send JOIN(v,x);
                                        Clusterhead := x;
                                        Ch(v) := false
                                      end
                end
end;
```

Figure 4 - Clusterhead Receive Method

3.3.5 Join Receive Method

Mentioned method has two parameters. It contains both joining ordinary node and the head node issue to the join action. Apart from the two nodes, every node receives the message that node is joining to the related head. The reason behind the idea is informing all nodes for join event since previous head has to remove this ordinary node.

```
On receiving JOIN(u, z);
begin
    if Ch(v)
        then if z = v then Cluster(v) := Cluster(v) ∪ {u}
                      else if u ∈ Cluster(v) then Cluster(v) := Cluster(v) \ {u}
        else if Clusterhead = u then
                  if {z ∈ Γ(v) : w_z > w_v  ∧  Ch(z)} ≠ ∅
                    then begin
                           x := max_{w_z > w_v} {z : Ch(z)};
                           send JOIN(v,x);
                           Clusterhead := x
                         end
                    else begin
                           send CH(v);
                           Ch(v) := true;
                           Clusterhead := v;
                           Cluster(v) := {v};
                           if |{z ∈ Γ(v) : Ch(z)}| > k then
                               send RESIGN(min_k{w_z : z ∈ Γ(v)∧ Ch(z)})
                         end
end;
```

Figure 5 - Join Receive Method

If the receiving node is a head and the sender is willing to join it, it immediately adds the sender node to the cluster member list. If the joining node is in its cluster and joining to another head, current head removes it from the cluster member list. For another case, a head node may prefer to leave its head role and join to another cluster. (This may occur when head node receives a resign message.) The nodes being informed that its cluster head is leaving its role have to find a new cluster to join. If ordinary nodes can not find a head to join, it simply announces cluster head.

3.3.6 Resign Receive Method

When a node receives a resign message and it is a cluster head, it checks its weight with the threshold weight. If it has a weight smaller than threshold, it finds a new head to join and moves in to ordinary node role. As it is obvious, the procedures contain many resign messages which will result in dramatic cluster reorganization.

The idea behind the design of "resign message" is permitting head nodes to be directly connected to each other. Although this is permitted, it is also restricted by a predefined threshold value. As a result, to guarantee that a head node is one-hop neighbor to k head nodes; such message is sent if the number of head nodes exceed k. Receiving head node immediately resigns if it has weight smaller than the threshold. By the mentioned method, the head nodes having higher weights remain in cluster head set of total network and head nodes having weak weights are forced to give up that role.

```
On receiving RESIGN(w);
begin
   if Ch(v) and w_v ≤ w then begin
                                  x := max_{w_z > w_v}{z : Ch(z)};
                                  send JOIN(v,x);
                                  Clusterhead := x;
                                  Ch(v) := false
                               end
end;
```

Figure 6 - Resign Receive Method

3.4 Service Discovery Algorithm

The service discovery algorithm implemented in this thesis operates over a clustered network structure. Before introducing the service discovery algorithm in details, it is appropriate to give the preliminary definitions.

3.4.1 Definitions Used in Service Discovery Algorithm

Service: The word *service* is used in a wide range in this study. Any source residing in a node that is published to the network nodes is named as a "service". In the simulation environment, all services described in XML format with following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<services>
      <service>
            <id>1</id>
            <name>printing</name>
            <keywords>
                  <keyword>color</keyword>
                  <keyword>print</keyword>
            </keywords>
      </service>
      <service>
            <id>2</id>
            <name>memberFile</name>
            <keywords>
                  <keyword>employer</keyword>
                  <keyword>member</keyword>
            </keywords>
      </service>
      ....

      ....
<services>
```

Each service is owned by a node when the simulation sets up. A node might serve one or more services at the same time. The id property is preserved for simulator usage. The name and keyword set are used to query a service in the network.

Service Weight: In the algorithm, each node has a "*service weight*" which is increased on each service hit in the node. The service weight definition has an important role since it plays a connector bridge role between the clustering algorithm and the service discovery algorithm. Detailed usage of service weight can be found at section 3.5.2

BorderH: After clustering set up or in topology changes, the ordinary nodes sends the number of heads it is directly connected to its own clusterhead. This value is described as "*BorderH*" value.

BorderB: Similar to BorderH, the value is the count of ordinary neighbor nodes belonging to different clusters.

Intra-cluster Search: The service query process that is performed by the clusterhead in service repositories of cluster members.

Inter-cluster Search: The service query process that is performed in many clusters that is trigered by one clusterhead sending the query message to other clusterheads.

PhyToServiceWeight Ratio: Since the service popularity is projected as service weight and it is used to determine the total weight of the node, a ratio between two weight
value is needed.

Service Hit Increment: The service weight increment value if a service hit occurs in any node.

Service One-Hop Increment: The service weight increment value used in one-hop neighbor of the service provider node.

3.4.2 Service Discovery Algorithm Search Patterns

In Figure 7, a simple clustered network is shown. In the cluster, the clusterheads are presented by tag "H_i" where i is the head number. Similar to the head nodes, the ordinary nodes are presented by tag "B_{ij}" where i is the clusterhead number the node belongs to and j is number showing the node number in the related cluster.

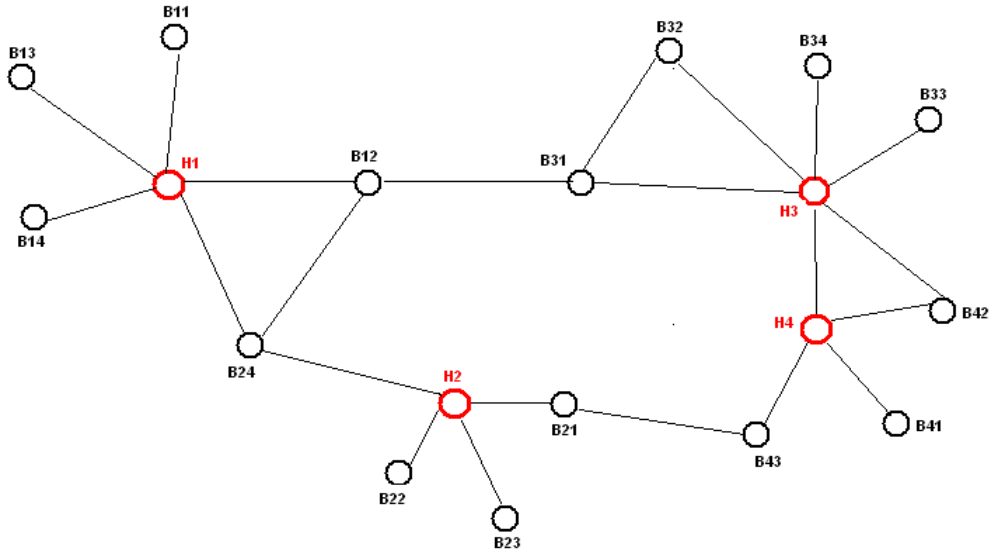

Figure 7 – A simple clustered network

To discover a service, four different hop patterns can be traversed. The patterns depend on reaching a clusterhead of a cluster. Mentioned patterns can be listed as following:

H1-H2:

In this pattern, a clusterhead is directly connected to another clusterhead. The example of mentioned pattern is H3-H4 connection in Figure 7.

·H1-B2-H2:

In this pattern, a normal node B2 connects two clusterheads and ordinary node belongs to H2. The example of the pattern in Figure 7 is H1-B24-H2 connection.

·H1-B1-H2:

The pattern is similar to previous pattern including three hops. Although it seems similar to H1-B2-H2 pattern, it has different usage in service discovery algorithm. The example of given pattern in Figure 7 is H2-B24-H1 connection.

·H1-B1-B2-H2:

This pattern is the longest pattern in the network that could be used to reach to another clusterhead. Two heads are connected by two ordinary nodes each belonging to one of the two clusters. The example of the pattern in Figure 7 is H1-B21-B43-H4 connection.

3.4.3 Core of the Service Discovery Algorithm

In the clustered network structure, a clusterhead caches the services provided by each cluster member. Any service request from a cluster member is first queried in cluster repository. This search is named as "*intra-cluster search*". Any service hit is immediate response is returned to the requester including the unique node number.

If the requested service can not be found by intra-cluster search, a forwarding list depending on previously mentioned search patterns is constructed. The patterns have priority one over other. H1-H2 is the most valuable pattern since it enables one clusterhead to reach the other clusterhead in absolutely **one hop**. As a result, a clusterhead checks whether it has another clusterhead among neighbors. In the second step, H1-B2-H2 pattern is checked. To find this pattern, a clusterhead checks whether any node belonging to another head is in the neighborhood by looking at BorderH values. In the next step, H1-B1-H2 pattern is searched by checking any cluster member has reported that it has clusterhead neighbor.

Although H1-B2-H2 and H1-B1-H2 consist of three hops, former one has priority over other. The reason is that each node on the pattern checks its local service repository before forwarding the message to the related head. In this situation, the local search of B1 in pattern H1-B1-H2 is ambiguous since the H1 has cached the services of B1 and queried them before starting an inter-cluster search. But in pattern H1-B2-H2, H1 forwards the query message to B2 and B2 has to start a local search before forwarding the message to its own head. If node B2 has a service hit, it will immediately prepare a response message without forwarding the query to clusterhead H2. Thus, the query will result in two hops instead of three hops query.

As a final step, H1-B1-B2-H2 pattern is searched among the BorderB values of cluster members.

Once the forward list is constructed, patterns are tried according to their priority starting with H1-H2 pattern in serial manner. In another words, the result of pattern *i* is waited before forwarding the query with pattern (*i+1*) although clustering algorithm and infrastructure support parallel query pattern. The idea behind this decision is decreasing message count and message density in unit time in order to prevent collision, packet drop and high traffic.

For patterns H1-B1-H2 and H1-B1-B2-H2, a forwarding list is constructed by node B1 since it may have different clusterhead in pattern H1-B1-H2 and many ordinary nodes belonging to different clusterheads in H1-B1-B2-H2. The response message of B1 is prepared when all queries in constructed list is done or when the service hit is found among the forwarding patterns. In H1-B2-H2 and H1-B1-B2-H2 pattern, in contrast to behavior of B1, the node B2 forwards the query to H2 instead of preparing a forwarding list. Since patterns can be used recursively in each clusterhead, a vector data structure which holds the clusterheads visited during the query traversal is also carried in the forward message as an attachment in order to prevent duplicate cluster searches for same service request.

When the service hit is achieved in an inter-cluster search, each node on the way back to the requesting node is pushed into a vector starting from the provider node. Thus, the provider also gets a path to the provider for further communication in addition to solely providing node information. Furthermore, if the service hit is achieved in head node by an intra-cluster search, a message is sent to the provider node in order to inform it that it will have a requester communication. Apart from informing the node, receiving provider node also increases its service weight after getting mentioned message.

3.5 Improved DMAC Algorithm

Message (or network traffic) density, which can be described as average message sent in unit time, is an important measurement in wireless networks since high message density will interrupt continuous communication between nodes. As a rule of thumb, the improvement philosophy in DMAC and design philosophy in service discovery algorithm depend on decreasing the message density to enable service discovery and further service communication to be fast and continuous.

In original DMAC algorithm, when the network is set up, many nodes may announce itself as a clusterhead since at the beginning there does not exist any available clusterhead. Thus, the situation causes many false clusterheads which will join to another clusterhead in a short period of time. As a result, many cluster reorganization occurs in the network which invalidates current stable working clusters. For any other algorithm that will operate on DMAC, the situation may not cause any problem; but for service discovery algorithm, it is an obvious matter. To solve the problem, the initialization method and the link failure methods -which is triggered in ordinary nodes after a head failure is detected- is enhanced.

The *init()* method in original DMAC algorithm forces the ordinary node to join an existing cluster. If there does not exist a clusterhead, the node announces itself as a clusterhead. In the improved algorithm, an ordinary node finds the maximum weighted node. If this node is a clusterhead having weight greater than its weight, the ordinary node joins to the existing clusterhead. Otherwise, there are two other options. First one is the executor node having the maximum weight among neighbors. The situation results in executor node be a clusterhead. In the latter one, the node has neither has a clusterhead around nor it has the maximum weight among neighbors. At this point, the node does not finalize the *init()* procedure and waits a clusterhead message from maximum weighted neighbor having no clusterhead. In theory, the wait situation may lead to an “undefined role” syndrome among nodes if the weight of the nodes increase / decrease regularly from one side to other side in

network layout. But random experimental results with 30 and 50 nodes show that none of the nodes stay in an undefined role. They are either in head role or ordinary node role.

link_failure() method is also improved to decrease cluster reorganization or false clusterhead situation. When a head failure is detected, the members of the same clusterhead are marked as headless in each node. Every node checks any available head is around. If any head with greater weight can not be obtained, the node with maximum weight among others becomes a clusterhead. During this process, any ordinary node having a neighbor with a better weight waits the neighbor to announce itself as a clusterhead. If the candidate head joins to another head, re-election is done among remaining nodes. The procedure is similar to *init()* method.

In original DMAC algorithm, apart from *init()* and *new_link()* methods, the changes in weight of the nodes are not sensed by the algorithm which may result in inadequate clusterheads. In the improved algorithm, the dynamic changes in the weight of nodes are detected and an ordinary node checks whether it is an adequate node for being a clusterhead. If such situation occurs, the node immediately announces itself as a clusterhead. The "service weight" property of service discovery algorithm connects both algorithm and it is considered as another parameter while calculating the total weight of the nodes. It may change in two situations:

- A node may have a service hit informed by the message received from clusterhead or a node may realize that a service hit is occurred in local search while forwarding the search pattern.
- If the node is just one hop to the provider node and resulting path is passing over it, the service weight of gate node also increases.

The service weight increase in two scenarios are different from each other. The increment in first scenario is expected to be greater than the second one but since it is a parametric value, it could be changed according to the needs of the network.

Simulation result section can be reviewed for results of the experiments done by changing such parametric values.

In the second scenario, the one-hop neighbor of the provider node deletes the head node from path vector in order to shorten the path and free the head node being on a service path. The decision behind the idea is relieving the duty of the head nodes. Since a head node in the cluster has duties such as caching the cluster services, performing the intra-cluster search, preparing inter-cluster search structures and managing the inter-cluster search. As a result, the power consumption in head nodes is obviously higher than normal nodes. To aid the clusterheads and to shorten the path, one hop neighbor of the provider node discards the head node from the path vector if it is already passing over itself. Thus, the head nodes are kept out of the service paths as much as possible.

After a service weight increment either due to first or scenario, the node immeaditely check if its weight exceeds the weight of its current head. If the mentioned situation occurs, a reorganization in the cluster is performed. Improved versions of aforementioned method are as following:

procedure init()

```
begin
      max = MAX(neighbors)
      if (max is nil) {
            clusterhead:=me
            initCompletedFlag:=1
      }elseif (totalWeight(me) > totalWeight(max))
      begin
            clusterhead:=me
            initCompletedFlag:=1
      end elseif (totalWeight(max) > totalWeight(me) and ch(max))
      begin
            join(max)
            initCompletedFlag:=1
      end else
            initCompletedFlag:=2

end
```

```
procedure determineNewHead()
begin
      max = MAX_HEAD(neighbors)
      if (max not nil and totalWeight(max) > totalWeight(me))
            join(max)
      else begin
            max = MAX(neighbors)
            if (max is nil)   begin
                  clusterhead:=me
                  determinationFlag:=1
            end
            elseif (totalWeight(max) < totalWeight(me)) begin
                  clusterhead:=me
                  determinationFlag:=1
            end
            else
                  determinationFlag:=2
      end
end
```

CHAPTER 4

SIMULATIONS

In this chapter, various simulation environments are briefly introduced. Reasons behind selecting OMNET++ environment and features of Mixim are explored. After the introduction of simulation environment; the simulation parameters used during the tests, the decision behind selecting mentioned parameters, the experiments run for the implemented solution and the discussion about the experiment results are presented.

4.1 Simulation Environments

Before presenting deep information about OMNET++ simulation environment, a brief research is done about simulation environments. Key points and explanations are presented at Table 1. Being open source or at least being free, Windows compatibility, being user friendly and strong wireless networking support are the important key features while selecting the simulation environment.

Table 1 - Simulation Environments

Simulator	Brief Description	Wind. Comp.	Linux Comp	Free	Special Note
Cnet [31]	Developed at University of Australia, event driven approach, for simulating point-to-point and 802.3 Ethernet	No	Yes	Open source, For academic use	
GloMoSim [32]	Developed at UCLA, parallel discrete-event simulation for wired &	Yes	Yes	Academic usage, restricted	

Table-1 Continued

GloMoSim [32]	wireless networks, supporting CSMA and 802.11	Yes	Yes	university distributio n	
GTNetS [30]	Developed By GeorgiaTec h, mostly for TCP/IP based networks, C ++ based.	Yes (Prerequisite : Visual Studio 2005)	Yes	Open source	Claiming 802.11 support but manual does not contain description (at [30], page 115)
NCTuns [33]	Developed by Network and System Laboratory at NCTU. For IP based wired and wireless networks. IEEE 802.3 CSMA/CD MAC, IEEE 802.11b support	No	Yes (restriction: highly dependent to fedora kernel)	Free	
NetSim [34]	Developed by Tetcoc. IEEE 802.3, 802.4, 802.5, 802.11b	Yes	Yes	Commerci al	Only academic demo available
	Developed by University of Southern California, C++ based, discrete event sim.	Yes	Yes	Free	Very user unfriendly, although open source, difficult to add /change modules

Table-1 Continued

OMNeT++ [36]	Developed by OMNeT ++ Community Site. **component-based, modular and open-architecture,** C++ based simulation environment	Yes	Yes	Free, open source	Eclipse based IDE, strong and user friendly GUI and interactive simulation support, various wireless network simulation modules
OPNET [37]	Developed by OPNET Technologies. For design and study communication networks,devices, protocols, and applications	Yes	Yes	Commercial	formal registration is required for academic use
PARSEC [38]	Developed by UCLA Parallel Computing Lab. C-based simulation environment for sequential and parallel execution of discrete event simulation.	Yes (Prerequisite : Visual Studio 2005)	Yes	Free, open source	Does not contain clear information about wireless network support

OMNET++ simulation environment is started to be developed at the end of 1990s. It is C++ based, component oriented, modular simulation environment and it has Eclipse based IDE. For researchers who has a knowledge of C++, Java and Object oriented view of understanding, simulation environment is very easy to adapt. Unlike ns2, it has very user friendly GUI, simple set up directions for both Linux-based and Windows based platforms. It is open source, hence it is very easy to implement own auxiliary code and build the core. Furthermore, there are various wireless and mobile adhoc networking simulation modules. Some of them can be listed as following:

- INET Framework
- INETMANET Framework
- MF
- Castalia
- Mixim Framework

Detailed information about first four framework can be found at [36]. The fifth one is the mobility module used in this thesis. It uses OMNET simulation engine and adds additional features to simulate mobile and adhoc networks. The Mixim framework merges ChSim, MacSimulator, Mobility Framework and Positive Framework.

In Mixim, a network is modeled in layers. At the bottom, Nic layer resides and it combines MAC and physical modules. On the next layer, network module is implemented. Finally, application layer is mounted on network layer. Additional to basic modules; arp, mobility and battery module are combined for specific implementations. Since this thesis focuses on mobility, various mobility patterns are tested against clustering algorithms.

Apart from pre-implemented mobility modules such as LinearMobility, MassMobility, BonnMotionMobility, TurtleMobility, it is easy to implement specific mobility modules extending from BaseMobility module which defines base

mobility features for extender classes. Like in mobility module case, the other modules such as mac layer, application layer and battery modules have base classes defining features of mentioned layers and enabling developers to extend Mixim framework easily for their own simulation purposes. On the run time, ConnectionManager and -simulation- World modules manages interaction between aforementioned layers and modules. To give a general view of Mixim and introduce how this thesis benefits from modules, a brief presentation is done in thesis borders. Detailed structural and functional information about Mixim can be found at [39].

Furthermore, as a parallel understanding of OMNET++, Mixim Framework has a good user interface for simulations. Figure 8 shows a start up screen shot for 10 nodes in 400x400 unit playground. To collect statistics, OMNET can log scalar entities, histograms, vectors and errors. Scalar and vector entities further can be used in statistical graphing. The IDE has a basic and powerful tool for drawing graphs, bar charts and collaborating data sets. The common parameters and input parameters for modules can be set either embedding into code or from omnetpp.ini files.

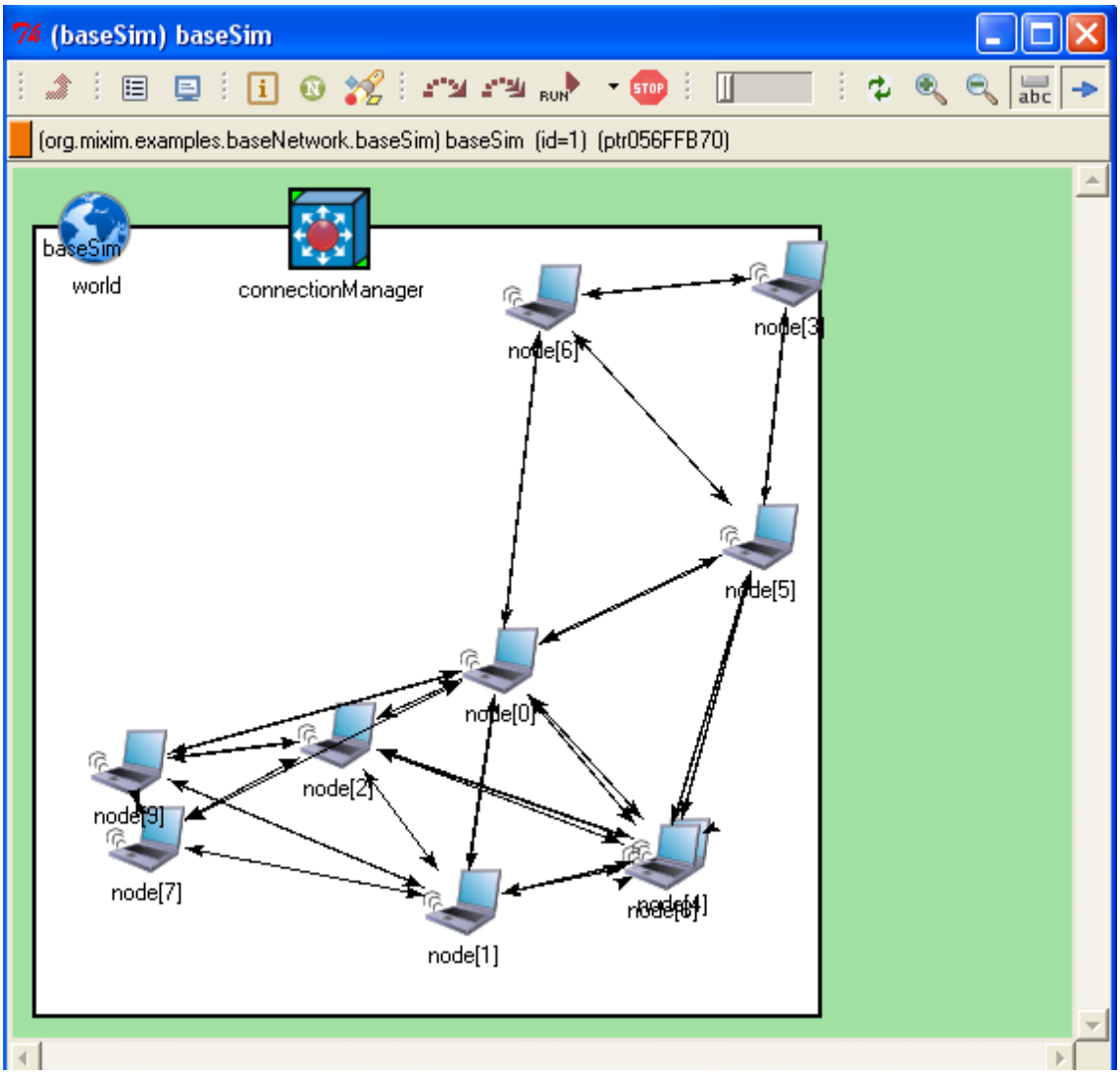

Figure 8 Simulation Screenshot

4.2 Simulations

In this section, results of simulation are presented. Before exploring results, it is appropriate to examine simulation parameters. Table-2 shows the mentioned values. A Cartesian product is done for parameters having multiple values. Thus, for one main scenario, (P to S) X (H to O) Cartesian results in 4 different sub-scenarios under main scenario.

Table 2 – Simulation Parameters

Playground	1200meters X 660meters
Node count	24
Speed (Spd)	double_uniform(1,2) mps
Angle	int_uniform(0-360) degrees
Service Provider Node Count	5
Service Requester Node Count	20
Request Interval	4 seconds
Ratio of Physical Weight to Service Weight (P to S)	(20% to 80%), (80% to 20%)
Service Hit Weight Increment to Service One-hop Increment (H to O)	(200-100), (100-200)
Simulation Duration	120 seconds

To examine the impacts of the improvements on DMAC algorithm and the performance of the service discovery algorithm, both the improved and original DMAC algorithm are tested against all scenarios and the arguments listed below are measured:

- Average hop count for total try (HTT)
- Average hop count for hit try (HHT)
- Total join event count (JEC)
- Total clusterhead event count (CEC)

To analyze the behavior of both algorithms, three different main scenarios are run. In the first scenario, the head nodes appear in the end point of the network and all other normal nodes belonging to clusters appear in random scenarios. In the second one, the head nodes mostly appear in middle points with cluster members around. The third scenario is a free scenario where speed of the nodes, the service request intervals, the provider and requester nodes are randomly distributed.

4.2.1 Experiments and Results

In the first scenario, head nodes appear at the end points after cluster initialization. The normal cluster members reside in center of the network. In the initial cluster set up, 4-5 clusters are formed. There does not exist any cluster fully disconnected from the rest of the network. The maximum hop length is 6-7 from left to right, 3-4 from up to down. Each head node owns 3,4 or 5 cluster members around. Table 3 shows the result for both algorithms. To examine the impact of service weight on total weight, first and second sub-experiments are run under 80% service weight. To remove the effect of service weight, the ratio is reversed.

The third and fourth sub-scenarios are run to examine the behavior of the implementation relatively having less service weight. Similar to the service weight and physical weight ratios; the promotion points of service provider and one-hop neighbor of provider is 200 and 100 respectively. To figure out the effect of such promoting, the promotion points are reversed as executed in case of physical and service weight. In the simulation environment, the total weight of a node may vary between 1 and 999. The promotion point is 10% or 20% of maximum available total weight which is 999.

Table 3 – Results of Experiment 1

Experiment -1		Heads appear in the endpoints							
		DMAC + Service Discovery				**Improved DMAC + Service Discovery**			
PtoS	HtoO	HTT	HHT	JEC	CEC	HTT	HHT	JEC	CEC
20-80	200-100	21.3	7.0	55	17	17.3	6.2	37	10
20-80	100-200	21.3	7.0	55	17	15.8	5.8	41	11
80-20	200-100	21.3	7.0	55	17	19.4	6.6	25	7
80-20	100-200	21.3	7.0	55	17	19.4	6.6	25	7

In the second scenario, head nodes appear mostly at the middle points after cluster initialization. The members of the clusters surround the head nodes. The idea behind the scenario is pre-executing the head shift operation which is executed in first experiment by the simulation code. In main scenario of this experiment, the task that would be accomplished by the simulation is already defined in the set up. In other words, the gain that would be obtained by simulation is given in the initialization.

Although significant improvement is not expected, the experiment is run to examine if the solution invalidates any acceptable cluster organization in the network. In other words, the experiment is run in order to check that the solution keeps the network in stable situation if the initial set up is already stable. If the case could not have been verified, additional improvements should have been designed on the solution. Similar to first experiment, the impact of service weight and physical weight on total weight are examined by reversing the ratios. In addition to weight ratio, promotion points for service provider and one hop neighbor of provider are also reversed in sub-scenarios. Table 4 shows the result for both algorithms.

Table 4 – Results of Experiment 2

Experiment 2		Heads appear in the middle points							
		DMAC + Service Discovery				**Improved DMAC + Service Discovery**			
PtoS	HtoO	HTT	HHT	JEC	CEC	HTT	HHT	JEC	CEC
20-80	200-100	15.9	5.6	47	13	15.2	5.2	32	6
20-80	100-200	15.9	5.6	47	13	15.2	5.2	33	6
80-20	200-100	15.9	5.6	47	13	15.2	5.2	31	7
80-20	100-200	15.9	5.6	47	13	15.2	5.2	35	6

The third scenario is a free scenario where speed of the nodes, the service request intervals, the provider and requester nodes are randomly distributed. Apart from the previous scenarios, this scenario is run 7200 seconds with 50 nodes, producing service requests in interval uniformly distributed between 2 and 4 seconds. The

positions of the head nodes and member nodes are not predefined. In other words, some head nodes appear in end points whereas some of them appear in middle points. Similar to other parameters, the position of head nodes are also random.

The idea behind the experiment is to examine the behavior of the solution under random parameters. In addition to random value tests, the solution is also tested against the heavy load of service request in long run time. By the help of the last experiment, it is also examined if the solution executes in a stable manner in long execution time. Te experiment result show that improved DMAC can still execute with less network reorganization and shorter paths and shorter traversal hops. Table 5 shows the result for both algorithms.

Table 5 – Results of Experiment 3

Experiment 3		Random head occurrence							
		DMAC + Service Discovery				**Improved DMAC + Service Discovery**			
PtoS	HtoO	HTT	HHT	JEC	CEC	HTT	HHT	JEC	CEC
20-80	200-100	38.1	15.6	7910	1546	17.8	12.3	7091	1128
20-80	100-200	38.1	15.6	7910	1546	19.1	13	7187	1206
80-20	200-100	38.1	15.6	7910	1546	20.5	13.7	7003	1098
80-20	100-200	38.1	15.6	7910	1546	20.6	13.9	7103	1001

4.3 Observations and Analysis of Experiment Results

In all scenarios, it is obvious that JEC, CEC values of improved DMAC algorithm are lower than JEC, CEC values original DMAC algorithm. The reason behind the difference of values between two algorithm is the improvements done in the second algorithm. The details of the improvements can be found at chapter 3 in section 3.5

In the first experiment, where heads appear at endpoints, the improved algorithm senses the service weight changes in provider nodes or in one-hop neighbor of

provider nodes and after a few hits in same nodes, the head of the cluster is shifted from endpoint to relatively middle points by clusterhead announcement of one-hop neighbor of provider nodes. Thus, the provider node which was acting as a gateway to connect the its cluster to the remaining network, now becomes a clusterhead. As being the new clusterhead of the cluster, it is able to query the cluster members when a query is forwarded and immediately return the answer of the query. Since the query message travels less hops, average query hop (HTT) and hit hop (HHT) of the improved algorithm are shorter than the original algorithm.

When the numbers of the first algorithm are compared to itself in sub-scenarios of first experiment, it is observed that there has not been any change. This is an expected result since the original DMAC algorithm does not senses increase or decrease in service weights. On the other hand, in the improved algorithm, the service weight increases when service hit occurs in nodes and the increase is reflected to the total weight of the nodes resulting new head announcements.

In the first and second sub-scenario of experiment 1, the ratio of the service weight is extremely higher than the ratio of physical weight. The impact of service hit weight increment and service one-hop increment come on the seen very fast. The impact can be understood from the increment in CEC and JEC values. Since the service weight increases, new head announcements occur in the network resulting the CEC and JEC value to rise.

In the third and fourth sub-scenario of experiment 1, the ratio of the physical weight is extremely higher than the ratio of service weight. As a result, the head nodes can not be shifted to the middle point although the service weights increases in nodes. The HTT and HHT values of reflect the behavior of the situation. Both values are higher than the first and second sub-scenario results.

The results of experiment 2 shows that improved algorithm only shifts one or two head node rather than all head nodes since the clusterheads are already in the gateway role. Despite a few shifts, HTT and HHT values are still better than the

original algorithm. The constant values of original algorithm is another expected result as in experiment 1.

In the final experiment, in contrast to situation in first experiment, the service provider nodes are promoted instead of one hop neighbors of the providers. This is easily understood from HTT and HHT values of first two sub-scenarios. The result is an expected one in the scope of this study since one-hop neighbors of provider or provider itself is promoted during service hits.

On the overall view, the results of the experiments are in the expected scope of the thesis study.

CHAPTER 5

CONCLUSION and FUTURE WORK

As stated in abstract and introduction section, service discovery in mobile and adhoc networks is cumbersome and painful process due to the mobility. To accomplish mentioned task, numerous studies and many protocols have been done in the field. Each study is worked in a restricted point of view and accomplishes a particular task solving a restricted set of problem.

The view point in this thesis rely on preventing flooding or flooding-like messaging among nodes to enable the network communicate in continuous manner. Since the previous studies show that a heavy caching does not provide expected service hit rates, a lightweight caching mechanism is used.

To provide a lightweight infrastructure for non-centralized and infra-structureless MANET architectures; improved version of DMAC, which depends on solely one-hop neighbor information, highly adaptive to mobility and could reorganize the network in architecture changes is implemented. On the mentioned clustered structure, a flexible service discovery algorithm, in which simple forwarding patterns are used with priority options is integrated. To complete the service discovery algorithm, lightweight caching mechanism which caching is done only in clusterheads is integrated.

In order to enable the service discovery algorithm and DMAC algorithm communicate each other, service popularity is added as a weight parameter. Thus, the clustering algorithm adapts the changes in service discovery algorithm and supports it by shifting the clusterhead role to appropriate nodes resulting in shorter service hit paths and shorter message hop travels.

The service request where the requester lefts the cluster is also recovered up to 2 hops to support mobility in service discovery layer. The service requests which stays in hang situation due to mobility or cluster changes are scanned by a routine and one more request is done over the current clusterhead.

Unlike traditional wired networks, MANETs have many parameters and restrictions to accomplish mentioned tasks. It is difficult to fix all parameters to perform best performance since there exists numerous restrictions. A set of parameters are improved to enable mobile nodes to benefit from the network resources while considering the restrictions of the MANETs.

For future work about this thesis subject, particular improvements on caching can be studied to shorten the HTT and HHT values. In fact, full caching in which every node caches all request passing over itself does not solve the problem since path of the service may change due to the mobility. Instead, a study on figuring out the best performing cache threshold can be done to shorten aforementioned values and service response times.

As a second step, an estimation mechanism can be constructed in order to predict whether the reorganization of the clusters in accordance to the service discovery layer is worth trying it. Hence, total reorganizations in the network is decreased enabling the network to stay more stable.

REFERENCES

[1] Islam N. Shaikh Z.A., A novel approach to service discovery in mobile adhoc network, IEEE, 2008, pp. 58-62.

[2] Rownston A. Druschel P., Pastry: Scalable, decentralized object location and routing for large scale peer-to-peer systems, In Proceedings of IFIP/ACM Middleware 2001, Nov. 2001, pp. 329-350

[3] Ratnasamy S. Francis P. Handley M. Karp R. Schenker S., A Scalable Content – Addressible Network, In Proceedings of SIGCOMM, Aug. 2001, pp. 161-172

[4] JXTA home page, https://jxta.dev.java.net/, Last accessed on October 2009.

[5] Stoica I. Morris R. Karger D. Kaashoek M.F. Balakrkshnan H., Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, In Proceedings of SIGCOMM, Aug. 2001, pp. 1-14

[6] Zhao B.Y. Kubiatowicz J.D. Joseph A.D., Tapestry: A Resilient Global-scale Overlay for Service Deployment, IEEE Journal on Selected Areas in Communications, Vol.22, No.1, January. 2001, pp. 1-15,

[7] Guttman E. Perkins C. Veizades J. Day M., Service Location Protocol, Version 2, *IETF RFC 2608*, June 1999.

[8] Understanding UpnP: A White Paper, http://www.upnp.org/resources/whitepapers.asp, Last accessed on 7 October 2009.

[9] Introduction to Jini, http://www.jini.org/wiki/Category:Introduction_to_Jini, Last accessed on 6 October 2009.

[10] Bettstetter C. Renner C., A comparison of service discovery protocols and implementation of the service location protocol, In Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications, 2000.

[11] SDP document, www.jeyatheepan.com/btsms/documents/sdp.doc, Last accessed on 27 October 2009.

[12] Lee C. Helal S., Protocols for service discovery in dynamic and mobile networks, International Journal of Computer Research Volume 11, Number 1, 2002, pp. 1-12.

[13] Helal S. Desai N. Verma V. Lee C., Konark - A service discovery and delivery protocol for adhoc networks, IEEE, 2003, pp. 2107-2113.

[14] Kozat C. Tassiulas L., Service discovery in mobile ad hoc networks: an overall perspective on architectural choices and network layer support issues, Ad Hoc Networks 2, 2004, pp. 23-44.

[15] Klein M. König-Reis B. Obreiter P., Lanes – A lightweight overlay for service discovery in mobile ad hoc networks, 3rd Workshop on Applications and Services in Wireless Networks (ASWN2003), 2003, pp. 1-26.

[16] Klein M. König-Reis B., Multi-layer clusters in ad hoc networks – an approach to service discovery, Networking 2002 Workshops, LNCS 2376, 2002, pp. 187-201.

[17] Motegi S. Yoshihara K. Horiuchi H., Service discovery for wireless ad hoc networks, IEEE, 2002, pp. 232-236.

[18] Li L. Lamont L., A lightweight service discovery mechanism for mobile ad hoc pervasive environment using cross layer design, Proceedings of the 3rd International Conference on Pervasive Computing and Communications Workshops, IEEE, 2005, pp. 1-5.

[19] Frank C. Karl H., Consistency challenges of service discovery in mobile ad hoc networks, MSWIM, 2004, pp. 105-114.

[20] Engelstad P.E. Zheng Y., Evaluation of service discovery architectures for mobile ad hoc networks, Proceedings of the second annual conference on wireless on-demand network systems and services, IEEE, 2005, pp. 1-14.

[21] Basagni S., Distributed and mobility adaptive clustering for ad hoc networks, Technical report UTD / EE-02-98, 1998, pp. 1-14.

[22] Ramachandran L. Kapoor M. Sarkar A. Aggarwal A., DIAL M Workshop, 2000, pp. 54-63.

[23] Chatterjee M. Das S.K. Turgut D., WCA: A weighted clustering algorithm for mobile ad hoc networks, Cluster Computing 5, 2002, pp. 193-204.

[24] Leng S. Zhang L. Fu H. Yang J., A novel location-service protocol based on k-hop clustering for mobile ad hoc networks, IEEE transactions on vehicular technology, vol. 56, no. 2, March 2007, pp. 810-817.

[25] Zhang Y. Ng J. M. Low C. M., A distributed group mobility adaptive clustering algorithm for mobile ad hoc networks, Computer Communications 32, 2008, pp. 189-202

[26] Liang J.C. Chen J.C. Zhang T., Mobile service discovery protocol (MSDP) for mobile ad hoc networks, 8th International Symposium on Autonomous Decentralized Systems, 2007, pp. 1-8.

[27] Yuhan J. Hui T. Zemin L., Layered service discovery approach for MANETs, IEEE, 2008, pp. 1-4.

[28] Marin-Perianu R.S. Scholten J. Havinga P.J.M. Hartel P.H., Cluster-based service discovery for heterogeneous wireless sensor networks, International Journal of Parallel, Emergent and Distributed Systems 23:4, 2008, pp. 325-346.

[29] Islam N. Shaikh Z.A., A novel approach to service discovery in mobile ad hoc networks, Networking and Communications Conference, IEEE International, 2008, pp. 58-62.

[30] GTNETS Manual, http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/docs/GTNetS_manual.pdf, Last accessed on 15 December 2009.

[31] CNET home page, http://www.csse.uwa.edu.au/cnet/index.html, Last accessed on 15 December 2009.

[32]GloMoSimManual, http://pcl.cs.ucla.edu/projects/glomosim/GloMoSimManual.html, Last accessed on 15 December 2009.

[33] NCTUns manual, http://www.csie.nctu.edu.tw/~shieyuan/publications/AllChapter.pdf, Last accessed on 15 December 2009.

[34] Netsim manual, http://www.tetcos.com/brochure_av.pdf, Last accessed on 15 December 2009.

[35] NS2 home page, http://www.isi.edu/nsnam/ns/, Last accessed on 15 December 2009.

[36] OMNET++ home page, http://www.omnetpp.org/, Last accessed 16 December 2009.

[37] OPNET Academic page, http://www.opnet.com/university_program/research_with_opnet/, Last accessed on 10 December 2009.

[38] PARSEC home page, http://pcl.cs.ucla.edu/projects/parsec/, Last accessed on 16 December 2009.

[39] Köpke A. Swigulski M. Wessel K. Willkomm D. Klein H.P.T. Parker T.E.V. Visser O.W. Lichte H.S. Valentin S., Simulating Wireless and Mobile Networks in OMNeT++ : The MiXiM Vision, OMNeT++ 2008, March 2008, pp. 1-8.

Appendix A

PSEUDO-CODES of SERVICE DISCOVERY ALGORITHM

The following procedure is triggered when a head node receives service request from cluster members.

```
·procedure memberRequest (member, keyword)
begin
     service:=intraClusterSearch(keyword)
     if (service is nil)
     begin
         service:=interClusterSearch(keyword)
         sendReply(service, member)
      end
end
```

The following procedure is used by head nodes while searching the service repository of cluster members.

```
·procedure intraClusterSearch (keyword)
begin
       for all cluster_members do
      begin
         service:=localSearch(member,keyword)
      end
end
```

The following procedure is triggered when the requested service can not be found in repository of cluster members.

```
·procedure interClusterSearch(keyword)
begin
    add(me, visitedList);
    constructForwardSchema(keyword)
    forwardPattern:=getFirstUnforwardedPattern()
    sendMessage(keyword, forwardPattern.node)
end
```

The following procedure is used to built up a forwarding schema when an inter-cluster search will be started.

```
·procedure constructForwardSchema(keyword)
begin
for all neighbors do
            begin
                  if isHead(neighbor)
                      addToList(neighbor, H1_H2)
            end
            for all neighbors do
            begin
                  if (notIsHead(neighbor) and head(neighbor)<>me)
                      addToList(neighbor, H1_B2_H2)
            end
            for all cluster_members do
            begin
                  if (borderH(member)>0)
                     addToList(member, H1_B1_H2)
                  if (borderH(member)=0 and borderB(member)>0)
                     addToList(member, H1_B1_B2_H2)

end
```

The following procedure is used to forward a service request message to adequate neighbor.

```
·procedure forwardMessage(keyword, pattern, requester, visitedList)
begin
   service:=localSearch(keyword)
   if (service is not nil)
       replyMessage(service, requester)
   elseif (pattern=H1_B2_H2 and notVisited(head))
   begin
       add(head,visitedList)
       sendMessage(head, keyword, pattern,visitedList)
   end
   elseif  (pattern= H1_B1_B2_H2 and requester<>head)
   begin
      add(head,visitedList)
      sendMessage(head, keyword, pattern,visitedList)
   end
   elseif (pattern=H1_B1_B2_H2  and requester=head)
      begin
         list:=getBorderBNodes(me)
         node:=getUnvisitedNode(list.node.head);
          add(list.node.head, visitedList)
          sendMessage(node, keyword, pattern,visitedList)
      end
      elseif (pattern=H1_B1_H2  and requester=head)
```

```
      begin
          list:=getBorderHNodes(me)
          node:=getUnvisitedNode(list);
          add(node,visitedList)
           sendMessage(node, keyword, pattern,visitedList)
      end
end
```

The following procedure is used when any serach pattern can not find the requested service. From the previosly constructed forward list, a new serach pattern is selected.

```
·procedure replyMessageReceived(service, requester, sender)
begin
    if (service is nil)
      begin
         forwardPattern:=getFirstUnforwardedPattern()
         sendMessage(keyword, forwardPattern.node)
      end
    else
      sendMessage(requester, service)
end
```