MESSAGE COMPLEXITY ANALYSIS OF MOBILE AD HOC NETWORK (MANET) ADDRESS AUTOCONFIGURATION PROTOCOLS

By

SANG-CHUL KIM

Bachelor of Electrical Engineering and Computer Science
Kyungpook National University
Daegu, Korea
1994

Master of Science
Changwon National University
Changwon, Korea
1998

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 2005

MESSAGE COMPLEXITY ANALYSIS OF MOBILE AD

HOC NETWORK (MANET) ADDRESS

AUTOCONFIGURATION PROTOCOLS

Dissertation Approved:

PREFACE

This dissertation proposes a novel method to perform a quantitative analysis of message complexity and applies this method in comparing the message complexity among the mobile ad hoc network (MANET) address autoconfiguration protocols. The original publications on the address autoconfiguration protocols had many incomplete parts making them insufficient to use on practical MANETs. Therefore, the first objective of the executed research was to complete the address autoconfiguration protocols by filling in all the missing gaps to make them operational. The missing procedures that were filled in have been developed based on the most logical procedures being faithful to the original protocol publications. In this dissertation, to obtain the upper bound of the message complexity of the protocols, the *O*-notation of a MANET group of *N* nodes has been applied. To asymptotically calculate the total number of messages generated by a protocol's step or procedure, an investigation on the nodes broadcasting, unicasting, relaying, and receiving messages is conducted and used in obtaining the upper bound of the message complexity for each protocol.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1. Background

Wireless communication systems continue to show rapid growth as a result of significant advancements in digital communications, commercial laptop computers, and semiconductor technologies. The most popular networks of the traditional wireless model are cellular and mobile IP networks, which have been configured with a wired backbone, where the last hop is a wireless link, essentially a point-to-point wireless channel between the base station and the mobile user. In the wireless cell domain, the base station provides centralized control for the mobile users to access the medium. Some representative specifications are IS-54 (the first generation of the digital standard with TDMA technology), IS-95 (the standard for CDMA), GSM, cdma2000, and W-CDMA.

For the past few years, mobile ad hoc networks (MANETs) have been emphasized as an emerging research area due to the growing demands for mobile and pervasive computing, where the dynamic topology for the rapid deployment of independent mobile users becomes a new factor to be considered. For instance, mobile users across a campus can transmit data files to each other, group members of a search, disaster rescue, recovery team, or military solders in a battlefield can communicate in order to coordinate their actions, without using a base station. Especially, in battlefield circumstances, the infrastructure may not be built in advance for soldiers to communicate with each other. These example networks are called ad hoc wireless networks where centralized and organized connectivity can not be possible. The examples show that MANETs need to have the ability to provide for establishing survivable, efficient, dynamic communication for emergency, search-and-rescue operations, disaster relief efforts, law enforcement in crowd control and military networks. One of the outstanding features of MANETs could be the self-creating, self-administrating, self-organizing, and self-configuring multihop wireless network characteristic.

MANETs differ from conventional cellular networks because all links are wireless and the mobile users communicate with each other without using a base station. Several basic properties of MANETs are described below. An autonomous collection of mobile users composes a MANET, where they communicate over relatively bandwidth constrained wireless links. MANETs use peer-to-peer wireless connections, where the packets from a source node are transmitted via intermediate nodes called relay nodes towards a destination node. A MANET topology dynamically changes as mobile users join, leave, or rejoin the network. Sometimes, radio links in a MANET may not be usable due to the node mobility.

Most research and development (R&D) funding for MANET applications are for military applications for defense and security systems, where the Office of Naval Research (ONR) and the Defense Advanced Research Project Agency (DARPA) of the U.S. Department of Defense (DoD) have been leading the research in this area. Other MANET related applications are found in government-industry funded projects such as the R&D of Intelligent Transportation Systems (ITS). One important issue in MANETs is the time-varying network topology which may be unpredictable over time and, therefore, MANET routing algorithms must keep updating their neighbor discovery data and inform the nodes of the network topology change due to node mobility. The MANET working group (WG) of the Routing Area of the Internet Engineering Task Force (IETF) has defined and standardized IP routing functionality suitable for MANET wireless routing applications within both static and dynamic topologies. Several MANET routing protocols ([1] and [2]) have been accepted as Internet Standards or are under development as Internet drafts ([3], [4], [5], [6], and [7]) under the IETF.

MANET routing protocols can be classified into proactive and on-demand. Proactive routing protocols, using periodic *Neighbor Discovery* message and *Topology Update* message, give route information to each node before a node sends data packets to a destination. The Fisheye Scope Routing (*FSR*) [3], Topology Broadcast Based on Reverse Path Forwarding (*TBRPF*) [4], Fuzzy Sighted Link State Routing (*FSLS*) [5], Optimized Link State Routing Protocol (*OLSR*) [1], and Landmark Ad Hoc Routing (*LANMAR*) [6] are currently being developed as MANET proactive routing protocols. On-demand routing protocols, such as Dynamic Source Routing (*DSR*) [7] and Ad hoc On-demand Distance Vector (*AODV*) [2], issue *Route Discovery* mechanism messages only when a node needs to send data to a destination node. Because these protocols do not use any periodical message exchange such as the neighbor discovery message used in proactive routing protocols, they do not hold any route information at each node before a node sends data towards a destination node. Therefore, they need *Route Request* and *Route Reply* messages to find out and maintain a route until it is needed. Based on [1], [2], [3], [4], [5], [6], and [7], the advantages and disadvantages of the proactive and on-demand routing protocols can be summarized below. The main advantage of the proactive routing protocols is that whenever a node sends a data packet, it obtains route information to a destination by searching its route table. Therefore, the route is already known and can be used immediately. In addition, there is no delay time to determine the route in the source node. However, a portion of network resources in MANETs should be allocated to handle the periodic *Neighbor Discovery* and *Topology Update* messages, and this increases the network traffic load. The main advantage of On-demand routing protocols is that because there are no message exchanges before the start of data communications, the network traffic overhead can be reduced. However, the delay time caused by *Route Discovery* mechanism to find a route to a destination could be a significant factor when considering MANETs routing performance. As the node population and mobility increase, the routing control overhead in the MANET area is also increased, and this is an important factor to be considered in limited wireless bandwidth applications. The authors of [8] studied the scalability issues in MANET's proactive and on-demand routing protocols.

1.2. Motivation

MANET nodes have the capability to cooperate in routing each others' data packets. Due to the lack of any centralized control and possible node mobility in MANETs, many issues at the network, medium access, and physical layers currently become new research topics since no counterparts in the wired networks like Internet, or in cellular networks can satisfy these MANET requirements.

1.2.1. Network Layer in MANET

At the network layer, the main problem is that of routing, which is awfully deteriorated by the time-varying network topology, power constraints, and the characteristics of the wireless channel. MANET routing protocols can be categorized into proactive, reactive, hybrid, hierarchical and location-based protocols. For a MANET to have trustworthy routing protocol, several factors should be considered.

1. The mechanisms for neighbor discovery, topology update, route discovery, route maintenance, data forwarding, link error-checking, and link error recovery when nodes power up, reboot, join, leave and rejoin the network, could be definitely one consideration factor in MANET routing protocols.
2. Performance issues such as network-imposed delay, delay variance, reliability defined as the average loss ratio of the medium by the routing/switching design, the number of hops per route, route discovery time, routing traffic (bps), end-to-end delay, hop-by-hop and end-to-end packet delivery ratio, number of data packets transmitted, number of control bytes transmitted, effect of the traffic load on the routing protocol, and effect of node mobility on the routing protocol should also be studied to satisfy a guaranteed QoS.
3. The technology to setup self-organized wireless interconnection of communication devices in dynamic topologies also needs to be considered.
4. Standardization for different MANET routing protocols to implement interoperability in heterogeneous MANET networks will be demanded to be incorporated within other MANET routing protocols. As node mobility increases, the control routing overhead also increases dramatically, which causes problems

with the allocation of network resources. The scalability issues in MANET routing protocols have been continuously studied to reduce routing control overhead. Due to the different link characteristics in opposite directions in a wireless link, the implementation of a symmetric route from a source to a destination in MANET routing protocol becomes one challenging research topic.

5. With well-defined MANET protocol models, wireless network capacity could be mathematically analyzed to define the maximum network throughput. One of the on-going research areas focuses on defining the maximum throughput in MANETs to be used as a reference guide.
6. Interoperability issues with the other layer stacks such as TCP/UDP protocols, and RSVP/LDP signaling protocols, is one of the primary challenges to design compatible MANET routing protocols.
7. To reduce the number of vulnerabilities, security mechanisms which include authorization and admission control are being developed, and it is an open research area.
8. Assigning IP address to mobile nodes can be expanded as another research topic in MANETs.
9. Another research topic in MANETs focuses on the implementation of multicasting, in which a MANET node can send a data packet to multiple destinations in a group.

1.2.2. Medium Access Control Layer in MANET

MANETs consist of a number of mobile users that communicate with each other over a wireless channel, which causes an issue regarding on how to share a wireless medium among all the users. Due to the time-varying network topology and the lack of centralized control, the choice of the medium control access (MAC) scheme technology is also difficult in ad hoc networks. The ultimate purpose of the MAC is to establish the mechanism for traffics, and to provide the classification for the different requirements of each traffic class. Two types of multiple access protocols such as the contention-based protocol and the contention-free protocol have been developed. TDMA (Time Division Multiple Access), FDMA (Frequency Division Multiple Access), CDMA (Code Division Multiple Access), Token Ring, and DQDB (Distributed Queue Dual Bus) are widely used as the contention-free multiple access protocols. On the other hand, Contention-based protocols can be classified into random access and collision resolution methods based on the methods used to resolve the packet collisions. ALOHA, CSMA (Carrier Sensing Multiple Access), BTMA (Busy Tone Multiple Access), ISMA (Idle Signal Multiple Access) are well-known random access contention-based protocols, in which they use a random delay before resending conflicted packets. The TREE and WINDOW protocols use a sophisticated method to solve the packet retransmission instead of using a random delay. The increasing complexity in TDMA due to the non-centralized control or dynamic assignment of frequency bands in FDMA could not be the best solution in MANETs. The FDMA scheme in MANETs tends to be inefficient when the MANET becomes densely populated. One issue in CDMA is the need to keep track of frequency-hopping patterns and/or spreading codes of the time-varying neighborhood. CSMA/CA

(Carrier Sensing Multiple Access/Collision Avoidance) has been standardized in the IEEE 802.11 (IEEE stands for Institute of Electronics and Electrical Engineering) and can be one of the possible MAC protocols for MANETs. Since the birth of the ALOHA protocol, several variants have been developed. The difference between ALOHA and CSMA is that in ALOHA protocol, when a sender transmits a packet, it does not check whether the channel is busy or not. However, in CSMA protocol when the transmitter sends a packet, it listens and checks whether the channel is busy or not in order to prevent potential packet collisions. Therefore, the CSMA protocol could achieve a better throughput accomplished by listening to the channel before transmitting a packet. Even though two transmitters in the CSMA protocol detect the collision when they transmit packets, they don't stop transmitting packets, but continue sending packets and complete the packet transmission, and this transmission occupies the wireless medium uselessly for an entire packet time and the transmitted packets eventually collide. However, in the CSMA/CD protocol, whenever two nodes detect a collision when transmitting packets, they stop the transmissions immediately. CSMA/CD can only detect packet collisions and there is no collision avoidance mechanism. However, CSMA/CA protocol has a special request to send (RTS) and clear to send (CTS) hand-shaking mechanism to avoid packet collisions before sending a packet. The distributed coordination function (DCF) defines the mechanism of RTS and CTS frames prior to the transmission of the actual data frame. Based on the above discussions, for a MANET to have trustworthy medium access control protocol, several factors should be considered as follows.

1. The selection of the layer 2 mechanisms among TDMA, FDMA, DTDD (Dynamic Time Division Duplex), CDMA, ALOHA, CSMA/CD and CSMA/CA

is a difficult choice for MANET medium access control. Many simulation results based on OPNET, MATLAB, ns-2, GlomoSim and QualNet with various scenario analyses are published to show the various performances of the MAC schemes.

2. Performance issues such as utilization of bandwidth, packet delay rate, channel busy probability, packet blocking rate, packet dropping rate, packet error rate, and throughput could be measured to evaluate the various MAC schemes.
3. Various scheduling issues have also been studied to provide QoS differentiation over the scheduling switch.
4. The technology of preemptive or nonpreemptive priority-based access control scheme for broadband MANETs has also been considered by many researchers.
5. Error control schemes such as automatic repeat request (ARQ) to achieve reliable data transmission over wireless transmission links have been proposed.

1.2.3. Physical Layer in MANET

At the physical layer, power control is one of the most important issues, and the focus is on getting the sufficient transmission range of a node, which needs to be controlled so that it is wide enough to reach the intended receiver, while causing minimal interference to other nodes.

1. Based on the parameters coming from the physical layer, several efficient power-aware routing protocols and battery cost routing protocols have been proposed.

2. To get higher wireless channel capacity, Multi-Input-Multi-Output (MIMO) systems, in which both the transmitter and the receiver have multiple antennas, are currently under study by many researchers.
3. To get different QoS levels in the physical layer, Dynamic Time Division Duplex (DTDD) can be used, where portions of the downlink and uplink bandwidth in cellular network are dynamically assigned. In Static TDD (STDD) the portions of downlink and uplink bandwidth in cellular networks are fixed. This could be one challenging research topic.
4. To get better signal-to-interference (SIR) ratio at the receiver, many research studies to avoid the co-channel interference (CCI) and inter-symbol interference (ISI) have been addressed.

1.3. Dissertation Outline

In this dissertation, the technical review of several MANET routing protocols based on their messages is analyzed and message complexity analysis of MANET address autoconfiguration protocols is studied. Due to the lack of any centralized control and possible node mobility in MANETs, many issues at the network, medium access, and physical layers currently remain as research topics since no counterparts in the wired networks or cellular networks can satisfy these MANET requirements.

Therefore, the objective of this research is to propose a novel method to perform a quantitative analysis of message complexity as one of performance measures to overcome the issues at the network, medium access, and physical layers. The contribution of this research is that the analytical framework used in deriving the upper bound of the message complexity, which is represented in this dissertation, can be widely adapted to a wide variety of protocols. The remainder of this dissertation is organized as follows. Chapter 2 gives the technical review of 12 MANET routing protocols based on their messages. In chapter 3, the comparison of the message complexity among the MANET address autoconfiguration protocols is studied. Chapter 3 also presents several *Lemmas* and their proofs used in deriving the message complexity of Strong DAD (Duplicate Address Detection), Weak DAD, and MANETconf respectively and it contains the numerical results, performance analysis and conclusion.

2. MANET ROUTING PROTOCOLS

2.1. Introduction

This chapter introduces the technical review of several MANET routing protocols based on their messages. To come up with the dynamic topology of networks in proactive routing protocols, the *Topology Update* message is one of the essential messages used to update the MANET topology. On-demand Routing protocols need the *Route Request* message to set up a path to a destination.

Fisheye State Routing (*FSR*) is a link state proactive routing protocol. Instead of using *Link State* flooding over the network, update routing table in ad hoc network is periodically broadcasted to its neighbor. Depending on the hop distance from the current node, the frequency update is different such that the *FSR* produces accurate information about the immediate neighbor nodes but less accurate information about the nodes that are farther away from the current node. A change in link state outside predefined scope from the current node does not necessarily cause a change in the routing table at the current node. Each node running *FSR* maintains a topology table and a routing table. *Link state* messages construct the topology table where the distance information is calculated from the routing table and is used to classify the node similar to a fisheye scope [3].

Topology Broadcast Based on Reverse Path Forwarding (*TBRPF*) is a link state proactive routing protocol composed of the *TBRPF neighbor discovery module* (*TND*) and the *TBRPF routing module*. The *TND* only senses one hop neighbors; however the *TBRPF* *routing module* discovers two-hop neighbors. The *TND* is periodically performed through differential *Hello* messages that report only the changes in neighbors. With the different fields of *Neighbor_Request*, *Neighbor_Reply* and *Neighbor_Lost_of_Hello* messages, each *TBRPF* node detects the bi-directional links between its local interface and its neighbor interface, detects loss of such links and keeps links to its one hop neighbors. In the *TBRPF routing module*, based on the periodic and differential topology updates with *Hello* message, each node creates the shortest paths to all reachable nodes. *Link State* updates that form the spanning tree in the reverse direction from the source are propagated. Full topology updates and differential topology updates are two types of topology updates of the *TBRPF* with the *Topology_Update* message, *Interface*, *Host* and *Network Prefix Association* messages. Differential topology updates are sent more frequently to update the link changes such as additions and deletions of nodes [4].

Fuzzy Sighted Link State Routing (*FSLS*) is a link state proactive routing protocol for a flat network structure where a balance between the proactive link update overhead and the overhead due to suboptimal routing is considered. In *FSLS*, the nodes transmit *Link Status Updates* (*LSU*) only at particular instants that are multiples of some constant time value. Several link changes are collectively sent in a single packet with a Time to Live (TTL) value [5].

Dynamic Source Routing (*DSR*) is an on-demand (reactive) routing protocol which does not need any periodic routing advertisement, link status sensing or neighbor detection packet. *DSR* is composed of a *Route Discovery* procedure and a *Route Maintenance* procedure.

When node *S* wants to send a packet to node *D*, but does not know a route to *D*, node *S* initiates a *Route Discovery* with *Route Request* (*RREQ*) message flooding. Each node adds its own identifier when a *RREQ* is forwarded by each node. If the node is the destination node of the *RREQ*, it stops forwarding the *RREQ*, and the destination replies by sending a *Route Reply* (*RREP*) message which is sent on a route obtained by receiving the route appended to receive *RREQ* if a bi-directional link is allowed. Otherwise, *RREP* may need to scan a new route discovery for *S* from node *D* unless node *D* already knows a route to *S*. If a route discovery is initiated by *D* for a route to *S*, the *RREP* is piggybacked on the *RREQ* generated from *D*. *RREP* includes the route from *S* to *D* on which *RREQ* was received by node *D*. When *S* receives the *RREP*, it caches the route from *S* to *D* included in the *RREP*. When *S* sends a data packet to *D*, the entire route is included in the data packet header from which the name of Source Routing comes. Intermediate nodes use the source route included in the packet header to determine to whom a packet should be forwarded to. With the use of Route Caching at each node, which learns a new route and caches it, the performance of *DSR* can be optimized. If a node finds a broken link when it attempts to forward the data packet, the node initiates and sends a *Route Error* (*REER*) message in the reverse path along the source route. The nodes that hear the *REER* update their route cache to remove the broken link [7].

Zone Routing Protocol (*ZRP*) is a hybrid routing protocol that provides a balance between the proactive and reactive routing protocols. Within the network area of the routing zone, a distance-vector and a link state of the proactive *IntrAzone Routing Protocol* (*IARP*) is used to deliver the topology update. When a change occurs in the topology, *IARP* packets (messages) are exchanged and appropriate changes are made in the routing tables of the nodes in the zones. The packets are restricted from being transmitted to nodes outside a zone with the help of the hop count parameter. The *IntErzone Routing Protocol* (*IERP*) is responsible for acquiring routes to destinations that are located beyond the routing zone. The *IERP* uses *Route Query*/*Route Reply* fields of the *IERP* message to discover a route to nodes located outside the Zone [9].

Optimized Link State Routing Protocol (*OLSR*) is a link state proactive routing protocol, in which the topology information of the network is exchanged periodically. Each node selects a set of nodes in its neighborhood which is called the *Multipoint Relay* (*MPR*) set of that node. Only *MPRs* of node *N* can send the *Link State* broadcast message and the neighbor nodes of node *N* which are not in the set of *N*'s *MPR* can receive and process broadcast messages but do not retransmit *Link State* broadcast messages received from node *N*. Therefore, *OLSR* can reduce duplicate retransmissions of flooding messages in the same region. Since only *MPR* nodes can send routing messages that flood the network and only partial *Link States* are flooded, the number of *Link State* Updates and the length of the *Link State Updates* are reduced. A *Hello* message is used for neighbor discovery mechanism. To get the local link information and the neighborhood information, periodic exchange of *Hello* messages is performed. Each node with multiple interfaces should periodically announce information describing its interface configuration to other nodes in the network using the message *Multiple Interface Declaration* (*MID*). The *Topology Control* (*TC*) message is used for the distribution of topology update information of each node. A node which is associated with hosts or/and networks should periodically generate the *Host* and *Network Association* (*HNA*) message, containing pairs of (network address, netmask) corresponding to the connected hosts and networks [1].

Landmark Ad Hoc Routing (*LANMAR*) is a proactive routing protocol, which combines the features of Fisheye State Routing (*FSR*) and Landmark Routing. Landmarks are selected for each set of nodes that move as a group. A packet that has to reach a remote destination initially aims at the landmark of the remote group and then as it gets closer to the landmark, it switches to Fisheye State Routing that is more accurate. Both scalability and mobility problems are solved as the line and storage overhead are kept low. To periodically update the landmark distance vector (*LMDV*) and the drifter list (*DFDV*), the *LANMAR Update* message is exchanged with the neighbors [6].

Location-Aided Routing (*LAR*) is a geographical routing protocol that uses location information supported by Global Positioning System (*GPS*) to limit the scope of *Route Request* flood. Expected zone is calculated as a region that is expected to hold current location of the destination. Assume that there is a node S that needs to communicate with a destination node D. Suppose S knows the location L of D at time t_0, then the *Expected Zone* of the node D can be estimated at time t_1 and is given by the circular region of radius $v(t_1 - t_0)$ centered at location L, where v is the average speed of node D. *Route Request* message is limited to a *Request Zone* that contains the *Expected Zone* and the location of the sender node. The *Request Zone* is usually larger than the *Expected Zone* to increase the probability of the message reaching the destination. When receiving *Route*

Request message, the destination node responds by sending a *Route Reply* message to the sender. A node along the path sends a *Route Error* message if the node finds that there is a broken link on the route to its next hop [10].

Greedy Perimeter Stateless Routing (*GPSR*) is a geographical routing protocol which uses only positions of a router's immediate neighbors and a packet's destination to make routing decisions. In cases where Greedy forwarding is not possible, routing around the perimeter of the region takes place. Geographic routing helps routers to be almost stateless. A *GPSR* node just has to know the neighboring nodes by exchange of the *Beacon* message which is periodically broadcasted to its neighbor nodes. The *GPSR* has two kinds of method for forwarding packets - greedy forwarding and perimeter forwarding. Greedy forwarding is used as much as possible and only in cases where Greedy forwarding is not possible Perimeter forwarding is used [11].

Ad hoc On Demand Distance Vector (*AODV*) is a distance vector on-demand (reactive) routing protocol. *DSR* includes the source routes in packet header which results in large header and degrading performance. *AODV* provides the mechanism to hold the routing table at the nodes, so that data packets do not have to contain source routes in its header. The *Hello* message is periodically exchanged with neighbor nodes for connectivity. The absence of the *Hello* message for *Allowed_Hello_Loss* · *Hello_Interval* milliseconds results in an indication of link failure. When node *S* wants to send a packet to node *D*, but does not know a route to *D*, node *S* sends a *Route Request* (*RREQ*) message which is flooded into the network. When the destination receives a *RREQ*, it replies by sending a *Route Reply* (*RREP*) message which travels along the reverse path set up when a *RREQ* is forwarded. Forwarding links at each intermediate node are installed when *RREP* travels along the reverse path. When a *RREP* arrives at source node *S*, *S* sends *Route Reply Acknowledgement* (*RREP-ACK*) message to *D*. Each node along the reverse path caches the next hop links in its routing table. Whenever link failures to the next hop are found at each node, the *Route Error* (*RERR*) message is propagated to all active nodes including the source node *S*. When the node *S* receives the *RERR*, it initiates a new route discovery destined for *D* [2].

Hierarchical State Routing (*HSR*) is a hierarchical link state routing protocol in which the networks are managed by a hierarchical structure. Two types of node partition are provided in *HSR*. Physical partitioning is based on geographical node position whereas the logical partitioning is based on node functionality. Two address schemes are used in *HSR* network. An IP address is used for identification for packet delivery and a *Hierarchical ID* (*HID*) address is used for routing in *HSR*. The home agent keeps up-to-date mobility binding for each subnet member based on the *HID* registration process, which is triggered in periodic and event driven circumstances. To register a *HID* at the home agent, each subnet member sends a *Registration* message to its home agent. Upon receiving a *Registration* message, the home agent creates a mobility binding for the mobile node, or modifies the mobility binding accordingly. If the registered *HID* address is not refreshed within a certain time, it is timed out and erased from the home agent. To send a data packet from source *S* to destination *D*, *S* sends the packet directly to the destination if *S* has the valid *HID* address of the destination. Otherwise, through *HID Reply* message from the home agent, *S* knows the registered address of *D* and this mechanism is called the *HID* finding process [12].

Clusterhead Gateway Switch Routing (*CGSR*) is a hierarchical distance vector routing protocol in which a cluster member table and a distance vector (*DV*) routing table are updated at the each mobile node. When *S* sends a data packet, *S* initially finds the clusterhead of the destination node *D* from the cluster member table. Then *S* looks up its routing table to find the next hop to the clusterhead of the destination and forwards the packet to the clusterhead of the destination. When the *CGSR* network forwards the packets, it routes packets alternatively between clusterheads and gateways. When the packets finally reach the clusterhead of the destination, the clusterhead of the destination forwards the packet to the destination node *D* [13].

2.2. Fisheye State Routing (FSR)

2.2.1. Introduction

FSR is a proactive routing protocol for the flat network structure and it is a variation of the *Link State* type routing protocol where each node periodically updates its *Link State* to other nodes instead of being event driven. The events are usually triggered by the *Link State* exchange such as link breaks. If this mode of exchange was followed in the unstable mobile ad hoc network (MANET), it will lead to many *Link State* updates. To reduce *Link State* overhead in *FSR*, even though the *Link State* updates are sent periodically, not all the updates are sent at the same frequency. The frequency of the updates depends on the hop distance from the current node. Due to the different period of *Link State* update in *FSR*, there is accurate information about the immediate neighborhood but less accurate information about the nodes that are farther away from the current node [3] [14].

2.2.2. Topology representation of FSR

The network is modeled as a undirected graph, $G(V,E)$, where V is the set of nodes and E is the set of links that connect nodes in V. A unique identifier identifies each node. An undirected link is formed between two nodes, say i and j, if they are within the transmission range. In *FSR* routing algorithm, each node maintains three tables and a list. There is a neighbor list $\boldsymbol{A_i}$, a topology table $\boldsymbol{TT_i}$, a next hop table $\boldsymbol{NEXT_i}$, a distance table $\boldsymbol{D_i}$. $\boldsymbol{A_i}$ is the set of nodes that are one hop away from the node i. Each destination j has an entry in $\boldsymbol{NEXT_i}$ comprising of two parts – the link state information sent by j and the time stamp at which the information was generated by j. $\boldsymbol{NEXT_i}$ $(\boldsymbol{j})$ represents the next hop from i to node j through the shortest path. D_i (j) represents the distance of the shortest path between i and j.

2.2.3. Link State (LS) message

FSR is composed of one *Link State* (*LS*) message that not only gives a neighbor discovery mechanism but also yields a topology update mechanism for a node to calculate the efficient routes. Based on topology map at each *FSR* node, the node calculates the efficient routes in MANET.

FSR nodes are classified with a different hop distance from a source node which generates *Link State* message. The source node uses a different period to update the *Link State* message to its scoped destination nodes based on the classified hop distance. The node sends the *Link State* message within the smallest scope with highest frequency. The rest of the scoped nodes will receive the *Link State* message with lower frequencies. After performing neighbor discovery, a node can send a *Link State* update message to a destination node to which the node can calculate the efficient route based on the information of the discovery. If the node does not receive a *Link State* message from its neighbor within a *Neighbor_Timeout* interval, the node deletes the neighbor from its neighbor list. After receiving a *Link State* message from its neighbor, the node updates its topology. Based on the information of the topology table, the node calculates the shortest path towards each destination and stores the path information in its routing table. The distance information for each destination in the node's routing table is used to classify the scope of each destination node. The node uses the next hop information in its routing table to forward the packets destined for a certain destination.

Fig. 1 shows the *Link State* message of *FSR*. *Destination Address* in the *Link State* message is the node’s IP address which is located at the hop distance which is equal to the hop distance of the source node's neighbor. The hop distance of the source node's neighbor is used in neighbor discovery mechanism or beyond the hop distance of the source node's neighbor, which is used in the topology update mechanism. *Destination Sequence Number* is the last sequence number received from the destination in the past by the source. *N_neighbors n* indicates the number of neighbors of the destination node *n*. *Neighbor Address* 1,⋯, *Neighbor Address N_neighbors n* is the list of neighbors of the destination node *n* [3] [14].

```
0                                       1                                       2                                       3
0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5   6   7   8   9   0   1
```

Packet Length	*Reserved*
Destination Address 1	
Destination Sequence Number 1	*N_neighbors* 1
Neighbor Address 1	
. . .	
Neighbor Address N_neighbors 1	
Destination Address 1	
Destination Sequence Number 2	*N_neighbors* 2
Neighbor Address 1	
. . .	
Neighbor Address N_neighbors 2	
. . .	

Figure 1 FSR Link State Messag

Refer to the field description in [3] for further details.

2.2.4. Algorithm for FSR

1. *Each node is initialized with the following values.*

   A. *An empty neighbor list* $\mathbf{A_i}$.

   B. *An empty topology table* $\mathbf{TT}_i$.

   C. *All the entries in the distance table* $\mathbf{D_i}$ *are set to infinity.*

   D. *All the entries in the* $\mathbf{NEXT_i}$ ($\boldsymbol{j}$) *table are set to* –1.

   E. *All the entries of the time stamps in the topology table are also set to* –1.

2. *From each packet received, the node i tries to learn about its neighbors from the Sender ID*

   A. *It reads the ID of the received packet and adds it to the neighbor list* $\mathbf{A_i}$.

   B. *The node i tries to determine if the information from the node j is the latest by examining the time stamp information in the packet.*

C. *If the timestamp in the received packet is later than that is there in the topology table* ***TT****, then the link state information and the timestamp information from the received packet are updated.*

3. *After knowing the neighbors and updating the topology table, the node will use Dijkstra's algorithm to find the shortest path and updates the distance table* $\boldsymbol{D_i}$.
4. *The entries in the distance table are compared with the fisheye radius, whichever node has a distance less than the fisheye radius, and then the link state information of that node will be included in the update message.*
5. *This message is broadcasted to all the nodes in the set* $\boldsymbol{A_i}$*. The update message is sent out to the nodes in the fisheye radius based on the distance of the nodes from the node i. Information about nodes that are nearer are sent out very often and those which are farther away are not sent that very often.*

2.2.5. Pseudo Code for FSR

```
While TRUE do
    For each node I in the network
    $A_i = \phi$
    $TT_i = \phi$
    $D_i = \infty$
    $NEXT_i (j) = -1$
    $TT_i.SEQ(j) = -1$
    Next Node
```

```
For each received Packet Pkt
Ai = Ai ∪ Pkt.SenderID
If Pkt.SEQ(j) > $TT_i$.SEQ(j)
        $TT_i$.SEQ(j) = Pkt.SEQ(j)
        $TT_i$.LS(j) = Pkt.LS(j)
End if
 Next Packet
```

2.3. Topology Broadcast Based on Reverse Path Forwarding (TBRPF)

2.3.1. Introduction

Topology Broadcast Based on Reverse Path Forwarding is a proactive routing strategy in which each node has the information about the entire network. The messages are not flooded into the network, but are forwarded along the reverse path forwarding mode. The Dijsktra Topology Broadcast Based on Reverse Path Forwarding (*TBRPF*) comprises of two modules namely the neighbor discovery module and the routing module. Each node selects its next hop on the minimum hop path to reach the source node *S*. The link state updates are not flooded through the network and only those updates that form the spanning tree in the reverse direction from the source are propagated. There are also two types of topology updates used in *TBRPF* – full topology update and differential topology update. Differential topology updates are sent more frequently to update the link changes, such as additions and deletions [4] [15].

2.3.2. Message Summary

TBRPF is composed of a neighbor discovery message (*Hello* message) and routing messages (*Topology Update, Interface, and Host and Network Prefix Association Messages*). *TBRPF* nodes periodically report *Hello* message to their neighbors for the discovery of neighbors. *TBRPF* node updates the *Topology Update* message periodically to report the partial source tree at each node to its neighbors. *Interface, Host and Network Prefix Association* messages inform the neighbors of the information of the interface, host and network prefix. All these messages are constructed between the initiator node of messages and initiators' neighbor nodes and are periodically updated [4] [15].

Hello message

The *TBRPF Hello* message as shown in Fig. 2 includes the fields of the *Neighbor Interface* to represent the interface addresses of the neighbor nodes. For example, it is assumed that when the network is powered up, there are only two neighboring nodes *A* and *B*, node *A* can send a *Hello Neighbor Request* message including no list of neighbor interface address. As soon as node *B* receives the *Hello Neighbor Request* message, it knows that the link between node *B* and node *A* is a unidirectional, and that node *A* is its neighbor and sends a *Neighbor Request* message including *A*'s address as the neighbor interface address to respond to *A*'s *Hello Neighbor Request*. After node *A* receives the *Hello Neighbor Request* message from node *B*, node *A* can decide the link between node *A* and node *B* is bi-directional since it receives a *Hello Neighbor Request* and sends a *Hello Neighbor Request* on its interface. Node *A* sends the next periodic *Hello Neighbor Request* message including no list of neighbor interface address. The message piggybacks a *Hello Neighbor Reply* message including one neighbor interface address which represents *B*'s interface address. After node *B* receives the *Hello Neighbor Request* message from node *A*, node *B* can decide that the link between node *B* and node *A* is bi-directional since it receives a *Hello Neighbor Request* and sends a *Hello Neighbor Request* on its interface. If *B* moves away to out-of-range of node *A*, and therefore, node *A* does not receive *Hello* messages from node *B* within the *NBR HOLD TIME*, node A sends a *Hello Neighbor Request* message including no list of neighbor interface address.

The message piggybacks a *Hello Neighbor Lost* message including one neighbor interface address which represents *B*'s interface address. The role of the neighbor interface address list in a *Hello Neighbor Request* message is to convert the status from unidirectional link to bidirectional link if the nodes successfully receive a *Hello Neighbor Reply* in response to the *Hello Neighbor Request*.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

0	*TYPE*	*HSEQ*	*PRI*	*n*
Neighbor Interface Address (1)				
Neighbor Interface Address (2)				
⋯				
Neighbor Interface Address (*n*)				

Figure 2 TBRPF Hello message

Refer to the field description in [4], [15] for further details.

Topology Update message

Topology Update (*TU*) message as shown in Fig. 3 includes the metrics of the links belonging to the sending router's reported subtree *RT* and this message is sent to the neighbors.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

M	*D*	0	0	*TYPE*	*n*	*NRL*	*NRNL*
Router ID of u							
*Router ID of v*_1							
⋯							
Router ID of v_n							
Metric 1					*Metric* 2	⋯	

Figure 3 TBRPF Topology Update message

Refer to the field description in [4], [15] for further details.

Interface Association (*IA*) *message or Host Association* (*HA*) *message*

Interface Association (TYPE=8) message or *Host Association* (TYPE=9) message as shown in Fig. 4 contains the router *ID* and IP addresses of Interfaces or Hosts that are associated with the router ID to inform the neighbor nodes.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
ST		0		*TYPE*				*Reserved*								*n*															
Router ID																															
IP Address																															
IP Address																															
…																															

Figure 4 TBRPF Interface Association message

Refer to the field description in [4], [15] for further details.

Network Prefix Association message

Network Prefix Association (*NPA*) message shown in Fig. 5 includes the router *ID* of the originator, and network prefixes to inform the neighbor nodes.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
ST		0		*TYPE*				*Reserved*								*n*															
Router ID																															
PrefixLength								*Prefix byte* 1								*Prefix byte* 2								…							
…								*PrefixLength*								*Prefix byte* 1								*Prefix byte* 2							
…																															

Figure 5 TBRPF Network Prefix Association message

Refer to the field description in [4], [15] for further details.

2.3.3. Pseudocode for TBRPF

The following pseudocode describes the network-level procedures performed at node *i* by TBRPF [4], [15]. The notation *LSU*(*update_list*) represents a link-state-update message that includes the updates (*u*,*v*,*c*,*sn*) in *update_list*.

```
Process_Update(i, nbr, in_message)
{
    (Called when an update message in_message is received from nbr.)
    Update_Topology_Table(i, nbr, in_message, update_list).
    Update_Parents(i).
    For each node src in TT_i
    {
        Let update_list(src) consist of all tuples (k,l,c,sn)  in update_list
        such that k = src.
        If update_list(src) is nonempty
        {
            Send message LSU(update_list(src)) to
            children_i(src).
        }
    }
}
```

```
Update_Topology_Table(i, nbr, in_message, update_list)
{
    Set update_list to empty list.
    For each ((u,v,c,sn) in in_message)
    {
        If (p_i(u) == nbr)
        {
            If ((u,v) is in TT_i and sn > TT_i(u,v).sn)
            {
                Add (u,v,c,sn) to update_list.
                Set TT_i(u,v).sn = sn.
                Set TT_i(u,v).c = c.
                If (sn > sn_i(u)) Set sn_i(u) = sn.
            }
            If ((u,v) is not in TT_i)
            {
                Add (u,v,c,sn) to TT_i.
                Add (u,v,c,sn) to update_list.
                If (sn > sn_i(u)) Set sn_i(u) = sn.
            }
        }
    }
}
```

```
Link_Change(i,j)
{
        (Called when the cost of link (i,j) changes.)
        If (|TT_i(i,j).c - cost(i,j)|/TT_i(i,j).c > epsilon)
        {
                Set TT_i(i,j).c = cost(i,j).
                Set TT_i(i,j).sn = current time stamp SN_i.
                Set update_list = {(i, j, TT_i(i, j).c, TT_i(i, j).sn)}
                Send message LSU(update_list) to children_i(i).
        }
}

Link_Down(i,j)
{
        (Called when link (i,j) goes down.)
        Remove j from N_i.
        Set TT_i(i,j).c = infinity.
        Set TT_i(i,j).sn = current time stamp SN_i.
        Update_Parents(i).
        For each (node src in TT_i) remove j from children_i(src).
        Set update_list = {(i,j, infinity, TT_i(i,j).sn)}.
        Send message LSU(update_list) to children_i(i).
}
```

```
Link_Up(i,j)
{
    (Called when link (i,j) comes up.)
    Add j to N_i.
    Set TT_i(i,j).c = cost(i,j).
    Set TT_i(i,j).sn = current time stamp SN_i.
    Update_Parents(i).
    Set update_list = {(i, j, TT_i(i,j).c, TT_i(i,j).sn)}.
    Send message LSU(update_list) to children_i(i).
}


Update_Parents(i)
{
    Compute_New_Parents(i).
    For each (node k in N_i)
        Set cancel_src_list(k), src_list(k), and sn_list(k) to empty.
    For each (node src in TT_i such that src != i)
    {
        If (new_p_i(src) != p_i(src))
        {
            If (p_i(src) != NULL)
            {
                Set k = p_i(src).
```

```
                    Add src to cancel_src_list(k).
                }
                Set p_i(src) = new_p_i(src).
                If (new_p_i(src) != NULL)
                {
                    Set k = new_p_i(src).
                    Add src to src_list(k).
                    Add sn_i(src) to sn_list(k).
                }
            }
        }

        For each (node k in N_i)
        {
            If (src_list(k) is nonempty)
                Send message NEW PARENT(src_list(k), sn_list(k)) to k.
            If (cancel_src_list(k) is nonempty)
                Send message CANCEL PARENT(cancel_src_list(k))  to k.
        }
    }

    Compute_New_Parents(i)
    {
```

```
    For each (node src in TT_i such that src != i)
        Set new_p_i(src) = NULL.
    Compute min-hop paths using Dijkstra.
    For each (node src in TT_i such that src != i)
        Set new_p_i(src) equal to the neighbor of node i along the minimum hop
path from i to src.}


Process_New_Parent(i, nbr, src_list, sn_list)
{
    (Called when node i receives a
    NEW PARENT(src_list, sn_list) message from nbr.)
    Set update_list to empty list.
    For each (node src in src_list)
    {
        Let sn_list.src denote the sequence number
        corresponding to src in sn_list.
        Add nbr to children_i(src).
        Set new_updates = {(k,l,c,sn) in TT_i such that k = src
        and sn > sn_list.src}.
        Add new_updates to update_list.
    }
    Send message LSU(update_list) to nbr.
}
```

```
Process_Cancel_Parent(i,nbr,src_list )
(
    (Called when node i receives a CANCEL PARENT(src_list) message from nbr.)
    For each (node src in src_list) remove nbr from children_i(src).
}

Send_Periodic_Updates(i)
{
    Set update_list to empty.
    For each (j in N_i such that TT_i(i,j). c != infinity)
    {
        Set TT_i(i,j).sn = current time stamp SN_i.
        Add (i, j, TT_i(i,j).c, TT_i(i,j).sn) to update_list.
    }
    Send message LSU(update_list) to children_i(i).
}
```

2.4. Fuzzy Sighted Link State Routing (FSLS)

2.4.1. Introduction

A balance between the proactive link update overhead and the overhead due to suboptimal routing is struck in *FSLS*. *FSLS* is a proactive routing protocol for the flat network structure. In *FSLS*, the nodes transmits *Link Status Updates* (*LSU*) only at particular instants that are multiples of some constant time value t_e. Several link changes are collectively sent in a single packet with a *TTL* value. The node wakes up every $2^{i-1} * t_e$ (i =1,2,3…) seconds and transmits a *LSU* with *TTL* set to s_i if there has been a link status change in the last $2^{i-1} * t_e$ seconds [5].

2.4.2. Message Summary

Fuzzy Sighted Link State (*FSLS*) routing protocol uses the space and time limitation to reduce the control overhead in MANET. A *FSLS* node propagates messages to the nodes based on the value of the *TTL* field, which is called as the *Space Limitation*. Also *FSLS* disseminates messages at particular time instants (t_e), which is called as the *Time Limitation*. *FSLS* is composed of one *LSU* message whose exact format is not defined yet [5].

2.4.3. Pseudocode

Node Set S_i comprise of the nodes that are located at a Distance S_i from the node of interest [5]

```
For I = 0 to 1000
    WakeUp_Time = ($2^{i-1} * t_e$)
    If (Link change in the Wakeup_Time) then
        Transmit a TTL to the node Set $S_i$
    Else
        Wait for ($2^i * t_e$) Time to send the TTL
        End if
Next I

If $S_i$ is greater than the distance from any other node
        TTL value is set to infinity – a Global LSU
        Reset all the Counters and Timers
A Periodic Timer is started which transmits the Global LSU at regular intervals
End if
```

2.5. Dynamic Source Routing (DSR)

2.5.1. Introduction

To send a packet to another host, the sender constructs a *source route* in the packet's header, giving the address of each host in the network through which the packet should be forwarded in order to reach the destination host. The sender then transmits the packet over its wireless network interface to the first hop identified in the source route. When a host receives a packet, if this host is not the final destination of the packet, it simply transmits the packet to the next hop identified in the source route in the packet's header. Once the packet reaches its final destination, the packet is delivered to the network layer software on that host. There are two basic operations that take place in *DSR* – namely, route discovery and route maintenance [7], [16].

2.5.2. Message Summary

DSR protocol is composed of the mechanisms of *Route Discovery* and *Route Maintenance* which operate totally on-demand. When a source node (*S*) wants to send a packet towards a destination (*D*), *S* finds out an explicit source route to follow on its way to *D* in its *Route Cache*. When a route out of the *Route Cache* provides the route to *D*, *S* can use the route to send packets if the *Route Cache* has the valid route. If *S* can not find the route in its *Route Cache*, *S* initiates the *Route Discovery* mechanism via a *DSR Route Request* (*RREQ*) and a *DSR Route Reply* (*RREP*) options. *S* propagates the *RREQ* option that includes the destination address to its neighbors. When intermediate nodes propagate the *RREQ* option, they record their addresses to the *Address* fields of the *RREQ* option. When *D* receives the *RREQ* option, it should reply with the *DSR RREP* option that includes a copy of the accumulated route record list from the *RREQ* option. When *S* receives the *RREP* option, *S* stores the route record list in its *Route Cache*, puts the source route into the header of the packets, and sends the packet that has the routing information in their headers [7], [16].

RREQ message

When *S* can not find the route in its *Route Cache*, *S* initiates the *Route Discovery* mechanism via a *Route Request* (*RREQ*) message. The *Target Address* field indicates the IP address of a destination. *Address* [1], *Address* [2], …, *Address* [*n*] fields are accumulated when a *RREQ* is relayed at one of the relaying nodes between *S* and *D*.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Option Type								*Opt Data Len*								*Identification*															
Target Address																															
Address [1]																															
Address [2]																															
…																															
Address [*n*]																															

Figure 6 DSR Route Request message

Refer to the field description in [7], [16] for further details.

RREP message

When *D* receives the *RREQ* option, it should reply with the *DSR RREP* option that includes a copy of the accumulated route record list from the *RREQ* option. *S* uses *Address* [1], *Address* [2], …, *Address* [*n*] fields as a source route in order to send data packet to *D*.

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
								Option Type								*Opt Data Len*								*L*	*Reserved*						
Address [1]																															
Address [2]																															
…																															
Address [*n*]																															

Figure 7 DSR Route Reply message

Refer to the field description in [7], [16] for further details.

RERR message

When a node finds a link error while it attempts to forward a packet, it generates a *RERR* option. *Error Source Address* field indicates a node which generates a *RERR* message.

Error Destination Address field indicates a node to which a *RERR* message should be delivered. *Type-Specific Information* includes the detail of error contents.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Option Type	*Opt Data Len*	*Error Type*	*Reserved*	*Salvage*
Error Source Address				
Error Destination Address				
Type-Specific Information				

Figure 8 DSR Route Error message

Refer to the field description in [7], [16] for further details.

DSR Source Route message

Data packet includes a *DSR Source Route option.* Each active node forwards the data packet based on the *DSR Source Route option*.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Option Type	*Opt Data Len*	*F*	*L*	*Reserved*	*Salvage*	*Segs Left*
Address [1]						
Address [2]						
…						
Address [*n*]						

Figure 9 DSR Source Route message

Refer to the field description in [7], [16] for further details.

2.5.3. Algorithm for DSR

Originating a Data Packet

When node A originates a packet, the following steps must be taken before transmitting the packet:

1. If the destination address is a multicast address, piggyback the data packet on a Route Request targeting the multicast address. The following fields must be initialized as specified:

IP.Source_Address = Home address of node A

IP.Destination_Address = 255.255.255.255

Request.Target_Address = Multicast destination address

2. Otherwise, call Route_Cache.Get() to determine if there is a cached source route to the destination.

3. If the cached route indicates that the destination is directly reachable over one hop, no Routing Header should be added to the packet. Initialize the following fields:

IP.Source_Address = Home address of node A

IP.Destination_Address = Home address of the Destination

4. Otherwise, if the cached route indicates that multiple hops are required to reach the destination, inserts a Routing Header into the packet.

5. Otherwise, if no cached route to the destination is found, insert the packet into the Send Buffer and initiate Route Discovery

Processing a Route Request Option

When a node A receives a packet containing a Route Request option, the Route Request option is processed as follows:

1. If Request.Target_Address matches the home address of this node, then the Route Request option contains a complete source route describing the path from the initiator of the Route Request to this node.

(*a*) *Send a Route Reply.*

(*b*) *Continue processing the packet in accordance with the Next Header value contained in the Destination Option extension header.*

2. Otherwise, if the combination (*IP.Source_Address, Request.Identification*) *is found in the Route Request Table, then discard the packet, since this is a copy of a recently seen Route Request.*

3. Otherwise, if Request.Target_Address is a multicast address then:

(*a*) *If node A is a member of the multicast group indicated by Request.Target_Address, then create a copy of the packet, setting IP.Destination_Address = REQUEST.Target_Address, and continue processing the copy of the packet in accordance with the Next Header field of the Destination option.*

(*b*) *If IP.TTL is non-zero, decrement IP.TTL, and retransmit the packet.*

(*c*) *Otherwise, discard the packet.*

4. Otherwise, if the home address of node A is already listed in the Route Request (*IP.Source_Address or Request.Address*[ ]), *then discard the packet.*

5. Let

m = number of addresses currently in the Route Request option

$n = m + 1$

6. Otherwise, append the home address of node A to the Route Request option (*Request.Address*[*n*]).

7. Set Request.IN_Index[*n*] *= index of interface packet was received on.*

8. If a source route to Request.Target_Address is found in our Route Cache, return a Cached Route Reply

9. Otherwise, for each interface on which the node is configured to participate in a DSR ad hoc network:

(*a*) *Make a copy of the packet containing the Route Request.*

(*b*) *Set Request.OUT_Index*[*n*+1] = *index of the interface.*

(*c*) *If the outgoing interface is different from the incoming interface, then set the C bit on both Request.OUT_Index*[*n*+1] *and Request.IN_Index*[*n*]

(*d*) *Link-layer re-broadcasts the packet containing the Route Request on the interface jittered by T milliseconds, where T is a uniformly distributed, random number between* 0 *and BROADCAST_JITTER.*

Originating a Route Reply

1. If REQPacket.Request.Address[ ] *does not contain any hops, then node A is only a single hop from the originator of the Route Request. Build a Route Reply packet as follows:*

REPPacket.IP.Source_Address	= *REQPacket.Request.Target_Address*
REPPacket.Reply.Target	= *REQPacket.IP.Source_Address*
REPPacket.Reply.OUT_Index[1]	= *REQPacket.Request.OUT_index*[1]
REPPacket.Reply.OUT_C_bit[1]	= *REQPacket.Request.OUT_C_bit*[1]
REPPacket.Reply.Address[1]	= *The home address of node A*

GOTO step 3.

2. Otherwise, build a Route Reply packet as follows:

REPPacket.IP.Source_Address	= *The home address of node A*

REPPacket.Reply.Target = *REQPacket.IP.Source_Address*

REPPacket.Reply.OUT_Index[1..*n*] = *REQPacket.Request.OUT_index*[1..*n*]

REPPacket.Reply.OUT_C_bit[1..*n*] = *REQPacket.Request.OUT_C_bit*[1..*n*]

REPPacket.Reply.Address[1..*n*] = *REQPacket.Request.Address*[1..*n*]

3. Send the Route Reply jittered by T milliseconds, where T is a uniformly distributed random number between 0 *and BROADCAST_JITTER* [7], [16].

2.6. Zone Routing Protocol (ZRP)

2.6.1. Introduction

Zone Routing Protocol is a mix of both proactive and reactive schemes. It is proactive with respect to the neighborhood and reactive with respect to the whole network. Each node has and maintains a routing zone comprising of its neighbors using the *MAC* layer protocols or through *MAC*-level *Neighbor Discovery* protocol. The routing zones of the neighbors overlap. *ZRP* maintains the zones using a protocol called *IntrAzone Routing Protocol* (*IARP*) where zones are defined by *Zone Radius*. *Zone radius* is nothing but number of hops. All nodes that can be reached within this zone radius belong to a zone. When a change occurs in the topology, *IARP* packets are exchanged and appropriate changes are made in the routing tables of the nodes in the zones. The packets are restricted from being transmitted to nodes outside a zone with the help of hop count. The *IntErzone Routing Protocol* (*IERP*) is responsible for acquiring routes to destinations that are located beyond the *routing zone*. The *IERP* uses route query (*RREQ*) / route reply (*RREP*) packets to discover a route to modes outside the Zone [9].

2.6.2. Message Summary

The *ZRP* is composed of proactive and reactive routing protocols. Proactive protocols use a *routing zone* which is defined in hops as the minimum distance from a node. *Distance Vector* (*DV*) *IARP message or Link State* (*LS*) *IntrAzone Routing Protocol* (*IARP*) message have the characteristic of proactive routing protocols. *ZRP IARP* node sends the *IARP* messages to nodes whose distance from the source node is exactly same as the radius of the source node's *routing zone*. This mechanism can reduce the packet control overhead compared to the conventional neighbor flooding. To send packets to the peripheral node, *ZRP* nodes use a series of IP unicast or an IP multicast, which is called *bordercasting*.

Distance Vector (*DV*) *IntErzone Routing Protocol* (*IERP*) message has the characteristic of a reactive routing protocol. *ZRP* nodes use the proactive maintenance within the routing zones through its proactive *IARP* messages and support the on-demand route determination between nodes located at distances beyond the routing zones by using reactive *IERP Query*/*Reply* messages. For example, when a destination is within a *routing zone*, the path to the destination is already known. If the destination is not within source's routing zone, the source bordercasts a *Route Query* to its peripheral nodes. All the peripheral nodes check if the destination is within their *routing zones*. If one of the peripheral nodes finds the destination, it replies by sending *Route Reply* towards the source. Otherwise, the peripheral nodes forward the query to its peripheral nodes, which performs the same routing zone algorithm. While bordercasting the *Route Query*, the accumulated route to the destination is stored in the query packet which is used for the destination to send back the *Route Reply* to the sender. The *Route Reply* packet follows the reverse sequence of IP addresses specified in the *Route Query* packet. To reduce the size of the *Route Query* in the intermediate nodes, the nodes can buffer the address information accumulated in the *Route Query*. Therefore, the address information can be removed from *Route Query* packet, which reduces the size of the *Route Query* packet and improves the query traffic and query response time.

Distance Vector (*DV*) *IARP message*

In the routing zone, each node periodically exchanges *DV IARP* message including the routing table of each node in order to update the network topology and calculate the shortest path.

0		1		2			3
0 1 2 3 4 5 6 7	8 9	0 1 2 3 4 5	6 7 8 9	0 1 2 3 4 5 6 7 8 9			0 1
Destination Address							
Destination Subnet Mask (*Optional*)							
Next Hop #1 *Address*							
Next Hop #2 *Address*							
Reserved	*Metric Type*		*Metric Value*				
…							
Reserved	*Metric Type*		*Metric Value*				
Hop Count							

Figure 10 ZRP Distance Vector IARP message

Refer to the field description in [9] for further details.

Link State (*LS*) *IARP message*

ZRP nodes depend on the *Neighbor Discovery*/*Maintenance* (*NDM*) protocol for the basic neighbor discovery. Each node knows the nodes within the *routing zone*. After discovering its neighbors, each node can calculate the shortest path to a certain destination node within the routing zone. To report the shortest path and *Link State* information between a source node and a destination node within the routing zone, *ZRP* nodes use *LS IARP* message. *Link Source Address* indicates the IP address of the source node which propagates the *Link State* update. *Link Destination Address* indicates the IP address of the destination node to which the *LS IARP Link State* update message should be delivered. *Packet Source Address* indicates the IP address of the intermediate node within the source's *routing zone* which relays the source's *LS IARP*. *Zone Radius* indicates the Routing *Zone Radius* of the link's source node.

```
0                                       1                                       2                                       3
0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5   6   7   8   9   0   1
```

Link Source Address			
Link Destination Address			
Packet Source Address			
Link State ID		*Zone Radius*	*Flags*
Link Destination Subnet Mask (*Optional*)			
Reserved	*Metric Type*	*Metric Value*	
Reserved	*Metric Type*	*Metric Value*	
…			
Reserved	*Metric Type*	*Metric Value*	

Figure 11 ZRP Link State IARP message

Refer to the field description in [9] for further details.

IERP message

The *IERP* messages are based on the query/reply mechanism, which are used to find out and maintain routes to source and destination nodes that are beyond a node's routing zone. *ZRP* nodes use source routes accumulated in the *IERP* packets and cached routes stored in *ZRP* nodes. The cached routes in *ZRP* nodes provide a faster response to find out the destination, because queries are terminated before reaching the query destination based on the cached routes of node which finds the query destination. The *Query Extension* can be implemented for the node which reported cached routes to inform the destination about getting the reverse path towards the source. The requested *Quality of Service* (*QoS*) may need the explicit source route to propagate the *QoS* packets, which is provided by the source route mechanism from the *IERP* packets. The type in *IERP* packet indicates the type of *IERP* packet which can be *Route Query*, *Query Extension* or *Route Reply*. *TTL* shows the number of hops that a *Route Query* may continue to propagate. Hop count indicates the hop count from the source to the current node in *Route Query, Query Extension* messages or the hop count from the source to route destination in *Route Reply*, *Route Accumulation* and *Route Optimization* messages [9].

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Type	*TTL*	*Hop Count*	*Flags*
Current	*ROF Ptr*	*Num Dests = D*	*Num Nodes = N*
Query ID		*Reply ID*	
Query/Route Source Address			
Replying Node Address			
Bad Link Source Address			
Query/Route Destination (1) *Address*			
Query/Route Destination (2) *Address*			
…			
Query/Route Destination (*D*) *Address*			
Next IERP Address			
Next BRP Address			
Prev IERP Address			
Intermediate Node (1) *Address*			
Intermediate Node (2) *Address*			
…			
Intermediate Node (*N*) *Address*			
Node	*Metric Type*	*D*	*Metric Value*
Node	*Metric Type*	*D*	*Metric Value*
…			
Node	*Metric Type*	*D*	*Metric Value*
Route Optimization Flags (*Node* 0 == *Source*)			
Route Optimization Flags (*Node* 1)			
…			
Route Optimization Flags (*Node N*)			
Route Optimization Flags (*Node N* +1== *Dest*)			

Figure 12 ZRP IERP message

Refer to the field description in [9] for further details.

Bordercasting Resolution Protocol

To forward *IERP* route queries, *ZRP* nodes use *BRP* providing the *bordercasting* message delivery service. A bordercasting node relays the *BRP* messages to its peripheral nodes.

2.6.3. Pseudocode

Update Intrazone Routing Table

```
if (packet arrived)
    {host, route->next_hop,route->hop_count} <-- packet
else
{
        {host} <-- intrpt
        route->next_hop=host
        if (type(intrpt) == "Neighbor Found")
        // indicates that direct contact with the host has been confirmed
        {
                for recorded_host = each host in Intrazone_Routing_Table
                {
                        best_route = Intrazone_Routing_Table[recorded_host,0]
                        if (best_route->hop_count <  ROUTING_ZONE_RADIUS)
```

```
                    {
                              packet <-- {recorded_host,my_id,hop_count+1}
                              send(packet,host)
                              // send is the standard function used in IP
                    }
               }
          route->hop_count=1
     }
     else
     // Neighbor lost-> used by the neighbor Discover/Maintenance
     //Protocol to indicate that direct with the connection host is lost.
               route->hop_count=INF
}

former_best_route = Intrazone_Routing_Table[host,0]
if (route->hop_count < INF)
     add(Intrazone_Routing_Table[host], route)
else
     remove(Intrazone_Routing_Table[host],route)
  best_route = Intrazone_Routing_Table[host,0]
  if (best_route != NULL)
  {
          if (best_route->hop_count != former_best_route->hop_count
               && best_route->hop_count < ROUTING_ZONE_RADIUS)
          {
              packet <-- {host, my_id, best_route->hop_count+1}
              broadcast(packet)
          }
     }
  else
  {
          force_intrpt("IERP","Host Lost",{host})
              // "Host Lost" is used by IARP to notify IERP that node no longer exists
          packet <-- {host, my_id, INF}
          broadcast(packet)
     }
```

Intrazone Node Lost

```
{lost_host} <-- intrpt
for host = each host in Interzone Routing Table
{
    m=0
    while (Interzone_Routing_Table[host,m] != NULL)
    {
        route=Interzone_Routing_Table[host,m]
```

```
          if (lost_host (EXISTS IN) route)
                remove(Interzone_Routing_Table[host,m])
            else
                m++
    }
}
force_intrpt("IERP","repair",{lost_host})
```

Initiate Route Discovery

```
{dest} <-- intrpt
req_id++
route(0)=my_id
last_hop=0
if (type(intrpt) == "repair")
    max_hops=MAX_REPAIR_HOPS
else
    max_hops=MAX_REQUEST_HOPS

packet<-- {REQUEST, req_id, last_hop, NULL, max_hops, dest, route}
bordercast(packet)
add (Processed_Request_List, {my_id, req_id})
```

Report Route Failure

```
{route,dest} <-- intrpt
```

```
last_hop=0
while (route(last_hop) != my_id)
    last_hop++
packet <-- {FAILURE, NULL, last_hop, last_hop, NULL, dest, route}
send(packet, route(last_hop-1))
```

Receive IERP Packet

```
{pk_type, req_id, last_hop, bad_hop, max_hops, route} <-- packet
{overheard} <-- intrpt
switch(pk_type)
{
case:  REQUEST
    add({Processed_Request_List, source, req_id)
    LSP1_terminate = FALSE
    n=0
    while (!LSP1_terminate && n < last_hop)
    {
        if (Intrazone_Routing_Table[route(n)]!=NULL)
            LSP1_terminate = TRUE
        n++
    }
```

```
LSP2_terminate = (Processed_Request_List[source,req_id]!= NULL)
if (!overheard && !LSP1_terminate && !LSP2_terminate &&
        last_hop < max_hops)
{
      last_hop++
      route(last_hop)=my_id
      if (dest (EXISTS IN) Intrazone_Routing_Table)
      {
            packet<--{REPLY,req_id,last_hop,bad_hop,max_hops,
                         dest,route}
            send(packet, route(last_hop-1))
      }
      else
      {
            packet<--{REQUEST,req_id,last_hop,bad_hop,max_hops,
                         dest,route}
            bordercast(packet)
      }
}
break
```

case: ***REPLY***

case: ***FAILURE***

```
          if (route(0) == my_id)
        {
          if (pk_type == ROUTE_REPLY)
                      add(Interzone_Routing_Table, route)
          else
                {
                      link(0)=route(bad_hop)
                      link(1)=route(bad_hop+1)
                      remove(Interzone_Routing_Table,link)
                }
        }
        else
        {
          last_hop --
                packet <-- {pk_type,req_id,last_hop,bad_hop,max_hops,
                            dest,route}
                send(packet, route(last_hop-1))
        }
}
```

2.7. Optimized Link State Routing Protocol (OLSR)

2.7.1. Introduction

The *OLSR* protocol uses *Multipoint relays* (*MPR*). *MPRs* are the set of nodes that connect the current node to nodes that are 2 hops away. This *MPR* set information is passed to the neighbors. By using *MPRs*, the number of *Link State Updates* and the length of the *Link State Updates* are reduced. The length of the *Link State Updates* are reduced because not all the links in the network are advertised and only the links that lead to the *MPR* set are sent to the neighbors. *OLSR* is optimal for dense networks rather than for sparse networks. In a fast changing mobile environment, the frequency at which the control packets are transferred can be increased in *OLSR* [1].

2.7.2. Message Summary

Optimized Link State Routing Protocol (OLSR) node selects *Multipoint Relays* (*MPRs*) which are the nodes within one hop range of the node. Only the *MPR* set can resend the node's broadcast messages to reduce the control overhead in the classical flooding mechanism. The neighbors which are not selected as the *MPR* node receive and process the node's broadcast messages but do not retransmit the messages.

Hello message

The node selects the *MPR* set based on the value of the willingness field of *Hello* message flooded from the node's neighbors. *OLSR* has characteristics of a proactive and a hop-by-hop routing algorithms for mobile ad-hoc network (MANET), where the route could be prepared immediately at the *OLSR* node and each *OLSR* node should decide the next hop to forward the data packet based on its local routing information, without any information of source route in the header of the data packet. A *MPR* selector node can use a *Hello* message to inform its neighbors of a new *MPR* set if *MPR* selector node selects another *MPR* set due to a change in the previous *MPR* set over time.

0 0 1 2 3 4 5 6 7	 8 9	1 0 1 2 3 4 5	 6 7 8 9	2 0 1 2 3	 4 5 6 7 8 9	3 0 1
Reserved			*Htime*		*Willingness*	
Link Code	*Reserved*		*Link Message Size*			
Neighbor Interface Address						
Neighbor Interface Address						
…						
Link Code	*Reserved*		*Link Message Size*			
Neighbor Interface Address						
Neighbor Interface Address						
…						

Figure 13 OLSR Hello message

Refer to the field description in [1] for further details.

Topology Control message

Topology Control (*TC*) message advertises the all the main addresses of the *MPR* selectors of the originator node.

0 0 1 2 3 4 5 6 7 8 9 1 0 1 2 3 4 5	6 7 8 9 2 0 1 2 3 4 5 6 7 8 9 3 0 1
ANSN	*Reserved*
Advertised Neighbor Main Address	
Advertised Neighbor Main Address	
…	

Figure 14 OLSR Topology Control message

Refer to the field description in [1] for further details.

Host and Network Association message

OLSR node uses a *Host and Network Association* (*HNA*) message to advertise the interface addresses of the associated networks and hosts that do not support the *OLSR* routing protocol to other nodes in the MANET network.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|                        Network Address                        |
+---------------------------------------------------------------+
|                            Netmask                            |
+---------------------------------------------------------------+
|                        Network Address                        |
+---------------------------------------------------------------+
|                            Netmask                            |
+---------------------------------------------------------------+
|                              ...                              |
+---------------------------------------------------------------+
```

Figure 15 OLSR Host and Network Association message

Refer to the field description in [1] for further details.

2.7.3. Pseudocode

Thus, upon receiving a basic packet, a node performs the following tasks for each encapsulated message:

1. *If the time to live of the message is less than or equal to*'0' (*zero*), *the message MUST silently be dropped.*
2. *If there exists a tuple in the duplicate set, where:*

   D_addr == Originator Address, AND

   D_seq_num == Message Sequence Number

   then the message has already been completely processed and MUST silently be ignored.
3. *Otherwise, if the node implements the Message Type of the message, the message MUST be processed according to the specifications for the message type.*

4. *Otherwise, if the node does not implement the Message Type of the message the message SHOULD be processed according to the following algorithm:*

1. *If the sender interface address of the message is not detected to be in the symmetric neighborhood of the node, the message MUST silently be dropped.*
2. *If the sender interface address of the message is detected to be in the symmetric neighborhood of the node, an entry in the duplicate set is recorded with:*

   D_addr = originator address

   D_seq_num = Message Sequence Number

   D_time = current time + D_HOLD_TIME.
3. *If the sender interface address is an interface address of a MPR selector of this node and if the time to live of the message is greater than '*1*', the message MUST be forwarded according to the following:*

   3.1 *The TTL of the message is reduced by one.*

   3.2 *The hop-count of the message is increased by one*

   3.3 *The message is broadcasted on all interfaces*

The Neighbor Set should be updated as follows:

1. *Upon receiving a HELLO message, if there exists no neighbor tuple with N_if_addr == Sender Interface Address and N_if_id == identifier of the Receiver Interface, a new tuple is created with*

   N_if_addr = Sender Interface Address

   N_if_id = identifier of the Receiver Interface

N_SYM_time = *current time* - 1 (*expired*)

N_time = *current time* + *NEIGHB_HOLD_TIME*

2. *The tuple* (*existing or new*) *with:*

    N_if_addr == *Sender Interface Address and*

    N_if_id == *identifier of the Receiver Interface,*

    is then modified as follows:

    2.1 *N_main_addr* = *Originator Address.*

    2.2 *N_willing* = *Originator Willingness*

    2.3 *N_time* = *max* (*N_time, current time* + *NEIGHB_HOLD_TIME*);

    2.4 *N_ASYM_time* = *current time* + *NEIGHB_HOLD_TIME;*

    2.5 *if the node finds the Receiver Interface Address among the addresses listed in the HELLO with:*

    Link Interface Address == *Sender Interface Address, then, the tuple is modified as follows:*

    if Link Type == *LOST_LINK then N_SYM_time* = *current time* - 1 (*i.e. expired*)

    else:

    N_SYM_time = *current time* + *NEIGHB_HOLD_TIME,*

    N_time = *current time* + 2 **NEIGHB_HOLD_TIME.*

The rule for setting N_time is the following: a link loosing its symmetry should still be advertised during at least NEIGHB_HOLD_TIME. This allows neighbors to detect the link breakage.

The 2*-hop Neighbor Set is updated as follows:*

1. *For each* 2*-hop interface address listed in the HELLO message with Link Type SYM_LINK or MPR_LINK, a* 2*-hop tuple is created with:*

   N_main_addr = Originator Address;

   N_if_addr = Link Interface Address corresponding to the 2*-hop interface address;*

   N_2hop_addr = the interface address of the 2*-hop neighbor;*

   N_time = current time + 2*HOP_HOLD_TIME.*

   N_2hop_main_addr = the main address of the node, extracted from the multiple interface association information base; if no address is available, the interface address N_if_addr is used.

   This tuple may replace an older similar tuple with same N_if_addr and N_2hop_addr values.

2. *For each* 2*-hop interface address listed in the HELLO message with Link Type LOST_LINK or ASYM_LINK, all the* 2*-hop tuples where:*

   N_if_addr == Link Interface Address corresponding to the 2*-hop interface address, and*

   N_2hop_addr == the 2*-hop interface address are deleted.*

Based on the information obtained from the HELLO messages, each node constructs its MPR selector set.

Thus, upon receiving a HELLO message, if a node finds one of its interface addresses in a list with a link type of "MPR", it MUST update the MPR selector set to contain updated information about the sender of the HELLO message:

1. *If there exists no MPR selector tuple with:*

   MS_if_addr   == Link Interface Address and

   MS_main_addr == Originator Address

2. *If there exists no MPR selector tuple with:*

   MS_if_addr   == Link Interface Address then a new tuple is created with:

   MS_if_addr   = Link Interface Address

3. *The tuple is then modified as follows:*

   MS_main_addr = Originator Address,

   MS_time   = current time + NEIGHB_HOLD_TIME [1].

2.8. Landmark Ad Hoc Routing (LANMAR)

2.8.1. Introduction

LANMAR is a proactive routing scheme, which combines the features of *Fisheye State Routing* (*FSR*) and *Landmark Routing*. Landmarks are used for each set of nodes that move as a group. A packet that has to reach a remote destination initially aims at the landmark of the remote group and then as it gets closer to the landmark, it switches to *Fisheye State Routing* that is more accurate. Both scalability and mobility problems are solved as the line and storage overhead are kept low.

Each node has a unique physical identifier such as a 48-bit Ethernet address, a unique logical identifier consisting of a subnet field and a host field and a landmark *ON/OFF* flag. The flag shows whether the node is a landmark or not. There is a list and three tables for each node *i*. There is a neighbor list A_i , a topology table TT_i , a next hop table $NEXT_i$ and a distance table D_i. TT_i consists of two parts – link state information provided by a node *j* and a time stamp indicating the age of the link state information [6].

2.8.2. Landmark Election

Initially no landmarks exist. *FSR* functionality is used and a node will then learn from the *FSR* table that a particular number of group members are in the group. This node then becomes the landmark of the group. In case more than one node declares itself as the landmark, the node whose group has a larger number of members wins. In case of a tie, the lowest ID wins. A landmark may lose its role as the structure of mobile networks keeps changing. Death of landmarks is detected using timeouts. This results in a new landmark election.

2.8.3. Message Summary

Landmark Routing Protocol (*LANMAR*) uses landmarks which are selected in each logical subnet. Each node knows routes to all nodes within a scoped range predefined by the maximum number of hops. Above the scoped range, a node sends a packet to a landmark node representing a destination subnet. When the landmark node receives the packet, it forwards the packet to the destination node in its subnet. The routing table in each node includes the nodes within the scope and the landmark nodes to reduce the routing traffic overhead. The routing update packets are propagated only within the scoped area. *LANMAR* uses only one *LANMAR Update* (*LMU*) message. *N_landmarks* field indicates the number of landmarks in a subnet. *N_drifters* field indicates the number of drifters in a subnet. *Landmark Address* 1 field indicates the IP address of a landmark of a subnet. *Next Hop Address* field indicates one of the neighboring nodes of the *landmark* node. *Distance* 1 field indicates the distance between a *landmark* and *Next Hop Address* 1. *Drifter Address* 1 field indicates the IP address of the drifter of a subnet.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Landmark Flag	*N_landmarks*	*N_drifters*	*Reserved*
Landmark Address 1			
Next Hop Address 1			
Distance 1	*N_numbers* 1	*Sequence Number* 1	
…			
Drifter Address 1			
Next Hop Address 1			
Distance 1	*Drifter Sequence Number* 1		…
…			

Figure 16 LANMAR Update message

Refer to the field description in [6] for further details.

2.8.4. Pseudocode

```
//Choosing landmarks
choose-landmark(id,level)
    selected = -1
    closest = 0
    for each landmark l in DV table
        if(l.level == level)
        d=abs(hash(l.id)-hash(id))
        if (d<closest or selected == -1)
        closest = d
        selected = l.id
    return selected

//Choosing childs for landmarks
choose-child(id,parent,level)
    selected = -1
    closest = 0
    for each landmark l in DV table
        if(l.level == level and l.parent == parent)
        d=abs(hash(l.id)-hash(id))
        if (d<closest or selected == -1)
        closest = d
        selected = l.id
    return selected

//Print landmark table.
```

```
RoutingLanmarPrintLandmarkTable(Node *node, LanmarData *lanmar)
start
    char addrString[MAX_STRING_LENGTH]
    char clockStr[MAX_STRING_LENGTH]
    LanmarLandmarkTableRow *row

    row = lanmar->landmarkTable.row
    print "Landmark table content:"
    begin if (row == NULL)
            print "empty"
    end if
    begin while (row != NULL)
         IO_ConvertIpAddressToString(row->destAddr, addrString)
        Print “destAddr = ", addrString
        IO_ConvertIpAddressToString(row->nextHop, addrString)
        Print "nextHop = ", addrString
                        assert(row->nextHop != 0);
        print "numLandmarkMember =”,
               row->numLandmarkMember)
        print "distance = ",
               row->distance
        print "sequenceNumber = ",
               row->sequenceNumber
        ctoa(row->timestamp, clockStr)
        print "timestamp =", clockStr
        row = row->next;
```

```
     end while
end


//Print drifter table.
RoutingLanmarPrintDrifterTable(Node *node, LanmarData *lanmar)
begin
    char addrString[MAX_STRING_LENGTH]
    char clockStr[MAX_STRING_LENGTH]
    LanmarDrifterTableRow *row
    row = lanmar->drifterTable.row
    print "Drifter table content:"
    begin if (row == NULL)
            print "empty"
    end if
    begin while (row != NULL)
        IO_ConvertIpAddressToString(row->destAddr, addrString);
        Print "destAddr = ", addrString
        IO_ConvertIpAddressToString(row->nextHop, addrString);
        Print "nextHop =", addrString
        Print "distance = ",
               row->distance
         Print "sequenceNumber = ",
               row->sequenceNumber
        ctoa(row->lastModified, clockStr)
         Print "timestamp = ", clockStr
              row = row->next
    end while
end [6]
```

2.9. Location-Aided Routing (LAR)

2.9.1. Introduction

In Location-Aided Routing (*LAR*), location information is used to narrow down the routing area to a "*Request zone*". This reduces the routing overhead. The *Global Positioning System* (*GPS*) may be used to provide the required location information. In the real world, some amount of error is associated with the location information. The errors that occur are the differences between the actual coordinates that give the location and the coordinates calculated by *GPS*. Assume that there is a node *S* that needs to communicate with a destination node *D*. Suppose *S* knows the location *L* of *D* at time t_0, then the "*Expected zone*" of the node *D* can be estimated at time t_1 and is given by the circular region of radius $v(t_1 - t_0)$ centered at location *L*, where *v* is the average speed of node *D*. The "*Request zone*" is usually larger than the "*Expected zone*" to increase the probability of the message reaching the destination.

2.9.2. Message Summary

LAR uses the *GPS* as the *Route Discovery* mechanism to limit the number of flooding messages of route discovery. It is assumed that a source *S* wants to find a route to a destination *D*. At time t_0, *S* knows the location *L* of *D* by using *GPS* position information. If *D* moves with velocity *v*, at time t_1, S could expect the location of *D* by using the equation of $v(t_1 - t_0)$. The circular region with radius of $v(t_1 - t_0)$ centered at the location L is the *Expected Zone* of D. Therefore, *Route Request* message from S can flood to this limited *Expected Zone* to find out the route to D. Only the nodes located in the *Request Zone* can forward the *Route Request* to D. The nodes outside the *Request Zone* discard the *Route Request*. When D receives the *Route Request*, D replies by sending a *Route Reply* towards S. The path of *Route Reply* follows the reverse path of the *Route Request*. After S finds the route to D, S starts sending data packet. If the intermediate node finds that its next hop is broken, it generates the *Route Error* message and the *Route Error* message is forwarded to S. The exact formats of *LAR* messages are not defined yet.

2.9.3. Pseudocode

```
//Flooding
check
begin if
    node $\in$ request zone
     forward request to node
else
     do not forward
end if

//s is the source and i is the next hop, $\delta$ is a parameter
nexthop($\delta$,d)
begin
            begin if
```

```
        $d_s + \delta >= d_i$
        request forwarded
        else
        request discarded
        end if
end
```

2.10. Greedy Perimeter Stateless Routing (GPSR)

2.10.1. Introduction

Greedy Perimeter Stateless Routing (*GPSR*) uses only positions of a router’s immediate neighbors and a packet’s destination to make routing decisions. Greedy forwarding is implemented in cases where Perimeter forwarding is not possible. Geographic routing helps routers to be almost stateless, they just have to know the neighboring routers and they need not have any other information on network topology.

The Greedy Perimeter Stateless Routing comprises of two methods for forwarding packets – *greedy forwarding* and *perimeter forwarding*. *Greedy forwarding* is used as much as possible and only in cases where *Perimeter forwarding* is not possible.

By keeping only the local topology state, *GPSR* scales better in per-router state than shortest-path and ad hoc routing protocols as the number of network destinations increases. When topology changes frequently due to mobility, *GPSR* can use local topology information to find the new correct routes quickly.

2.10.2. Message Summary

Greedy Perimeter Stateless Routing (*GPSR*) uses a *Beacon* message piggybacked by data packets to update a node's topology with node's neighbors only. *GPSR* uses *GPS* as the *Route Discovery* mechanism. The advantage of using *GPS* is that a source node trying to find a route to a destination knows the coordinates of the destination's physical location in advance. Therefore, the source can use the position information of the designation to calculate the shortest path from the one of its neighbors to the destination. The source node knows its neighbor information by using periodical *Beacon* message. Therefore, the source can select the one of its neighbor nodes which provides the source node with the shortest path between the neighbor and the destination. The format of *GPSR Beacon* message is not defined yet.

2.10.3. Pseudocode

```
//comparing neighbors
function static int NeighbEntCmp(const void *a, const void *b)
start
  nsaddr_t ia = ((const NeighbEnt *) a)->dst
  nsaddr_t ib = (*(const NeighbEnt **) b)->dst
  begin if
          (ia > ib) return 1
  end if
  begin if
          (ib > ia) return –1
  end if
  return 0
end

function NeighbTable(GPSR_Agent *mya)
start
  int i
  nents = 0
  maxents = 100
  tab = new (NeighbEnt *)[100]
  a = mya
  for (i = 0; i < 100; i++)
    tab[i] = new NeighbEnt(a)
  end for
end

begin case
    switch (mode)
     case GPSRH_BEACON:
      begin if (address())
            // don't receive my own beacons; I'm not my neighbor
            beacon(p, gpsrh)
      break
    case GPSRH_PPROBE:
      peri(p, gpsrh)
      break
    case GPSRH_DATA_GREEDY:
      as = Address::instance().print_nodeaddr(addr_)
      print(stderr, "greedy data pkt @ %s:GPSR_PORT!\n",  as)
      fflush(stderr)
      delete[ ] as
      break
    case GPSRH_DATA_PERI:
      as = Address::instance().print_nodeaddr(addr_)
      print(stderr, "peri data pkt @ %s:GPSR_PORT!\n",  as)
      fflush(stderr)
      delete[ ] as
      break
    default:
      as = Address::instance().print_nodeaddr(addr_)
      print(stderr, "unk pkt type %d @ %s:GPSR_PORT!\n", gpsrh->mode_, as)
      fflush(stderr)
      delete[ ] as
      break

  end case
```

```
 else
   forward Packet
```

2.11. Ad hoc On Demand Distance Vector (AODV)

2.11.1. Introduction

AODV is capable of both unicast and multicast routing. *AODV* uses sequence numbers to ensure the freshness of routes. It is self starting, loop-free, and scales to large numbers of mobile nodes. AODV builds routes using a route request / route reply query cycle. When a source node desires a route to a destination for which it does not already have a route, it broadcasts a *Route Request* (*RREQ*) packet across the network. Nodes receiving this packet update their information for the source node and set up backwards pointers to the source node in the route tables. In addition to the source node's IP address, current sequence number, and broadcast *ID*, the *RREQ* also contains the most recent sequence number for the destination of which the source node is aware. A node receiving the *RREQ* may send a *Route Reply* (*RREP*) if it is either the destination or if it has a route to the destination with corresponding sequence number greater than or equal to that contained in the *RREQ*. If this is the case, it unicasts a *RREP* back to the source. Otherwise, it rebroadcasts the *RREQ*. Nodes keep track of the *RREQ*'s source IP address and broadcast *ID*. If they receive a *RREQ* that they have already processed, they discard the *RREQ* and do not forward it.

As the *RREP* propagates back to the source, nodes set up forward pointers to the destination. Once the source node receives the *RREP*, it may begin to forward data packets to the destination. If the source later receives a *RREP* containing a greater

sequence number or contains the same sequence number with a smaller hop count, it may update its routing information for that destination and begin using the better route.
The route is maintained as long as there are data packets periodically being sent from the source to the destination along that path. Once the source stops sending data packets, the links will time out and eventually be deleted from the intermediate node routing tables. If a link break occurs while the route is being used for transmission of data, the node upstream of the break propagates a *Route Error* (*RERR*) message to the source node to inform it of the now unreachable destination(s). After receiving the *RERR*, if the source node still desires the route, it can reinitiate route discovery [2].

2.11.2. Message Summary

Ad hoc On-demand Distance Vector (*AODV*) routing protocol is composed of a *Hello*, a *Route Request* (*RREQ*), a *Route Reply* (*RREP*), a *Route Error* (*RERR*), and a *Route Reply Acknowledgement* (*RREP-ACK*) message. To reduce the data packet overhead, *AODV* nodes store routing information in the node routing table instead of using source route added into the data packet such as in *DSR*. *AODV* nodes check the link status of next hops in active routes for the route maintenance. If a link break is detected, the node which finds the link break sends a *RERR* message to notify other nodes that the link was broken.

Route Request message

An *AODV* node increases the sequence number by one whenever the node triggers an action to do *RREQ*, *RREP*, *RERR*, or *RREP-ACK*. It is assumed that when a source node finds a route towards a destination, previous valid route to the destination is expired in its routing table, and it does not have a valid route to the destination. The destination sequence number in *RREQ* message is copied from the source's routing table which stores the last known destination sequence number. If a source node generating a *RREQ* message does not know the sequence number of the destination, it sets the *U* bit (*unknown sequence number*) in its *RREQ* message. The field of *RREQ ID* is increased by one from the last *RREQ ID* issued by the source node. When the source node receives the *RREQ* message again from its neighbors due to the local broadcasting property of MANET, it just discards the *RREQ* message after comparing the *RREQ ID* and originator IP address in the received *RREQ* message and the *RREQ ID* and originator IP address stored in the source node's *Path_Discovery_Time* buffer.

To make the bidirectional communication between a source node and a destination node, the source node must not only know the route to the destination, but also the destination node should know a route back to the source node. As one of *RREQ* propagation scenarios, one of the intermediate nodes having a valid route to the destination can reply by sending a *RREP*, therefore, the destination node does not receive the *RREQ* message from the source node and can not make a route back to the source node. To compensate this situation, the source node sends the *RREQ* message including the bit of *G* (*gratuitous RREP flag*) that is set, which notifies the intermediate node generating the *RREP* message to unicast a gratuitous *RREP* to the destined destination node. To find out a destination, a source node uses an expanding ring search algorithm to avoid the network-wide dissemination of *RREQ*s, which can be implemented by the *TTL* value in the *RREQ* IP header. If a source node does not receive a *RREP* message, it resends a *RREQ* message with the *TTL* increased by *TTL_Increment*. This will continue until the value of *TTL* in the *RREQ* reaches *TTL_Threshold*. The first-in, first-out (*FIFO*) scheme is used to buffer the data packet which waits for a *RREP* after a *RREQ* has been sent.

A source node should wait for a *RREP* message by using a binary exponential backoff mechanism to reduce network congestion. If the source node does not receive a *RREP* message within the *Net_Traveral_Time* after sending a *RREQ* message, the source node resends the second *RREQ* message. In this case, the source node should wait for the *RREP* message for a duration of 2**Net_Traveral_Time* which is two times longer than the first *Net_Traveral_Time*. If the source node does not receive the *RREP* message, it can resend the third *RREQ* message up to *RREQ_Retries*. For this case, the new waiting time for the source node is calculated by multiplying 2 into the previous waiting time, which is 4* *Net_Traveral_Time*. When an intermediate node receives a *RREQ* message, it checks whether it receives a *RREQ* with the same *Originator* IP Address and *RREQ ID*. It discards the *RREQ* if it receives such a *RREQ*. If it did not receive such a *RREQ* before, it first increases the value of the hop count field in the received *RREQ* by one, then it search a reverse route to the *Originator* IP Address. If it finds a reverse route to the *Originator* IP Address, the sequence number of the route in its routing table is copied from the value of the current sequence number of the *RREQ*. When the intermediate node receives a *RREP* message for the response of the *RREQ* message, it should have a reverse route to send the received *RREP* message towards the *Originator* IP Address. To carry out the refresh mechanism of the reverse route, when an intermediate node receives a *RREQ* message, it sets the value of the lifetime of the reverse route entry for the *Originator* IP Address as the maximum of (*ExistingLifeTime*, *MinimalLifeTime*).

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

Type	*J*	*R*	*G*	*D*	*U*	*Reserved*	*Hop Count*
RREQ ID							
Destination IP Address							
Destination Sequence Number							
Originator IP address							
Originator Sequence Number							

Figure 17 AODV Route Request message

Refer to the field description in [2] for further details.

2.11.3. Route Reply message

When a destination receives a *RREQ* message or an intermediate node which has an active route to a destination, it should respond by sending a *RREP* message towards the source node. The following section describes the case that the destination receives the *RREQ*. The fields of *Destination IP Address* and the *Originator IP Address* of the *RREP* message come from the corresponding fields in the *RREQ* message. The unicast route for the *RREP* towards the source follows the reverse path from which the *RREQ* is delivered. The value of the *Hop Count* field is increased by one at each intermediate node towards the source. The value of *My_Roure_Timeout* in the destination is copied into the value of *Lifetime* field. When the intermediate node sends the *RREP* message, the field value for the *Destination Sequence Number* comes from the value of its *destination sequence number* which can be extracted from its routing table. The value of the *Hop Count* field is calculated from the distance in hops from the destination to the intermediate node. The value obtained from subtracting the current time from the expiration time in the route table entry can be the value of the *Lifetime* field. The *forward route entry* in the precursor list is copied from the source IP address of the received *RREQ*, which indicates the last hop node from which the intermediate node receives the *RREQ*. The *reverse route entry* in the precursor list is copied from the next hop towards the destination stored in its routing table.

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+-+-+-----------------+---------+---------------+
|     Type      |R|A|    Reserved     |Prefix Sz|   Hop Count   |
+---------------+-+-+-----------------+---------+---------------+
|                    Destination IP Address                     |
+---------------------------------------------------------------+
|                 Destination Sequence Number                   |
+---------------------------------------------------------------+
|                    Originator IP Address                      |
+---------------------------------------------------------------+
|                           Lifetime                            |
+---------------------------------------------------------------+
```

Figure 18 AODV Route Reply message

Refer to the field description in [2] for further details.

Route Error message

Whenever a node detects a failure of next link in an active route, it should generate a *RERR* message which is broadcasted if there are many precursors or unicasted if there is only one precursor. The fields of the *Unreachable Destination IP Address* and *Unreachable Destination Sequence Number* are the *Destination IP Address* and *Destination Sequence Number* in an active route where the next link of this node is broken to its neighbor. The node runs a local recovery and it does not receive any *RREP* message within a recovery period; it informs a source node of the link failure destined for the destination by sending a *RERR* message.

```
0                                       1                                       2                                       3
0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5   6   7   8   9   0   1
```

Type (bits 0–7)	*N* (bit 8)	*Reserved* (bits 9–23)	*DestCount* (bits 24–31)
Unreachable Destination IP Address (1)			
Unreachable Destination Sequence Number (1)			
Additional Unreachable Destination IP Address (*if needed*)			
Additional Unreachable Destination Sequence Number (*if needed*)			

Figure 19 AODV Route Error message

Refer to the field description in [2] for further details.

Route Reply Acknowledgment message

Whenever the node receives a *RREP* message which sets the bit of '*A*' field, it should respond with *Route Reply Acknowledgement* message.

```
0                                       1
0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5
```

Type (bits 0–7)	*Reserved* (bits 8–15)

Figure 20 AODV Route Reply Acknowledgment message

Hello message

An *AODV* node, which is one of the members on an active route, can use either an appropriate layer 2 message or a *RREQ* with *TTL* value of one in the *RREQ* IP header, which can be used as a *Hello* message. *AODV* nodes should send a *RREQ* or an appropriate layer 2 message for every *Hello_Interval* to refresh the active link to its neighbor node. If it does not send any *RREQ* message within the last *Hello_Interval*, it should broadcast a *RREP* with *TTL* value of one with the following modifications, which is called as an *AODV Hello* message. The field value of *Destination IP Address* in *RREP* message is the node's IP address. The field value of *Destination Sequence Number* is the node's latest sequence number. The field value of *Hop Count* is zero. The field value of

Lifetime is *Allowed_Hello_Loss * Hello_Interval*. If a node does not receive any *AODV Hello* or other messages from its neighbors within *Allowed_Hello_Loss * Hello Interval*, it can decide the link to its neighbor is broken.

2.11.4. Pseudocode

```
Action taken at a node that desires to forward data Packets
1. Check if there is route to the node of interest
2. If yes, send the message
3. If no, Broadcast a RREQ Packet through the Network
4. And wait for a RREP to forward the data packets
5. Upon receiving the RREP forward the data packets to the destination
6. If another RREP is received with a greater sequence number or same sequence number with a smaller hop count, start using this better route

Action taken at a node on receiving a RREQ Packet
1. Update the information for the source node
2. Set up backwards pointers to the source node in the route tables
3. Check if the current node is the destination or it has a route to the destination with corresponding sequence number greater than or equal to that contained in the RREQ
4. If yes, send a unicast RREP back to the source and set the forward pointers to the source.
5. If no, rebroadcast the RREQ.
```

6. *Keep track of the RREQ's source IP address and broadcast ID.*

7. *If the received RREQ has been already processed, discard the RREQ and do not forward it.*

2.12. Clusterhead Gateway Switch Routing (CGSR)

2.12.1. Introduction

This protocol is a multicast protocol based on the *Core Based Tree* approach for Internet. In this protocol multicast groups are initiated and maintained by a multicast server. Whenever a node wants to join a multicast group it will query the server. All the entries about the members of a particular multicast group are maintained in the server. The construction and maintenance of the tree is by using '*graft*' and '*prune*' messages [13], [17].

2.12.2. Message Summary

The exact format of *CGSR* message is not defined yet.

2.12.3. Pseudo code for electing the Cluster head

```
Check if the member is within the transmission range of the node of reference
If yes then the member falls into the same cluster
If no it falls into a different cluster

//Code to elect the Clusterhead
```

```
For each node in the cluster
    Is the ID of the node lowest?
    If yes store the value as possible clusterhead
    If no proceed to the next node
Next node

Set the cluster head as the node that is stored as the possible clusterhead

Criterion for changing the cluster is
If two cluster heads come within range of each other
If the existing clusterhead gets disconnected [13], [17]
```

2.12.4. Pseudocode for the CGSR algorithm

1. *Use lowest-id cluster algorithm or highest-connectivity cluster algorithm to create initial clusters.*

2. *When a non-clusterhead node in cluster i moves into a cluster j, no clusterhead in cluster i and j will be changed* (*only cluster members are changed*).

3. *When a non-clusterhead node moves out its clusters and doesn't enter existing cluster,*

4. *It becomes a new clusterhead, forming a new cluster.*

When clusterhead C(*i*) *from cluster i moves into the cluster j, it challenges the corresponding clusterhead C*(*j*)*. Either C*(*i*) *or C*(*j*) *will give up its clusterhead position according to lowest-id or highest-connectivity* (*or some other well defined priority scheme*).

5. *Nodes which become separated from a cluster will recompute the clustering according to lowest-id or highest-connectivity* [13], [17].

2.12.5. Pseudocode for channel access

1. *Initially, the clusterhead gets the permission token to access the radio channel. It transmits any messages it has in its transmission queue.*
2. *The clusterhead passes the token to one of its neighbors according to a separately defined scheduling algorithm.*
3. *The cluster node* (*regular node or gateway*) *returns the token to its clusterhead after it has transmitted its message*(*s*) (*if any*).
4. *Repeat* 1 *to* 3 [13], [17].

2.12.6. Multicast Tree graft and prune

The construction and maintenance of the *Core Based Tree* is receiver-oriented. When node *i* wants to join a multicast group *G*, it first gets the corresponding *Mid* and *MSid* either from its database or from the directory server. Then, it sends a *Join Request* to *MSid*. The *Join Request* will be routed to *MSid* (core) , using *CGSR,* until it reaches any node *j* which is already a member of the host group of *G*. Node *j* terminates the *Join* process by sending a *Join Ack* back to node *i*. A node joins the multicast group and grafts a branch to the multicast tree (core-based tree) upon being traversed by *Join Ack.* Since *CGSR* routing is used, the internal nodes of the multicast tree are all clusterheads and gateways. Regular nodes can be found only at the leaves of the tree. When internal node *n* (a clusterhead or gateway) is traversed by *Join Ack*, it records the upstream and downstream node of *Join Ack*. This information will be used to reconstruct the tree when the links in the tree break due to mobility or crash. The clusterhead of node *i* will record node *i* as a member of *G* after it forwards *Join Ack* to node *i*. When a leaf node wants to quit the group *G*, it sends a *Quit Request* to its clusterhead. The clusterhead will update its membership information and then acknowledge this request with a *Quit Ack*. A leaf clusterhead leaves *G* and sends a *Quit Request* to its upstream member when all of its downstream members have quit *G*. A nonleaf node cannot quit until it becomes a leaf.

We allow *Join Ack* to follow a different path (from *Join Request*), if so provided by routing tables; and use *Join Ack* to graft links into the tree. The *Join Ack* strategy is more adaptive to a higher mobile situation where routes may change between *Req* and *Ack*. In this case, we want to choose the most current route [13], [17].

2.12.7. Member migration

It is necessary to reconfigure the multicast tree when its group members move or change node-type. A group member can detect the change of its multicast tree by monitoring its connectivity to upstream and downstream members. A member node reconnects to the tree by sending a *Join Request* to its multicast server (core) when its upstream member moves out of range or changes node type. For example a cluster head member will send a *Join Request* to the *MSid* in order to reconstruct the tree, if its upstream member (a gateway) changes to a regular node, or becomes disconnected. When a regular node member (a leaf) moves out of a cluster C_i and enters into a cluster C_j, the clusterhead of C_i will drop it from its descendant list. The regular node will send a *Join Request* to its new clusterhead of C_j. The clusterhead of C_i will send a *Quit Request* to its upstream member if it has become a leaf itself [13], [17].

2.13. Hierarchical State Routing

2.13.1. Introduction

HSR is a hierarchical link state based routing protocol. There are two types of partitioning in *HSR*, namely physical and logical. Physical partitioning is based on geographical position whereas the logical partitioning is based on functionality. Logical partitions play a key role in mobility management. The physical portioning has a dotted representation of the address and this hierarchical address is sufficient to deliver a packet to its destination from anywhere in the network using *HSR* tables. The drawback of *HSR* with respect to flat link state routing is the need of continuously updating the cluster hierarchy and the hierarchical address as nodes move. Nodes are also assigned logical addresses of the type- <*subnet, host* > where members of a subnet have common characteristics. The advantage of this logical partition is the separation of mobility management from physical hierarchy. The mobility management in *HSR* involves the maintenance of a dynamic mapping database for the hierarchical address of each IP subnet member at the home agents. Two functional components namely *HID* registration and *HID* finding are described in the following sections [12], [18], [19].

2.13.2. HID registration

HID registration is the process of updating the *HID* address database of a logical IP subnet member at its home agent. The other is the *HID* used for routing in *HSR*. Through the *HID* registration process, the home agent maintains up-to-date mobility binding for each subnet member.

Each subnet member registers with its home agent by sending a *Registration* message to the home agent. The *Registration* message contains two fields: its own host *ID* and current *HID*. Each member of a logical subnetwork knows the *HID* of its home agent; therefore, it uses that *HID* to send the *Registration* message to the home agent directly.

Upon receiving a *Registration* message, the home agent creates a mobility binding for the mobile node if there is none, or modifies the mobility binding accordingly. Registration is both periodic and event driven. At the home agent, the registered address is timed out and erased if not refreshed. Since, in most applications, the members of the same subnet move as a group, they tend to reside in neighboring clusters. Thus, registration overhead is modest.

Instead of sending the small *Registration* messages individually by each node, the cluster head can in fact periodically send to the home agent of a subnet. *Registration* message includes the registration information for all mobile nodes from the same subnet in the cluster. The aggregation of small individual *Registration* messages into larger blocks improves the *MAC* (medium access control) layer efficiency by minimizing the large per block overhead in most wireless *MAC* layers [12], [18], [19].

2.13.3. HID finding

HID finding is the process for the sender to learn the *HID* of the destination node it wants to communicate with. Each node maintains a *HID* mapping cache. When a source wants to send a packet to a destination of which it knows the IP address, it first checks its local *HID* cache. It will send the packet directly to the destination if its cache has the valid *HID* of the destination. Otherwise, it extracts the subnet address from the IP address. From its internal list (or from the top hierarchy) it obtains the hierarchical address of the corresponding home agent (recall that all home agents advertise their *HID*s to the top level hierarchy). It then sends the packet to the home agent using the obtained hierarchical address. The home agent finds the registered address from the host ID (in the IP address) and delivers the packet to the destination. The home agent then sends the registered address to the sender through the *HID Reply* message. Once the source and destination have learned each other’s hierarchical addresses, packets can be delivered directly without involving the home agent. *HSR* utilizes the address mapping cache to avoid the need to perform the *HID* finding for every packet. Aggressive caching helps minimizing the cost incurred by routing the packets through the home agent first. We assume that nodes can operate in a *promiscuous* receiving mode on their wireless network interface hardware, causing the interface to pass all heard packets to the network layer without filtering based on the destination address. Although our mobility management does not require this feature to work properly, this facility is common in current *LAN* hardware for broadcast media, including wireless. Overheard control packets do help the optimization of the mobility management. Whenever a node overhears the *Registration* and *HID Reply* messages, it will update its cache entries accordingly.

Furthermore, when the home agent sends the *HID Reply* message to the sender, it could return multiple most recently refreshed *HID*s besides the *HID* for the destination the sender is interested in. It could include as many *HID*s as the payload allows for a *MAC* frame. The penalty for caching is primarily due to the wasted effort in delivering a packet using an incorrect address. We adopt soft state approach to maintain the cache, i.e., each entry of the *HID* address (mobility binding) is purged if it is not refreshed [12], [18], [19].

2.13.4. Message Summary

HSR uses the mechanism of physical and logical hierarchical clustering to achieve scalability in MANET. The Physical partitioning of *HSR* is composed of a cluster-head node, a gateway node, and an internal node. The logical partitioning of *HSR* is composed of a home agent node and a subnet member node. *HSR* nodes have both *Hierarchical Identifier* (*HID*) *address* and *logical IP address*. The *HID* of each node is the sequence of the *MAC* addresses of the nodes on the path from the top hierarchy to the node itself. The logical IP address of each node has the format of [*subnet*, *host*] that follows the scheme of private IP address. Each subnet has its own home agent and contains the group definition for different mobility formations. *HSR* node uses a *Registration* message to refresh its *HID* in its home agent, which has the property of periodicity and to inform the new home agent of its *HID* when the node moves to the new cluster area, which has the property of event driven. The exact formats of *HSR Registration* and *HSR Reply* messages are not defined yet [12], [18], [19].

2.13.5. Pseudocode

[12], [18], [19]

```
Update_All()
{
// Called periodically every MIN_UPDATE_INTERVAL seconds
        Set msg_list = empty.
        Generate_HELLO(msg_list). // Adds HELLO to msg_list.
        Expire_Links().
        Update_Routes().
        Update_RN().
        If (current_time >= next_periodic) {
          Generate_Periodic_Update(msg_list).
          Set next_periodic = current_time + PER_UPDATE_INTERVAL.}
       Else Generate_Diff_Update(msg_list).
         Broadcast msg_list to neighbors.
         Set old_T = T and old_RN = RN.
      }


      Update_Routes() {
        For each node v in TT {
          Set d(v) = INFINITY, pred(v) = NULL.
          Set old_p(v) = p(v), p(v) = NULL. }
```

```
  Set $d(i) = 0$, $p(i) = i$, $pred(i) = i$.
  Set $S = \{i\}$ (labeled nodes).
 For each node j in N {
   // $c(u,v) = 1$ for min-hop routing
   Set $d(j) = c(i,j)$, $pred(j) = i$, $p(j) = j$. }
 While there exists a node u in TT - S s.t. $d(u) < INFINITY$ {
   Let u be a node in TT - S that minimizes $d(u)$.
   // A heap should be used to find u efficiently.
   Add u to S.
   If $p(u)$ != old_p(u) {  // parent has changed
     For each link (u,v) in TG {
       If (p(u) is not in r(u,v) AND reported(u,v) = 1) {
         Set reported(u,v) = 0. // (u,v) is not reported by p(u)
Set nr_expire(u,v) = current_time + PER_UPDATE_INTERVAL.}}
     If (p(u) is in r(u)) {
       Set tg_expire(u) = rt_expire(j,u).
       For each link (u,v) such that p(u) is in r(u,v) {
         Add (u,v) to TG.
         Set reported(u,v) = 1. }} // (u,v) is reported by p(u)
   }
   For each node v s.t. (u,v) is in TG {
     // Penalize links (u,v) not reported by p(u).
     If (reported(u,v) = 0 OR p(u) is not in r(v))
```

```
        Set cost = $c(u,v)$ + NON_REPORT_PENALTY.
      If ($d(u)$ + cost < $d(v)$ OR
          [$d(u)$ + cost = $d(v)$ AND ID($u$) < ID(pred($v$))]) {
        Set $d(v) = d(u) + c(u,v)$.
        Set pred($v$) = $u$.
        Set $p(v) = p(u)$. }}
  }
  Set T = empty set.
  For each node u in TT - i {
    Set dist($u$) = $d(u)$.  // route metric

If $p(u)$ != NULL, set next($u$) = $p(u)$. // route table entry
    Add link (pred($u$), $u$) to T. }
}

Update_RN() {
  // RN is the set of nodes reported by node i.
  Set RN = empty.
  // A neighbor j is in RN if some other neighbor s
  // would select node i as $p(j)$.
  For each neighbor s in N s.t. j is in $r(s)$ {
    // Initialize to run Dijkstra for source s, for 2 hops
    For each node j in N+{i},
    Set d(j) = INFINITY, par(j) = NULL.
   Set d(s) = 0, par(s) = s.
   For each link (s,j) s.t. s is in r(s,j) and j is in N+{i} {

    Set d(j) = 1, par(j) = j.
    For each link (j,k) s.t. j is in r(j,k) and k in in N {
     Set cost = 1.
     If (1 + cost < d(k) OR
         (1 + cost = d(k) AND ID(j) < ID(par(k)))) {
      Set d(k) = 1 + cost, par(k) = j. }}}
   // End of Dijkstra for source s
   For each neighbor j in N {
    // Add neighbor j to RN if its parent is i.
    If par(j) = i, add j to RN. }
  } // End of selection of neighbors in RN
  Add i to RN. // Node i is always in RN
  // A non-neighbor node u is in RN if p(u) is in RN.
  For each node u in TT - i,
   If p(u) is in RN, add u to RN.
}

Generate_Periodic_Update(msg_list) {
// Generates updates describing the reportable subtree RT.
```

```
   For each node u in RN that is not a leaf of T,
   add the update (FULL, n, NRL, NRNL, u, v_1,⋯, v_n)
   to msg_list, where:
     v_1, ⋯, v_n are the nodes v such that (u,v) is in T,
     the first NRL of these are nodes in RN that are leaves of T,
the next NRNL of these are nodes in RN that are not leaves of T,
     and the last n-(NRL+NRNL) of these are not in RN.
  }

  Generate_Diff_Update(msg_list) {
  // Generates updates reporting changes to the reportable
  // subtree RT.
   For each node u in RN {
     If u is not in old_RN and is not a leaf of T {
       // u was added to RN
       Add the update (FULL, n, NRL, NRNL, u, v_1, ⋯, v_n)  to msg_list, where v_1, ⋯,
       v_n, NRL, and NRNL are defined  as above for periodic updates.
     }
     Else if u is in old_RN and is not a leaf of T {
       Let v_1, ⋯, v_n be the nodes v s.t. (u,v) is in T AND {
         [(u,v) is not in old_T] OR
         [v is in RN - old_RN AND v is a leaf] OR
         [v is in old_RN - RN] }
```

```
      If this set of nodes is nonempty {
        Add the update (ADD, n, NRL, NRNL, u, v_1, ⋯, v_n)
        to msg_list, where NRL and NRNL are defined as above.}
    }
    If (u is in old_RN) {
      Let v_1, ⋯, v_n be the nodes v s.t. (u,v) is in old_T - TG
      AND [pred(v) is NULL or is not in RN].
      // If pred(v) is in RN, the delete is implied by an add.
      If this set of nodes is nonempty, add the update
      (DELETE, n, u, v_1, ⋯, v_n) to msg_list.
    }
  }
}

Process_Updates(j, msg_list) {
// Processes a list of update messages from node j.
  Set update_routes_flag = 0.
  // Flag will be set to 1 if a link in T is deleted.
  For each update = (subtype, n, NRL, NRNL, u, v_1, ⋯, v_n)
  in msg_list {
    Create an entry for u in TT if it does not exist.
    Create an entry for u in TT_j if it does not exist.
    If (subtype = FULL)
```

```
      Process_Full_Update(j, update).
    If (subtype = ADD)
      Process_Add_Update(j, update).
    If (subtype = DELETE)
      Process_Delete_Update(j, update).
    // Set update_routes_flag.
    If update_routes_flag = 0 {
      For each link (u,v) in TT(u) {
        If (u,v) is in T but not in TG {
          Set update_routes_flag = 1.
          Break. }}}
  } // End for each update
  If (update_routes_flag = 1) Update_Routes().
}


Process_Full_Update(j, update) {
  // update = (FULL, n, NRL, NRNL, u, v_1, ⋯, v_n)
  Add u to RN.
  Set rt_expire(j,u) = current_time + TOP_HOLD_TIME.
  For each link (u,v) s.t. j is in r(u,v) {

Remove j from r(u,v).
    If pred(j,v) = u, set pred(j,v) = NULL. }
  If (j = p(u) OR p(u) = NULL) {
    Set tg_expire(u) = current_time + TOP_HOLD_TIME.
    For each v s.t. (u,v) is in TG,
      // Delete old reported links before adding new ones.
      If reported(u,v) = 1, remove (u,v) from TG.
  }
  Process_Add_Update(j, update).
}

Process_Add_Update(j, update) {
  // update = (subtype, ADD, n, NRL, NRNL, u, v_1, ⋯, v_n)
  For m = 1, ⋯, n {
    Let v = v_m.
    Create an entry for v in TT if it does not exist.
    Create an entry for v in TT_j if it does not exist.
    Add j to r(u,v).
    If (j = p(u) OR p(u) = NULL) {
      Add (u,v) to TG.
      Set reported(u,v) = 1.
    }
    // Process implicit delete
    Set w = pred(j,v).
    Set pred(j,v) = u.
    If ($w$ != NULL AND $w$ != $u$) {
      Remove $j$ from $r(w,v)$.
      If ($j = p(w)$) remove $(w,v)$ from TG.
    }
    If ($m$ <= NRL) { // $v$ is a reported leaf
      Set leaf_update = (FULL, 0, 0, 0, $u$)
      Process_Full_Update($j$, leaf_update).
    }
    If ($m$ > NRL + NRNL) { // $v$ is not reported by $j$
       Remove $j$ from $r(v)$.
       Set rt_expire($j$,$v$) = 0.
       For each node $w$ s.t. $j$ is in $r(v,w)$
         Remove $j$ from $r(v,w)$.
       If ($j = p(v)$)
         For each node $w$ s.t. $(v,w)$ is in TG {
           Set reported($v$,$w$) = 0. // $(v,w)$ is not reported by $p(v)$
           Set nr_expire($u$,$v$) = current_time + PER_UPDATE_INTERVAL.}
    }
  }
}

Process_Delete_Update($j$, update) {
  // update = (DELETE, $n$, NRL, NRNL, $u$, $v$_1, ⋯, $v$_$n$)
  For m = 1, ⋯, n {
    Let v = v_m.
    Remove j from r(u,v).
    If pred(j,v) = u, set pred(j,v) = NULL.
    If (j = p(u)) remove (u,v) from TG.
  }
}

Link_Up(j) {
// Called when a link to j is discovered
  If j is not in N {
    Add j to N.
    Add (i,j) to TG.
    Set report(i,j) = 1.
  }
}

Link_Down(j) {
// Called when the link to neighbor j is lost
  If j is in N {
    Remove j from N.
    Remove (i,j) from TG.
    If lost link is due to link-layer failure indication {
      Update_Routes(i).
      Update_RN(i).
      Set msg_list = empty.
      Generate_Diff_Update(i, msg_list).
      Broadcast msg_list to neighbors.
      Set old_T = T and old_RN = RN.
    }
    Else Update_Routes(i).
  }
}

Expire_Links() {
  For each node u in TT - i {
    If (tg_expire(u) < current_time) {
      For each v s.t. (u,v) is in TG,
        Remove (u,v) from TG. }
    Else for each v s.t. (u,v) is in TG {
      If (report(u,v) = 0 AND nr_expire(u,v) < current_time)
        Remove (u,v) from TG. }
    For each node j in r(u) {
      If (rt_expire(j,u) < current_time) {
        Remove j from r(u).
        For each link (u,v) s.t. j is in r(u,v)
        Remove $j$ from $r(u,v)$. }}
  }
}
```

3. Message Complexity Analysis of Mobile Ad Hoc Network (MANET) Address Autoconfiguration Protocols

3.1. Introduction

MANETs are self-organizing wireless networks where mobile nodes have routing capabilities to be able to forward packets to communicate with one another over multi-hop wireless links without any fixed communication infrastructure, such as a base station or an access point [20]. Therefore, it is essential for all nodes to be able to perform the operations required for configuration of unique addresses to execute proper routing of data packets in a MANET. Address autoconfiguration is an important issue in MANETs since address pre-configuration is not always possible. MANETs currently depend on the mechanism of checking IP addresses of nodes to decide whether the connection and identification of nodes participating in a MANET are established or not [21].

In conventional networks, address autoconfiguration can be classified as either a stateless or stateful protocol [22]. The stateless approach is used when a network is not especially required to control the exact IP address assignments provided that the addresses are unique and routable. The stateful approach is used when a network demands exact IP address assignments [23]. Dynamic Host Configuration Protocol (DHCP) is an example of a stateful protocol where a DHCP server assigns unique addresses to unconfigured nodes and keeps state address information in an address allocation table. However, in stateless protocols, a node can select an address by itself and verify its uniqueness in a distributed manner using duplicate address detection (DAD) algorithms [24]. By using DAD algorithms, a node in a MANET, which lacks an IP address in the MANET, can determine whether a candidate address selected by itself is available or not. A node already equipped with an IP address also depends on DAD in order to protect its IP address from being accidentally used by another node in the MANET [25].

Based on the conventional method stated in [26], DAD can be classified as Strong DAD and Weak DAD. Strong DAD uses an address discovery mechanism where a node randomly selects an address and requests the address within a MANET by checking if the address is being used in the MANET. Based on a reply for the claimed request, which needs to arrive at the node within a finite bounded time interval, the node can detect an address duplication in the MANET [21]. Weak DAD is proposed by [26], where ad hoc routing protocols are used to detect address duplication by modification of the routing protocol packet format. MANET routing protocols can be classified into proactive and on-demand. Proactive routing protocols using periodic neighbor discovery messages and topology update messages give route information to each node before a node sends data packets to a destination. The Fisheye Scope Routing (FSR) [3], Topology Broadcast Based on Reverse Path Forwarding (TBRPF) [4], Fuzzy Sighted Link State Routing (FSLS) [5], Optimized Link State Routing Protocol (OLSR) [1], and Landmark Ad Hoc Routing (LANMAR) [6] are currently being developed as MANET proactive routing protocols. On-demand routing protocols such as Dynamic Source Routing (DSR) [7] and the Ad hoc On Demand Distance Vector (AODV) [2] issue route discovery mechanism messages only when a node needs to send data to a destination node. Because these protocols do not use any periodical message exchange, such as the neighbor discovery message used in proactive routing protocols, they do not hold any route information at each node before a node sends data towards a destination node. Therefore, they need Route Request and Route Reply messages to find and maintain a route when it is needed.

As a stateful protocol, MANETconf [27] uses a mutual exclusion algorithm for a node to acquire a new IP address. Therefore, if a requester wants to acquire an IP address, the IP address should be granted by all nodes in a MANET.

In other related research, Weniger and Zitterbart summarized the current approach and future directions of address autoconfiguration in MANETs [24]. Jeong *et al.* studied hybrid ad hoc IP address autoconfigurations in [25]. The authors of [28] proposed an IP address configuration for Zeroconf. Mohsin and Prakah [29] introduced IP address assignment in a MANET. Zhou and Mutka [30] investigated prophet address allocation for large scale MANETs. Additionally, Weniger proposed a passive autoconfiguration for MANETs [22].

Due to the lack of any centralized control and possible node mobility in MANETs, many issues at the network, medium access, and physical layers currently remain as research topics since no counterparts in the wired networks or cellular networks can satisfy these MANET requirements. At the network layer, the main problem is that of routing, which is awfully deteriorated by the time-varying network topology, power constraints, and the characteristics of the wireless channel. MANETs consist of mobile users that communicate with each other over a wireless channel, which cause an issue with regards to sharing a wireless medium among all the users. Due to the time-varying network topology and the lack of centralized control, the choice of the medium access control (MAC) scheme technology is also challenging in ad hoc networks. The ultimate purpose of the MAC is to establish the mechanism that can support multiuser traffic, and provide services for the different requirements of each traffic class. At the physical layer, power control is one of the most important issues, and focuses on getting the required transmission range of a node and quality of a communication link, which needs to be controlled so that it is wide enough to reach the intended receiver, while causing minimal interference to other nodes.

Based on the many considering factors of a MANET, the reduction of routing overhead is a main concern when a MANET routing protocol is developed. Therefore, one essential measure of the quality of a MANET routing protocol is the scalability in regards to an increase of the MANET nodes. Message complexity is defined as a performance measure where the overhead of an algorithm is measured in terms of the number of messages needed to satisfy the algorithm's request. The authors of [31] use the message complexity and synchronization delay to measure the performance of a mutual exclusion algorithm which is used to effectively share resources in distributed systems. In [32], Shen uses the message complexity to statistically measure the performance of the Cluster-based Topology Control (CLTC) protocol. The authors in [8] calculate the storage complexity and communication complexity to analyze the scalability of various MANET routing protocols and introduce the routing overhead of periodically updated *LS* messages, which follow the order of $O(N^2)$, where N indicates the number of nodes in a MANET. However, the detailed investigation to derive the upper bound of *LS* messages has not been justified by a mathematical form and currently the message complexity analysis and comparison among the IP address autoconfiguration protocols for MANETs has not been conducted yet. Therefore, in this chapter, the upper bounds of the message complexity of the IP address autoconfiguration protocols for MANETs are derived.

The analytical framework used in deriving the upper bound of the message complexity, which is represented in this chapter, can be widely adapted to a wide variety of protocols. The general methodology of analysis is based on [33], which uses a flowchart to analyze the time complexity of an image segmentation algorithm based on the recursive shortest spanning tree (RSST). The authors of [34] point out that time complexity is one of the most important factors to measure or compare the performance of different algorithms, and therefore, should be considered when an algorithm is being developed. Based on the complexity analysis method of [33], the message complexity of MANET address autoconfiguration algorithms is investigated. Each node strictly follows a *procedure*, which is a sequence of *steps* in the algorithm, where each step guides a node to make a general decision such as whether to generate a message or not, whether to take a same procedure or not (which is called a *recursive procedure*), whether to branch to a different procedure or not, and whether to stop a step or not. The method of adding the upper bounds of the time complexity measured at each step can be adapted in the proposed algorithm since MANET address autoconfiguration algorithms are composed of a sequence of discrete distinctive procedures where each step has its own message complexity. Therefore, by adding the message complexity measured at each step, the message complexity of a procedure can be calculated. Correspondingly, the method of adding the time complexity measured at each node to get the time complexity of *n* nodes can be adapted in the proposed algorithm since MANET address autoconfiguration algorithms are composed of recursive procedures. Therefore, by adding the message complexity measured at each procedure for each node, the message complexity of a MANET operation can be calculated.

This chapter is organized as follows. Section 3.2 summarizes the related work from existing publications. Section 3.3 presents a system model that is used in the derivations and analysis of the following sections and introduces the approach method used in analyzing the message complexity in this chapter. Sections from 3.4 to 3.6 present several *Lemmas* and their proofs used in deriving the message complexity of Strong DAD, Weak DAD, and MANETconf, respectively. Section 3.7 contains numerical results and performance analysis. Section 3.8 states the conclusion.

3.2. Summary of Address Autoconfiguration Protocols

The acronyms of messages and nomenclatures of the retry count variables used in this chapter are summarized in Table 1.

***Table 1 Acronym Table* [**: Variable*]**

Acronym	***Message***	***Acronym***	***Message***
AB	*Abort*	*IQ*	*Initiator Request*
AC	*Address Cleanup*	*LS*	*Link State*
AD	*Advertised*	*NR*	*Neighbor Reply*
AE	*Address Error*	*NQ*	*Neighbor Query*
AL	*Allocated*	*RR*	*Route Reply*
AO	*Allocation*	*RQ*	*Route Request*
AP	*Address Reply*	*RT*	*Requester Request*
AQ	*Address Request*	*M*	*DAD retry count limit**
IR	*Initiator Reply*	*n*	*retry count limit**

To perform address discovery, Strong DAD depends on the messages of address autoconfiguration such as *AQ* and *AP*. When a node needs to verify an IP address, it broadcasts an *AQ* to check if there is another node having the requested address. When a node finds a duplicated address, it replies by sending an *AP* message in response to the requested address. However, Strong DAD does not guarantee the uniqueness of an IP address in partitioned networks [21]. In addition, in [21], to consider the scalability issue in the Strong DAD protocol, the broadcast storm of DAD messages is mentioned but the method to either measure the scalability or solve the broadcast storm of DAD messages have not been provided.

In Weak DAD, when a node receives a *LS* message as a normal routing procedure, it compares the IP address and the associated key of the IP address of the *LS* message just received to the IP addresses and the keys of the IP addresses which already have been stored in its entries of the routing table, where the comparison is repeated for the number of IP addresses in the *LS* message. If the values of two keys per an IP address are different, it decides that the IP address is duplicated, otherwise, it keeps running the normal routing procedure. Since the unique key plays an important role in the routing table of a node, it is necessary for MANET nodes to keep the unique key in their routing tables in order to take additional steps [26] or to issue messages (such as the *AE* [25]) when a node detects a duplicated IP address. To make the mutual exclusion algorithm possible in MANETconf, an *Initiator*, which is one of the neighbor nodes of the *Requester,* broadcasts an *IQ* message towards all nodes in the MANET. Recipient nodes should reply with an affirmative or a negative message in response to the *IQ* message. When the *Initiator* receives affirmative replies from all nodes, the *Initiator* assigns the IP address to the *Requester*. If the initiator receives a negative response, the initiator selects another IP address and repeats the broadcasting of an *IQ* message until the retry count reaches *n*. When the initiator does not receive any reply from the nodes, it defines the nodes which do not reply a response as the abruptly departed nodes. The *Initiator* unicasts *IQ* messages to the abruptly departed nodes until the retry count reaches a threshold. After the *Initiator* finishes the repetition, it concludes that the nodes which did not reply a response as the nodes that abruptly left a MANET and broadcasts an *AC* message. However, since a *Requester* should wait a certain amount of time before it gets an approved IP address, MANETconf may have a significant time delay before being able to transmit urgent data. Moreover, MANETconf is composed of many steps or procedures, which results in broadcasting or unicasting of messages, which causes a severe overhead in MANET. Figs. 24, 25, 26, and 27 show the protocol operational flowcharts of Strong DAD, Weak DAD, and MANETconf in single node joining and MANET groups merging cases respectively.

3.3. Message Complexity Analysis

A MANET is represented as an undirected graph $G(\boldsymbol{V}, \boldsymbol{E})$ where $\boldsymbol{V}$ is a finite nonempty set of nodes, which can be represented as $\boldsymbol{V}= \{V_1^G, V_2^G, \cdots, V_W^G\}$ where $|\boldsymbol{V}|=W$ and $\boldsymbol{E}$ is a collection of pairs of distinct nodes from $\boldsymbol{V}$ that form a link, which can be represented as $\boldsymbol{E}= \{E_1^G, E_2^G, \cdots, E_W^G\}$ [35]. A connected, acyclic, undirected graph which contains all nodes is defined as a free tree. $\boldsymbol{V}$ can be partitioned into several subgraphs $\boldsymbol{V_1}$, $\boldsymbol{V_2}, \cdots,$ $\boldsymbol{V_k}, \cdots, \boldsymbol{V_n}$ where each partition subgraph is called as a free tree and $|\boldsymbol{V_1}+\boldsymbol{V_2}+\cdots+\boldsymbol{V_n}|= W$. A partitioned subgraph $\boldsymbol{V_k}$ is represented as a free tree $P(V,E)$, in which a node set V is represented as $\{V_1,V_2, \cdots, V_N\}$and $|V|$ equals N containing all nodes in the partitioned subgraph $\boldsymbol{V_k}$, where $N \leq W$. Fig. 21 shows MANET nodes at an instantaneous time where it is composed of six partitioned subgraphs.

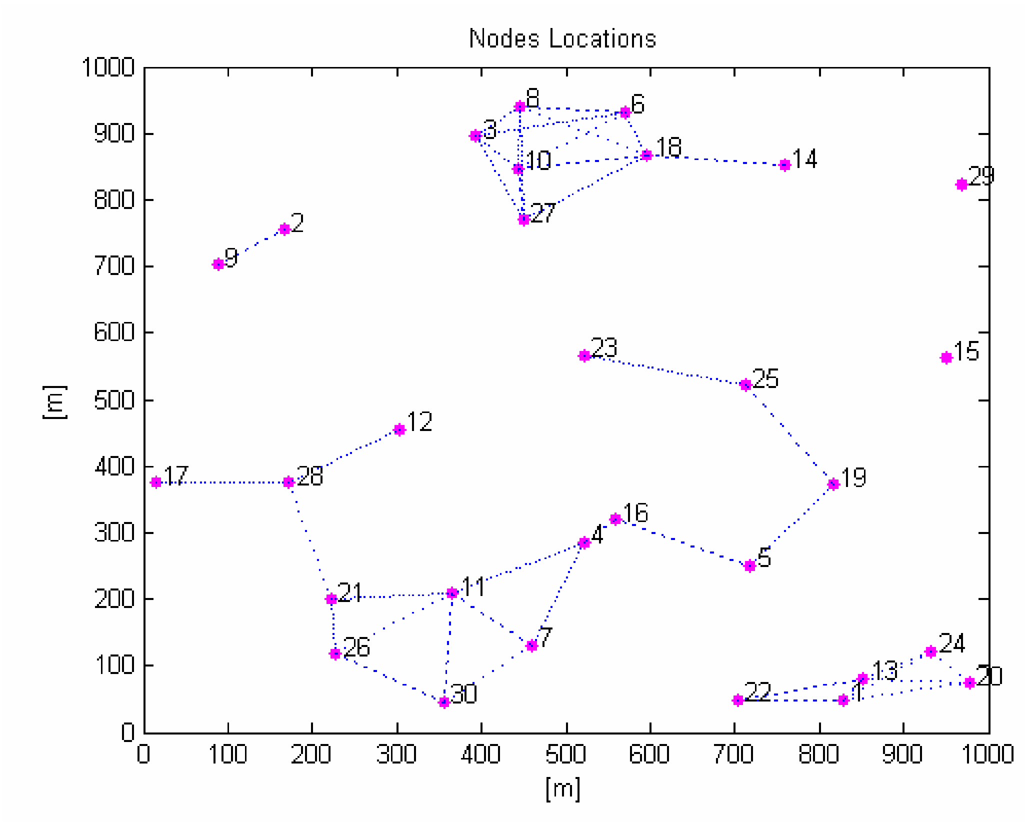

Figure 21 Mobile ad hoc network example (***MANET nodes are represented as dots and the dashed lines represent connected wireless links***)

To observe the effects of message complexity based on the procedures of the protocol's flowcharts, this chapter is organized into several subsections as per MANET address autoconfiguration protocol. Each subsection represents a scenario composed of a step or a procedure which consists of several steps where a step makes a decision and correspondingly broadcasts or unicasts a message. Each subsection includes several *Lemmas* and their proofs to verify the message complexity derivations of the steps and the procedures. When a node triggers a message based on a protocol's step, the number of nodes in a free tree to perform the step are first calculated. When all or some nodes in a free tree trigger a message from the network layer, the nodes identically adapt to one of the contention-based protocols or contention-free protocols at the MAC layer to effectively share the wireless medium channel to communicate with each other.

Moreover broadcasting, unicasting, relaying, or receiving a message is defined as a considerable activity. A node receiving a *NQ* message replies with a *NP* message in MANETconf. Based on the number of nodes, the maximum number of messages is calculated where the duplicated messages that have been already counted are not counted again in the number of messages. In order to derive an upper bound of the message complexity of each subsection, the maximum number of messages is calculated through an *O*-notation. In a procedure, the message complexity analysis of the next step is based on the message complexity analysis of the current step analyzed by the worst case scenario. In [32], the average number of nodes per cluster is used to calculate the message complexity. Therefore, it is assumed that the nodes in each step is independent of the dynamic network topology in order to make it possible to analyze and compare the message complexities among the steps or the procedures of the corresponding MANET address autoconfiguration protocols.

Fig. 22 shows the key steps of a single node joining case in address autoconfiguration protocols. In steps 1 and 2, a node *I* which needs to join the MANET group broadcasts an *AQ* message in Strong DAD; *LS* or *RQ* message in Weak DAD; or unicast *RR* message in MANETconf. In step 3, node *A* (*Initiator*) broadcasts an *IQ* message in MANETconf or node *A* relays an *AQ* message in Strong DAD; or *LS*/*RQ* message in Weak DAD. In step 4, all nodes in the MANET group relay *AQ*, *LS*, *RQ*, or *IQ* messages. In step 5, when node *B* finds a duplicated IP address, it will unicast an *AP* message towards the node *I* in

Strong DAD, or *AE* message in Weak DAD. In step 6, every node in MANETconf unicasts an *IR* message towards node *A* (*Initiator*) in MANETconf.

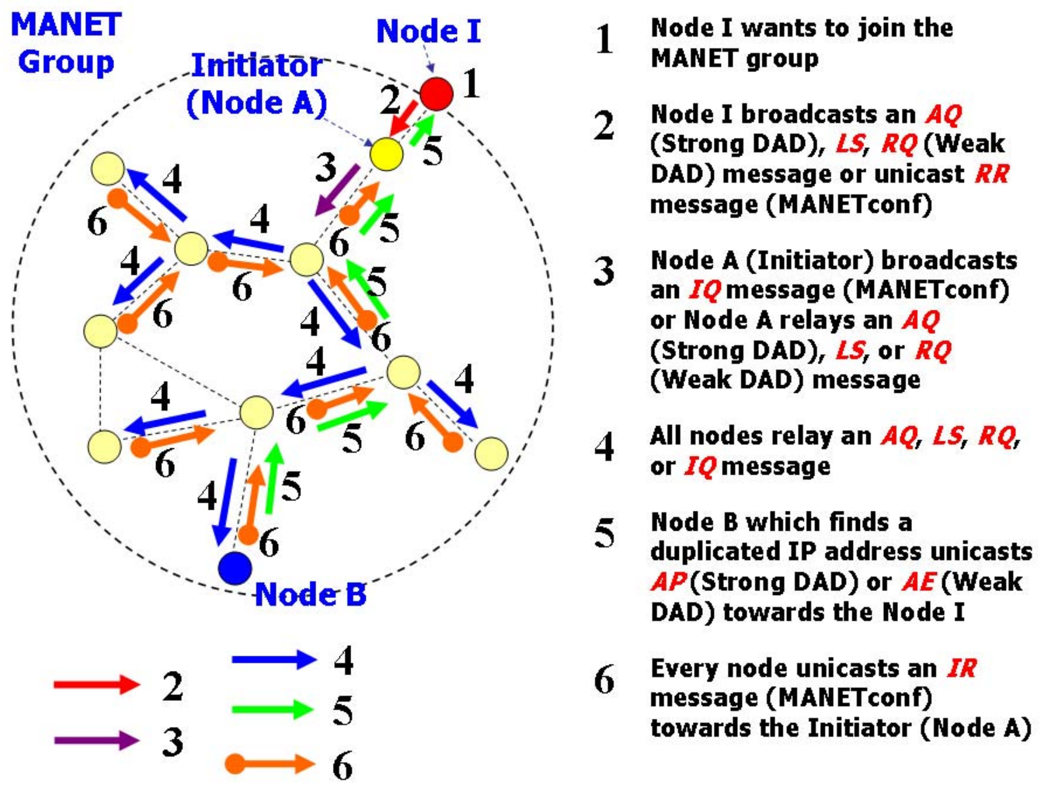
MANET Group
Initiator (Node A)
Node I
Node B
1 Node I wants to join the MANET group
2 Node I broadcasts an AQ (Strong DAD), LS, RQ (Weak DAD) message or unicast RR message (MANETconf)
3 Node A (Initiator) broadcasts an IQ message (MANETconf) or Node A relays an AQ (Strong DAD), LS, or RQ (Weak DAD) message
4 All nodes relay an AQ, LS, RQ, or IQ message
5 Node B which finds a duplicated IP address unicasts AP (Strong DAD) or AE (Weak DAD) towards the Node I
6 Every node unicasts an IR message (MANETconf) towards the Initiator (Node A)
2
3
4
5
6

Figure 22 The key steps of a single node joining node case in address autoconfiguration protocols.

Fig. 23 shows the key steps of a MANET group merging case in MANETconf. In step 1, when the two nodes *I* and *J* in two different MANET groups become neighboring nodes to each other, they exchange their *Partition Identity*. In step 2, each MANET group (N_1 and N_2) broadcasts *AL* messages to the other MANET group. In step 3, node *A* (*Initiator*), whose neighbor node (*Requestor*) finds a duplicated IP address, will broadcast an *IQ* message. In step 4, all recipient nodes will unicast *IR* messages in response to the *IQ* message towards node *A* (*Initiator*).

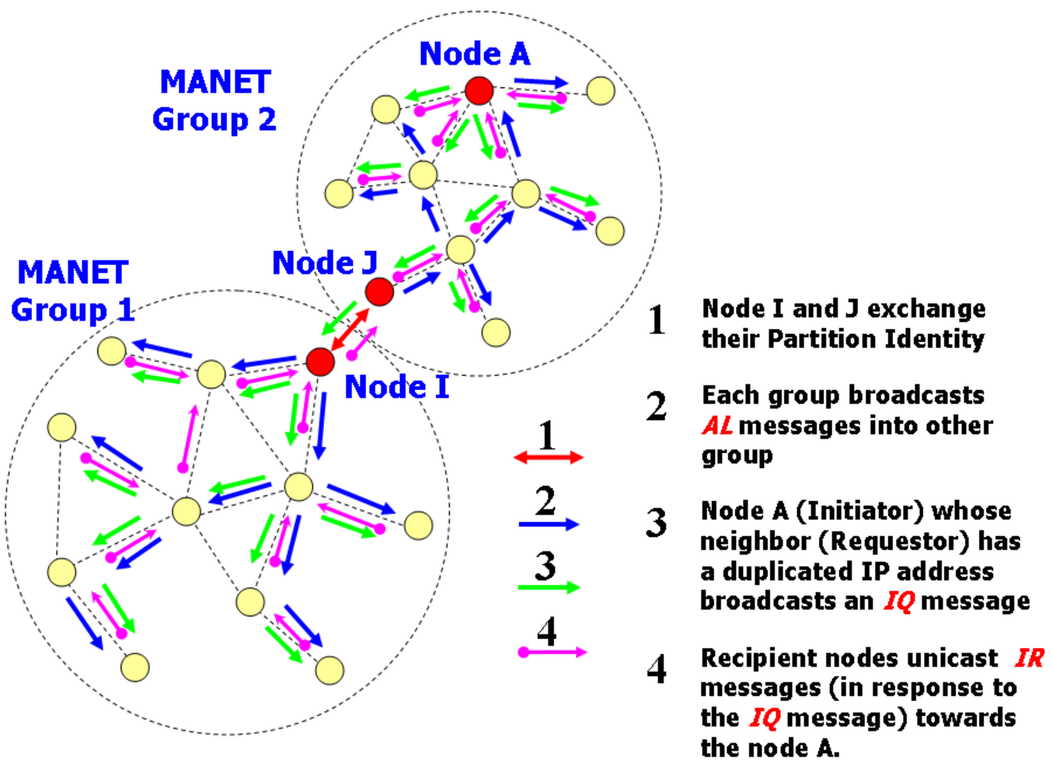
MANET Group 2
Node A
MANET Group 1
Node J
Node I
1
2
3
4
1 Node I and J exchange their Partition Identity
2 Each group broadcasts AL messages into other group
3 Node A (Initiator) whose neighbor (Requestor) has a duplicated IP address broadcasts an IQ message
4 Recipient nodes unicast IR messages (in response to the IQ message) towards the node A.

Figure 23 The key steps of MANET group merging case in MANETconf

Based on the above structure in a MANET, the following terms can be defined.

***Definition* 1** In a free tree *P*(*V*,*E*), broadcasting an *Address Query* (e.g., *AQ* message in Strong DAD, *LS* and *RQ* messages in Weak DAD, or *IQ* message in MANETconf) message by a node is defined as a *trial*.

1. A *success trial* is defined as an event in which after a node broadcasts an *Address Query* message, it does not receive any *Address Reply* message (e.g., *AP* message in Strong DAD, *AE* in Weak DAD, or negative *IR* message in MANETconf) within a specific time period.

2. A *failure trial* is defined as an event in which after a node broadcasts an *Address Query* message, it receives at least one *Address Reply* message within a specific time period.
3. A *successful IP verification procedure* is defined from *m* consecutive success trials.
   A. Therefore, for a node to get a verified IP address, the node has to perform a sequence of *m* independent trials where each trial has to become a *success trial*.
   B. In Strong DAD, *m* is defined as a positive number which is greater than 1 ($m>1$).
   C. In Weak DAD and MANETconf, since *m* is set to one ($m=1$), the *successful IP verification procedure* is same as the *success trial*.
4. An *IP verification procedure* including any *failure trial* results in a *failure IP verification procedure*.
   A. In Strong DAD, a *failure IP verification procedure* is composed of consecutive x-1 times of *success trials* and a *failure trial* at the x^{th} *trial* where $x = 1,2,\cdots, m$.
   B. In Weak DAD and MANETconf, since *m* is set to one ($m=1$), the *failure IP verification procedure* is same as the *failure trial*.
5. A *session* is defined as a sequence of *successful* or *failure procedures*. The maximum number of procedures executed in the *session* is limited by *n* in Strong DAD, Weak DAD, and MANETconf.

A. When computing the upper bound in Strong DAD, the worst case of a *successful session* composes of n-1 consecutive *failure IP verification procedures* and a *successful IP verification procedure* at the n^{th} *IP verification procedure*. A *failure session* is composed of n *failure IP verification procedures*.

B. When computing the upper bound in Weak DAD and MANETconf, the worst case of a *successful session* composes of n-1 consecutive *failure trials* and a *success trial* at the n^{th} *IP verification procedure*. A *failure session* composes of n *failure trials*. ■

The most common flooding method is used to broadcast an *Address Query* message where every node retransmits an *Address Request* message to its entire 1-hop neighbors whenever it receives the first copy of the *Address Query* message [36].

Based on a node that broadcasts an *Address Query* message in a free tree $P(V,E)$, the notations of broadcasting nodes are ascendingly rearranged such that u nodes, which are $\{V_1, V_2, \cdots, V_u\}$, broadcast or relay an *Address Query* message and N-u nodes, which are $\{V_{u+1}, V_{u+2}, \cdots, V_n\}$, relay *Address Query* messages. Since each member node in a free tree will relay the *Address Query* message initiated at node V_i, assuming that the duplicated packet discard scheme is applied, the maximum number of nodes relaying an *Address Query* message is N-1, where the rule of discarding duplicated messages at a node is adopted. Therefore, the maximum number of *Address Query* messages broadcasted or relayed in the free tree is N, which can be represented as $O(N)$. The above content can now be generalized into the following definition.

***Definition* 2** For a MANET with N nodes, $O(N)$ is the upper bound of the maximum number of broadcasted or relayed *Address Query* messages when a node broadcasts the *Address Query* message. ■

Next we consider the case where a node unicasts an *Address Reply* message in response to an *Address Query* message in a free tree $P(V,E)$. The variable t is defined as the largest number of nodes in a communication path based on the routing tree, including the source node. In addition, the maximum iteration number of the routing path setup algorithm at a node denotes the largest path length, which is t. If an address duplication is detected at node V_j, based on the t nodes which give a node set $\{V_j, V_k, \cdots, V_i\}$, an *Address Reply* message will be sent through the reverse path where V_j becomes the source node of the *Address Reply* message. The maximum number of nodes in a path which unicast or relay an *AP* message is denoted by $d(j,i)$, which equals the number of hops between the node V_j unicasting an *Address Reply* message and the node V_i which broadcasts an *Address Query* message.

***Lemma* 1** For a MANET routing tree with t nodes in the maximum length path, $O(t)$ is the upper bound of the maximum number of unicasted or relayed *Address Reply* messages when a node unicasts an *Address Reply* message.

Proof. Since each member node in a path of $d(j,i)$ hops relays an *Address Reply* message initiated by an *Address Reply* source node, the maximum number of nodes relaying an *Address Reply* message is t-2 where the rule of discarding duplicated messages at a node is adapted and the node V_i does not relay an *Address Reply* message. Therefore, the maximum number of *Address Reply* messages unicasted or relayed in the free tree is t-1, where the message complexity bound can be represented as $O(t)$. ■

3.4. Strong DAD

In order to derive the upper bound of the message complexity of the Strong DAD protocol, the flowchart of Strong DAD, as shown in Fig. 24, is used. The message complexity of Strong DAD can be considered in two cases, the single node joining case and the MANET group merging case.

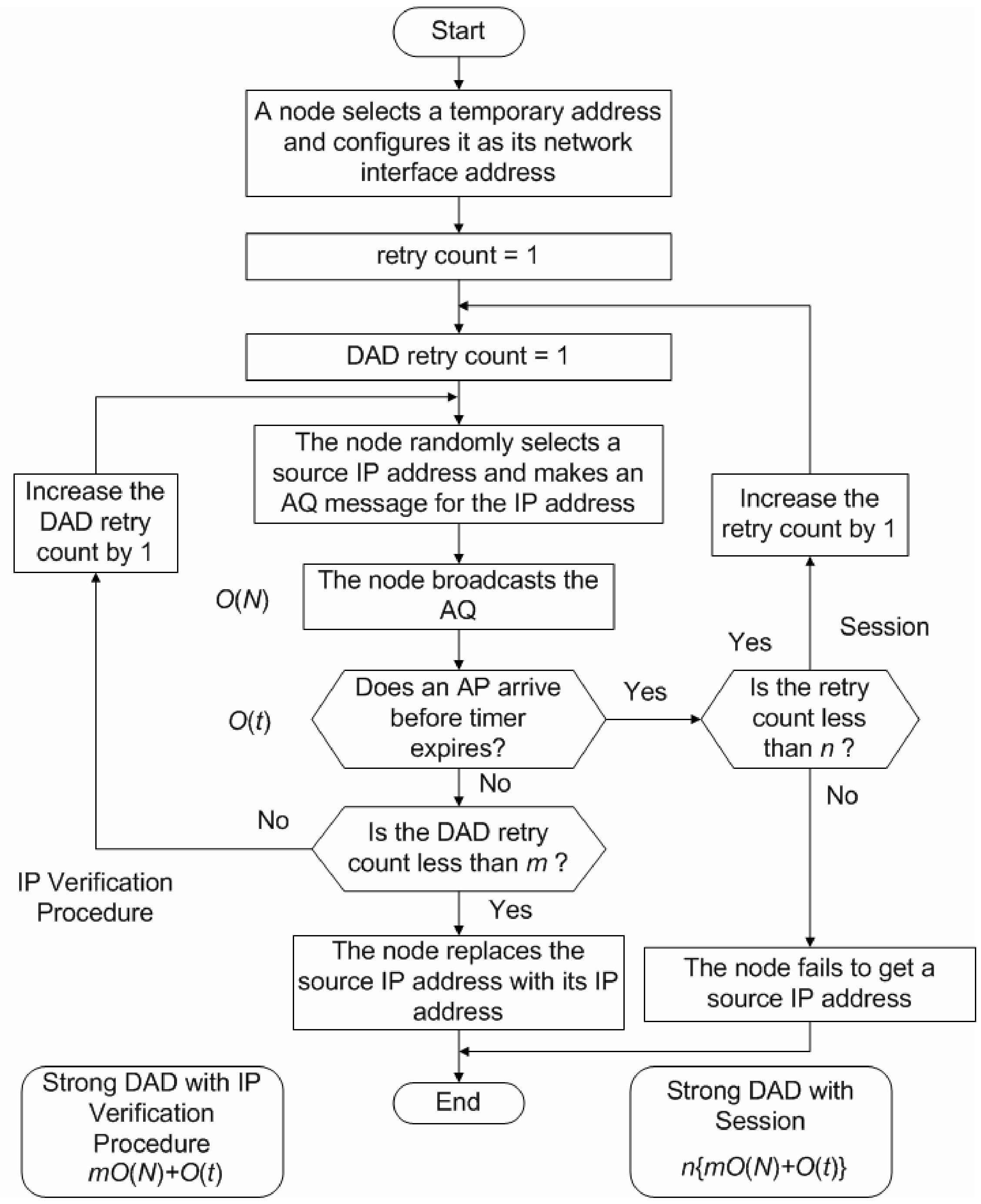
Start
A node selects a temporary address and configures it as its network interface address
retry count = 1
DAD retry count = 1
The node randomly selects a source IP address and makes an AQ message for the IP address
Increase the DAD retry count by 1
Increase the retry count by 1
O(N)
The node broadcasts the AQ
Session
Yes
O(t)
Does an AP arrive before timer expires?
Yes
Is the retry count less than n ?
No
No
No
Is the DAD retry count less than m ?
IP Verification Procedure
Yes
The node replaces the source IP address with its IP address
The node fails to get a source IP address
Strong DAD with IP Verification Procedure mO(N)+O(t)
End
Strong DAD with Session n{mO(N)+O(t)}

Figure 24 The flowchart of Strong DAD operations

3.4.1. A single node joining case in Strong DAD

In this section, a single node joining case is considered. To compute the upper bound of the message complexity, a scenario where a node experiences a *failure IP verification procedure* is considered. Since the procedure is composed of a total of (m-1) number of *success trials* and a *failure trial* at the *m*th *trial*, the message complexity of a *failure IP verification procedure* can be represented as $mO(N)+O(t)$. Based on the above result, the following *Lemma* is given.

***Lemma* 2** In an IP address verification procedure, $mO(N)+O(t)$ is the upper bound of the maximum number of broadcasted/relayed *AQ* messages and unicasted/relayed *AP* messages when a node needs to verify its IP address in a MANET with the Strong DAD protocol.

Proof. The *IP verification procedure* including a *failure trial* at the *m*th *trial* is composed of m-1 *success trials*, which gives $(m\text{-}1)O(N)$ number of broadcasted or relayed *AQ* message based on *Definition* 2, and a *failure trial* at the *m*th *trial*, which gives $O(N)$ number of broadcasted or relayed *AQ* message based on *Definition* 2, and $O(t)$ unicasted or relayed *AP* message based on *Lemma* 1. Therefore, the message complexity of the *failure IP verification procedure* can be represented as $(m\text{-}1)O(N)+O(N)+O(t)$, which sums the upper bound of the maximum number of broadcasted, unicasted, and relayed *AQ* and *AP* messages in m-1 *success trials* and a *failure trial* at the *m*th *trial*. Rearranging $(m\text{-}1)O(N)+O(N)+O(t)$ yields $mO(N)+O(t)$.

■

***Lemma* 3** In a session, $n(mO(N)+O(t))$ is the upper bound of the maximum number of broadcasted/relayed *AQ* messages and unicasted/relayed *AP* messages using the Strong DAD protocol.

Proof. Strong DAD has a *session* and the maximum number of retries of the *IP verification procedure* is limited by n in the *session*. Since the *session* consists of n maximum number of *IP verification procedures* and the upper bound of the maximum number of *IP verification procedures* is $mO(N)+O(t)$, based on *Lemma* 2, the message complexity of the session can be represented as $n(mO(N)+O(t))$. ■

3.4.2. MANET group merging case in Strong DAD

In this section, the MANET group merging case is considered. A scenario is considered where two MANET groups $\boldsymbol{V_i}$ and $\boldsymbol{V_j}$, where $|\boldsymbol{V_i}| = N_1$, $|\boldsymbol{V_j}| = N_2$, and $N_1 \leq N_2$, merge into each other and a node in the MANET group $\boldsymbol{V_i}$ finds an IP address that is duplicated based on the *AQ* message received. Since the message complexity in the MANET group $\boldsymbol{V_j}$ already has been defined as $n(mO(N_2)+O(t))$ based on *Lemma* 3 in the worst case, all nodes in MANET group $\boldsymbol{V_i}$ can find the duplicated IP addresses. Therefore, the message complexity of a MANET group merging case in Strong DAD can be represented as $nN_1(mO(N_2)+O(t))$. Based on the results, the following *Lemma* can be derived.

***Lemma* 4** For a two MANET group merging case where each group has N_1 and N_2 nodes respectively (where $N_1 \leq N_2$), $nN_1(mO(N_2)+O(t))$ is the upper bound of the maximum number of broadcasted/relayed *AQ* messages and unicasted/relayed *AP* messages of the two MANET groups merging case with Strong DAD.

Proof. Since the message complexity of the MANET group $\boldsymbol{V_j}$ is defined as $n(mO(N_2)+O(t))$ based on *Lemma* 3, and N_1 nodes need to verify their IP address in a merged MANET with Strong DAD, the message complexity of the MANET group merging case can be represented as $nN_1(mO(N_2)+O(t))$, where each node in $\boldsymbol{V_i}$ generates the message complexity of $n(mO(N_2)+O(t))$. ■

3.5. Weak DAD

In order to derive the upper bound of the message complexity of the Weak DAD protocol, the flowchart of Weak DAD, as shown in Fig. 25, is used. The message complexity of Weak DAD can be considered in two cases, the single node joining case and the MANET group merging case.

3.5.1. A single node joining case in Weak DAD

In this section, a single node joining case is considered. In Weak DAD with proactive MANET routing protocols, nodes periodically broadcast *LS* messages to inform other nodes of the network topology. In Weak DAD with on-demand MANET routing protocols, only when a source node needs to send data to a destination node where the source node does not have a route to the destination, the source node broadcasts a *RQ* message to find a route to a destination node and a node which is the destination node or a node having a fresh enough route unicasts a *RR* messages in response to the *RQ* message.

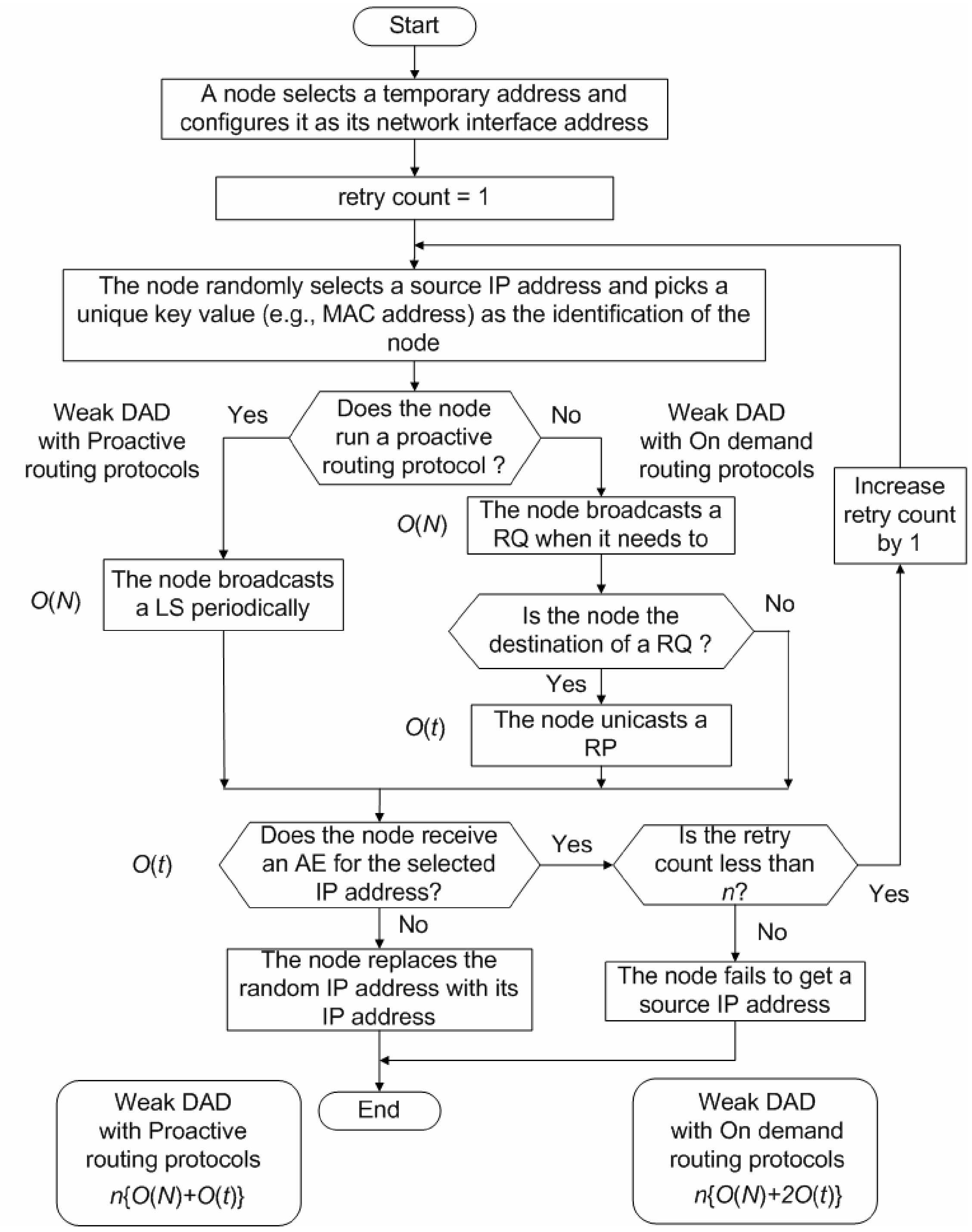
Start
A node selects a temporary address and configures it as its network interface address
retry count = 1
The node randomly selects a source IP address and picks a unique key value (e.g., MAC address) as the identification of the node
Weak DAD with Proactive routing protocols
Yes
Does the node run a proactive routing protocol ?
No
Weak DAD with On demand routing protocols
Increase retry count by 1
O(N)
The node broadcasts a RQ when it needs to
O(N)
The node broadcasts a LS periodically
Is the node the destination of a RQ ?
No
Yes
O(t)
The node unicasts a RP
O(t)
Does the node receive an AE for the selected IP address?
Yes
Is the retry count less than n?
Yes
No
No
The node replaces the random IP address with its IP address
The node fails to get a source IP address
Weak DAD with Proactive routing protocols n{O(N)+O(t)}
End
Weak DAD with On demand routing protocols n{O(N)+2O(t)}

Figure 25 The flowchart of Weak DAD operations

When a node finds an IP address that is duplicated with an entry in its routing table after investigating an IP address in a *LS*, *RQ*, or *RR* message, the node takes additional steps to inform other nodes of the duplicated address [26]. In such a case, the node that was already using the IP address will unicast an *AE* message to the node that has the duplicated IP address [25]. If a node does not find any duplicated IP address after investigating an IP address in a *LS, RQ,* or *RR* message, the node normally relays the *LS, RQ,* or *RR* message. Based on the above specifications, the following *Lemmas* can be derived.

***Lemma* 5** In an *IP verification procedure,* $O(N)+O(t)$ is the upper bound of the maximum number of broadcasted/relayed *LS* messages and unicasted/relayed *AE* messages when a node needs to verify its IP address in a MANET using Weak DAD with proactive routing protocols.

Proof. The maximum number of messages occurs when the *IP verification procedure* results in a *failure trial*. Since, the *failure trial* gives $O(N)$ number of broadcasted or relayed LS messages based on *Definition* 2, and $O(t)$ unicasted or relayed *AP* message based on *Lemma* 1, the message complexity of the *failure trial* can be represented as $O(N)+O(t)$, which sums the upper bound of the maximum number of broadcasted and relayed *LS* messages and unicasted and relayed *AE* messages. ■

***Lemma* 6** In a session, $n(O(N)+O(t))$ is the upper bound of the maximum number of broadcasted/relayed *LS* messages and unicasted/relayed *AE* messages using Weak DAD with proactive routing protocols.

Proof. Weak DAD with proactive routing protocols has a *session* and the maximum number of retries of the *IP verification procedure* is limited by *n* in the *session*. Since the *session* consists of *n* maximum number of *IP verification procedures* and the upper bound of the maximum number of an *IP verification procedure* is *O*(*N*)+*O*(*t*) based on *Lemma* 5, the message complexity of the *session* can be represented as *n*(*O*(*N*)+*O*(*t*)), where *n* is the number of retry count of the *IP verification procedures*. ■

In Weak DAD with on demand routing protocols, a node broadcasts or relays a *RQ* message and it can unicast a *RP* message if it is a destination node based on the normal routing procedure. In addition, it unicasts an *AE* message when a node finds a duplicated IP address. Based on the above results, the following *Lemma* is given.

***Lemma* 7** In an *IP verification procedure*, *O*(*N*)+2*O*(*t*) is the upper bound of the maximum number of broadcasted/relayed *RQ* messages and unicasted/relayed *RP* messages and *AE* messages when a node wants to verify its IP address in a MANET using Weak DAD with on demand routing protocols.

Proof. The maximum number of messages occurs when the *IP verification procedure* results in a *failure trial*. Since, the *failure trial* gives *O*(*N*) number of broadcasted or relayed *RQ* messages based on *Definition* 2, and 2*O*(*t*) unicasted or relayed *RP* messages and *AE* messages based on *Lemma* 1, the message complexity of the *failure trial* can be represented as *O*(*N*)+2*O*(*t*), which sums the upper bound of the maximum number of broadcasted and relayed *RQ* and unicasted and relayed *RP* and *AE* messages.

■

***Lemma* 8** In a session, *n*(*O*(*N*)+2*O*(*t*)) is the upper bound of the maximum number of broadcasted/relayed *RQ* messages and unicasted/relayed *RP* messages and *AE* messages in Weak DAD with on demand routing protocols.

Proof. Weak DAD with on demand routing protocols has a *session* and the maximum number of retries of the *IP verification procedure* is limited by *n* in the *session*. Since the *session* consists of *n* maximum number of *IP verification procedures* and the upper bound of the maximum number of an *IP verification procedure* is $O(N)+2O(t)$ based on *Lemma* 7, the message complexity of the *session* can be represented as $n(O(N)+2O(t))$ where *n* is the number of retry count of the *IP verification procedure*. ■

3.5.2. MANET group merging case in Weak DAD

In this section, the MANET group merging case is considered. A scenario is considered where two MANET groups $\boldsymbol{V_i}$ and $\boldsymbol{V_j}$, where $|\boldsymbol{V_i}| = N_1$, $|\boldsymbol{V_j}| = N_2$, and $N_1 \leq N_2$, merge into each other and a node in the MANET group $\boldsymbol{V_i}$ finds an IP address that is duplicated based on the *LS*, *RQ*, or *RR* message received. Since the message complexity in the MANET group $\boldsymbol{V_j}$ has been defined as $n(O(N_2)+O(t))$ already, in the worst case, all nodes in the MANET group $\boldsymbol{V_i}$ can find duplicated IP addresses. Therefore, the message complexity of a MANET group merging case in Weak DAD with proactive routing protocols can be represented as $N_1(n(O(N_2)+O(t)))$. Based on the results, the following *Lemmas* can be derived.

***Lemma* 9** For a two MANET group merging case where each group has N_1 and N_2 nodes (where $N_1 \leq N_2$), then $nN_1(O(N_2)+O(t))$ is the upper bound of the maximum number of broadcasted/relayed *LS* messages and unicasted/relayed *AE* messages of the two MANET groups merging case using Weak DAD with proactive routing protocols.

Proof. Since the message complexity of the MANET group $\boldsymbol{V_j}$ is defined as $n(O(N_2)+O(t))$ based on *Lemma* 6, and N_1 nodes need to verify their IP address in a merged MANET with Weak DAD with proactive routing protocols, the message complexity of the MANET group merging case can be represented as $N_1(n(O(N_2)+O(t)))$ where each node in $\boldsymbol{V_i}$ generates the message complexity of $n(O(N_2)+O(t))$. Rearranging $N_1(n(O(N_2)+O(t)))$ yields $nN_1(O(N_2)+O(t))$. ■

***Lemma* 10** For a two MANET group merging case where each group has N_1 and N_2 nodes (where $N_1 \leq N_2$), respectively $nN_1(O(N_2)+2O(t))$ is the upper bound of the maximum number of broadcasted/relayed *RQ* messages and unicasted/relayed *RP* messages and *AE* messages of the two MANET groups merging case using Weak DAD with on demand routing protocols.

Proof. Since the message complexity of the MANET group $\boldsymbol{V_j}$ is defined as $n(O(N_2)+2O(t))$ based on *Lemma* 8, and N_1 nodes need to verify their IP address in a merged MANET with Weak DAD with on demand routing protocols, the message complexity of the MANET group merging case can be represented as $N_1(n(O(N_2)+2O(t)))$ where each node in $\boldsymbol{V_i}$ generates the message complexity of $n(O(N_2)+2O(t))$. Rearranging $N_1(n(O(N_2)+2O(t)))$ yields $nN_1(O(N_2)+2O(t))$ ■

3.6. MANETconf

The message complexity of MANETconf can be considered in two cases, the single node joining case and the MANET group merging case.

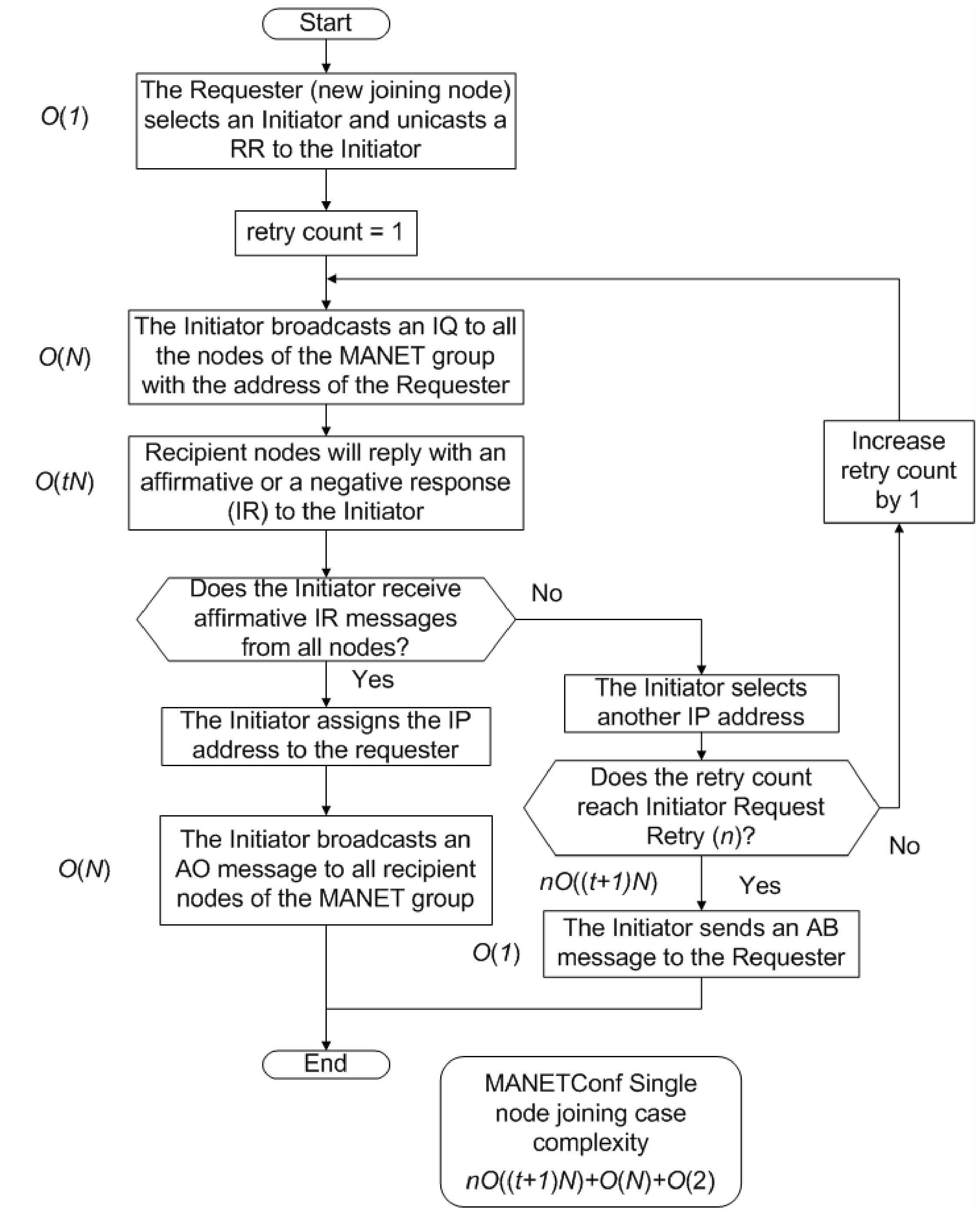
Start
O(1)
The Requester (new joining node) selects an Initiator and unicasts a RR to the Initiator
retry count = 1
O(N)
The Initiator broadcasts an IQ to all the nodes of the MANET group with the address of the Requester
O(tN)
Recipient nodes will reply with an affirmative or a negative response (IR) to the Initiator
Increase retry count by 1
Does the Initiator receive affirmative IR messages from all nodes?
No
Yes
The Initiator selects another IP address
The Initiator assigns the IP address to the requester
Does the retry count reach Initiator Request Retry (n)?
No
nO((t+1)N)
Yes
O(N)
The Initiator broadcasts an AO message to all recipient nodes of the MANET group
O(1)
The Initiator sends an AB message to the Requester
End
MANETConf Single node joining case complexity
nO((t+1)N)+O(N)+O(2)

Figure 26 The flowchart for message complexity derivation of MANETconf protocol for a single node joining case

3.6.1. A single node joining case in MANETconf

In this section, the single node joining case is considered. In order to derive the upper bound of the message complexity of the single node joining case in MANETconf, the flowchart as shown in Fig. 26 is used. When a node (which is a *Requestor*) tries to join a MANET and to obtain a verified IP address, it broadcasts a *NQ* message to its neighbors. When the *Requestor* does not receive any *NR* messages before the neighbor reply timer expires, it repeats broadcasting the *NQ* message by a threshold number. After finishing the repetition, the *Requestor* decides that there is only one node and configures itself with an IP address. The *Initialization* procedure of MANETconf described above is not considered into the message complexity since the message complexity is focused on the procedures of a single node joining into a MANET group and MANET group merging case.

If the *Requestor* receives *NR* messages, the *Requestor* selects an *Initiator* and unicasts a *RR* message to the *Initiator*. The message complexity of unicasting the *RR* message can be represented as $O(1)$. After receiving a *RR* message, the *Initiator* broadcasts an *IQ* message to all nodes of the MANET group in order to verify the IP address of the *Requestor*. The message complexity of broadcasting the *IQ* message can be represented as $O(N)$ based on *Definition* 2. Recipient nodes will reply with an affirmative or a negative response through the *IR* message, to the *Initiator*. The message complexity of unicasting the *IR* message by all nodes in the MANET group can be represented as $O(tN)$, since all N nodes unicast *IR* messages and each *IR* message has the message complexity $O(t)$ based on *Lemma* 1.

If the *Initiator* receives positive *IR* messages from all the recipient nodes, it broadcasts an *AO* message to all the recipient nodes of the MANET group. The message complexity of broadcasting the *AO* message can be represented as $O(N)$ based on *Definition* 2.

If the *Initiator* receives negative *IR* messages from the recipient nodes, it selects another IP address and repeats the step of broadcasting *IQ* and receiving *IR* messages until the retry count reaches the *Initiator Request Retry* which is set to n in this chapter. Based on the above results, the following *Lemma* can be derived.

***Lemma* 11** In an *IP verification procedure* of a single node joining case, $O((t+1)N)$ is the upper bound of the maximum number of broadcasted/relayed *IQ* messages and unicasted/relayed *IR* messages when a node needs to verify its IP address in a MANET with MANETconf.

Proof. The maximum number of messages occurs when the *IP verification procedure* results in a *failure trial*. Since, the *failure trial* gives $O(N)$ number of broadcasted or relayed *IQ* messages based on *Definition* 2, and $O(tN)$ unicasted/relayed *IR* messages based on *Lemma* 1, the message complexity of the *failure trial* can be represented as $O(N)+O(tN)$, which sums the upper bound of the maximum number of broadcasted and relayed *IQ* and unicasted and relayed *IR* messages. This can be rearranged as $O((t+1)N)$.

■

Therefore, the message complexity of broadcasting an *IQ* message and receiving *IR* messages until the retry count reaches n can be represented as $nO((t+1)N)$. After n times of repetitions, if the initiator receives negative *IR* messages, it sends *AB* messages to the

Requestor. The message complexity of unicasting the *AB* message can be represented as $O(1)$.

Therefore, the message complexity of a single node joining case can be represented as $nO((t+1)N)+O(N)+O(2)$ where $O(2)$ indicates the message complexity of unicasting *RR* and *AB* messages. Based on the above results, the following *Lemma* can be derived.

***Lemma* 12** *In* a session of a single node joining case, $nO((t+1)N)+O(N)+O(2)$ is the upper bound of the maximum number of broadcasted or relayed *IQ* and *AO* messages and unicasted or relayed *IR*, *RR*, and *AB* messages in MANETconf.

Proof. MANETconf has a *session* and the maximum number of retries of the *IP verification procedure* is limited by n in the *session*. Since the *session* consists of n maximum number of *IP verification procedures* and the upper bound of the maximum number of an *IP verification procedure* is $O((t+1)N)$ based on *Lemma* 11, the message complexity of the *session* can be represented as $nO((t+1)N)+O(N)+O(2)$ where n is the number of *IP verification procedures*, $O(N)$ indicates the message complexity of broadcasting the *AO* message and $O(2)$ indicates the message complexity of unicasting *RR* and *AB* messages. ■

3.6.2. MANET group merging case in MANETconf

In this section, the MANET group merging case is considered. In order to derive the upper bound of the message complexity of the MANET group merging case in MANETconf, the flowchart as shown in Fig. 27, is used.

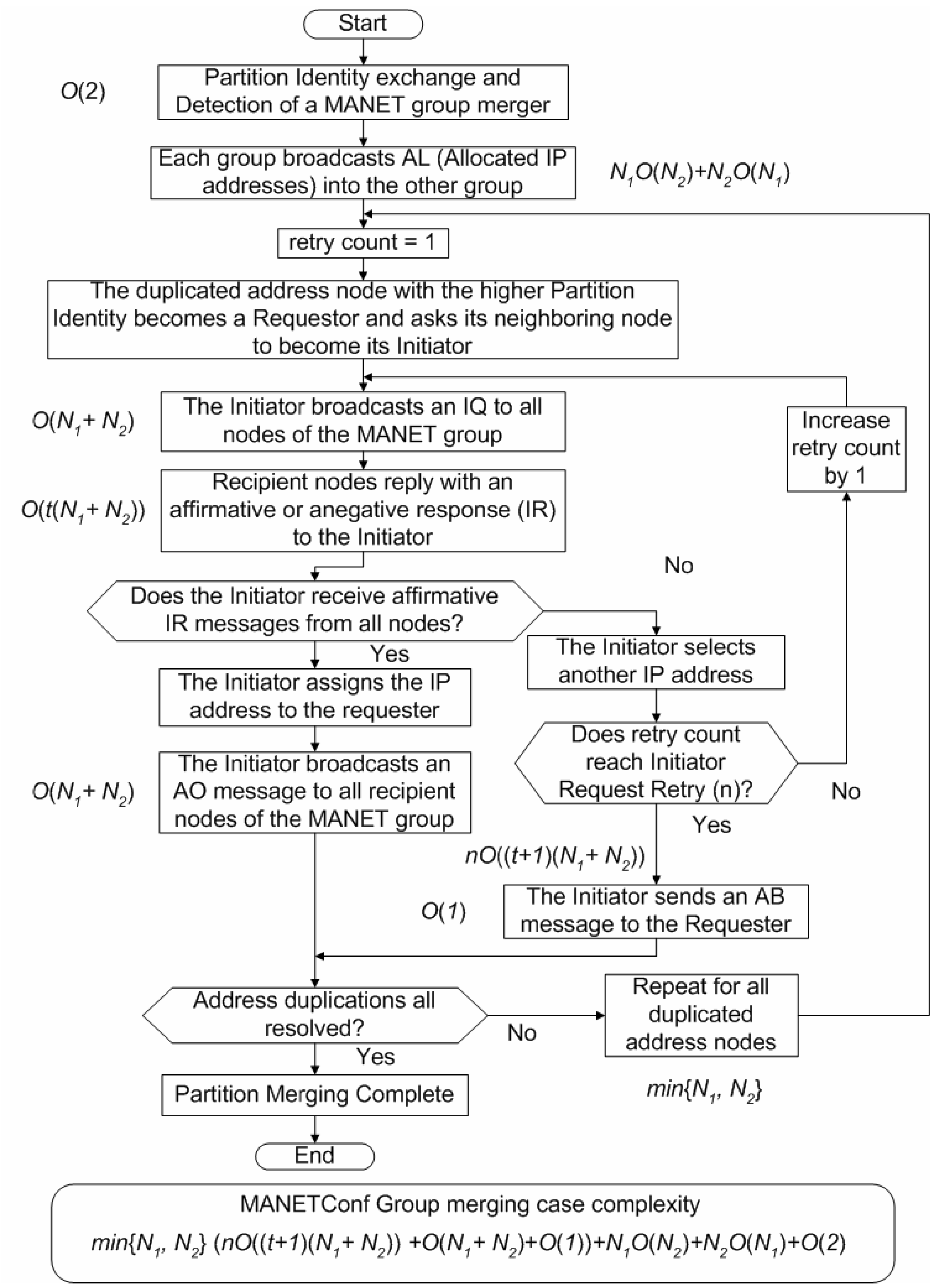
Start
Partition Identity exchange and Detection of a MANET group merger
O(2)
Each group broadcasts AL (Allocated IP addresses) into the other group
N1O(N2)+N2O(N1)
retry count = 1
The duplicated address node with the higher Partition Identity becomes a Requestor and asks its neighboring node to become its Initiator
The Initiator broadcasts an IQ to all nodes of the MANET group
O(N1+ N2)
Increase retry count by 1
Recipient nodes reply with an affirmative or anegative response (IR) to the Initiator
O(t(N1+ N2))
No
Does the Initiator receive affirmative IR messages from all nodes?
Yes
The Initiator selects another IP address
The Initiator assigns the IP address to the requester
Does retry count reach Initiator Request Retry (n)?
No
Yes
The Initiator broadcasts an AO message to all recipient nodes of the MANET group
O(N1+ N2)
nO((t+1)(N1+ N2))
The Initiator sends an AB message to the Requester
O(1)
Address duplications all resolved?
No
Repeat for all duplicated address nodes
Yes
min{N1, N2}
Partition Merging Complete
End
MANETConf Group merging case complexity
min{N1, N2} (nO((t+1)(N1+ N2)) +O(N1+ N2)+O(1))+N1O(N2)+N2O(N1)+O(2)

Figure 27 The flowchart for message complexity derivation of MANETconf protocol for MANET group merging case

When the two nodes *I* and *J* associated in two different MANET groups, which are $\boldsymbol{V_i}$, and $\boldsymbol{V_j}$ respectively, become neighboring nodes to each other, they exchange their *Partition Identity*. The message complexity of exchanging their *Partition Identity* can be represented as $O(2)$. In this section, the MANET group merging case is considered.

All nodes of a MANET group know their group's (*Lowest IP, UUID*), where the *Universal Unique ID* (*UUID*) is the MAC address of the lowest IP address node. The nodes *I* and *J* can detect the merger of two different MANET groups when the two nodes (*I* and *J* of each group) exchange their *AL* sets of IP addresses, which must contain the MANET group's *Partition Identity* (*Lowest IP*, *UUID*).

Since the *AL* message is composed of a list of IP addresses of a MANET group, the size of the *AL* message will be much larger than the *Maximum Transfer Unit* (*MTU*) permitted in a MANET. It is assumed that in the worst case, each IP address in the *AL* message is equivalent to the *MTU* size message permitted in a MANET packet if the other layers' overhead is considered. Therefore, in the upper bound case, an *AL* message from MANET group $\boldsymbol{V_j}$ is segmented into N_2 number of *MTU* sized messages and transmitted in MANET group $\boldsymbol{V_i}$. In addition, the *AL* message from MANET group $\boldsymbol{V_i}$ is segmented into N_1 number of *MTU* sized messages and transmitted in MANET group $\boldsymbol{V_j}$.

The algorithm requires all nodes in MANET group $\boldsymbol{V_i}$ to broadcast the *AL* messages transferred from MANET group $\boldsymbol{V_j}$, all nodes in MANET group $\boldsymbol{V_i}$ have to broadcast N_2 number of *AL* messages. As a result, the message complexity can be represented as $N_2O(N_1)$.

Likewise, all nodes in MANET group $\boldsymbol{V_j}$ need to broadcast the *AL* messages transferred from MANET group $\boldsymbol{V_i}$, and therefore, all nodes in the MANET group $\boldsymbol{V_j}$ have to broadcast N_1 number of *AL* messages. As a result, the message complexity can be represented as $N_1O(N_2)$.

Therefore, the message complexity due to broadcasting the *AL* messages in MANET groups $\boldsymbol{V_i}$ and $\boldsymbol{V_j}$ can be represented as $N_1O(N_2)+ N_2O(N_1)$.

The duplicated address node with the higher *Partition Identity* will become the *Requestor* asking its neighboring node to become its *Initiator*. Among the duplicated addresses nodes, the node of the MANET group that has the higher partition identity (i.e., comparing the lowest IP address of each group first, and if needed, also by comparing the *UUID* of each MANET group) will become the *Requestor* and chooses one of its neighbors with a non-conflicting address as its *Initiator* to send an *IQ* message.

Any nodes detecting conflicted IP addresses become *Initiators*, when each *Initiator* broadcasts an *IQ* message to all nodes of the MANET group with the address of the *Requester*. The message complexity upper bound of broadcasting *IQ* messages can be represented as $O(N_1+ N_2)$ since the *IQ* message is broadcasted into the merged MANET.

Recipient nodes will reply with an affirmative or a negative response (using *IR* message) to the *Initiator*. Therefore, the message complexity upper bound of unicasting *IR* messages can be represented as $O(t(N_1+ N_2))$ since all nodes ($N_1+ N_2$) unicast the *IR* message and each *IR* message has the message complexity upper bound of O(t) based on *Lemma* 1.

If the initiator receives positive *IR* messages from all recipient nodes, it broadcasts an *AO* message to all recipient nodes of the MANET group. The message complexity upper bound of broadcasting the *AO* message can be represented as $O(N_1+ N_2)$ based on *Definition* 2.

***Lemma* 13** In an *IP verification procedure* of the MANET group merging case, $O((t+1)(N_1+N_2))$ is the upper bound of the maximum number of broadcasted or relayed *IQ* messages and unicasted or relayed *IR* messages when a node needs to verify its IP address in a MANET with MANETconf.

Proof. The maximum number of messages occurs when the *IP verification procedure* results in a *failure trial*. Since, the *failure trial* gives $O(N_1+N_2)$ number of broadcasted or relayed *IQ* messages based on *Definition* 2, and $O(t(N_1+N_2))$ unicasted or relayed *IR* message based on *Lemma* 1, the message complexity of the *failure trial* can be represented as $O(N_1+N_2)+O(t(N_1+N_2))$, which sums the upper bound of the maximum number of broadcasted and relayed *IQ* and unicasted and relayed *IR* messages, and can be rearranged as $O((t+1)(N_1+N_2))$. ■

If the initiator receives any negative *IR* messages from its recipient nodes, it selects another IP address and repeats the steps of broadcasting *IQ* and receiving IR messages until the retry count reaches the *retry count limit* (n). Therefore, the message complexity of broadcasting *AO* messages and receiving *IR* messages until the retry count is less than n can be represented as $n(O(N_1+N_2)+O(t(N_1+N_2)))$. After n times of repetition, if the initiator receives negative *IR* messages, it sends an *AB* message to the requestor. The message complexity of unicasting the *AB* message can be represented as $O(1)$.

Based on the above results, the following *Lemma* can be derived.

***Lemma* 14** In a *session* of a MANET group merging case, $nO((t+1)(N_1+N_2))+O(N_1+N_2)+O(1)$ is the upper bound of the maximum number of broadcasted or relayed *IQ* and *AO* messages and unicasted or relayed *IR* and *AB* messages in MANETconf.

Proof. MANETconf has a *session* and the maximum number of retries of the *IP verification procedure* is limited by *n* in the *session*. Since the *session* consists of a maximum number of *n IP verification procedures* and the upper bound of the maximum number of an *IP verification procedure* is $O((t+1)(N_1+N_2))$ based on *Lemma* 13, the message complexity of the *session* can be represented as $nO((t+1)(N_1+N_2))+O(N_1+N_2)+O(1)$ where *n* is the number of the *retry count limit* of the *IP verification procedures*, $O(N_1+N_2)$ indicates the message complexity of broadcasting the *AO* message, and $O(1)$ indicates the message complexity of unicasting the *AB* message. ■

The above *session* procedure per a duplicated address node should be repeated until all duplicated address nodes are resolved. The repetition number of the *session* procedure in a merged MANET can be expressed as $min(N_1, N_2)$. Therefore, the message complexity of the MANET group merging case can be represented as $min(N_1, N_2)\{nO((t+1)(N_1+N_2))+O(N_1+N_2)+O(1)\}$.

Based on the above results, the following *Lemma* can be derived.

***Lemma* 15** In resolving all duplicated addresses of a MANET group merging case with MANETconf, $min(N_1,N_2)\{nO((t+1)(N_1+N_2))+O(N_1+N_2) + O(1)\} +O(2) +N_1O(N_2)+N_2O(N_1)$ is the upper bound of the maximum number of messages.

Proof. Resolving all duplicated addresses of the MANET group merging case consists of the steps of unicasting the *Partition Identity*, and broadcasting the *AL* message. The session procedures are repeated $min(N_1,N_2)$ times for all the duplicated addresses nodes, and the message complexity of resolving all duplicated addresses of the MANET group merging case can be obtained from the summation of the complexity of the *session* procedures (repeated $min(N_1,N_2)$ times), unicasting the *Partition Identity*, and broadcasting the *AL* message, which can be represented as $min(N_1,N_2)\{nO((t+1)(N_1+N_2))+O(N_1+N_2)+O(1)\}+O(2)+N_1O(N_2)+N_2O(N_1)$. ■

3.7. Numerical Results

To show the message complexity of each address autoconfiguration protocols in a standalone MANET where a MANET has no connection to an external network like the Internet [27], a computer based simulation where nodes are randomly distributed with uniform density in a network area of $1km^2$ is adopted. A discrete-event *MATLAB* simulator was developed in order to verify the various network topologies and to calculate the message complexity of each address autoconfiguration protocol. The random node generator in the *MATLAB* simulator was verified so that the average number of nodes per cluster in the *ADB* algorithm [32] was matched with the results in [32] performed by *QualNet*.

In [32], when the numbers of nodes are 100, 125, 150 and 175, the average numbers of nodes per cluster in the ADB algorithm are calculated as 7.79, 9.87, 12.36 and 13.94

respectively by using QualNet. In the simulator performed in this dissertation, with the same simulation environment, the average numbers of nodes per cluster are 7.94, 9.57, 11.26, and 12.88. A random generator provided in QualNet is used in [32] and a random generator provided in Matlab is used by the simulator in this thesis. Therefore, the simulation results performed in Duplicate Address Detection (DAD) algorithms, such as Strong DAD, Weak DAD with Proactive routing protocols and On Demand routing protocols and MANETconf could be verified based on Table 2 since error permission range is considered in the random generators in QualNet and Matlab and the maximum number of total nodes in a MANET is 50.

Table 2 Average numbers of nodes per cluster in ADB algorithms performed at QualNet and Matlab

No. of total Nodes in a network	*The average numbers of nodes per cluster in ADB (QualNet, [32])*	*The average numbers of nodes per cluster in ADB (Matlab)*	*|Difference|, (%)*
100	7.79	7.94	0.15, (1.9%)
125	9.87	9.57	0.27, (3.0%)
150	12.36	11.26	1.1, (8.9%)
175	13.94	12.88	1.04, (7.6%)

Based on the randomly distributed nodes in a network area and their transmission range, a MANET has been configured as several subgroups. At least two subgroups are generated in the simulator and considered in this simulation in order to analyze the single node joining case and the MANET group merging case. In addition, the two subgroups that are composed of the maximum number of nodes in MANET are considered. The following definition can be defined in order to explain the simulator for the MANET group merging case in detail.

***Definition* 3**

Several terms related with a graph, which are used in the computer based simulation, are defined as below. A MANET group can be represented as $G(\boldsymbol{V},\boldsymbol{E})$ and its subgraphs can be represented as $P(\boldsymbol{V_i},\boldsymbol{E_i})$ and $Q(\boldsymbol{V_j},\boldsymbol{E_j})$ [32].

1. A MANET G is *k-connected*, represented by $CONN(G,k)$, where deleting any k-1 vertices results in a graph that is still connected [32].
2. Disjoint subgraphs P and Q are *neighboring subgraphs* in G, represented by $NEIGHBOR_G(P, Q)$, if $\exists u \exists v(u \in \boldsymbol{V_i} \wedge v \in \boldsymbol{V_j} \wedge (u,v) \in \boldsymbol{E})$ [32].
3. Disjoint subgraphs P and Q are *neighboring k-connected subgraphs* in G, represented by $NBRCONN_G(P, Q, k)$, if $CONN(P,k)$, and $CONN(Q,k)$ and $\exists (u_1,v_1), \cdots, (u_k,v_k)$ such that $u_1,\cdots, u_k \in \boldsymbol{V_i}$ and $v_1,\cdots, v_k \in \boldsymbol{V_j}$. Therefore, $NBRCONN_G(P, Q, 1)$ equals $NEIGHBOR_G(P, Q)$ [32]. ■

Based on the above *Definition* 3, $NBRCONN_G(P, Q, 1)$, which is same as $NEIGHBOR_G(P, Q)$, is used in the computer simulation in order to perform the MANET group merging case.

A conflict probability is defined as the probability that a node requests its IP address, and the requested IP address is duplicated in a MANET group. The conflict probability depends on the size of the address and the number of nodes in a MANET group [22]. The conflict probability is selected from among 0.1, 0.3, 0.5, 0.7, 0.9 and 1. When the conflict probability is one, the selected or reselected IP address is always duplicated with one of the IP addresses in a MANET group, and this is called as the message complexity in the worst case scenario. It can be expected that in the simulation of running Weak DAD with on demand routing protocols, different occurrence probability of unicasting a *RP* message at a certain conflict probability will have various results in the message complexity. Therefore, it is assumed that the occurrence probability of unicasting a *RP* message in a node is not zero and is the same as the conflict probability of the requested IP address in the node for simplicity.

The most common flooding method is used in the simulation where every node retransmits an *Address Request* message to its entire one-hop neighbors whenever it receives the first copy of message [36].

Dijkstra's shortest path algorithm at each node is used to calculate the number of hops in unicasting or relaying a unicasted *Address Reply* message from a destination node to a source node. The transmission range of the nodes changes the number of hops. The upper bound of the message complexity is calculated based on the derived *Lemma's equation* where the maximum number of nodes in a reverse path at each unicast case is used to calculate $O(t)$ in each upper bound *Lemma's equation*.

In the Strong DAD protocol, five is used for *retry count limit* (n) and three is used for *DAD retry count limit* (m). In the Weak DAD and MANETconf protocols, five is used for *retry count limit* (n) and one is used for *DAD retry count limit* (m).

100m is selected as the transmission range of nodes. The number of nodes is varied from 10 to 50 for the transmission range.

Based on a random network topology scenario, the number of nodes and the network topology are not changed during the simulation in order to analyze the message complexity of each protocol and compare the mathematical derivation and the simulation result. Each node has its own scenario based on the steps of the flowchart step. Monte

Carlo computer simulation performing 2,000 iterations at each conflict probability is used to evaluate various network topologies for random scenarios and to calculate the average number of messages for each scenario where each topology is composed of a different number of free trees. In this chapter, based on the calculation of the control overhead (message complexity), the scalability issue of MANET address autoconfiguration protocols is analyzed with the increase in the number of nodes in a MANET [32].

3.7.1. Strong DAD: Single Node Joining Case

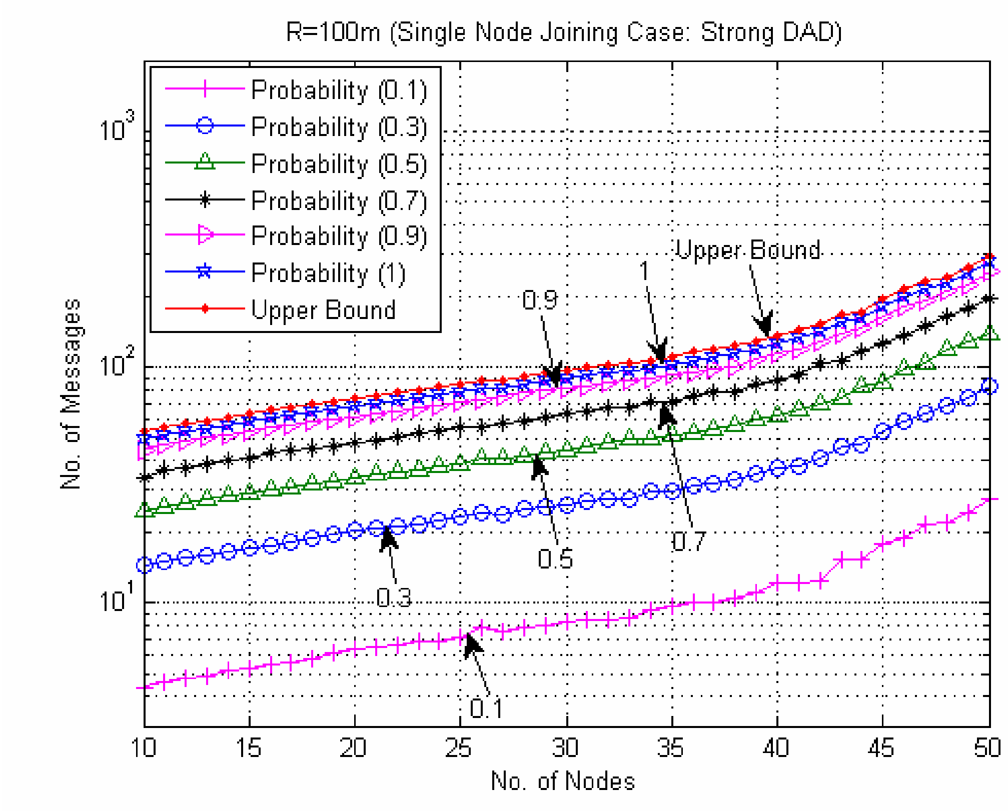

Figure 28 The message complexity (*Single Node Joining Case: Strong DAD*)

Fig. 28 shows the message complexity of the single node joining case in the Strong DAD protocol based on the different conflict probabilities. From Fig. 28 to Fig. 35, the horizontal axis shows the number of nodes in the network area and the vertical axis shows the number of messages at each case. As shown in Fig. 28, the different conflict probability and the different number of nodes result in different number of messages. As conflict probability is increased, it is shown that the number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $n(mO(N)+O(t))$, which is derived in *Lemma* 3.

3.7.2. Strong DAD: MANET Group Merging Case

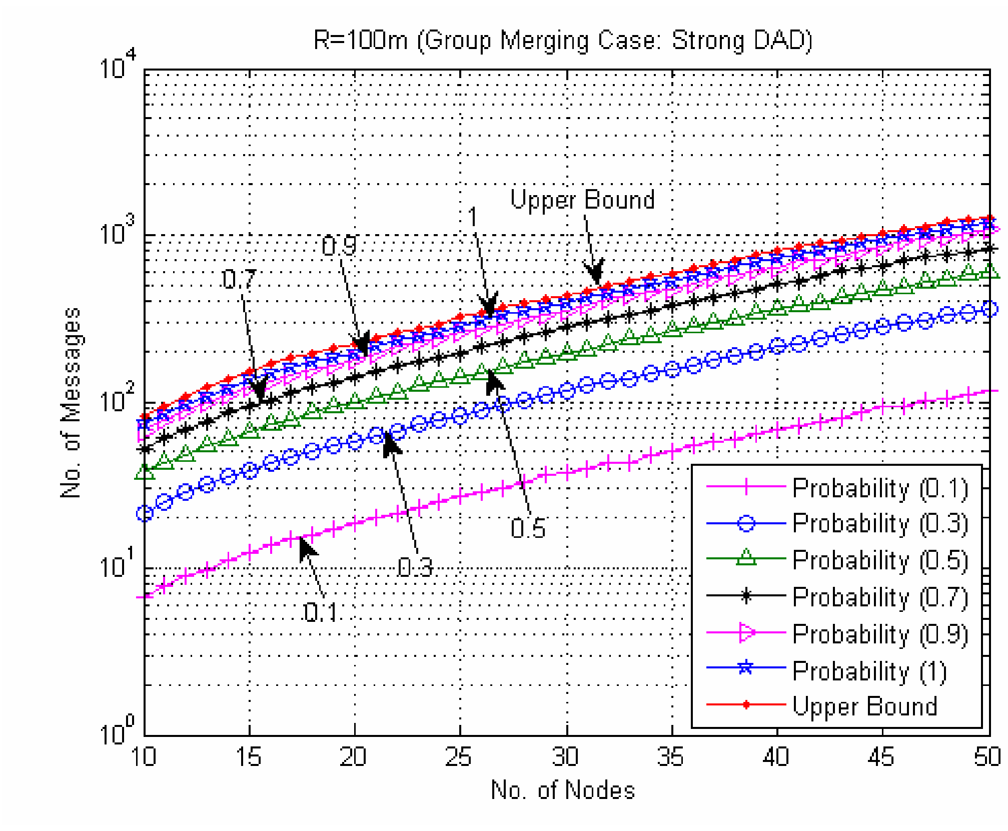

***Figure 29 The message complexity** (**MANET Group Merging Case: Strong DAD**)*

Fig. 29 shows the message complexity of the group merging case in the Strong DAD protocol based on the different conflict probabilities. As shown in Fig. 29, the different conflict probability and the different number of nodes result in different number of messages. As conflict probability is increased, it is shown that the number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $nN_1(mO(N_2)+O(t))$, which is derived in *Lemma* 4.

3.7.3. Weak DAD with Proactive Routing Protocols: Single Node Joining Case

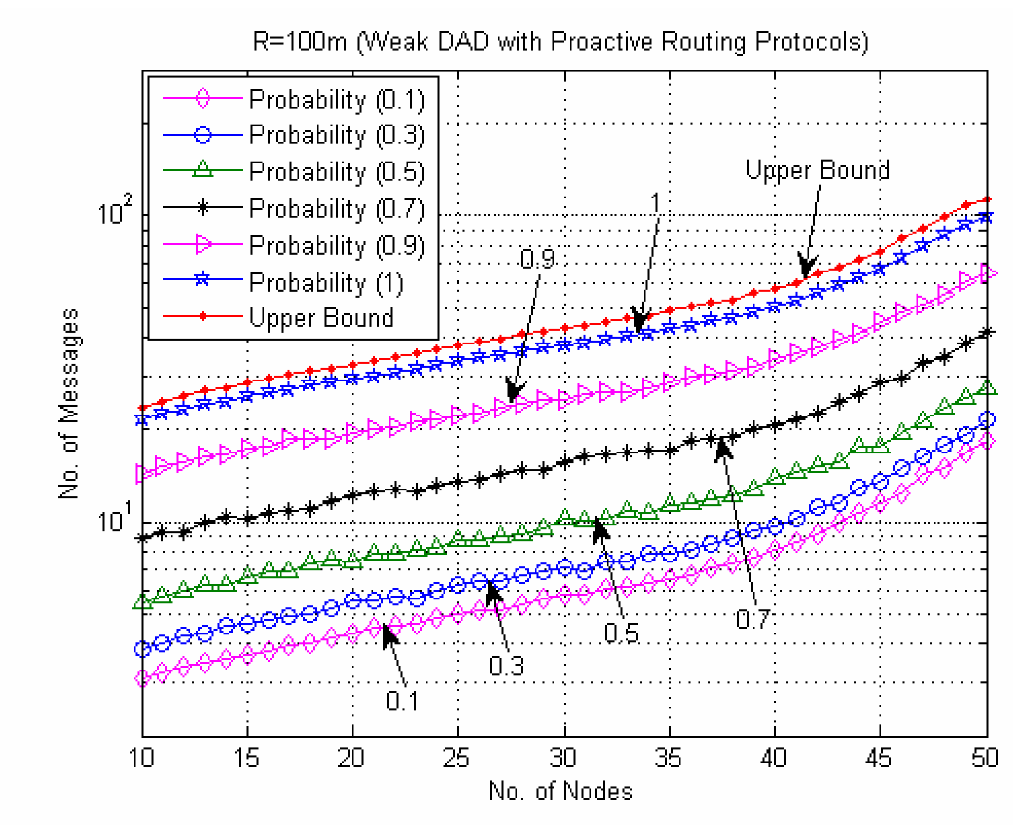

Figure 30 The message complexity of Weak DAD with Proactive Routing Protocols: Single Node Joining Case

Fig. 30 shows the message complexity of the single node joining case in the Weak DAD with proactive routing protocols based on the different conflict probabilities. As shown in

Fig. 30, the different conflict probability and the different number of nodes result in the different number of messages. As conflict probability is increased, it is shown that number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $n(O(N)+O(t))$, which is derived in *Lemma* 6.

3.7.4. Weak DAD with Proactive Routing Protocols: MANET Group Merging Case

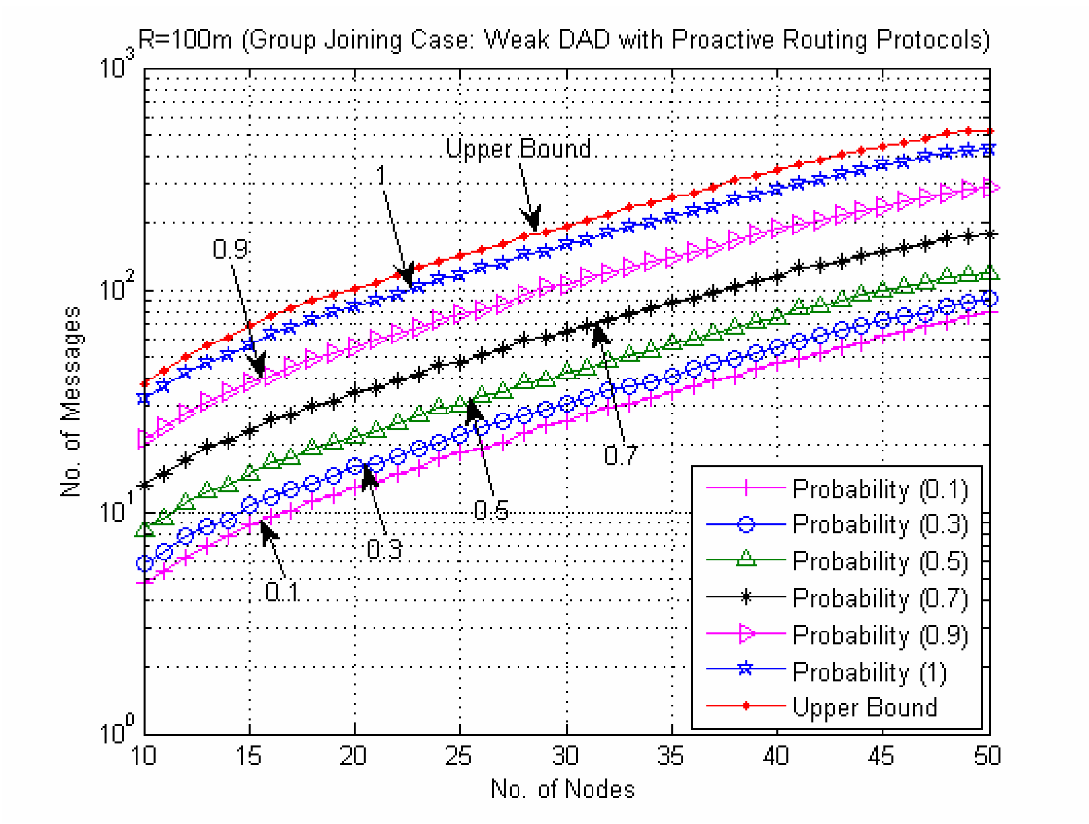

Figure 31 The message complexity of Weak DAD with Proactive Routing Protocols: MANET Group Merging Case

Fig. 31 shows the message complexity of the group merging case in the Weak DAD with proactive routing protocols based on the different conflict probabilities. As shown in Fig.

31, the different conflict probability and the different number of nodes result in different number of messages. As conflict probability is increased, it is shown that the number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $nN_1(O(N_2)+O(t))$, which is derived in *Lemma* 9.

3.7.5. Weak DAD with On Demand Routing Protocols: Single Node Joining Case

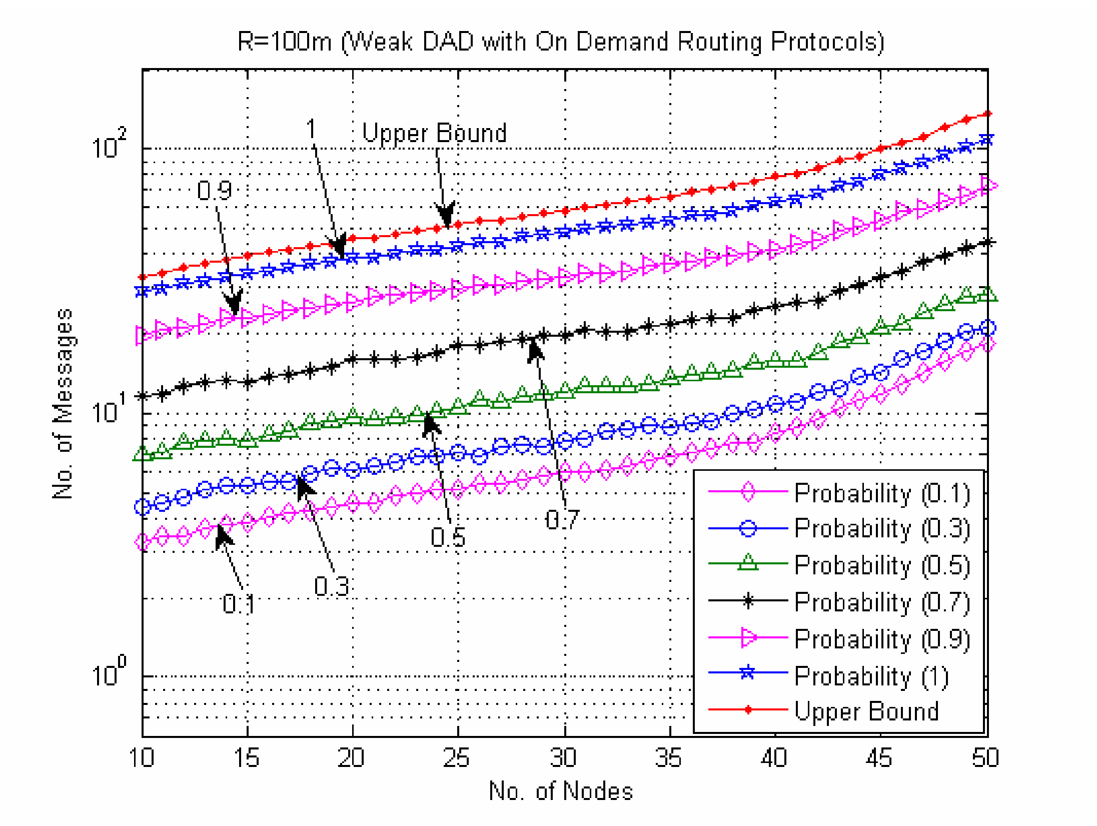

Figure 32 The message complexity of Weak DAD with On Demand Routing Protocols: Single Node Joining Case

Fig. 32 shows the message complexity of the single node joining case in the Weak DAD with on demand routing protocols based on the different conflict probabilities. As shown in Fig. 32, the different conflict probability and the different number of nodes result in different number of messages. As conflict probability is increased, it is shown that the number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $n(O(N)+2O(t))$, which is derived in *Lemma* 8.

3.7.6. Weak DAD with On Demand Routing Protocols: MANET Group Merging Case

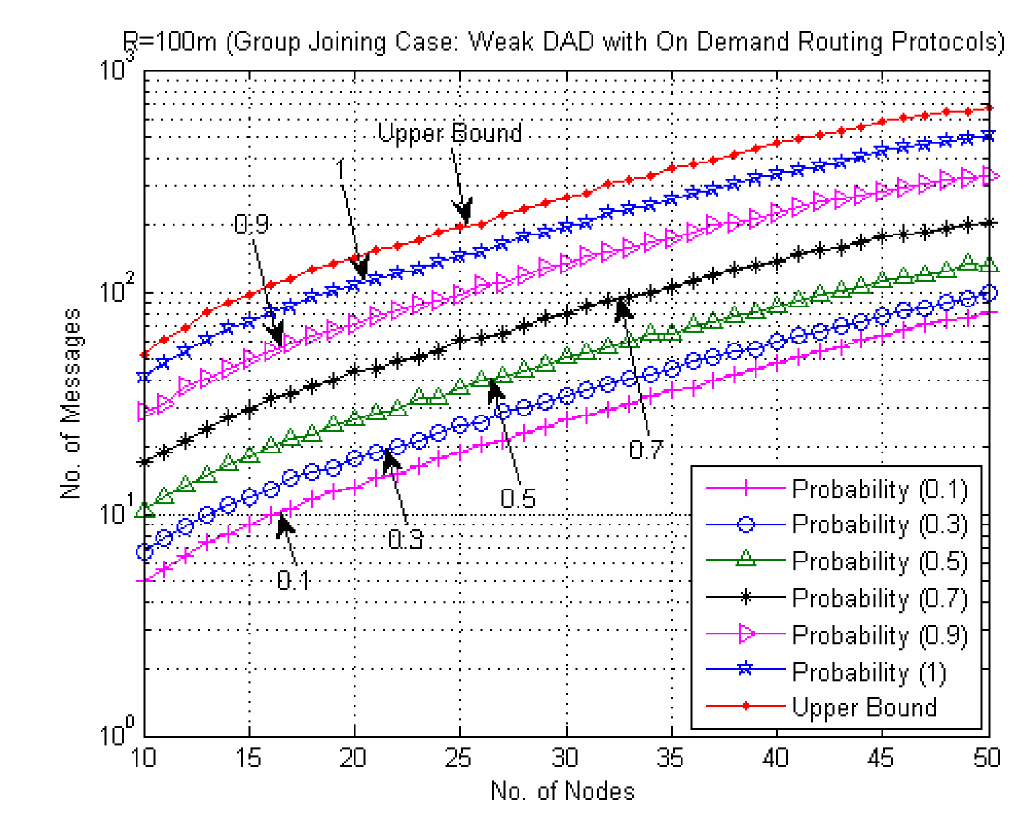

Figure 33 The message complexity of Weak DAD with On Demand Routing Protocols: MANET Group Merging Case

Fig. 33 shows the message complexity of the group merging case in the Weak DAD with on demand routing protocols based on the different conflict probabilities. As shown in

Fig. 33, the different conflict probability and the different number of nodes result in different number of messages. As conflict probability is increased, it is shown that the number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $nN_1(O(N_2)+2O(t))$, which is derived in *Lemma* 10.

3.7.7. MANETconf: Single Node Joining Case

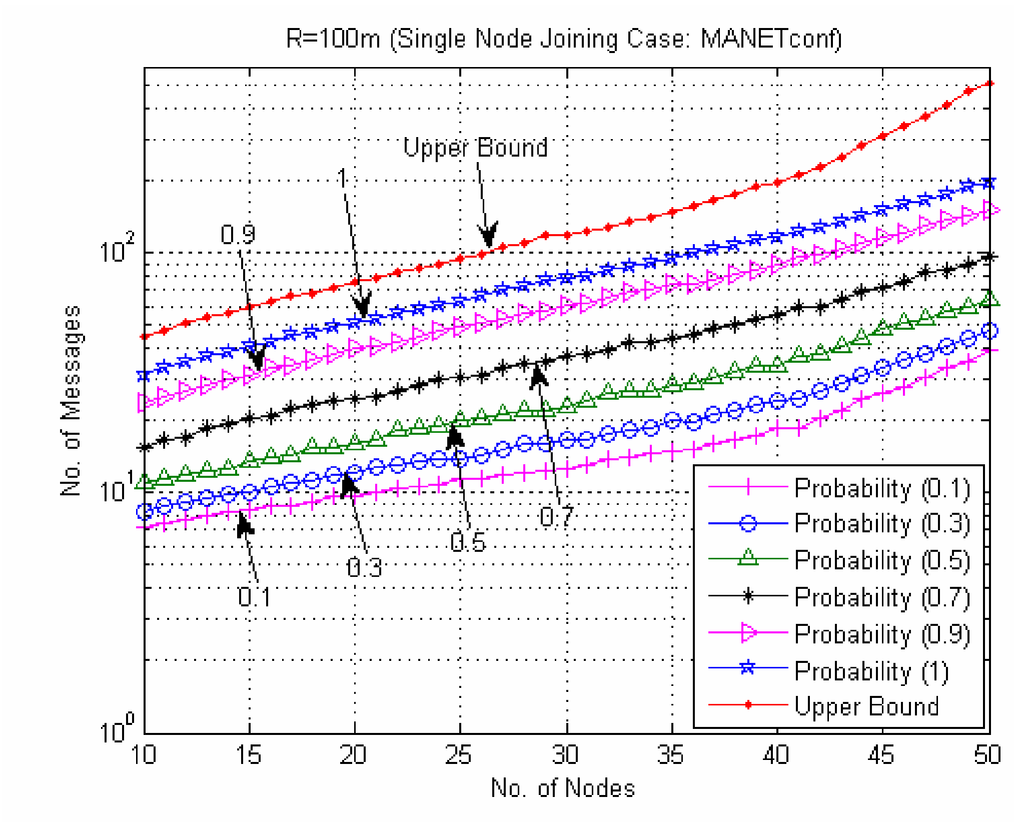

Figure 34 The message complexity of MANETocnf: Single Node Joining Case

Fig. 34 shows the message complexity of the single node joining case in the MANETconf protocol based on the different conflict probabilities. As shown in Fig. 34, the different conflict probability and the different number of nodes result in different number of messages. As conflict probability is increased, it is shown that the number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $nO((t+1)N)+O(N)+O(2)$, which is derived in *Lemma* 12.

3.7.8. MANETconf: MANET Group Merging Case

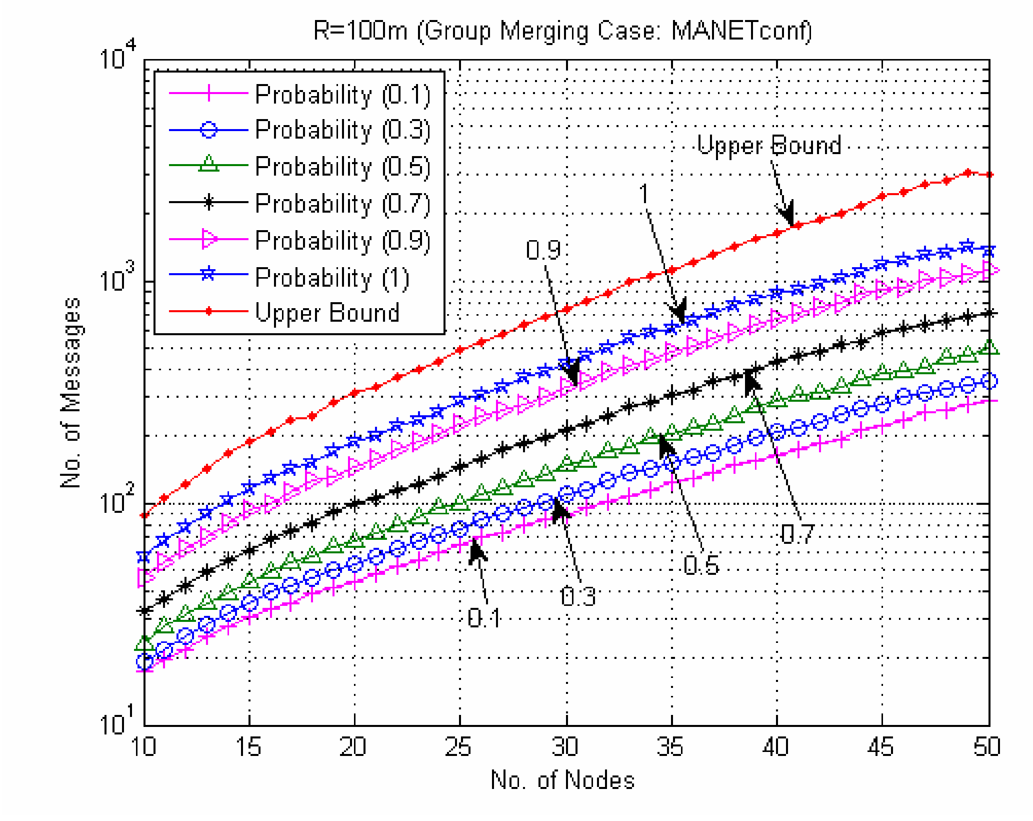

Figure 35 The message complexity of MANETocnf: MANET Group Merging Case

Fig. 35 shows the message complexity of the MANET group merging case in the MANETconf protocol based on the different conflict probabilities. As shown in Fig. 35, the different conflict probability and the different number of nodes result in different number of messages. As conflict probability is increased, it is shown that the number of messages to resolve the duplicated IP address is also increased. When the conflict probability equals one, the number of message complexity is under the upper bound message complexity, $min(N_1,N_2)\{nO((t+1)(N_1+N_2))+O(N_1+N_2)+O(1)\}+O(2)+N_1O(N_2)+N_2O(N_1)$, which is derived in *Lemma* 15.

3.8. Percentage overhead in the Single Node Joining Case

3.8.1. Percentage overhead comparison (Conflict probability of 0.5)

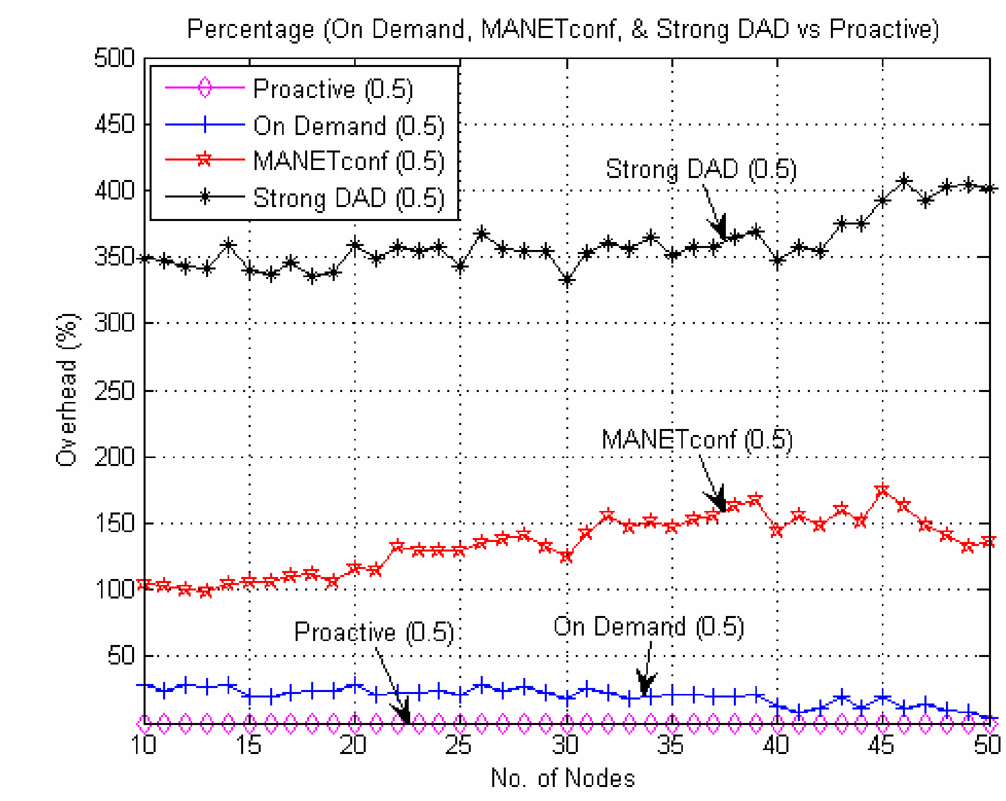

Figure 36 The percentage overhead comparison (***Single Node Joining Case, p*=0.5, *R*=100**)

Fig. 36 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in single node joining case with a conflict probability of 0.5. It is shown that as the number of nodes increases, the message complexity of Weak DAD with on demand routing protocols tends to converge to the message complexity of Weak DAD with proactive routing protocols.

The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 29.30 % and 20.22 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 3 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

***Table 3 Percentage Overhead* (*Weak DAD with on demand routing protocols*, *p=0.5*)**

***Percentage Overhead* (*Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p=0.5*)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	28.42	24.15	28.60	27.38	29.30	19.53	20.36	23.39	23.91
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	24.16	29.19	20.80	23.32	22.42	23.80	20.72	28.19	23.60
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	26.64	22.90	18.23	25.94	22.51	18.03	20.45	20.64	21.43
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	19.90	19.58	21.95	13.06	8.08	11.31	19.88	10.65	19.67
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	11.24	14.05	9.20	8.75	3.72	29.30		20.22	

The percentage overheads of message complexity of MANETconf are 174.04 % and 134.43 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 4 shows the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

Table 4 Percentage Overhead (MANETconf, p=0.5)

Percentage Overhead (MANETconf vs. Weak DAD with Proactive routing protocols, p=0.5)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	104.8	103.16	100.51	99.02	103.93	105.55	106.15	110	111.37
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	105.52	116.46	114.39	132.64	129.25	128.84	129.99	135.22	137.64
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	141.86	132.45	125.77	142.63	155.2	146.29	150.66	146.99	152.39
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	155.65	162.79	166.82	144.55	155.25	148.88	160.7	151.66	174.04
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	163.19	148.87	141.77	132.84	136.15	174.04		134.43	

The percentage overheads of message complexity of Strong DAD are 407.18 % and 360.08 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 5 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

Table 5 Percentage Overhead (Strong DAD, p=0.5)

Percentage Overhead (Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.5)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	348.41	347.80	342.52	341.74	358.56	339.28	337.22	345.99	334.91
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	338.17	359.11	348.76	357.42	354.64	357.83	342.82	368.38	355.39
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	354.90	354.33	333.02	352.67	359.88	355.49	364.65	351.41	356.90
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	357.69	365.16	368.90	346.95	357.25	354.83	375.42	374.84	393.16
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	407.18	392.11	402.35	403.95	401.48	407.18		360.08	

Therefore, it can be said that at a conflict probability of 0.5, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in the single node joining case. However, since the proactive routing protocols depend on a periodic message to update the network topology and on demand routing protocols does not have a periodic message, it is not fair for the two routing protocols to be compared using the number of messages. The result above can be used only in the case where Weak DAD protocol uses MANET routing protocols for nodes to configure its IP address and solve the duplicate IP address detection.

3.8.2. Percentage overhead comparison (Conflict probability of 0.7)

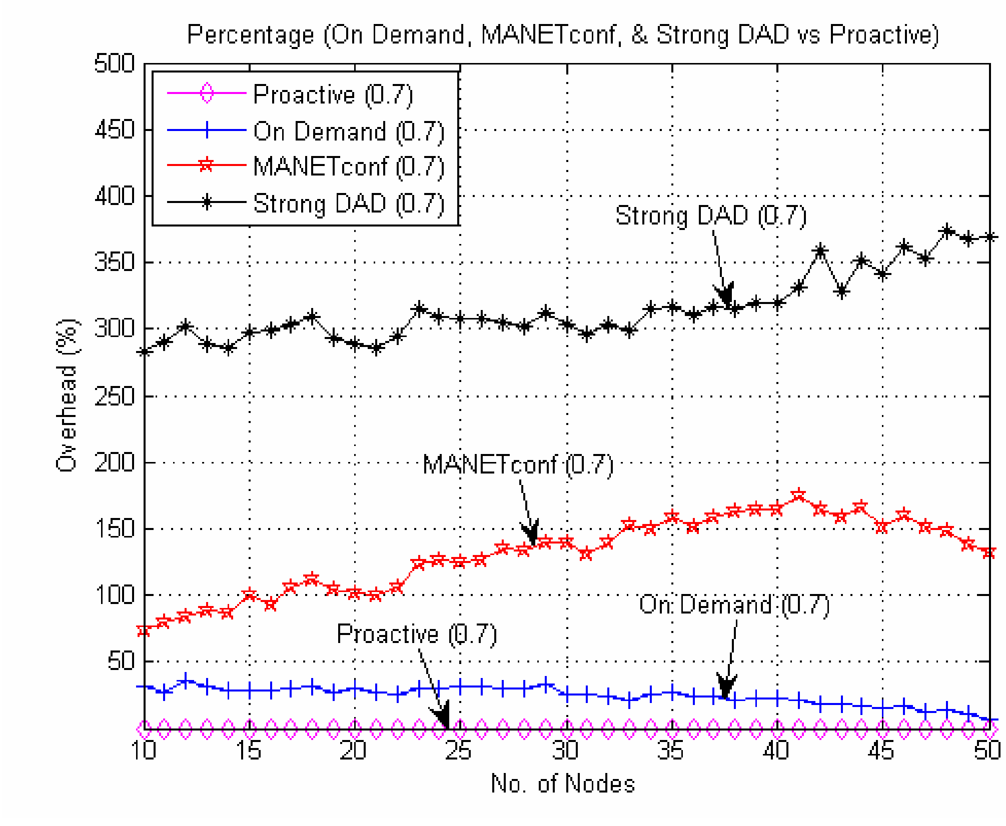

***Figure 37 The percentage overhead comparison** (**Single Node Joining Case, p=0.7, R=100**)*

Fig. 37 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in single node joining case with a conflict probability of 0.7. It is shown that as the number of nodes increases, the message complexity of Weak DAD with on demand routing protocols is inclined to converge to the message complexity of Weak DAD with proactive routing protocols.

The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 36.15 % and 24.86 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 6 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

***Table 6 Percentage Overhead* (*Weak DAD with on demand routing protocols*, *p=0.7*)**

***Percentage Overhead* (*Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p=0.7*)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	31.03	26.85	36.15	31.86	28.00	28.03	28.86	29.71	31.75
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	27.36	30.10	27.10	25.93	30.46	30.16	32.14	32.20	30.26
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	30.02	33.68	25.19	26.23	23.79	21.97	25.36	27.00	24.35
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	24.50	21.66	22.65	22.53	21.80	18.43	18.05	17.03	15.68
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	17.49	13.01	14.03	10.76	6.13	36.15		24.86	

The percentage overheads of message complexity of MANETconf are 174.56 % and 130.92 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 7 shows

the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

Table 7 Percentage Overhead (MANETconf, p=0.7)

Percentage Overhead (MANETconf vs. Weak DAD with Proactive routing protocols, p=0.7)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	74.27	79.71	83.82	88.32	87.25	100.11	93.21	106.37	111.94
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	105.27	101.05	100.54	105.66	123.53	126.80	124.52	126.52	135.12
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	134.05	139.18	139.92	131.17	139.43	152.29	149.39	158.64	151.88
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	159.04	163.75	163.88	164.18	174.56	164.84	158.85	165.49	152.01
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	160.10	152.06	148.41	137.62	132.82	174.56		130.92	

The percentage overheads of message complexity of Strong DAD are 374.24 % and 315.46 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 8 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

Table 8 Percentage Overhead (Strong DAD, p=0.7)

Percentage Overhead (Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.7)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	283.60	289.67	301.33	289.04	285.70	297.85	299.74	303.05	309.74
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	292.86	288.86	286.40	294.18	314.93	308.94	308.13	307.40	304.87
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	302.50	312.84	303.90	296.04	303.31	299.58	314.67	316.49	310.53
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	317.14	315.20	319.83	319.90	330.78	358.57	328.50	351.07	341.05
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	361.37	353.59	374.24	367.55	368.73	374.24		315.46	

Therefore, it can be said that at a conflict probability of 0.7, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in the single node joining case.

3.8.3. Percentage overhead comparison (Conflict probability of 0.9)

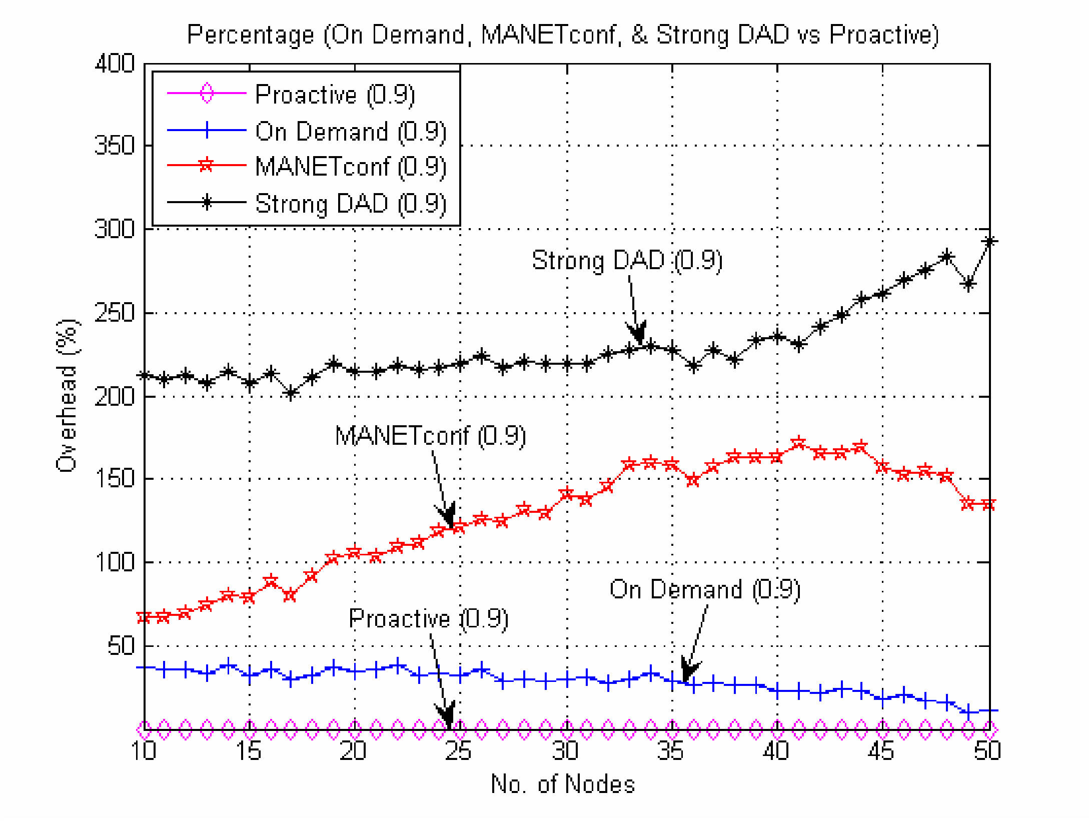

Figure 38 The percentage overhead comparison (***Single Node Joining Case, p*=0.9, *R*=100**)

Fig. 38 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in single node joining case with conflict probability of 0.9.

It is shown that as the number of nodes increases, the message complexity of Weak DAD with on demand routing protocols tends to converge to the message complexity of Weak DAD with proactive routing protocols.

The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 38.41 % and 28.70 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead

respectively. Table 9 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

Table 9 Percentage Overhead (Weak DAD with on demand routing protocol, p=0.9)

Percentage Overhead (Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p=0.9)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	37.36	36.19	35.64	33.98	38.41	32.26	35.53	30.32	32.38
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	36.90	35.16	35.49	38.41	32.69	33.80	32.67	36.29	29.19
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	30.04	28.18	29.40	31.22	27.94	30.20	32.91	28.77	26.09
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	27.76	26.85	26.46	22.37	22.36	21.92	24.15	23.39	18.75
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	21.11	16.70	16.21	9.78	11.53	38.41		28.70	

The percentage overheads of message complexity of MANETconf are 171.28 % and 127.74 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 10 shows the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

Table 10 Percentage Overhead (MANETconf, p=0.9)

Percentage Overhead (MANETconf vs. Weak DAD with Proactive routing protocols, p=0.9)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	67.31	67.33	69.70	74.18	79.64	78.85	88.41	79.92	92.38
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	102.83	105.85	103.59	109.26	111.46	118.39	121.13	125.95	124.34
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	131.99	129.31	140.96	137.88	145.91	158.24	160.01	158.93	149.26
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	157.53	162.72	163.13	163.34	171.27	165.94	165.07	168.86	157.30
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	152.28	155.36	151.53	134.75	135.25	171.28		127.74	

The percentage overheads of message complexity of Strong DAD are 292.58 % and 229.43 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 11 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

Table 11 Percentage Overhead (Strong DAD, p=0.9)

Percentage Overhead (Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.9)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	212.00	210.11	212.24	207.72	214.80	207.85	214.08	201.85	211.02
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	219.30	215.09	215.22	218.45	215.39	216.71	219.74	223.78	216.68
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	220.74	218.85	219.68	219.23	225.45	227.11	230.09	227.20	218.43
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	226.97	221.60	233.40	235.22	230.80	242.09	249.14	258.44	261.42
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	269.24	275.00	284.12	267.66	292.58	292.58		229.43	

Therefore, it can be said that at a conflict probability of 0.9, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in the single node joining case.

3.8.4. Percentage overhead comparison (Conflict probability of 1)

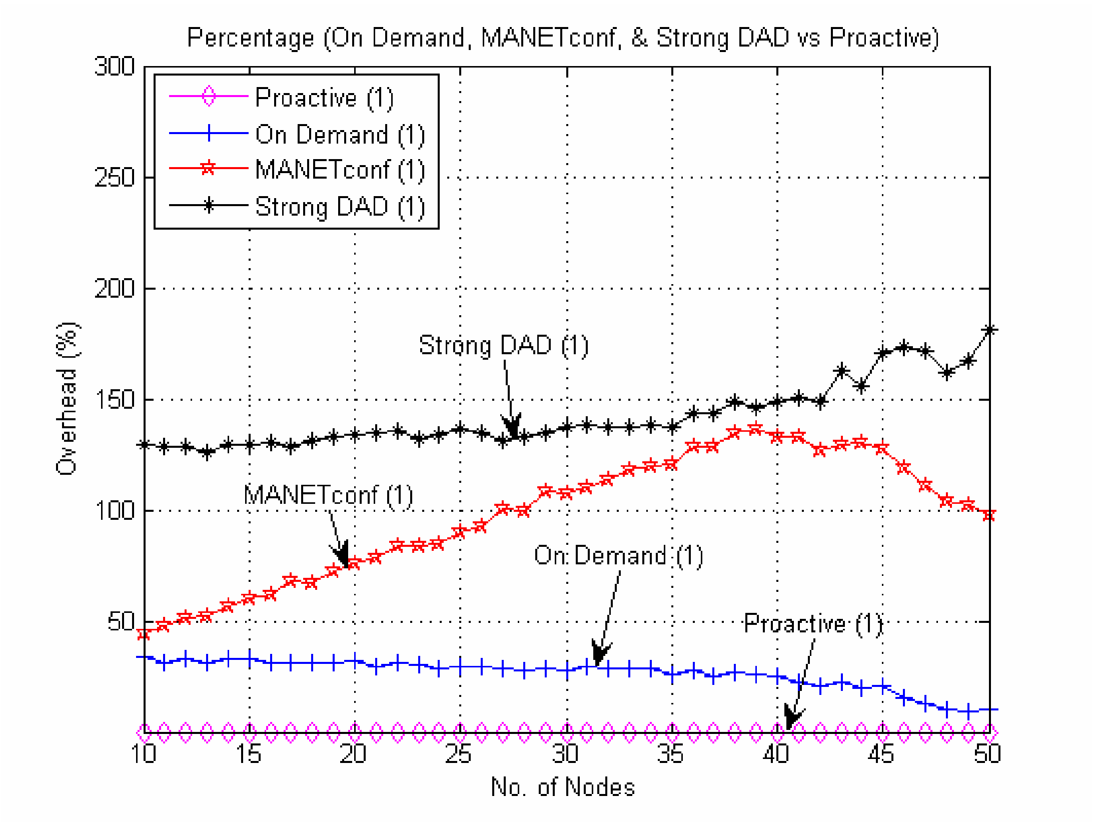

Figure 39 The percentage overhead comparison (***Single Node Joining Case, p*=1, *R*=100**)

Fig. 39 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in single node joining case with conflict probability of one.

It is shown that as the number of nodes increases, the message complexity of Weak DAD with on demand routing protocols is inclined to converge to the message complexity of Weak DAD with proactive routing protocols.

The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 33.71 % and 26.31 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead

respectively. Table 12 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

Table 12 Percentage Overhead **(*Weak DAD with on demand routing protocol*, *p*=1)**

***Percentage Overhead* (*Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p*=1)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	33.71	31.21	32.82	31.09	32.98	32.87	31.30	31.26	30.86
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	31.10	31.87	29.82	31.52	30.28	28.25	29.67	29.85	28.23
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	28.09	28.31	27.55	29.57	28.42	28.67	28.33	25.79	27.65
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	24.82	27.05	26.00	25.40	22.25	20.47	22.63	20.12	20.67
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	15.78	12.40	10.55	9.38	10.05	33.71		26.31	

The percentage overheads of message complexity of MANETconf are 136.64 % and 97.89 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 13 shows the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

Table 13 Percentage Overhead **(*MANETconf*, *p*=1)**

Percentage Overhead **(*MANETconf vs. Weak DAD with Proactive routing protocols, p*=1)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	44.29	47.60	51.37	52.60	56.79	59.91	62.00	68.03	67.12
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	72.85	75.82	78.67	83.77	83.75	84.71	89.93	92.72	100.57
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	99.96	108.44	107.70	110.15	113.59	117.79	119.42	120.83	128.48
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	128.56	134.91	136.64	132.95	133.22	127.17	129.07	130.14	127.68
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	118.94	111.38	103.69	102.42	98.05	136.64		97.89	

The percentage overheads of message complexity of Strong DAD are 180.99 % and 142.30 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 14 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

***Table 14 Percentage Overhead* (*Strong DAD*, *p*=1)**

***Percentage Overhead* (*Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.9*)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	129.84	128.51	128.86	126.09	129.16	129.14	130.39	128.38	131.32
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	132.86	133.77	134.50	135.33	132.37	133.57	136.23	134.30	131.02
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	132.77	135.02	137.29	138.57	137.49	137.36	138.60	137.38	143.35
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	143.31	148.94	146.11	148.78	150.07	148.90	163.10	155.86	170.74
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	173.02	171.80	161.96	167.25	180.99	180.99		142.30	

Therefore, it can be said that at a conflict probability of one, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in the single node joining case.

3.8.5. Observations in the single node joining case

Based on the observations from section 3.8.1 to section 3.8.4, it is shown that with the increase of the conflict probability from 0.5 to 1, the maximum overhead percentage of the message complexity of Weak DAD with on demand routing protocols is changed from 29.30 %, to 36.15 %, to 38.41 %, and to 33.71 %, the maximum overhead percentage of the message complexity of MANETconf is changed from 174.04 %, to 174.56 %, to 171.28 %, and to 136.64 % and the maximum overhead percentage of the message complexity of Strong DAD is decreased from 407.18 %, to 374.24 %, to 292.58 %, and to 180.99 % as shown in Table 15.

***Table 15 Maximum overhead percentage* [%]**

p	Weak DAD with on demand routing protocols	MANETconf	Strong DAD
0.5	29.30	174.04	407.18
0.7	36.15	174.56	374.24
0.9	38.41	171.28	292.58
1	33.71	136.64	180.99

As shown in Table 16, with the increase of the conflict probability from 0.5 to 1, the average overhead percentage of the message complexity of Weak DAD with on demand routing protocols is changed from 20.22 %, to 24.86 %, to 28.70 %, and to 26.31 %, the average overhead percentage of the message complexity of MANETconf is decreased from 134.43 %, to 130.92 %, to 127.74 %, and to 97.89 %, and the average overhead percentage of the message complexity of Strong DAD is decreased rapidly from 360.08 %, to 315.46 %, to 229.43 %, and to 142.30 %.

Table 16 ***Average overhead percentage*** **[%]**

p	**Weak DAD with on demand routing protocols**	**MANETconf**	**Strong DAD**
0.5	20.22	134.43	360.08
0.7	24.86	130.92	315.46
0.9	28.70	127.74	229.43
1	26.31	97.89	142.30

Since Strong DAD uses a *DAD retry count limit m*=3,, it can be expected that the percentage overhead of Strong DAD will be three times larger than the one of Weak DAD with proactive routing protocols. Therefore, the average overhead percentages with conflict probability of 0.5, 0.7, and 0.9 tend to follow the expected result (behavior); however, the average overhead percentage with conflict probability of one does not follow the expected result (behavior).

The maximum or average percentage overhead of the message complexity in the case of conflict probability of one has a little difference in value between MANETconf (97.89 %, in the case of average percentage overhead) and Strong DAD (142.30 %, in the case of average percentage overhead), which means that when the conflict probability is close to one, there is not much difference in the message complexity between MANETconf and Strong DAD.

Based on the percentage overhead of the message complexity of MANETconf, unicasting by all nodes causes approximately 135 % (175 %, from the results of maximum overhead percentage) more overhead than unicasting by a single node in the single node joining case.

Based on the percentage overhead of the message complexity of Weak DAD with on demand routing protocols, another unicasting mechanism causes approximately 29 % (39 %, from the results of maximum overhead percentage) more overhead compared to one of a single unicasting mechanism in the single node joining case.

Therefore, based on the maximum or average percentage overhead of the message complexity, a broadcast is a complex operation in comparison to a unicast since a broadcast causes more traffic messages and procedural operations than a unicast operation in MANETs.

3.9. Percentage overhead in MANET Group Merging Case

3.9.1. Percentage overhead comparison (Conflict probability of 0.5)

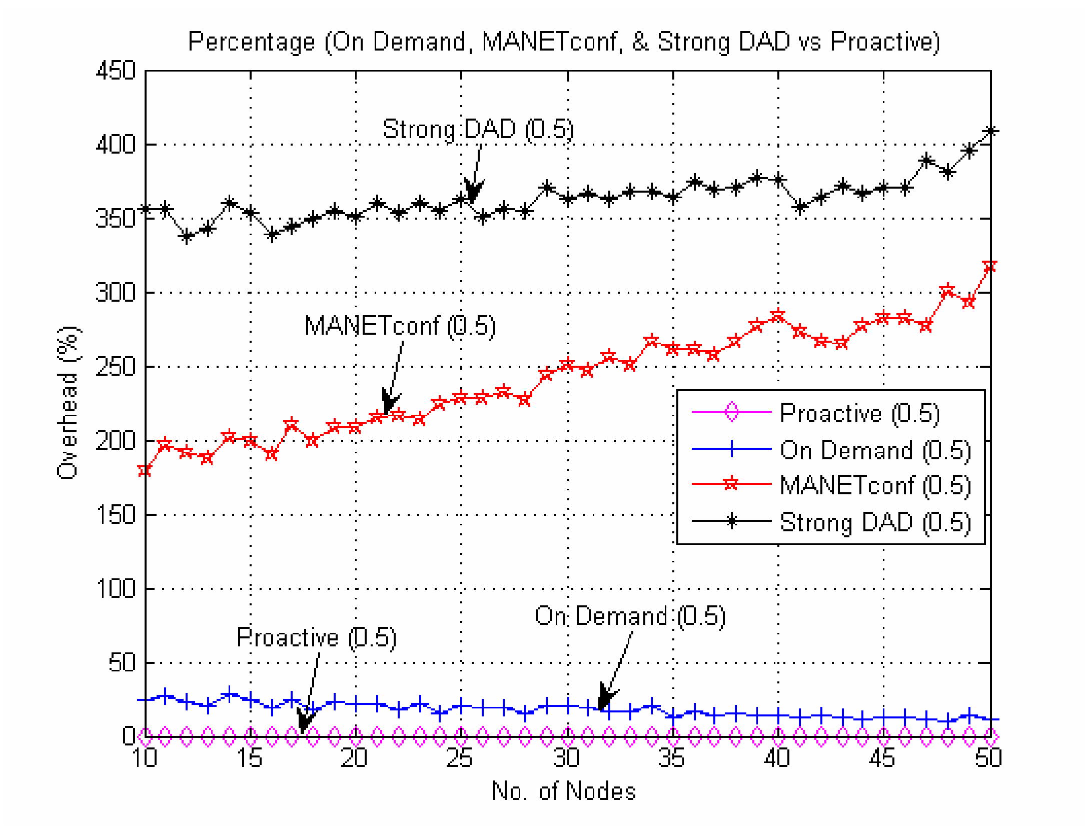

Figure 40 The percentage overhead comparison (***MANET Group Merging Case, p*=0.5, *R*=100**)

Fig. 40 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in MANET group merging case with a conflict probability of 0.5. It is shown that as the number of nodes increases, the message complexity of Weak DAD with on demand routing protocols converges to the message complexity of Weak DAD with proactive routing protocols. The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 28.41 % and 17.66 % compared to the

Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 17 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

***Table 17 Percentage Overhead* (*Weak DAD with on demand routing protocols*, *p=0.5*)**

***Percentage Overhead* (*Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p=0.5*)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	24.85	27.35	22.98	19.84	28.41	24.07	19.47	24.33	18.01
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	22.78	21.73	21.71	18.15	21.58	15.03	20.87	19.12	18.88
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	15.42	20.35	20.00	19.30	16.83	16.99	20.42	12.44	16.09
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	13.50	15.68	13.60	13.83	11.94	13.80	12.95	11.53	12.16
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	12.33	10.95	9.71	13.97	11.17	28.41		17.66	

The percentage overheads of message complexity of MANETconf are 328.10 % and 241.72 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 18 shows

the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

***Table 18 Percentage Overhead** (**MANETconf**, **p=0.5**)*

Percentage Overhead (MANETconf vs. Weak DAD with Proactive routing protocols, p=0.5)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	179.02	196.30	191.08	187.64	201.41	199.96	190.02	209.25	198.70
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	208.29	208.69	215.31	215.95	214.04	224.15	228.92	228.92	231.72
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	227.03	243.75	250.97	246.27	255.45	250.58	267.10	261.65	261.07
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	257.09	266.14	276.78	283.48	272.48	266.19	265.23	276.32	282.56
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	281.82	277.13	300.61	293.28	318.10	318.10		241.72	

The percentage overheads of message complexity of Strong DAD are 408.38 % and 363.36 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 19 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

Table 19 Percentage Overhead (Strong DAD, p=0.5)

Percentage Overhead (Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.5)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	356.13	356.13	337.32	343.25	359.76	352.92	339.01	343.48	348.86
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	354.43	351.19	359.41	353.74	360.16	355.00	363.00	351.11	355.28
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	355.03	370.25	362.46	366.01	362.79	368.22	367.66	364.35	374.74
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	368.60	370.33	377.56	375.01	357.72	363.95	371.29	366.82	370.65
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	370.97	388.85	380.96	395.08	408.38	408.38		363.36	

Therefore, similar to the case of single node joining case, it can be said that at the conflict probability of 0.5, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in MANET group merging case. However, since the proactive routing protocols depend on a periodic message to update the network topology and on demand routing protocols does not have a periodic message, it is not fair for the two routing protocols to be compared using the number of messages. The result above can be used only in the case that Weak DAD protocol uses MANET routing protocols for nodes to configure its IP address and solve the duplicate IP address detection in the MANET group merging case.

3.9.2. Percentage overhead comparison (Conflict probability of 0.7)

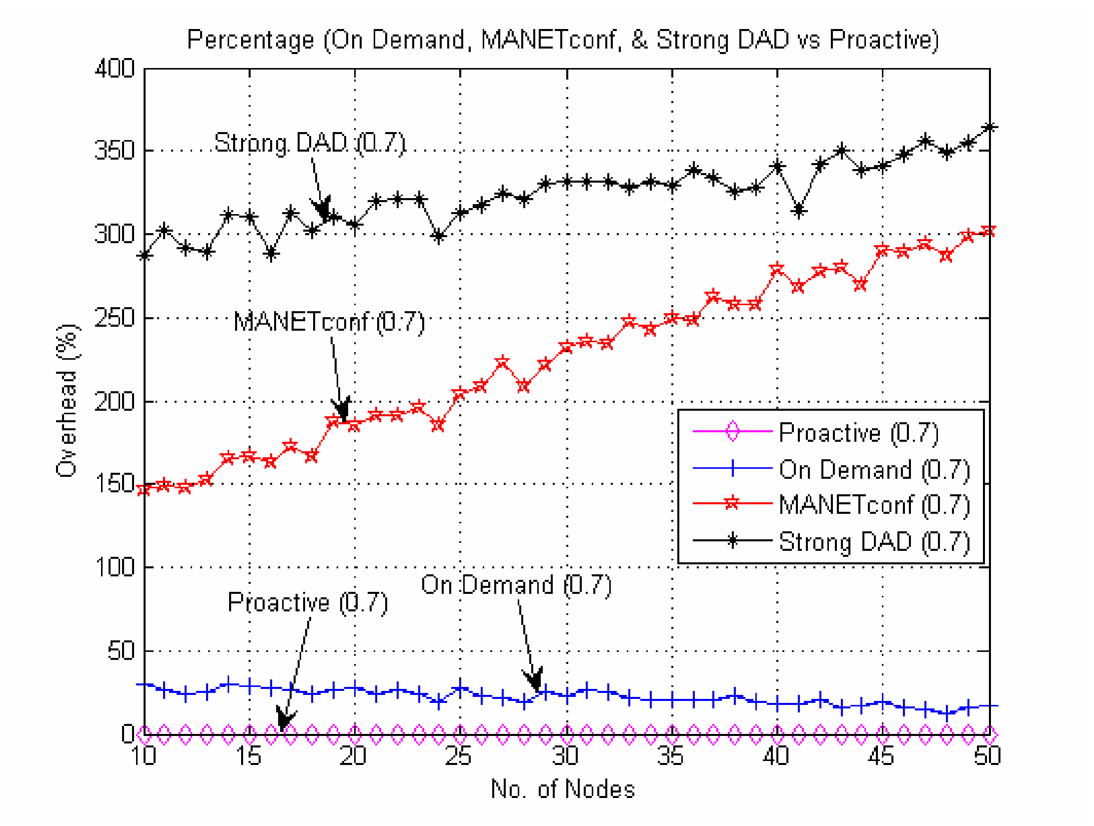

Figure 41 The percentage overhead comparison (*MANET Group Merging Case, p*=**0.7**, ***R***=**100**)

Fig. 41 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in MMANET group merging case with conflict probability of 0.7. It is shown that as the number of nodes increases, the message complexity of Weak DAD with on demand routing protocols converge to the message complexity of Weak DAD with proactive routing protocols.

The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 30.22 % and 22.15 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead

respectively. Table 20 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

***Table 20 Percentage Overhead** (**Weak DAD with on demand routing protocols, p=0.7**)*

***Percentage Overhead** (**Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p=0.7**)*									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	30.02	25.86	23.99	24.70	30.22	28.13	27.20	26.81	23.88
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	26.44	27.74	24.40	26.08	23.80	19.05	27.57	22.47	21.14
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	19.45	25.27	22.50	26.13	24.78	21.86	20.70	20.56	20.85
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	20.34	22.25	18.85	18.19	18.14	20.13	16.22	17.32	19.43
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	15.64	14.75	12.53	16.03	16.92	30.22		22.15	

The percentage overheads of message complexity of MANETconf are 302.70 % and 225.34 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 21 shows the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

Table 21 Percentage Overhead (MANETconf, p=0.7)

Percentage Overhead (MANETconf vs. Weak DAD with Proactive routing protocols, p=0.7)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	146.46	148.97	148.44	152.17	165.56	166.45	163.46	172.71	166.40
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	187.28	184.98	190.83	190.73	195.81	185.25	204.57	209.22	223.15
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	209.23	221.99	231.70	235.37	234.26	247.20	243.21	249.88	248.36
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	262.80	258.20	257.78	278.77	268.09	277.80	280.27	269.14	290.90
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	289.19	293.81	287.52	298.35	302.70	302.70		225.34	

The percentage overheads of message complexity of Strong DAD are 363.88 % and 324.03 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 22 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

Table 22 Percentage Overhead (Strong DAD, p=0.7)

Percentage Overhead (Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.7)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	287.28	302.01	291.56	289.20	311.65	310.48	288.42	313.21	302.23
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	311.05	305.28	319.88	321.24	321.20	298.67	313.06	317.78	324.51
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	320.47	330.44	331.43	331.60	331.89	327.66	331.77	329.65	338.06
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	333.58	325.98	327.95	340.36	313.84	341.73	350.11	338.73	340.60
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	347.80	355.83	348.71	354.60	363.88	363.88		324.03	

Therefore, it can be said that at the conflict probability of 0.7, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in the MANET group merging case.

3.9.3. Percentage overhead comparison (Conflict probability of 0.9)

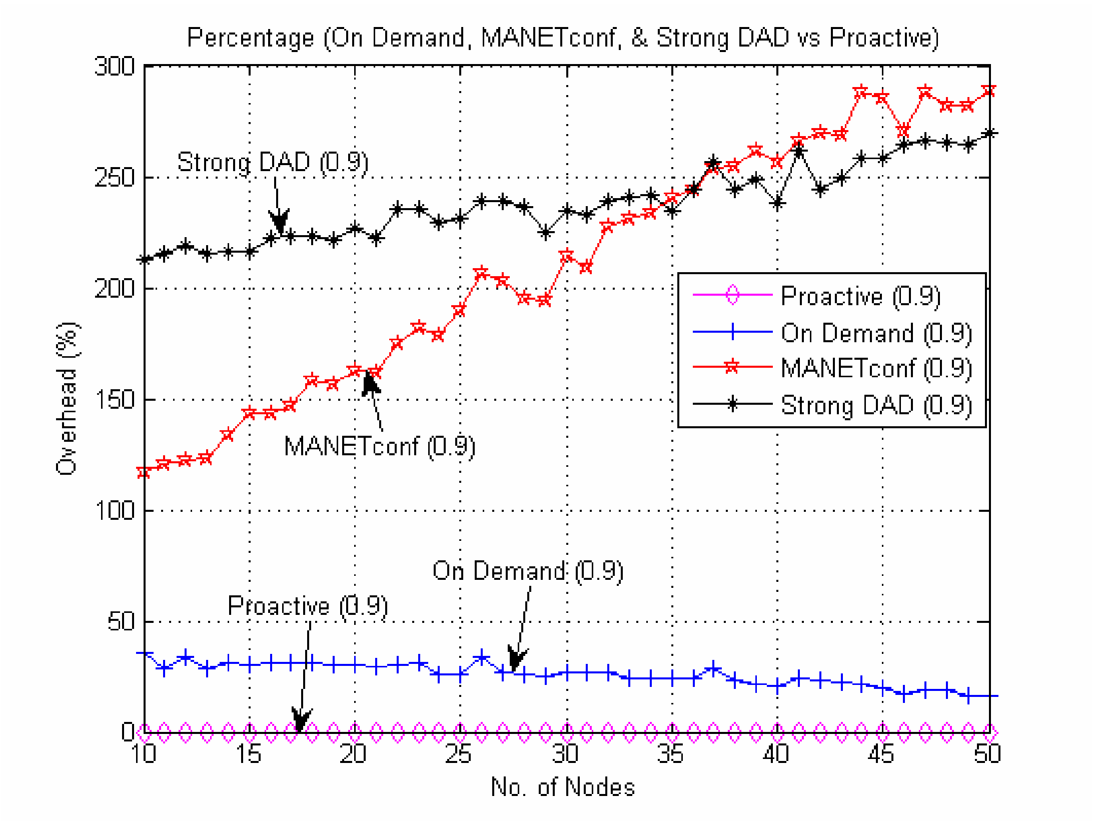

Figure 42 The percentage overhead comparison (***Single Node Joining Case, p*=0.9, *R*=100**)

Fig. 38 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in MANET group merging case with conflict probability of 0.9.

It is shown that as the number of nodes is increased, the message complexity of Weak DAD with on demand routing protocols tends to converge to the message complexity of Weak DAD with proactive routing protocols. In addition, the message complexity of MANETconf goes beyond the message complexity of Strong DAD when the number of nodes is around 35 nodes.

The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 35.96 % and 26.02 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 23 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

Table 23 Percentage Overhead (Weak DAD with on demand routing protocol, p=0.9)

Percentage Overhead (Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p=0.9)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	35.96	28.84	33.60	28.75	31.11	30.30	31.49	31.07	31.33
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	30.43	29.96	29.09	29.94	31.42	25.59	26.09	34.15	27.18
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	25.44	25.25	27.09	26.88	27.14	24.01	24.43	24.22	24.31
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	28.47	23.45	21.18	20.42	23.86	23.42	22.29	21.73	20.06
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	17.33	19.03	18.81	15.84	15.95	35.96		26.02	

The percentage overheads of message complexity of MANETconf are 289.03 % and 210.55 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 24 shows

the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

***Table 24 Percentage Overhead* (*MANETconf*, *p=0.9*)**

***Percentage Overhead* (*MANETconf vs. Weak DAD with Proactive routing protocols, p=0.9*)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	116.77	120.91	122.78	122.98	133.64	143.16	143.63	147.02	158.11
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	156.51	162.68	161.45	175.36	181.88	178.77	189.93	206.76	203.21
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	195.09	193.99	214.64	209.03	227.74	230.76	233.60	240.48	244.21
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	254.34	254.63	262.18	256.64	266.43	269.43	268.44	288.35	285.52
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	270.27	287.88	282.45	281.72	289.03	289.03		210.55	

The percentage overheads of message complexity of Strong DAD are 270.16 % and 238.16 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 25 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

***Table 25 Percentage Overhead** (**Strong DAD**, **p=0.9**)*

***Percentage Overhead** (**Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.9**)*									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	212.43	214.98	218.72	215.26	216.22	216.10	222.64	223.12	223.04
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	221.42	227.02	222.75	235.25	235.84	229.41	231.58	239.07	238.69
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	236.21	225.32	234.46	233.04	239.28	240.58	241.65	234.65	244.64
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	256.36	244.29	248.46	238.16	261.51	244.43	249.60	258.50	258.64
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	264.83	266.63	265.11	264.69	270.16	270.16		238.16	

Therefore, it can be said that at the conflict probability of 0.9, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in the MANET group merging case between 10 and 35 nodes. However, the message complexity of MANETconf is the largest when the number of nodes is above 35.

3.9.4. Percentage overhead comparison (Conflict probability of 1)

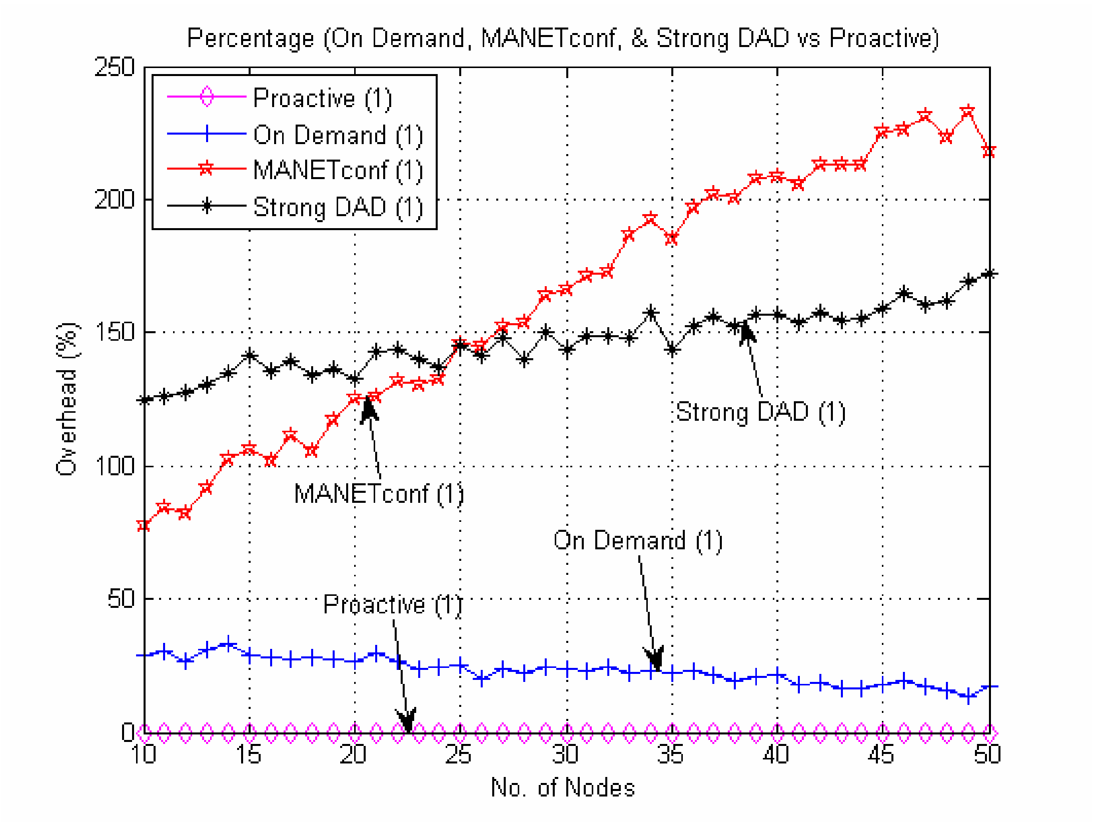

Figure 43 The percentage overhead comparison (***Single Node Joining Case, p*****=1,** ***R*****=100**)

Fig. 39 shows the percentage overhead comparison among the Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, MANWTconf and Strong DAD in MANET group merging case with conflict probability of one.

It is shown that as the number of nodes is increased, the message complexity of Weak DAD with on demand routing protocols is inclined to converge to the message complexity of Weak DAD with proactive routing protocols. In addition, the message complexity of MANETconf goes beyond the message complexity of Strong DAD when the number of nodes is around 25.

The percentage overheads of message complexity of Weak DAD with on demand routing protocols are 33.52 % and 23.22 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 26 shows the details of the percentage overhead between Weak DAD with on demand routing protocols and Weak DAD with Proactive routing protocols.

Table 26 Percentage Overhead (Weak DAD with on demand routing protocol, p=1)

Percentage Overhead (Weak DAD with on demand routing protocols vs. Weak DAD with Proactive routing protocols, p=1)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	28.73	30.62	26.76	30.81	33.52	28.59	27.88	27.32	28.47
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	27.48	26.76	29.71	26.70	23.65	24.30	25.37	19.95	23.97
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	22.42	24.80	23.65	23.19	24.32	22.37	23.37	22.03	22.96
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	21.22	19.55	20.80	21.21	17.85	18.50	16.66	16.70	17.89
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	19.26	16.99	15.36	13.55	16.99	33.52		23.22	

The percentage overheads of message complexity of MANETconf are 232.48 % and 163.06 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 27 shows the details of the percentage overhead between MANETconf and Weak DAD with Proactive routing protocols.

Table 27 Percentage Overhead (MANETconf, p=1)

Percentage Overhead (MANETconf vs. Weak DAD with Proactive routing protocols, p=1)									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	77.59	84.59	82.43	91.80	102.47	106.69	101.77	111.75	105.85
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	117.65	125.52	126.06	131.59	130.24	132.33	145.81	145.45	152.53
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	154.18	164.04	166.66	171.65	173.17	186.91	192.66	185.58	196.95
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	202.14	200.99	207.61	208.96	205.43	212.82	213.12	213.16	225.86
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	226.52	231.11	222.97	232.48	218.52	232.48		163.06	

The percentage overheads of message complexity of Strong DAD are 172.29 % and 146.98 % compared to the Weak DAD with proactive routing protocols at the maximum percentage overhead and the average percentage overhead respectively. Table 28 shows the details of the percentage overhead between Strong DAD and Weak DAD with Proactive routing protocols.

***Table 28 Percentage Overhead* (*Strong DAD*, *p*=1)**

***Percentage Overhead* (*Strong DAD vs. Weak DAD with Proactive routing protocols, p=0.9*)**									
No. of Nodes	10	11	12	13	14	15	16	17	18
Overhead (%)	124.96	126.03	127.45	130.59	135.15	141.51	135.32	139.52	134.27
No. of Nodes	19	20	21	22	23	24	25	26	27
Overhead (%)	136.32	132.71	142.96	143.73	140.13	136.97	145.34	141.09	147.81
No. of Nodes	28	29	30	31	32	33	34	35	36
Overhead (%)	140.09	150.37	143.80	148.55	148.81	148.38	157.39	143.89	152.11
No. of Nodes	37	38	39	40	41	42	43	44	45
Overhead (%)	156.32	152.19	156.80	156.64	154.19	157.85	154.89	155.20	158.78
No. of Nodes	46	47	48	49	50	*Max*		*Average*	
Overhead (%)	165.06	160.17	161.60	169.01	172.29	172.29		146.98	

Therefore, it can be said that at the conflict probability of one, Weak DAD with proactive routing protocols has the smallest message complexity and Strong DAD has the largest message complexity in the MANET group merging case between 10 and 25 nodes. However, the message complexity of MANETconf is the largest when the number of nodes is above 25.

3.9.5. Observations in the MANET group merging case

Based on the observation from section 3.9.1 to section 3.9.4, it is shown that with the increase of the conflict probability from 0.5 to 1, the maximum overhead percentage of the message complexity of Weak DAD with on demand routing protocols is changed from 28.41 %, to 30.22 %, to 35.96 %, and to 33.52 %, the maximum overhead percentage of the message complexity of MANETconf is decreased from 318.10 %, to 302.70 %, to 289.03 %, and to 232.48 % and the maximum overhead percentage of the message complexity of Strong DAD is decreased rapidly from 408.38 %, to 363.88 %, to 270.16 %, and to 172.29 % as shown in Table 29.

***Table 29 Maximum overhead percentage* [%]**

p	**Weak DAD with on demand routing protocols**	**MANETconf**	**Strong DAD**
0.5	28.41	318.10	408.38
0.7	30.22	302.70	363.88
0.9	35.96	289.03	270.16
1	33.52	232.48	172.29

As shown in Table 30, with the increase of the conflict probability from 0.5 to 1, the average overhead percentage of the message complexity of Weak DAD with on demand routing protocols is changed from 17.66 %, to 22.15 %, to 26.02 %, and to 23.22 %, the average overhead percentage of the message complexity of MANETconf is decreased from 241.72 %, to 225.34 %, to 210.55 %, and to 163.06 %, and the average overhead percentage of the message complexity of Strong DAD is decreased rapidly from 363.36 %, to 324.03 %, to 238.16 %, and to 146.98 %.

Table 30 ***Average overhead percentage*** **[%]**

p	**Weak DAD with on demand routing protocols**	**MANETconf**	**Strong DAD**
0.5	17.66	241.72	363.36
0.7	22.15	225.34	324.03
0.9	26.02	210.55	238.16
1	23.22	163.06	146.98

Since Strong DAD uses *DAD retry count limit n*=3, it can be expected that the percentage overhead of Strong DAD will be three times (300 %) larger than that of Weak DAD with proactive routing protocols. Therefore, the average overhead percentages with conflict probability of 0.5, 0.7, and 0.9 tend to follow the expected behavior (result); however, the average overhead percentage with conflict probability of 1 does not follow the expected behavior (result).

The maximum or average percentage overhead of the message complexity in the case of conflict probability of 1 has a little difference in value between MANETconf (232.48 %, 163.06 %, from the cases of maximum and average percentage overhead) and Strong DAD (172.29 %, 146.98 %, from the cases of maximum and average percentage overhead), which means that when the conflict probability is close to one, there is not much difference in the message complexity between MANETconf and Strong DAD.

Based on the percentage overhead of the message complexity of MANETconf, unicasting by all nodes causes approximately 242 % (319 %, from the results of maximum overhead percentage) more overhead than unicasting by a single node in the MANET group merging case.

Based on the percentage overhead of the message complexity of Weak DAD with on demand routing protocols, another unicasting mechanism causes approximately 27 % (36 %, from the results of maximum overhead percentage) more overhead compared to one of a single unicasting mechanism in the MANET group merging case.

Especially, based on the maximum or average percentage overhead of the message complexity as shown in Tables 29 and 30, it can be seen that as the conflict probability is increased (for example, when the conflict probability equals one) and the number of nodes is also increased, MANETconf has the largest message complexity in the MANET group merging case. Therefore, it could be said that even though a broadcast is a complex operation in comparison to a unicast, since a broadcast causes more traffic messages and procedural operations than a unicast operation in MANETs, in the MANET group merging case with MANETconf protocol, a unicast by all nodes becomes a complex operation in comparison to a broadcast when the conflict probability is increased and the number of nodes is also increased at the same time.

3.9.6. Comparison of percentage overhead difference between Single Node Joining Case and MANET Group Merging Case

Table 31 shows the maximum percentage overhead difference, where SNJC indicates Single Node Joining Case, MGMC indicates MANET Group Merging Case and |D| indicates the absolute value of the maximum percentage overhead difference between SNJC and MGMC at each protocol.

Table 31 ***Maximum percentage overhead difference*** **[%]**

	Weak DAD with on demand routing protocols			**MANETconf**			**Strong DAD**		
p	**SNJC**	**MGMC**	**|D|**	**SNJC**	**MGMC**	**|D|**	**SNJC**	**MGMC**	**|D|**
0.5	29.30	28.41	0.89	174.04	318.10	144.06	407.18	408.38	1.2
0.7	36.15	30.22	5.93	174.56	302.70	128.14	374.24	363.88	5.36
0.9	38.41	35.96	2.45	171.28	289.03	117.75	292.58	270.16	22.42
1	33.71	33.52	0.19	136.64	232.48	95.84	180.99	172.29	8.7

Based on |D|, MANETconf shows big differences between the SNJC percentage overhead and MGMC percentage overhead. The difference varies from 95.84 % to 144.06 %. However, Weak DAD with on demand routing protocols and Strong DAD do not show much variance between the SNJC percentage overhead and MGMC percentage overhead. The difference varies from 0.19 % to 22.42 %. Therefore, it could be said that when a MANET performs MGMC, MANETconf has the largest of increase rate in analyzing the message complexity among the Weak DAD with on demand routing protocols, Strong DAD and MANETconf. Table 32 shows the average percentage overhead difference for each protocol.

Table 32 ***Average percentage overhead difference*** **[%]**

	Weak DAD with on demand routing protocols			**MANETconf**			**Strong DAD**		
p	**SNJC**	**MGMC**	**|D|**	**SNJC**	**MGMC**	**|D|**	**SNJC**	**MGMC**	**|D|**
0.5	20.22	17.66	2.56	134.43	241.72	107.29	360.08	363.36	3.28
0.7	24.86	22.15	2.71	130.92	225.34	94.42	315.46	324.03	8.57
0.9	28.70	26.02	2.68	127.74	210.55	82.81	229.43	238.16	8.73
1	26.31	23.22	3.09	97.89	163.06	65.17	142.30	146.98	4.68

4. Conclusion

The main focus of this dissertation is to propose a novel method to perform a quantitative analysis of message complexity and to compare the message complexity among the mobile ad hoc network (MANET) address autoconfiguration protocols. To conduct a quantitative analysis of message complexity, the analysis of the worst case scenario is conducted in this dissertation. In the analysis procedures, each node or a set of nodes in a MANET is affiliated with steps in a flowchart of a MANET node operation of generating messages, which ultimately makes it possible to derive the upper bound of the message complexity.

Several results are derived in order to compare the performance of the address autoconfiguration protocols. First, the maximum number of messages is calculated since when the number of messages grows, the probability of message loss or corruption increases, and the time delay for a successful step also increases. Based on the analysis of the maximum number of messages, it was found that for a MANET group of N nodes operating under address autoconfiguration protocols, the message complexity of the single node joining case in Strong DAD, Weak DAD with proactive routing protocols, Weak DAD with on-demand routing protocols, and MANETconf is respectively $n(mO(N)+O(t))$, $n(O(N)+O(t))$, $n(O(N)+2O(t))$, and $nO((t+1)N)$, where m is the *DAD retry count limit*, n is the *retry count limit,* and t is the maximum length path.

In addition, the message complexity of the MANET group merging case in Strong DAD, Weak DAD with proactive routing protocols, Weak DAD with on-demand routing protocols, and MANETconf has been derived as $nN_1(mO(N_2)+O(t))$, $nN_1(O(N_2)+O(t))$, $nN_1(O(N_2)+2O(t))$, and $min(N_1,N_2)\{nO((t+1)(N_1+N_2))+O(N_1+N_2)+O(1)\}+O(2)+N_1O(N_2) + N_2O(N_1)$ respectively.

Based on a comparison of the message complexity between the single node joining case for a conflict probability value of one or less than one and the MANET group merging case for a conflict probability value of less than 0.9, it can be concluded that with the increase of the number of nodes, the message complexity of Strong DAD has the highest message complexity, while MANETconf and Weak DAD with the on demand routing protocol have the second highest message complexity. Weak DAD with the proactive routing protocol shows the lowest message complexity. In MANET group merging case, with the increase of the number of nodes, when the conflict probability equals 0.9 and above, the message complexity of MANETconf has the highest message complexity, while Strong DAD and Weak DAD with the on demand routing protocol have the second highest message complexity. Weak DAD with the proactive routing protocol shows the lowest message complexity.

Second, the number of unicast and broadcast messages is calculated and the analysis of this dissertation indicates that for a MANET group of N nodes operating the address autoconfiguration protocols, the upper bound of the message complexity broadcasting is $O(N)$, the upper bound message complexity of all nodes unicasting in response to a broadcast message is $O(tN)$, and the upper bound message complexity of some nodes unicasting in order to check the status of other nodes is $O(tN)$. It is known that a step that needs for all nodes to unicast a message is a complex operation since the results of this dissertation in the case of MANET group merging case when the conflict probability equals 0.9 or 1 show that the steps which requires for all nodes to unicast a message in response to a broadcast message and the steps where some nodes unicast in order to check the status of other nodes caused a lot of traffic messages and procedural operations in MANETs. This result is explained in section 3.9.5.

Third, since the number of recursive procedures is one of the main factors determining the time delay in a protocol, the effect of the number of recursive execution on the message complexity is studied. Since the number of recursive procedures has been suggested as a fixed value in the protocols, it was found that the number of recursive procedures is not related with an increase in the polynomial order in the *O*-notation, but it is strongly related with an increase in the total number of messages.

Based on a comparison of the worst case scenario in Strong DAD, MANETconf, Weak DAD with proactive routing protocols, and Weak DAD with on-demand routing protocols, when the network is composed of a large density of nodes, the maximum complexity of Strong DAD tends to increase linearly with an increase of MANET nodes compared to Weak DAD with proactive routing protocols. This is mainly because Strong DAD chooses broadcasting schemes in steps (or procedures) that cause the most traffic messages than other protocols.

In addition, the maximum complexity of MANETconf tends to increase linearly with an increase of MANET nodes compared to Weak DAD with proactive routing protocols. This is mainly because MANETconf uses a mutual exclusion algorithm to acquire a new IP address for a *Requestor*, and all nodes unicast *IR* messages to inform the *Initiator* of the duplication status of a requested IP address. Unicasting a message by all nodes results in the operation complexity upper bound case of a MANET.

Therefore, in the view point of the message complexity, when a MANET area is composed of a high density of communicating nodes, Weak DAD with MANET routing protocols becomes a more suitable protocol than MANETconf and Strong DAD since Weak DAD with MANET routing protocols can provide messages that can be used for both routing and address autoconfiguration and has much less message complexity in address autoconfiguration compared to MANETconf and Strong DAD. However, the reliability issue in duplicate address detection is a future research topic.

In the view point of the message complexity, when a MANET area is composed of a high conflict probability, Weak DAD with MANET routing protocols becomes a more suitable protocol than MANETconf and Strong DAD since Weak DAD with MANET routing protocols provides both routing and address autoconfiguration and much less message complexity compared to MANETconf and Strong DAD.

Tables 33 and 34 summarize the message complexity of a single node joining case and MANET group merging case in Strong DAD, Weak DAD with proactive routing protocols, Weak DAD with on demand routing protocols, and MANETconf respectively.

Based on the observation in section 3.9.6, when a MANET performs MANET group merging case, MANETconf has the largest rate of increase in analyzing the message complexity among Weak DAD with on demand routing protocols, Strong DAD and MANETconf.

Table 33 Comparison of the message complexity in single node joining case

Address Autoconfiguration Protocols	*Message Complexity*
Strong DAD	$n(mO(N)+O(t))$
Weak DAD with proactive routing protocols	$n(O(N)+O(t))$
Weak DAD with on demand routing protocols	$n(O(N)+2O(t))$
MANETconf	$nO((t+1)N)$

Table 34 Comparison of the message complexity in MANET group merging case

Address Autoconfiguration Protocols	*Message Complexity*
Strong DAD	$nN_1(mO(N_2)+O(t))$
Weak DAD with proactive routing protocols	$nN_1(O(N_2)+O(t))$
Weak DAD with on demand routing protocols	$nN_1(O(N_2)+2O(t))$
MANETconf	$min(N_1,N_2)\{nO((t+1)(N_1+N_2))+O(N_1+N_2)+O(1)\}+O(2)+N_1O(N_2) + N_2O(N_1)$.

In this chapter, the address autoconfiguration protocols (i.e., Strong DAD, Weak DAD, and MANETconf) have been investigated. The main contributions of this dissertation are based upon the following accomplishments.

1. The original publications on the address autoconfiguration protocols had many incomplete parts making them impossible to use on practical MANETs. Therefore, the first objective of the executed research was to complete the

address autoconfiguration protocols by filling in all the missing gaps to make them operational. The missing procedures that were filled in have been developed based on the most logistic procedures being trustful to the original protocol publications.

A. By introducing the *retry count limit* (*n*) of a *session* in Strong DAD, the possibility of an infinite loop from the original Strong DAD has been removed. The original Strong DAD does not prepare the maximum number of retries of the *IP verification procedure*.

B. By adapting the mechanism of the replying AE message introduced in [25], the additional step defined in the original Weak DAD protocol has been more accurate when a node finds a duplicated IP address.

C. In MANETconf, the duplicated address node with the higher *Partition Identity* will become the *Requestor* asking its neighboring node to become its *Initiator*.

2. So far, except for MANETconf computer simulation, none of the address autoconfiguration protocols have been investigated in reference to their complexity and scalability in MANET based operations. Therefore, the conducted research provides a detailed derivation of single node joining and partitioned group merging message complexity and extends the results to scalability and complexity analysis. Based upon the results, the following comparison on complexity and scalability can be concluded.

   A. The advantages of Strong DAD are as follows.

i. Since Strong DAD has been configured with *m* number of *DAD retry count limit* in the *verification procedure* in order to verify the required duplicated IP address in a MANET, the results are reliable compared to Weak DAD or MANETconf, and the results after executing the *verification procedure* could be more reliable compared to the Weak DAD or MANETconf.

ii. Strong DAD has been configured with a simple messaging scheme such as *Address Request* and *Address Reply* compared to MANETconf. Therefore, with this simple messaging scheme, it is able to complete its required operations.

B. The advantages of Weak DAD are as follows.

i. Weak DAD does not need any new message for the address autoconfiguration since the routing messages have been modified to include the functionalities needed for address autoconfiguration.

ii. In heavily dense networks, since Weak DAD provides both routing and address autoconfiguration mechanisms at the same time, Weak DAD with proactive routing protocols have good scalability characteristic compared to Strong DAD or MANETconf.

iii. From the message complexity comparison, Weak DAD with proactive routing protocol has the lowest message complexity.

C. The advantages of MANETconf are as follows.

i. MANETconf has a definite mechanism of partition and merging by using the *Partition Identity* of MANET groups. Therefore, partition and merging are handled through stable systematic procedures in group based activities.

ii. MANETconf has moderate message complexity even though it is a stateful approach.

D. The disadvantages of Strong DAD are as follows.

i. In addition to the large overhead of acquiring IP addresses by strong DAD, MANET nodes generate routing overhead to update their network topology by using MANET routing protocols procedures.

ii. Since Strong DAD uses *m* number of *DAD retry count limit* in the *verification procedure* in order to verify the required duplicated IP address in a MANET, it has the highest message complexity in both a single node joining case when the conflict probability is one or less than one and a MANET group merging case when the conflict probability is less than 0.9. In addition it could have a problem of a significant time delay in acquiring an IP address (especially for large values of *m*) compared to Weak DAD or MANETconf.

iii. Practically in Strong DAD, there is no identified procedure for group based partitioning or merging. In addition, there is no partition group ID that the nodes record. If MANET groups were to merge, each node would have to proceed with a joining procedure as a single node individually. Therefore, the concept of partition group control and management is weak compared to that of MANETconf.

E. The disadvantages of Weak DAD are as follows.

i. Weak DAD has a problem in the size of each node's routing table since the routing table should include the unique key value (MAC address) of each node.

ii. Practically in Weak DAD, there is no identified procedure for group based partitioning and merging. In addition, similar to Strong DAD, there is no partition group ID that the nodes record. If MANET groups were to merge, each node would have to proceed with a joining procedure as a single node individually. Therefore, the concept of partition group control and management is weak compared to the MANETconf protocol.

F. The disadvantages of MANETconf are as follows.

i. In addition to the overhead of acquiring an IP address using MANETconf, MANET nodes generate routing overhead to update their network topology by using MANET routing protocol procedures.

ii. When the conflict probability and network size grow in MANETconf, since all nodes reply with *IR* messages in response to the *IQ* message, the message complexity abruptly increases in comparison to the message complexity of Strong DAD and Weak DAD leading to scalability problems.

iii. When two networks merge in MANETconf, the total number of messages rapidly increases since all nodes reply with *IR* messages in response to the *IQ* message.

3. The following section introduces the observation based on the results of message complexity of Strong DAD, Weak DAD, and MANETconf.

   A. Strong DAD has two *retry count limits* which are the *DAD retry count limit* (*m*) and the *retry count limit* (*n*). The *retry count limit* (*n*) is used to indicate the maximum number of retries of the *session procedure* in Strong DAD, and the *DAD retry count* (*m*) is used to indicate the maximum trial number in an *IP verification procedure* in Strong DAD. Weak DAD has only one *retry count limit* which is the *retry count limit* (*n*) since the *DAD retry count limit* (*m*) equals one. Based on the *Lemmas* 2 ($mO(N)+O(t)$) and 5 ($O(N)+O(t)$), to verify an IP address, Strong DAD needs to run *m* trials, however, Weak DAD with proactive routing protocols needs to run one trial. Therefore, $(m-1)O(N)$ number of *Address Query* messages can be reduced in Weak DAD compared to Strong DAD. Therefore, the message complexity of Weak DAD ($O(N)+O(t)$) is much less than the message complexity of Strong DAD ($mO(N)+O(t)$). However, Strong DAD may have more accurate duplicate address detection than the one of Weak DAD but with a higher cost of message complexity.

   B. Based on the *Lemmas* 3 ($n(mO(N)+O(t))$) and 6 ($n(O(N)+O(t))$), when the message complexity in a *session* is compared, the message complexity of

Strong DAD is larger than the message complexity of Weak DAD since each *session* in Strong DAD includes a maximum of *m IP address verification procedures*, however, each *session* in Weak DAD is composed of only one *IP address verification procedure*.

C. By introducing the *retry count limit* (n) of a *session* in Strong DAD, the possibility of an infinite loop from the original Strong DAD has been removed. The original Strong DAD does not prepare the maximum number of retries of the *IP reselection* and its *IP verification procedure*.

D. When comparing the message complexity between Weak DAD with proactive routing protocols and Weak DAD with on demand protocols based on *Lemmas* 5 ($O(N)+O(t)$) and 7 ($O(N)+2O(t)$), and since a node can send a *RP* message during the *IP verification procedure* period when it is the destination node of other nodes, Weak DAD with on demand protocols has $O(t)$ more messages compared to Weak DAD with proactive routing protocols. To compare equal complexity, the periodicity of the *LS* messages is not considered. However, since the proactive routing protocols depend on a periodic message to update the network topology and on demand routing protocols does not have a periodic message, it is not fair for the two routing protocols to be compared with the number of messages. The result above can be used only in the case that Weak DAD protocol uses MANET routing protocols for nodes to configure its IP address and solve the duplicate IP address detection.

E. Based on the mathematical derivation in *Lemma* 2 ($mO(N)+O(t)$), *Lemma* 5 ($O(N)+O(t)$), *Lemma* 11 $O((t+1)N)$, since MANETconf has the mechanism that all recipient nodes must unicast *IR* messages when they receive *RQ* messages, it is shown that the message complexity of unicasting *IR* messages is linearly increased based on the number of MANET nodes (N) in MANETconf compared to unicasting *AP* and *IR* messages in Strong DAD and Weak DAD. Therefore, the comparison of the message complexity between the unicasting by all nodes in MANETconf and the broadcasting by a limited time in Strong DAD could be considered. Based on the simulation results, in single node joining case for a conflict probability which equals one or less than one and in MANET group merging case for a conflict probability which is less than 0.9, even though MANETconf has the mechanism that all recipient nodes must unicast *IR* messages when they receive *RQ* messages, it is shown that the message complexity of MANETconf has the second largest message complexity among Weak DAD, Strong DAD and MANETconf. Therefore, when a network demands an exact IP assignment, MANETconf can be used since it generates a moderate message complexity compared to Strong DAD.

4. Based on the simulation results and analysis of the message complexity in Tables 14 and 15, when nominal n, m, t, N values and transmission range have been assigned in a single node joining case with $p \leq 1$, the message complexity can be compared as follows:

   A. Weak DAD with proactive routing protocol < Weak DAD with on demand routing protocol < MANETconf < Strong DAD.

5. Based on the simulation results and analysis of the message complexity in Tables 28 and 29, when nominal *n*, *m*, *t*, *N* values and transmission range have been assigned in a MANET group merging case with $p = 0.5$ and 0.7 (from the results of the maximum overhead percentage) and $p = 0.5$, 0.7 and 0.9 (from the results of the average overhead percentage), the message complexity can be compared as follows:

   A. Weak DAD with proactive routing protocol < Weak DAD with on demand routing protocol < MANETconf < Strong DAD.

6. However, when conflict probability equals 0.9 and 1 (from the results of the maximum overhead percentage) and the conflict probability equals one (from the results of the average overhead percentage), the message complexity in a MANET group merging case can be compared as follows:

   A. Weak DAD with proactive routing protocol < Weak DAD with on demand routing protocol < Strong DAD < MANETconf.

5. List of References

[1] T. Clausen, Ed., and P. Jacquet, Ed., “Optimized Link State Routing Protocol (OLSR),” *RFC 3626*, *The Internet Society*, Oct., 2003.

[2] C. Perkins and E. Royer, “Ad hoc On Demand Distance Vector (AODV) Routing,” *RFC 3561*, *The Internet Society*, July, 2003.

[3] M. Gerla, X. Hong, and G. Pei, “Fisheye State Routing Protocol (FSR) for Ad Hoc Networks,” *Internet Draft*, June, 2002, http://www.ietf.org/proceedings/02jul/I-D/draft-ietf-manet-fsr-03.txt.

[4] R. Ogier, M. Lewis, and F. Templin, “Topology Broadcast Based on Reverse Path Forwarding (TBRPF),” *Internet Draft*, April. 2003, http://www.potaroo.net/ietf/old-ids/draft-ietf-manet-tbrpf-08.txt.

[5] C. A. Santiváñez, R. Ramanathan, and I. Stavrakakis, “Making link-state routing scale for ad hoc networks,” *Proc. of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, Long Beach, CA, USA, 2001.

[6] M. Gerla, X. Hong, L. Ma, and G. Pei, “Landmark Routing Protocol (LANMAR) for Large Scale Ad Hoc Networks”, *Internet Draft*, Nov. 2002, http://www.ietf.org/proceedings/01dec/I-D/draft-ietf-manet-lanmar-02.txt.

[7] D. Johnson, D. A. Maltz, and Y.-C. Hu, “The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR),” *Internet Draft*, April, 2003, http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-09.txt.

[8] X. Hong, K. Xu, and M. Gerla, “Scalable Routing Protocol for Mobile Ad Hoc Networks,” *IEEE Network*, pp.11-21, Jul./Aug. 2002.

[9] Z. J. Haas, M R. Pearlman, and Prince Samar, “The Zone Routing Protocol (ZRP) for Ad Hoc Networks,” *Internet Draft*, July, 2002, http://www.ietf.org/proceedings/02nov/I-D/draft-ietf-manet-zone-zrp-04.txt.

[10] Y.-B. Ko and N. H.Vaidya, “Location-Aided Routing (LAR) in Mobile Ad Hoc Networks,” *Proc. ACM/IEEE Mobile computing and networking*, 1998. pp.66-75.

[11] B. Karp and H.T.Kung, “GPSR: Greedy Perimeter Stateless Routing for Wireless Networks,” *Proc. ACM/IEEE Mobile computing and networking*, 2000. pp.243-253.

[12] G. Pei, M. Gerla, X. Hong, and C. C, Chiang, “A Wireless Hierarchical Routing Protocol with Group Mobility,” *Proc. IEEE WCNC '99*, New Orleans, LA, Sept. 1999.

[13] C.-C. Chiang and M. Gerla, “Routing in Clustered Multihop Mobile Wireless Networks,” *Proc. Information Networking* (*ICOIN11*), 3B-1.1-3B-1.9, Jan. 1997.

[14] A. Iwata and *et al*., “Scalable Routing Strategies for Ad-hoc Wireless Networks,” *IEEE JSAC*, 1369-79, Aug. 1999.

[15] B. Bellur and R. G. Ogier, “A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks,” *Proc. IEEE INFOCOM '99*, Mar. 1999.

[16] D. B. Johnson and D. A. Maltz, *Dynamic Source Routing in Ad Hoc Wireless Networks*, Kluwer, 1996.

[17] C.-C. Chiang, “Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel,” *Proc. IEEE SICON'97*, pp. 197-211, Apr. 1997.

[18] G. Pei and M. Gerla, “Mobility Management in Hierarchical Multi-hop Mobile Wireless Networks,” *Proc. IEEE ICCCN'99*, Oct. 1999.

[19] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla and T.-W. Chen, “Scalable Routing Strategies for Ad Hoc Wireless Networks,” *IEEE Journal on Selected Areas in Communications*, Vol. 17, No. 8, pp. 1369-1379, Aug. 1999.

[20] D. P. Agrawal and Q.-A. Zeng, *Introduction to Wireless and Mobile Systems*, Brooks/Cole-Thomson Learning, CA, 2003.

[21] C. E. Perkins, J. T. Malinen, R. Wakikawa, E. M. Royer, and Y. Sun, “IP address autoconfiguration for ad hoc networks,” *IETF draft*, 2001.

[22] K. Weniger, "Passive duplicate address detection in mobile ad hoc networks," *Proc. IEEE WCNC 2003*, Mar. 2003, New Orleans, USA.
[23] S. Thomson, and T. Narten," IPv6 Stateless Address Autoconfiguration," *RFC 2462*, Dec. 1998.
[24] K. Weniger, and M. Zitterbart, "Address autoconfiguration in mobile ad hoc networks: Current Approaches and Future Directions," *IEEE Network*, pp. 6-11, Jul./Aug. 2004.
[25] J-. P. Jeong, J. Park, H. Kim, and D. Kim, "Ad hoc IP address autoconfiguration," *IETF draft*, July 2004.
[26] N. H. Vaidya, "Weak duplicate address detection in mobile ad hoc networks," *Proc. ACM MobiHoc 2002*, pp. 206-216, June 2002, Lausanne, Switzerland.
[27] S. Nesargi and R. Prakash, "MANETconf: Configuration of hosts in a mobile ad hoc network," *Proc. IEEE Infocom 2002*, June 2002, New York, USA.
[28] S. Cheshire and B. Aboba, "Dynamic Configuration of IPv4 Link-Local Address," *Internet Draft*, June 2001.
[29] M. Moshin and R. Prakash, "IP address assignment in a mobile ad hoc network," *Proc. IEEE Milcom 2002*, pp. 856-861, Oct. 2002.
[30] H. Zhou, L. M. Ni, and M. W. Mutka, "Prophet Address Allocation for Large Scale MANETs," *Ad Hoc Networks Journal*, vol. 1, issue 4, pp. 423-434, Nov. 2003.
[31] G. Cao and M. Singhai, "A delay-optimal quorum-based mutual execution algorithm for distributed systems," *IEEE Trans. Parell and Distributed Systems*, vol. 12, no.12, pp. 1256-1268, Dec. 2001.
[32] C-. C. Shen, and C. Srisathapornphat, and R. L. Z. Huang, and C. Jaikaeo, and E. L. Lloyd, "CLTC: A cluseter-based topology control framework for ad hoc networks," *IEEE Trans. Mobile Computing*, vol. 3, no.1, pp. 18-32, Jan.-Mar. 2004.
[33] S. H. Kwok and A. G. Constantinides, "A fast recursive shortest spanning tree for image segmentation and edge detection," *IEEE Trans. Image Processing*, vol. 6, no.2, pp. 328-332, Feb. 1997.
[34] A. Boukerche, S. Hong, and T. Jacob, "An efficient synchronization scheme of multimedia streams in wireless and mobile systems," *IEEE Trans. Parell and Distributed Systems*, vol. 13, no.9, pp. 911-923, Sep. 2002.
[35] J. Gross and J. Yellen, *Graph Theory and Its Applications*, CRC Press, 1998.
[36] M. Sheng, J. Li, and Y. Shi, "Relative Degree Adaptive Flooding Broadcast Algorithm for Ad Hoc Networks," IEEE Trans. on Broadcasting, vol. 51, no. 2, pp. 216-222, June 2005.

VITA

Sang-Chul Kim

Candidate for the Degree of

Doctorate of Philosophy

Dissertation: MESSAGE COMPLEXITY ANALYSIS OF MOBILE AD HOC NETWORK (MANET) ADDRESS AUTOCONFIGURATION PROTOCOLS

Major Filed: Electrical Engineering

Biographical:

Personal Data: Born in Daegu, S. Korea, On January 25, 1970, the son of Jae-In Kim and Sun-Ju Park; married JiYoung Kim in 1995; SungHyun, daughter, was born in 1995.

Education: Graduated from Keysung Senior High School, Daegu, S. Korea in Feburary 1988; Received Bachelor of Engineering degree in Electronics Engineering from Kyungpook National University at Daegu, S. Korea in February 1994; Received Master of Science degree in Computer Science from Changwon National University, at Changwon, S. Korea in August 1998; Completed the requirements for the Doctorate of Philosophy degree with a major in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma in December 2005.

Experience: Research Assistant at Oklahoma State University from August 2000 to present; Graduate and Undergraduate Teaching Assistant at Oklahoma State University from August 2002 to May 2003; System Engineer at Samsung SDS Co. Ltd., at Seoul, S. Korea from January 2000 to July 2000; System Engineer at Samsung Aerospace Co. Ltd., at Changwon, S. Korea from January 1994 to December 1999; Instructor, Department of Computer Science, Changwon National University, at Changwon, S. Korea from August 1998 to December 1999.

Professional Membership: Institute of Electrical and Electronics Engineers, Phi Beta Delta

Name: Sang-Chul Kim Date of Degree: December, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: MESSAGE COMPLEXITY ANALYSIS OF MOBILE AD HOC NETWORK (MANET) ADDRESS AUTOCONFIGURATION PROTOCOLS

Pages in Study: 202 Candidate for the Degree of Doctor of Philosophy

Major Field: Electrical Engineering

Scope and Method of Study: Wireless communication systems continue to show rapid growth as a result of significant advancements in digital modulation technologies, network protocol development, and microprocessor technologies. Mobile ad hoc networks (MANETs) are self-organizing wireless networks where the mobile nodes have routing capabilities to be able to forward packets to communicate with one another over multi-hop wireless links without any fixed communication infrastructure. This dissertation presents a technical review of several MANET routing protocols based on their messages. In addition, a novel method to perform a quantitative analysis of the message complexity among the MANET address autoconfiguration protocols has been proposed. The upper bound of messages has been derived and justified through computer simulation. In reference to different conflict probabilities, a comparison among the IP address autoconfiguration protocols for MANETs has been conducted.

Findings and Conclusions: The first objective of the executed research is to complete the address autoconfiguration protocols by filling in all the missing gaps to make them operational. The missing procedures that were filled in have been developed based on the most logical procedures being faithful to the original protocol publications. In this dissertation, the upper bound of the message complexity of the IP address autoconfiguration protocols for MANETs is derived. To obtain the upper bound of the message complexity of the protocols, the *O*-notation of a MANET group of *N* nodes has been investigated. To asymptotically calculate the total number of messages generated by a protocol's step or procedure, an investigation on the nodes broadcasting, unicasting, relaying, and receiving messages is computed and used in obtaining the upper bound of the message complexity for each protocol.

ADVISER’S APPROVAL: Dr. Jong-Moon Chung