P-3597

ENHANCING ON-DEMAND MULTICAST ROUTING PROTOCOL IN MOBILE ADHOC NETWORKS

PROJECT REPORT

Submitted by

M.BHARATHIDASAN	**0710108007**
A.GOWTHAM	**0710108014**

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University of Technology, Coimbatore)

COIMBATORE – 641 049

APRIL 2011

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE-641 049

BONAFIDE CERTIFICATE

Certified that this project report entitled "**ENHANCING ON-DEMAND MULTICAST ROUTING PROTOCOL IN MOBILE ADHOC NETWORKS**" is the bonafide work of M.Bharathidasan and A.Gowtham who carried out the research under my supervision. Certified also, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

[illegible]

SIGNATURE

Mrs.R.Kalaiselvi, M.E.,

SUPERVISOR

Assistant Professor

Department of Computer

Science and Engineering

Kumaraguru College of Technology

Coimbatore-641049

[illegible]

SIGNATURE

Mrs.P.Devaki, M.E.,

HEAD OF THE DEPARTMENT

Department of Computer

Science and Engineering

Kumaraguru College of Technology

Coimbatore-641049

The candidate with University Register Nos. 0710108007 and 0710108014 were examined by us in the project viva-voce examination held on 20.04.2011

[illegible]

INTERNAL EXAMINER

[illegible]

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project entitled **"ENHANCING ON-DEMAND MULTICAST ROUTING PROTOCOL IN MOBILE ADHOC NETWORKS"** is a record of original work done by us and to the best of our knowledge, a similar work has not been submitted to Anna University or any Institutions, for fulfillment of the requirement of the course study.

The report is submitted in partial fulfillment of the requirement for the award of the Degree of Bachelor of Computer Science and Engineering of Anna University of Technology, Coimbatore.

Place: Coimbatore

Date: 19.04.2011

M. Bharathidasan

(M.BHARATHIDASAN)

[illegible]

(A.GOWTHAM)

ACKNOWLEDGEMENT

We are intend to express our heartiest thanks to our chairman **Arutselvar Dr.N.Mahalingam, B.sc., F.I.E** and the correspondent **M.Balasubramaniam, M.com., M.B.A.**, for given us this opportunity to embark on this project.

We extend our sincere thanks to our Principal, **Dr. S.Ramachandran,** Kumaraguru College of Technology, Coimbatore, for being a constant source of inspiration and providing us with the necessary facility to work on this project.

We would like to make a special acknowledgement and thanks to **Dr. S. Thangasamy,** Dean of Research and Development, for his support and encouragement throughout the project.

We are indent to express my heartiest thanks to **Mrs.P.Devaki, M.E,** Project coordinator, Head of the Department of Computer Science &Engineering, for her valuable guidance and useful suggestions during the course of this project.

We express deep gratitude and gratefulness to **our guide Mrs.R.Kalaiselvi M.E,** Assistant Professor Department of Computer Science & Engineering, for his supervision, enduring patience, active involvement and guidance.

We would like to convey our honest thanks to **all Faculty members** of the Department for their enthusiasm and wealth of experience from which we have greatly benefited.

We also thank our **friends and family** who helped us to complete this project fruitfully.

TABLE OF CONTENTS

ABSTRACT

With the advance of wireless communication technology, portable computers with radios are being increasingly deployed in common activities. Applications such as conferences, meetings, lectures, crowed control, search and rescue, disaster recovery, and automated battlefields typically do not have central administration. In Adhoc networks, each host must act as a router since routes are mostly multi hop. Nodes in such a networks move arbitrarily, thus network topology changes frequently. The protocol termed ODMRP is a mesh-based instead of tree based, multicast protocol that provides richer connectivity among multicast members. The major strengths of ODMRP are its simplicity. This project further improves its performance. New techniques to enhance the effectiveness and efficiency of ODMRP are provided in this project. Primary Goals of Enhanced ODMRP are to improve hop-by-hop transmission reliability, reduce delay and eliminate route acquisition latency.

LIST OF FIGURES

LIST OF ABBREVIATIONS

AODV	Ad-hoc On-Demand Distance Vector
CLR	Common Language Runtime
CLS	Common Language Specification
CTS	Common Type System
DSR	Dynamic Source Routing
FG_FLAG	Forwarding Group Flag
GPS	Global Positioning System
GSR	Global State Routing
IP	Internet Protocol
LET	Link Expiration Time
LRU	Least Recently Used
MANET	Mobile Ad-hoc Network
MIN_LET	Minimum Link Expiration Time
ODMRP	On Demand Multicast Routing Protocol
WAM	Wireless Adaptive Mobility
.NET	DotNET

CHAPTER 1

INTRODUCTION

1.1 WIRELESS AD-HOC NETWORK

Wireless ad-hoc networks are composed of autonomous nodes that are self managed without any infrastructure. In this way, ad-hoc networks have a dynamic topology such that node can easily join or leave the network at any time. They have many potential applications, especially, in military and rescue areas such as connecting soldiers on the battlefield or establishing a new network in a place of a network which collapsed after a disaster like an earthquake. Ad-hoc networks are suitable for areas where it is not possible to set up a fixed infrastructure, they provide the connectivity, and nodes use some routing protocols. Besides acting as a host, each node also acts as a router to discover a path and forward packets to the correct node in the network.

At a given point of time, depending on the nodes' positions and their transmitter and receiver coverage patterns, transmission power levels and co-channel interference levels, a wireless connectivity in the form of a random, multi-hop graph or "ad hoc" network exists between the nodes. An ad-hoc network is a collection of wireless mobile nodes dynamically forming a temporary network with out the presence of a wired support infrastructure. In this environment routing/multicasting protocols are faced with the challenge of producing multi-hop routes under host mobility and bandwidth constraints.

1.2 SALIENT FEATURES OF MANET

- *Dynamic topologies:* Nodes are free to move arbitrarily; thus, the network topology which is typically multi-hop may change randomly and rapidly at unpredictable times, and may consist of both bidirectional and unidirectional links.

- *Bandwidth-constrained, variable capacity links:* Wireless links will continue to have significantly lower capacity than their hardwired counterparts. In addition, the realized throughput of wireless communications - after accounting for the effects of multiple access, fading, noise, and interference conditions, etc., is often much less than a radio's maximum transmission rate.

- *Energy-constrained operation:* Some or all of the nodes in a MANET may rely on batteries or other exhaustible means for their energy. For these nodes, the most important system design criteria for optimization may be energy conservation.

- *Limited physical security:* Mobile wireless networks are generally more prone to physical security threats than are fixed cable nets. The increased possibility of eavesdropping, spoofing, and denial-of-service attacks should be carefully considered. Existing link security techniques are often applied within wireless networks to reduce security threats. As a benefit, the decentralized nature of network control in MANETs provides additional robustness against the single points of failure of more centralized approaches.

1.3 CLASSIFICATION OF ROUTING PROTOCOLS IN MANET

A major technical challenge in a MANET is the design of efficient routing protocols to cope with the rapid topology changes. Routing protocols in ad hoc networks vary depending on the type of the network.

Depending on the way routing update is carried out, the protocols can be put into two categories: Flooding based algorithms use flooding of routing updates in the network. Flooding levies an extra overhead. Most of the routing protocols are based on these methods (AODV, DSR, GSR etc.). Link reversal algorithms aim to save bandwidth. Any link breakage results in link reversal algorithm to be used at the site of the link failure to re-establish the path. It tries to localize effect. It gives many alternate paths to the destination. It not only save the bandwidth in updates, but also provides alternate paths in case of path failures.

Depending on the intended use, protocols may be categorized broadly as follows: One-to-One communication is called Unicasting. The packet from the source is intended for a specific recipient. It includes all such protocols which are used for end-to-end communication. Multicasting includes all such protocols, which can be used for one-to-many communication. One to all communication is referred as broadcasting. In the IP address, if all the bits are set to 1 it serves as the broadcast address. The packet is transmitted to each and every node of the network. Geocasting includes all such protocols which can be used for geographic.

Another way to classify routing protocols may be based on the network hierarchies: In flat protocols all nodes are on the same level. In hierarchical protocols nodes are logically or physically on a different level of hierarchy.

Based on the multicast topology, multicast routing protocols are grouped into two types: Tree-based protocols are those in which there exists only one possible path between a source-destination pair, whereas in mesh-based protocols, there may be more paths. Tree-based protocols are further categorized into shared-tree and source-tree topologies. Mesh-based Protocols are more robust to the changes in the network.

Many routing protocols optimized for mobile wireless ad hoc networks emerged recently and they can be classified into topology-based and location-aware protocols. The main category of MANET routing protocols can be further divided into proactive (table-driven), reactive (on-demand), and hybrid (combining both reactive and proactive mechanisms).

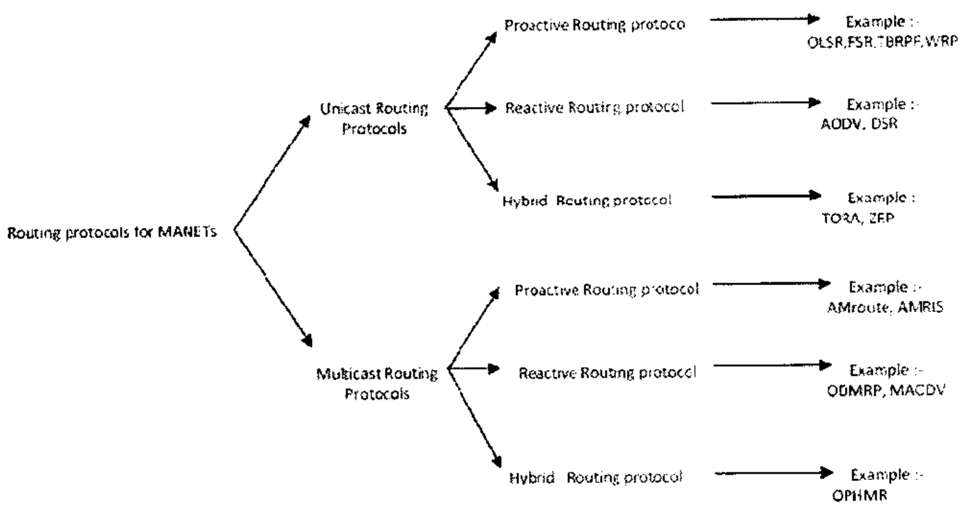

Routing protocols for MANETs
Unicast Routing Protocols
Multicast Routing Protocols
Proactive Routing protoco
Reactive Routing protocol
Hybrid Routing protocol
Example :- OLSR,FSR,TBRPF,WRP
Example :- AODV, DSR
Example : TORA, ZRP
Proactive Routing protocol
Reactive Routing protocol
Hybrid Routing protocol
Example :- AMroute, AMRIS
Example :- ODMRP, MACDV
Example :- OPHMR

Figure 1.1 Classification on routing Protocols for MANETs

Unicast and multicast routing protocols are divided into proactive, reactive, and hybrid protocols. Figure 1.1 gives a classification on routing protocol which is based on unicast and multicast routing protocol. In proactive routing routes available immediately. Reactive routing discovers the route when needed. Hybrid routing is combination of both, such as proactive for neighborhood, reactive for far away.

Multicasting in wireless ad –hoc network is a hot topic in recent years. In multicasting, packets are transmitted from one source or a group of sources to a group of one or more hosts that are identified by a single destination address. Multicasting greatly reduces the transmission cost when sending the same packet to multiple recipients. Multicast reduces the channel bandwidth, sender and router processing and delivery delay. In addition multicast gives robust communication whereby the receiver address is unknown or modifiable without the knowledge of the source within the wireless environment. In recent years, a number of new multicast routing protocols of different styles have been proposed for ad hoc networks.

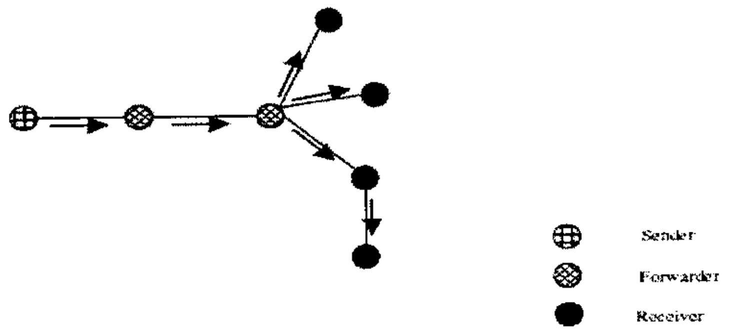


Figure 1.2 Multicast: Data packet replicated by the network

ODMRP (On Demand Multicast Routing Protocol), is a mesh based multicast protocol that provides richer connectivity among multicast members. By building a mesh and supplying multiple routes, multicast packets can be delivered to destinations on the face of node movements and topology changes. To establish a mesh for each multicast group, ODMRP uses the concept of forwarding group. The forwarding group is a set of nodes responsible for forwarding multicast data on shortest paths between any member pairs. ODMRP also applies on-demand routing techniques to avoid the channel overhead and improves scalability. The major strength of ODMRP is its simplicity.

In this project it is considered that the delay properties of one-hop networks with general interference constraints and multiple traffic streams with time-correlated arrivals. This project show that the well known maximal scheduling algorithm achieves average delay.

Chapter 2 with basic ODMRP, Chapter 3 defines the problems with the existing ODMRP, Chapter 4 provides the proposed ODMRP with the methodologies for enhancement, Chapter 5 concludes that the proposed ODMRP is efficient than existing ODMRP.

CHAPTER 2

LITERATURE SURVEY

2.1 ODMRP

On-Demand Multicast Routing Protocol (ODMRP) was developed by the Wireless Adaptive Mobility (WAM) Laboratory at University of California, Los Angeles. ODMRP applies "on-demand" routing techniques to avoid channel overhead and improves scalability. It uses the concept of "forwarding group," a set of nodes responsible for forwarding multicast data, to build a forwarding mesh for each multicast group. By maintaining and using a mesh instead of a tree, the drawbacks of multicast trees in mobile wireless networks (e.g., intermittent connectivity, traffic concentration, frequent tree reconfiguration, non-shortest path in a shared tree, etc.) are avoided. A soft-state approach is taken to maintain multicast group members. No explicit control message is required to leave the group (Yudhvir et.al,2010). The reduction of channel/storage overhead and the relaxed connectivity make ODMRP more attractive in mobile wireless networks. ODMRP is well suited for ad hoc wireless networks with mobile hosts where bandwidth is limited, topology changes frequently and rapidly.

2.2 ADVANTAGES OF ODMRP

i. Simplicity
ii. Low channel and storage overhead
iii. Usage of up-to-date shortest routes
iv. Reliable construction of routes and forwarding group
v. Robustness to host mobility
vi. Maintenance and exploitation of multiple redundant paths

vii. Exploitation of the broadcast nature of wireless environments
viii. Unicast routing capability
ix. Scalability using efficient flooding.

2.3 GENERAL TERMS

This section defines terminology used in ODMRP.

NODE: A device that implements IP.

NEIGHBOR: Nodes that are within the radio transmission range.

FORWARDING GROUP: A group of nodes responsible for forwarding multicast data on shortest paths between any member pairs. ODMRP also applies on-demand routing techniques to avoid the channel overhead and improve scalability.

MULTICAST MESH: The topology defined by the link connection between forwarding group members.

JOIN QUERY: The special data packet sent by multicast sources to establish and update group memberships and routes.

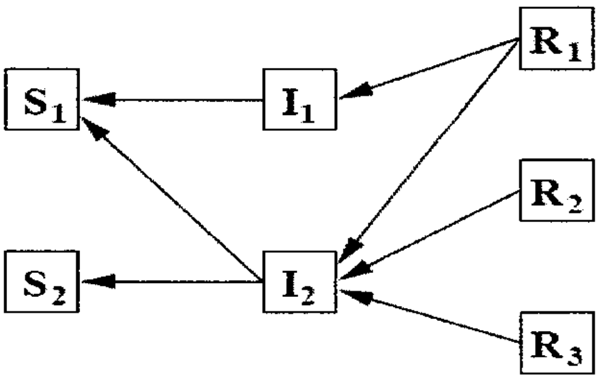
JOIN REPLY: The table broadcasted by each multicast receiver and forwarding node to establish and update group membership and routes

2.4. PROTOCOL OVERVIEW

2.4.1 Multicast route and mesh creation

In ODMRP, group membership and multicast routes are established and updated by the source on demand. Similar to on-demand unicast routing protocols, a request phase and a reply phase comprise the protocol. When a multicast source has packets to send but no route and group membership is known, it floods a member advertising packet with data payload piggybacked. This packet, called "JOIN QUERY". It is periodically broadcasted to the entire network to refresh the membership information and update the routes. When a node receives a JOIN QUERY packet, it stores the source address and the unique identifier of the packet to its "Message Cache" to detect duplicates. The upstream node address is inserted or updated as the next node for the source node in its "Routing Table." If the JOIN QUERY packet is not a duplicate and the Time-To-Live value is greater than zero, appropriate fields are updated and it is rebroadcast. When a JOIN QUERY packet reaches the multicast receiver, it creates and broadcasts a " JOIN REPLY " to its neighbors.

When a node receives a JOIN REPLY, it checks if the next node address of one of the entries matches its own address. If it does, the node realizes that it is on the path to the source and thus is part of the forwarding group; it sets the FG_FLAG (Forwarding group flag). It then broadcasts its own JOIN REPLY built upon matched entries. The next node address field is filled in by extracting the information from its routing table. This way, the JOIN REPLY is propagated by each forward group member until it reaches the multicast source via the selected path. This process constructs (or updates) the routes from sources to receivers and builds a mesh of nodes, the forwarding group.



Join Reply of Node R_1

Sender	Next Node
S_1	I_1
S_2	I_2

Join Reply of Node I_1

Sender	Next Node
S_1	S_1

Figure 2.1 JOIN REPLY

As an example of JOIN REPLY forwarding process (Lee et.al, 2002) nodes S1 and S2 are multicast sources, and nodes R1 and R2 are multicast receivers. Node R2 sends its JOIN REPLY to both S1 and S2 via I2, and R1 sends its packet to S1 via I1 and to S2 via I2. When receivers send their Join Replies to next hop nodes, an intermediate node I1 sets the FG_FLAG and builds its own JOIN REPLY since there is a next node ID entry in the JOIN REPLY received from R1 that matches its ID. Note that the JOIN REPLY built by I1 has an entry for sender S1 but not for S2 because the next node address for S2 in the received JOIN REPLY is not I1. In the meanwhile, node I2 sets the FG_FLAG constructs its own JOIN REPLY and sends it to its neighbors. Note that I2 broadcasts the JOIN REPLY only once even though it receives two Join Replies from the receivers because the second table arrival carries no new source information. Channel overhead is thus reduced dramatically in cases where numerous multicast receivers share the same links to the source.

After this group establishment and route construction process, a source can multicast packets to receivers via selected routes and forwarding groups. While outgoing data packets exist, the source sends JOIN QUERY every REFRESH_INTERVAL. This JOIN QUERY and JOIN REPLY propagation process refreshes forwarding group and routes. When receiving the multicast data packet, a node forwards it only when it is not a duplicate and the setting of the FG_FLAG for the multicast group has not expired. This procedure minimizes the traffic overhead and prevents sending packets through stale routes.

ODMRP uses location and movement information to predict the duration of time that routes will remain valid. With the predicted time of route disconnection, a "join data" packet is flooded when route breaks of ongoing data sessions are imminent. It reveals that ODMRP is better suited for ad hoc networks in terms of bandwidth utilization.

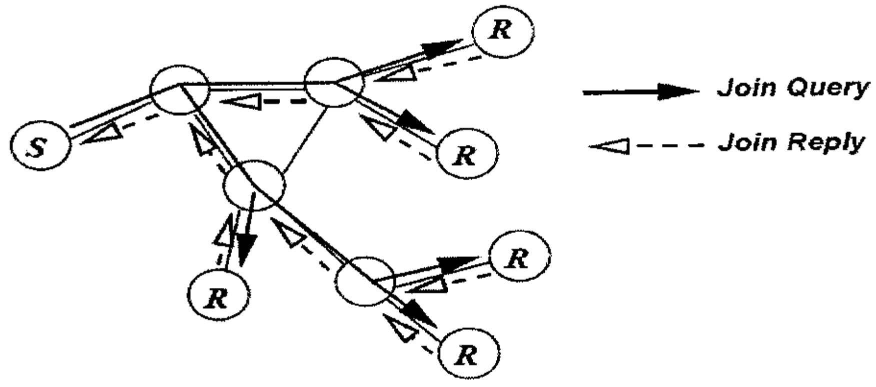


Figure 2.2 ODMRP Mesh Creation

2.4.2 Data forwarding

After the group establishment and route construction process, a multicast source can transmit packets to receivers via selected routes and forwarding groups(Baburaj and Vasudevan,2007). Periodic control packets are sent only when outgoing data packets are still present. When receiving a multicast data packet, a node forwards it only if it is not a duplicate and the setting of the FG_FLAG for the multicast group has not expired. This procedure minimizes traffic overhead and prevents sending packets through stale routes.

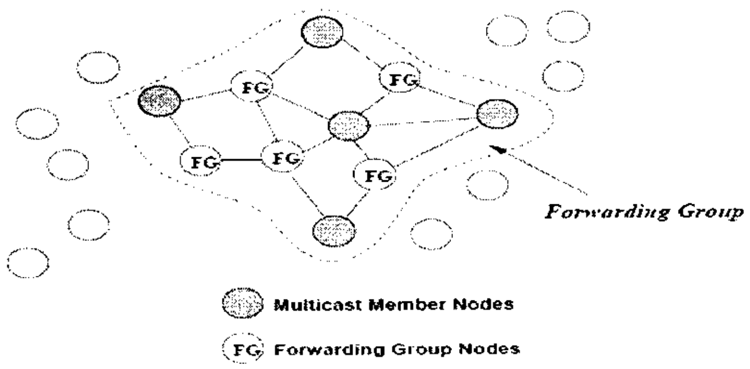


Figure 2.3 Forwarding group concepts

2.4.3 Soft state

In ODMRP, no explicit control packets need to be sent to join or leave the group (Narshima et.al, 2007). If a multicast source wants to leave the group, it simply stops sending JOIN QUERY packets since it does not have any multicast data to send to the group. If a receiver no longer wants to receive from a particular multicast group, it removes the corresponding entries from its Member Table and does not transmit the JOIN REPLY for that group. Nodes in the forwarding group are demoting to non-forwarding nodes if not refreshed (no Join Replies received) before they timeout.

2.4.4 Adapting the refresh interval via mobility prediction

ODMRP requires periodic flooding of JOIN QUERIES to build and refresh routes. Excessive flooding is, however, is not desirable in ad-hoc networks because of bandwidth constraints. Furthermore, flooding often causes congestion, contention, and collision. In the prediction method, assume a free space propagation model, where the received signal strength solely depends on its distance to the transmitter. Assume that all nodes in the network have their clock synchronized. Therefore, if the motion parameters of two neighbors are known, the duration of connection between these two nodes can be determined.

2.4.5 Network model

Assume two nodes i and j are within the transmission range r of each other. Let (x_i, y_i) be the coordinate of mobile host i and (x_j, y_j) be that of mobile host j. Also let v_i and v_j be the speeds, and T_i and T_j ($0<= T_i, T_j<2\pi$) be the moving direction of nodes i and j respectively, Then, the amount of time that they will stay connected, D_t is predicted by:

$$D_t = \frac{-(ab+cd)+\sqrt{(a^2+c^2)r^2-(ad-bc)^2}}{(a^2+c^2)}$$ ..... Equation (2.1)

where ,

$a = v_k \cos \theta_k - v_j \cos \theta_j$

$b = x_k - x_j$

$c = v_k \sin \theta_k - v_j \sin \theta_j$

$d = y_k - y_j$

r = Transmission radius

Figure 2.4 Distance calculation

When $v_k = v_j$ and $\theta_k = \theta_j$, D_t is set to ∞. The value of D_t gives the link expiration time between the neighbor nodes k and j. The direction of movement of the node can be calculated as:

$$\tan\theta = \frac{y_2 - y_1}{x_2 - x_1}$$ ...... Equation (2.2)

Where, θ is the direction of movement of the node. (x_1, y_1) and (x_2, y_2) are the positions of the nodes at two different time instants.

2.4.6 Data structures

Network hosts running ODMRP are required to maintain the following data structures.

Member table: Each multicast receiver stores the source information in the Member Table. For each multicast group the node is participating in, the source ID and the time when the last JOIN REQUEST is received from the source are recorded. If no JOIN REQUEST is received from a source within the refresh period, that entry is removed from the Member table.

Routing table: A Routing table is created on demand and is maintained by each node. An entry is inserted or updated when a non-duplicate JOIN REQUEST is received. The node stores the destination (i.e., the source of the JOIN REQUEST) and the next hop to the destination (i.e., the last node that propagated the JOIN REQUEST). The Routing table provides the next hop information when transmitting Join tables.

Forwarding group table: When a node is a forwarding group node of the multicast group, it maintains the group information in the Forwarding group table. The multicast group ID and the time when the node was last refreshed are recorded.

Message cache: The Message Cache is maintained by each node to detect duplicates. When a node receives a new JOIN REQUEST or data, it stores the source ID and the sequence number of the packet. Note that entries in the Message Cache need not to be maintained permanently. Schemes such as LRU (Least Recently Used) or FIFO (First in First Out) can be employed to expire and remove old entries and prevent the size of the Message Cache to be extensive.

CHAPTER 3

PROBLEM DEFINITION

The main disadvantage of ODMRP is its excessive delay due to interference. Therefore, ODMRP may suffer scalability issue. Another drawback of ODMRP is that the mesh contains a lot of redundancy and thereby the number of transmissions of a specific multicast data packet is unnecessarily high and leads to a high bandwidth consumption. Also, the delay required to obtain a route is also a drawback of ODMRP. This route acquisition latency makes on-demand protocols less attractive in networks where real-time traffic is exchanged.

The proposed system develops order-optimal delay results for a single hop in ad-hoc networks with general interference set constraints. This system provides a flow control technique that works together with maximal scheduling and yields an explicit utility-delay tradeoff. Thereby it removes route acquisition latency. The usage of maximal scheduling achieves average delay that grows at most logarithmically in the largest number of interferers at any link. The usage of Markov chains also prove that average delay is independent of the number of nodes and links in the network, and hence is order-optimal.

CHAPTER 4

PROPOSED SYSTEM

4.1 PROPOSED TECHNIQUE

This system derives average delay bounds for single hop in wireless ad-hoc networks that use maximal scheduling subject to a general set of interference constraints. In order to reduce the scheduling complexity, queue grouping techniques are implemented. In particular, when arrival processes are modulated by independent Markov processes, average delay grows at most logarithmically in the number of nodes in the network. Existing work provides explicitly computable and order-optimal delay bounds for time-correlated arrivals. The proposed work addresses the issues of general interference constraints and time-correlated traffic simultaneously.

4.2 QUEUEING ANALYSIS

Queue grouping techniques have been used in reducing scheduling complexity in switches and wireless networks. They are used to provide order-optimal delay for opportunistic scheduling in a single server downlink. Near order-optimal delay is established for packet switches in. This project particularly treats delay in wireless networks with general constraint sets and time-correlated arrivals. These are perhaps the first delay bounds for controlled queuing networks that explicitly incorporate such statistical information. This allows delay to be understood in terms of general models for network traffic.

Define $Q_l(t)$ as the number of queued packets waiting for transmission over link l during slot t. Let Q (t) = $(Ql(t))_{l\in L}$ be the vector of queue. Define μl (t)ε{0,1} as the transmission rate offered to the link during slot t (in units of packets/slot). That is, μl(t) = 1 if link l is scheduled for transmission on slot t, and μl(t) = 0 otherwise. The scheduler only schedules a link l that does not violate the interference constraints and that has a packet ready for transmission (so that Ql(t) > 0). Let μ(t) = (μl(t))l∈L represent the transmission rate vector for slot t. Define X(t) as the set of feasible transmission vectors for slot t, representing all μ(t) rate vectors that conform to the constraints defined by the interference sets Sl and the additional constraint that μl(t) = 1 only if Ql(t) > 0 (for each l ∈ L). The goal is to observe the queue every slot and make scheduling decisions μ(t) ∈ X(t) so as to support all incoming traffic with average delay as small as possible.

4.3 MAXIMAL SCHEDULING

Define the network capacity region Λ as the closure of the set of all arrival rate vectors $(\lambda l)_{l\in L}$ that can be stably supported, considering all possible scheduling algorithms that conform to the above constraints. It is well known that scheduling according to a generalized max-weight rule every timeslot ensures stability and maximum throughput whenever arrival rates are interior to the capacity region. However, the max-weight rule involves an integer optimization that may be difficult to implement, and has delay properties that are difficult to analyze. In this project scheduling is done according to a simpler maximal scheduling algorithm. Specifically, given a queue vector Q(t), a transmission vector μ(t) is maximal if it satisfies the interference constraints and is such that for all links $l \in L$, if Ql(t) > 0 then μ_ω(t) = 1 for at least one link $\omega \in S_l$. It means that if link l has a packet, then either link l is selected for transmission, or some other link within the interference set S1 is selected. There is much recent interest in maximal scheduling because of its implementation and its ability to support input rates within a constant factor of the capacity region for wireless networks.

One way to achieve a maximal scheduling is as follows: First select any non-empty link $l \in L$ and label it "active." Then select any other non-empty link that does not conflict with the active link l (i.e., that is not within Sl). Label this second link "active." Continue in the same way, selecting new non-empty links that do not conflict with any previously selected links, until no more links can be added. It is not difficult to see that this final set of links labeled "active" has the desired maximal property. Maximal link selections are not unique, and can alternatively be found in a distributed manner, where multiple nodes attempt to activate their non-conflicting, non-empty links simultaneously, and contentions are resolved locally. This distributed implementation also requires multiple iterations before the set of selected links becomes maximal.

In this project, it is assumed that transmission decisions are made every slot according to any maximal scheduling. For convenience, further assume that the maximal scheduling has a well defined probabilistic structure given the queue backlog vector, so that the entire queuing system can be viewed as an ergodic Markov chain with a countably infinite state space. The inequality is the only additional property of maximal scheduling required in this analysis.

4.4 DELAY ANALYSIS

Assume that the arrival process is deterministic and vehicle arrives at a uniform rate. Further, assume that the system in unsaturated -- that is, the total number of vehicles that arrive in a period is less than the total number of vehicles that can be served by the system. These two assumptions mean that the arrival rate is such that all the vehicles that come in a cycle are cleared within the same cycle.

The average delay to vehicles for this case then can be easily determined.. The figure shows a typical cumulative arrival / departure graph against time for an unsaturated, uniform arrival rate approach to an intersection. The slope of the cumulative arrival line is v, where v is the uniform arrival rate in vehicles per unit time. The slope of the cumulative departure line is sometimes zero (when the light is red) and sometimes s(when the light is green); where s is the saturation flow rate obtained as the reciprocal of the saturation headway explained earlier; s is expressed as vehicles per hour of green per lane.

4.5 FLOW CONTROL

In data communications, flow control is the process of managing the pacing of data transmission between two nodes to prevent a fast sender from outrunning a slow receiver. It provides a mechanism for the receiver to control the transmission speed, so that the receiving node is not overwhelmed with data from transmitting node. Flow control should be distinguished from congestion control, which is used for controlling the flow of data when congestion has actually occurred. Flow control mechanisms can be classified based on whether the receiving node sends feedback to the sending node or not.

Flow control is important because it is possible for a sending computer to transmit information at a faster rate than the destination computer can receive and process them. This can happen if the receiving computers have a heavy traffic load in comparison to the sending computer, or if the receiving computer has less processing power than the sending computer.

A Markov chain is a mathematical system that transits from one state to another (out of a finite or countable number of possible states) in a chainlike manner. It is a random process endowed with the Markov property: that the next state depends only on the current state and not on the past. Markov chains have many applications as statistical models of real-world processes. A "discrete-time" random process means a system which is in a certain state at each "step", with the state changing randomly between steps. The steps are often thought of as time, but they can equally well refer to physical distance or any other discrete measurement; formally, the steps are just the integers or natural numbers, and the random process is a mapping of these to states. The Markov property states that the conditional probability distribution for the system at the next step (and in fact at all future steps) given its current state depends only on the current state of the system, and not additionally on the state of the system at previous steps. Since the system changes randomly, it is generally impossible to predict the exact state of the system in the future. However, the statistical properties of the system's future can be predicted. In many applications it is these statistical properties that are important.

The changes of state of the system are called transitions, and the probabilities associated with various state-changes are called transition probabilities. The set of all states and transition probabilities completely characterizes a Markov chain. By convention, assume all possible states and transitions have been included in the definition of the processes, so there is always a next-state and the process goes on forever.

4.6 SOFTWARE ENVIRONMENT

The software environment in which the proposed system is presented is .NET Framework.

4.6.1 Features Of .Net

Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There's no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate. ".NET" is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual

Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services, and so on).

4.6.2 The .Net Framework

The .NET Framework has two main parts:

1. The Common Language Runtime (CLR).
2. A hierarchical set of class libraries.

The CLR is described as the "execution engine" of .NET. It provides the environment within which programs run. The most important features are

- Conversion from a low-level assembler-style language, called Intermediate Language (IL), into code native to the platform being executed on.
- Memory management, notably including garbage collection.
- Checking and enforcing security restrictions on the running code.
- Loading and executing programs, with version control and other such features.

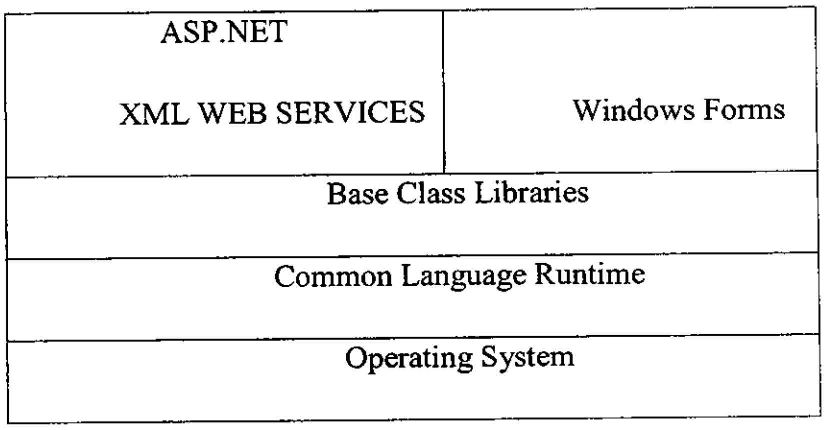


Figure 4.1 .NET Framework

4.6.3 Common Type System

The CLR uses something called the Common Type System (CTS) to strictly enforce type-safety. This ensures that all classes are compatible with each other, by describing types in a common way. CTS define how types work within the runtime, which enables types in one language to interoperate with types in another language, including cross-language exception handling. As well as ensuring that types are only used in appropriate ways, the runtime also ensures that code doesn't attempt to access memory that hasn't been allocated to it.

4.6.4 Common Language Specification

The CLR provides built-in support for language interoperability. To ensure that you can develop managed code that can be fully used by developers using any programming language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

4.6.5 The Class Library

.NET provides a single-rooted hierarchy of classes, containing over 7000 types. The root of the namespace is called System; this contains basic types like Byte, Double, Boolean, and String, as well as Object. All objects derive from System. Object. As well as objects, there are value types. Value types can be allocated on the stack, which can provide useful flexibility. There are also efficient means of converting value types to object types if and when necessary. The set of classes is pretty comprehensive, providing collections, file, screen, and network I/O, threading, and so on, as well as XML and database connectivity.

The class library is subdivided into a number of sets (or namespaces), each providing distinct areas of functionality, with dependencies between the namespaces kept to a minimum.

4.6.6 Languages Supported By .Net

The multi-language capability of the .NET Framework and Visual Studio .NET enables developers to use their existing programming skills to build all types of applications and XML Web services. The .NET framework supports new versions of Microsoft's old favorites Visual Basic and C++ (as VB.NET and Managed C++), but there are also a number of new additions to the family.

Visual Basic .NET has been updated to include many new and improved language features that make it a powerful object-oriented programming language. These features include inheritance, interfaces, and overloading, among others. Visual Basic also now supports structured exception handling, custom attributes and also supports multi-threading. Visual Basic .NET is also CLS compliant, which means that any CLS-compliant language can use the classes, objects, and components you create in Visual Basic .NET.

Managed Extensions for C++ and attributed programming are just some of the enhancements made to the C++ language. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework. C# is Microsoft's new language. It's a C-style language that is essentially "C++ for Rapid Application Development". Unlike other languages, its specification is just the grammar of the language. It has no standard library of its own, and instead has been designed with the intention of using the .NET libraries as its own.

Microsoft Visual J# .NET provides the easiest transition for Java-language developers into the world of XML Web Services and dramatically improves the interoperability of Java-language programs with existing software written in a variety of other programming languages.

Active State has created Visual Perl and Visual Python, which enable .NET-aware applications to be built in either Perl or Python. Both products can be integrated into the Visual Studio .NET environment. Visual Perl includes support for Active State's Perl Dev Kit.

CHAPTER 5

CONCLUSION

In this project a new technique to improve the performance of ODMRP is presented using .NET. The analysis in this project particularly treats delay in wireless networks with general constraint sets and time-correlated arrivals. This allows delay to be understood in terms of general models for network traffic. The proposed system develops order-optimal delay results for a single hop in ad-hoc networks with general interference set constraints. This system provides a flow control technique that works together with maximal scheduling and yields an explicit utility-delay tradeoff. Thereby it removes route acquisition latency.

APPENDICES

A.1 SAMPLE CODING

```
// Maximal Scheduling
//program.cs
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace Maximal_Scheduling
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}

//form.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
namespace Maximal_Scheduling
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    int priyar1 = 0;
    int priyar2 = 0;
    int priyar3 = 0;
    private void pictureBox5_Click(object sender, EventArgs e)
    {
      backgroundWorker1.RunWorkerAsync();
      lblstatus.Text = "Runing...";
    }
    receiver obj = new receiver();
    private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs
e)
    {
      obj.StartServer();
    }
private void Form1_Load(object sender, EventArgs e)
    {
        label1.Text =
Convert.ToString(Environment.GetFolderPath(Environment.SpecialFolder.Desktop)
);
        receiver.receivedPath = label1.Text;
        for (int i = 1; i <= 3; i++)
        {
          if (System.IO.Directory.Exists(label1.Text + "/MS Client " + i) == false)
          {
            System.IO.Directory.CreateDirectory(label1.Text + "/MS Client " + i);
          }
        }

      }
      int[] arr = new int[3]; string pri = "Yes";
      private void timer1_Tick(object sender, EventArgs e)
      {
        if (receiver.curMsg == "Received")
        {
          receiver.curMsg = "";
          l1.Text = receiver.fsize1.ToString();
          l2.Text = receiver.fsize2.ToString();
          l3.Text = receiver.fsize3.ToString();
          if (receiver.fsize1 != 0 && receiver.fsize2 == 0 && receiver.fsize3 == 0
 && receiver.str1 == "start")
          {
            receiver.str1 = "Stop";
            System.Threading.Thread.Sleep(500);
            arr[0] = receiver.fsize1;
            int temp;

            if (receiver.fsize1 == arr[0])
            {
              priyar1 = 1;
            }
            shedul();
            pri = "No";
          }
          if (receiver.fsize1 == 0 && receiver.fsize2 != 0 && receiver.fsize3 == 0
&& receiver.str2 == "start")
          {
            System.Threading.Thread.Sleep(500);
            arr[0] = receiver.fsize2;
            int temp;

            if (receiver.fsize2 == arr[0])
            {
              priyar2 = 1;
            }
            shedul();
            pri = "No";
          }
          if (receiver.fsize1 == 0 && receiver.fsize2 == 0 && receiver.fsize3 != 0
&& receiver.str3 == "start")
          {
            System.Threading.Thread.Sleep(500);
            arr[0] = receiver.fsize3;
            int temp;

            if (receiver.fsize3 == arr[0])
            {
              priyar3 = 1;
            }
            shedul();
            pri = "No";
```

```
        }
        if (receiver.fsize1 != 0 && receiver.fsize2 != 0 && receiver.fsize3 == 0
&& receiver.str1 == "start" && receiver.str2 == "start")
        {
          arr[0] = receiver.fsize1;
          arr[1] = receiver.fsize2;
          int temp;
          for (int i = 0; i <= 2; i++)
          {
            for (int j = i + 1; j <= 2; j++)
            {
              if (arr[i] < arr[j])
              {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
              }
            }
          }
          if (receiver.fsize1 == arr[0])
          {
            priyar1 = 1;
          }
          else if (receiver.fsize2 == arr[0])
          {
            priyar2 = 1;
          }

          if (receiver.fsize1 == arr[1])
          {
            priyar1 = 2;
          }
```

```
            else if (receiver.fsize2 == arr[1])
            {
               priyar2 = 2;
            }


            shedul();
            pri = "No";
         }
         if (receiver.fsize1 == 0 && receiver.fsize2 != 0 && receiver.fsize3 != 0
&& receiver.str2 == "start" && receiver.str3 == "start")
         {
            arr[0] = receiver.fsize2;
            arr[1] = receiver.fsize3;
            int temp;
            for (int i = 0; i <= 2; i++)
            {
               for (int j = i + 1; j <= 2; j++)
               {
                  if (arr[i] < arr[j])
                  {
                     temp = arr[i];
                     arr[i] = arr[j];
                     arr[j] = temp;
                  }
               }
            }
            if (receiver.fsize1 == arr[0])
            {
               priyar1 = 1;
            }
            else if (receiver.fsize2 == arr[0])
            {
                priyar2 = 1;
              }

              if (receiver.fsize1 == arr[1])
              {
                priyar1 = 2;
              }
              else if (receiver.fsize2 == arr[1])
              {
                priyar2 = 2;
              }

              shedul();
              pri = "No";
            }
            if (receiver.fsize1 != 0 && receiver.fsize2 == 0 && receiver.fsize3 != 0
&& receiver.str1 == "start" && receiver.str3 == "start")
            {
              arr[0] = receiver.fsize1;
              arr[1] = receiver.fsize3;
              int temp;
              for (int i = 0; i <= 2; i++)
              {
                for (int j = i + 1; j <= 2; j++)
                {
                  if (arr[i] < arr[j])
                  {
                    temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                  }
                }
            }
            if (receiver.fsize1 == arr[0])
            {
              priyar1 = 1;
            }
            else if (receiver.fsize2 == arr[0])
            {
              priyar2 = 1;
            }

            if (receiver.fsize1 == arr[1])
            {
              priyar1 = 2;
            }
            else if (receiver.fsize2 == arr[1])
            {
              priyar2 = 2;
            }

            shedul();
            pri = "No";

          }
          if (receiver.fsize1 != 0 && receiver.fsize2 != 0 && receiver.fsize3 != 0
&& receiver.str1 == "start" && receiver.str2 == "start" && receiver.str3 == "start")
          {
            arr[0] = receiver.fsize1;
            arr[1] = receiver.fsize2;
            arr[2] = receiver.fsize3;
            int temp;
            for (int i = 0; i <= 2; i++)
            {
```

```
for (int j = i + 1; j <= 2; j++)
{
   if (arr[i] < arr[j])
   {
      temp = arr[i];
      arr[i] = arr[j];
      arr[j] = temp;
   }


 }
}
if (receiver.fsize1 == arr[0])
{
   priyar1 = 1;
}
else if (receiver.fsize2 == arr[0])
{
   priyar2 = 1;
}
else if (receiver.fsize3 == arr[0])
{
   priyar3 = 1;

}
if (receiver.fsize1 == arr[1])
{
   priyar1 = 2;
}
else if (receiver.fsize2 == arr[1])
{
   priyar2 = 2;
```

```
        }
        else if (receiver.fsize3 == arr[1])
        {
            priyar3 = 2;


        }
        if (receiver.fsize1 == arr[2])
        {
            priyar1 = 3;
        }
        else if (receiver.fsize2 == arr[2])
        {
            priyar2 = 3;
        }
        else if (receiver.fsize3 == arr[2])
        {
            priyar3 = 3;
        }
        shedul();
        pri = "No";
    }
    timer1.Enabled = true;
  }
}
private void lbl1()
{
    lblsts1.Text = "ON";
    lblsts1.ForeColor = Color.Green;
    lblsts2.Text = "OFF";
    lblsts2.ForeColor = Color.Red;
    lblsts3.Text = "OFF";
    lblsts3.ForeColor = Color.Red;
    lblid1.Visible = true;
    lblid2.Visible = false;
    lblid3.Visible = false;
}
private void lbl2()
{
    lblsts1.Text = "OFF";
    lblsts1.ForeColor = Color.Red;
    lblsts2.Text = "ON";
    lblsts2.ForeColor = Color.Green;
    lblsts3.Text = "OFF";
    lblsts3.ForeColor = Color.Red;
    lblid1.Visible = false;
    lblid2.Visible = true;
    lblid3.Visible = false;
}
private void lbl3()
{
    lblsts1.Text = "OFF";
    lblsts1.ForeColor = Color.Red;
    lblsts2.Text = "OFF";
    lblsts2.ForeColor = Color.Red;
    lblsts3.Text = "ON";
    lblsts3.ForeColor = Color.Green;
    lblid1.Visible = false;
    lblid2.Visible = false;
    lblid3.Visible = true;
}
private void shedul()
{
    timer1.Enabled = false;
    byte[] ok = Encoding.ASCII.GetBytes("Ok");
```

```
int a, b, c;
if (priyar1 == 1)
{
   lbl1();
   Application.DoEvents();
   a = 3;
   for (int i = 1; i <= a; i++)
   {
      if (progressBar1.Value != 100)
      {
         progressBar1.Value += 10;
         Application.DoEvents();
         send(ok, 1);
         System.Threading.Thread.Sleep(3000);
      }
      else
      {
         //receiver.str1 = "Stop";
         //receiver.fsize1 = 0;
         // priyar1 = 0;
      }
   }
}
else if (priyar1 == 2)
{
   lbl1();
   Application.DoEvents();
   a = 2;
   for (int i = 1; i <= a; i++)
   {
      if (progressBar1.Value != 100)
      {
          progressBar1.Value += 10;
          Application.DoEvents();
          send(ok, 1);
          System.Threading.Thread.Sleep(2000);
        }
        else
        {
          // receiver.str1 = "Stop";
          //receiver.fsize1 = 0;
          // priyar1 = 0;
        }
      }
    }
    else if (priyar1 == 3)
    {
      lbl1();
      Application.DoEvents();
      a = 1;
      for (int i = 1; i <= a; i++)
      {
        if (progressBar1.Value != 100)
        {
          progressBar1.Value += 10;
          Application.DoEvents();
          send(ok, 1);
          System.Threading.Thread.Sleep(2000);
        }
        else
        {
          //receiver.str1 = "Stop";
          // receiver.fsize1 = 0;
          // priyar1 = 0;
        }
      }
    }

    if (priyar2 == 1)
    {
      lbl2();
      Application.DoEvents();
      a = 3;
      for (int i = 1; i <= a; i++)
      {
        if (progressBar2.Value != 100)
        {
          progressBar2.Value += 10;
          Application.DoEvents();
          send(ok, 2);
          System.Threading.Thread.Sleep(2000);
        }
        else
        {
          //receiver.str2 = "Stop";
          // receiver.fsize2 = 0;
          // priyar2 = 0;
        }
      }
    }
    else if (priyar2 == 2)
    {
      lbl2();
      Application.DoEvents();
      a = 2;
      for (int i = 1; i <= a; i++)
    {
      if (progressBar2.Value != 100)
      {
        progressBar2.Value += 10;
        Application.DoEvents();
        send(ok, 2);
        System.Threading.Thread.Sleep(2000);
      }
      else
      {
        //receiver.str2 = "Stop";
        // receiver.fsize2 = 0;
        // priyar2 = 0;
      }
    }
  }
  else if (priyar2 == 3)
  {
    lbl2();
    Application.DoEvents();
    a = 1;
    for (int i = 1; i <= a; i++)
    {
      if (progressBar2.Value != 100)
      {
        progressBar2.Value += 10;
        Application.DoEvents();
        send(ok, 2);
        System.Threading.Thread.Sleep(2000);
      }
      else
      {
            // receiver.str2 = "Stop";
            // receiver.fsize2 = 0;
            // priyar2 = 0;
          }
        }
      }

      if (priyar3 == 1)
      {
        lbl3();
        Application.DoEvents();
        a = 3;
        for (int i = 1; i <= a; i++)
        {
          if (progressBar3.Value != 100)
          {
            progressBar3.Value += 10;
            Application.DoEvents();
            send(ok, 3);
            System.Threading.Thread.Sleep(2000);
          }
          else
          {
            //receiver.str3 = "Stop";
            // receiver.fsize3 = 0;
            // priyar3 = 0;
          }
        }
      }
      else if (priyar3 == 2)
      {
        lbl3();
```

```
Application.DoEvents();
a = 2;
for (int i = 1; i <= a; i++)
{
   if (progressBar3.Value != 100)
   {
      progressBar3.Value += 10;
      Application.DoEvents();
      send(ok, 3);
      System.Threading.Thread.Sleep(2000);
    }
    else
    {
       //receiver.str3 = "Stop";
       //receiver.fsize3 = 0;
       //priyar3 = 0;
    }
  }
}
else if (priyar3 == 3)
{
   lbl3();
   Application.DoEvents();
   a = 1;
   for (int i = 1; i <= a; i++)
   {
      if (progressBar3.Value != 100)
      {
         progressBar3.Value += 10;
         Application.DoEvents();
         send(ok, 3);
         System.Threading.Thread.Sleep(2000);
```

```
            }
            else
            {
               //receiver.str3 = "Stop";
               //receiver.fsize3 = 0;
               //priyar3 = 0;
            }
          }
        }
        timer1.Enabled = true;
        //receiver.str1 = "Stop";
        //receiver.str2 = "Stop";
        //receiver.str3 = "Stop";
      }
      public void send(byte[] data, int port)
      {
         try
         {
            IPAddress[] ipAddress = Dns.GetHostAddresses("192.168.1.2");
            IPEndPoint ipEnd = new IPEndPoint(ipAddress[0], port);
            Socket clientSock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.IP);

            clientSock.Connect(ipEnd);
            clientSock.Send(data);
            clientSock.Close();
         }

         catch (Exception ex)
         {
            if (ex.Message == "A connection attempt failed because the connected
party did not properly respond after a period of time, or established connection
failed because connected host has failed to respond")
            {
                //lblError.Text = "";
                //lblError.Text = "No Such System Available Try other IP";
            }
            else
            {
                if (ex.Message == "No connection could be made because the target
machine actively refused it")
                {
                    //lblError.Text = "";
                    //lblError.Text = "File Sending fail. Because server not running.";
                }
                else
                {
                    //lblError.Text = "";
                    //lblError.Text = "File Sending fail." + ex.Message;
                }
            }
        }
    }

    private void label5_Click(object sender, EventArgs e)
    {

    }
}
class receiver
{
    IPEndPoint ipEnd;
```

```
Socket sock;
string ser1;
string fileDes, fileini;
int len;
public static string[] path = null;
byte[] c1data1; byte[] c2data1; byte[] c3data1;
byte[] c1data2; byte[] c2data2; byte[] c3data2;
byte[] c1data3; byte[] c2data3; byte[] c3data3;
byte[] c1data4; byte[] c2data4; byte[] c3data4;
byte[] c1data5; byte[] c2data5; byte[] c3data5;
byte[] c1data6; byte[] c2data6; byte[] c3data6;
byte[] c1data7; byte[] c2data7; byte[] c3data7;
byte[] c1data8; byte[] c2data8; byte[] c3data8;
byte[] c1data9; byte[] c2data9; byte[] c3data9;
byte[] c1data10; byte[] c2data10; byte[] c3data10;
int cseg1 = 0;
int cseg2 = 0;
int cseg3 = 0;
public receiver()
{
    IPHostEntry ipEntry = Dns.GetHostEntry(Environment.MachineName);
    IPAddress IpAddr = ipEntry.AddressList[0];
    ipEnd = new IPEndPoint(IpAddr, 4);
    sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.IP);
    sock.Bind(ipEnd);
}
public static string receivedPath;
public static string curMsg = "Stopped";
public static string str1 = "";
public static string str2 = "";
public static string str3 = "";
public static byte[] clientData;
public static int receivedBytesLen;
public static int fsize1 = 0;
public static int fsize2 = 0;
public static int fsize3 = 0;
int count = 0;
public void StartServer()
{
    try
    {
        //curMsg = "Starting...";
        sock.Listen(100);

        // curMsg = "Running and waiting to receive file.";
        Socket clientSock = sock.Accept();

        clientData = new byte[1024 * 15000];

        receivedBytesLen = clientSock.Receive(clientData);

        System.Threading.Thread.Sleep(1000);
        string who = Encoding.ASCII.GetString(clientData, 0, 2);
        if (who == "c1")
        {
            if (fsize1 == 0)
            {
                fsize1 = Convert.ToInt32(Encoding.ASCII.GetString(clientData, 2,
receivedBytesLen - 2));
            }
            else
            {
                int fileNameLen =
Convert.ToInt32(Encoding.ASCII.GetString(clientData, 2, 2));
                string fileName = Encoding.ASCII.GetString(clientData, 4,
fileNameLen);

                BinaryWriter bWrite = new BinaryWriter(File.Open(receivedPath +
"/MS Client 1/" + fileName, FileMode.Append)); ;
                bWrite.Write(clientData, 4 + fileNameLen, receivedBytesLen - 4 -
fileNameLen);

                curMsg = "Saving file...";

                bWrite.Close();
                if (fileName.EndsWith(".E"))
                {
                   MargeUp(receivedPath + "/MS Client 1/" + fileName);
                }
                str1 = "start";
              }
            }
            else if (who == "c2")
            {
               if (fsize2 == 0)
               {
                  fsize2 = Convert.ToInt32(Encoding.ASCII.GetString(clientData, 2,
receivedBytesLen - 2));
               }
               else
               {

                  int fileNameLen =
Convert.ToInt32(Encoding.ASCII.GetString(clientData, 2, 2));
```

```
                string fileName = Encoding.ASCII.GetString(clientData, 4,
fileNameLen);

                BinaryWriter bWrite = new BinaryWriter(File.Open(receivedPath +
"/MS Client 2/" + fileName, FileMode.Append)); ;
                bWrite.Write(clientData, 4 + fileNameLen, receivedBytesLen - 4 -
fileNameLen);

                curMsg = "Saving file...";

                bWrite.Close();
                if (fileName.EndsWith(".E"))
                {
                   MargeUp(receivedPath + "/MS Client 2/" + fileName);
                }
                str2 = "start";
              }
            }
            else if (who == "c3")
            {
              if (fsize3 == 0)
              {
                 fsize3 = Convert.ToInt32(Encoding.ASCII.GetString(clientData, 2,
receivedBytesLen - 2));
              }
              else
              {

                 int fileNameLen =
Convert.ToInt32(Encoding.ASCII.GetString(clientData, 2, 2));
                 string fileName = Encoding.ASCII.GetString(clientData, 4,
fileNameLen);
```

```
                BinaryWriter bWrite = new BinaryWriter(File.Open(receivedPath +
"/MS Client 3/" + fileName, FileMode.Append)); ;
                bWrite.Write(clientData, 4 + fileNameLen, receivedBytesLen - 4 -
fileNameLen);

                curMsg = "Saving file...";

                bWrite.Close();
                if (fileName.EndsWith(".E"))
                {
                  MargeUp(receivedPath + "/MS Client 3/" + fileName);
                }
                str3 = "start";
              }
            }
            curMsg = "Received";
            //BinaryWriter bWrite = new BinaryWriter(File.Open(receivedPath + "/" +
  fileName, FileMode.Append)); ;
            //bWrite.Write(clientData, 4 + fileNameLen, receivedBytesLen - 4 -
  fileNameLen);

            //curMsg = "Saving file...";

            //bWrite.Close();
            clientSock.Close();
            //send();
            StartServer();

            // curMsg = "Reeived & Saved file; Server Stopped.";

          }
```

```
        catch (Exception ex)
        {
            curMsg = "File Receving error.";
        }
    }
    public void MargeUp(string firstFileName)
    {
        if (firstFileName.Length < 1)
            return;

        string endPart = firstFileName;
        string orgFile = "";

        orgFile = endPart.Substring(0, endPart.LastIndexOf("."));
        endPart = endPart.Substring(endPart.LastIndexOf(".") + 1);

        if (endPart == "E")//If only one slice is there
        {
            orgFile = orgFile.Substring(0, orgFile.LastIndexOf("."));
            endPart = "0";
        }

        if (File.Exists(orgFile))
        {
            if (MessageBox.Show(orgFile + " already exists, do you want to delete it",
"", MessageBoxButtons.YesNo) == DialogResult.Yes)
                File.Delete(orgFile);
            else
            {
                MessageBox.Show("File not assembled. Operation cancelled by user.");
                return;
            }
        }

        //Assembling starts from here
        BinaryWriter bw = new BinaryWriter(File.Open(orgFile,
FileMode.Append));
        string nextFileName = "";
        byte[] buffer = new byte[bw.BaseStream.Length];



        int counter = int.Parse(endPart);
        while (true)
        {
           nextFileName = orgFile + "." + counter.ToString();
           if (File.Exists(nextFileName + ".E"))
           {
              //Last slice
              buffer = File.ReadAllBytes(nextFileName + ".E");
              bw.Write(buffer);
              if (File.Exists(nextFileName + ".E"))
                 File.Delete(nextFileName + ".E");
              break;
           }
           else
           {
              buffer = File.ReadAllBytes(nextFileName);
              bw.Write(buffer);
           }
           counter++;
           if (File.Exists(nextFileName))
              File.Delete(nextFileName);
        }
        bw.Close();
```

```
  //MessageBox.Show("File assebled successfully");
}

private void client1()
{
  if (cseg1 == 0)
  {
    cseg1++;
    c1data1 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c1data1, 0, receivedBytesLen - 3);

  }
  else if (cseg1 == 1)
  {
    cseg1++;
    c1data2 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c1data2, 0, receivedBytesLen - 3);

  }
  else if (cseg1 == 2)
  {
    cseg1++;
    c1data3 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c1data3, 0, receivedBytesLen - 3);

  }
  else if (cseg1 == 3)
  {
    cseg1++;
    c1data4 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c1data4, 0, receivedBytesLen - 3);
```

P - 3597

```
}
else if (csegl == 4)
{
   csegl++;
   cldata5 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, cldata5, 0, receivedBytesLen - 3);

}
else if (csegl == 5)
{
   csegl++;
   cldata6 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, cldata6, 0, receivedBytesLen - 3);

}
else if (csegl == 6)
{
   csegl++;
   cldata7 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, cldata7, 0, receivedBytesLen - 3);

}
else if (csegl == 7)
{
   csegl++;
   cldata8 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, cldata8, 0, receivedBytesLen - 3);

}
else if (csegl == 8)
{
   csegl++;
```

```
            c1data9 = new byte[receivedBytesLen - 3];
            Array.Copy(clientData, 3, c1data9, 0, receivedBytesLen - 3);


        }
        else if (cseg1 == 9)
        {
            cseg1++;
            c1data10 = new byte[receivedBytesLen - 3];
            Array.Copy(clientData, 3, c1data10, 0, receivedBytesLen - 3);
            if (c1data1 != null && c1data2 != null && c1data3 != null && c1data4 !=
null && c1data5 != null && c1data6 != null && c1data7 != null && c1data8 !=
null && c1data9 != null && c1data10 != null)
            {
                byte[] wrdata = new byte[c1data1.Length + c1data2.Length +
c1data3.Length + c1data4.Length + c1data5.Length + c1data6.Length +
c1data7.Length + c1data8.Length + c1data9.Length + c1data10.Length];
                //Array.Copy(c1data1, 0, data1, 0, fsize);
                //Array.Copy(c1data2, 0, data2, 0, fsize);
                //Array.Copy(c1data3, 0 + fsize, data3, 0, fsize);
                //Array.Copy(c1data4, 0 + fsize + fsize, data4, 0, fsize);
                //Array.Copy(c1data5, 0 + fsize + fsize + fsize, data5, 0, fsize);
                //Array.Copy(c1data6, 0 + fsize + fsize + fsize + fsize, data6, 0, fsize);
                //Array.Copy(c1data7, 0 + fsize + fsize + fsize + fsize + fsize, data7, 3,
 fsize);
                //Array.Copy(c1data8, 0 + fsize + fsize + fsize + fsize + fsize + fsize,
 data8, 3, fsize);
                //Array.Copy(c1data9, 0 + fsize + fsize + fsize + fsize + fsize + fsize +
 fsize, data9, 3, fsize);
                //Array.Copy(c1data10, 0 + fsize + fsize + fsize + fsize + fsize + fsize +
 fsize + fsize, data10, 3, fsize + Convert.ToInt32(g));
```

```
            Array.Copy(c1data1, 0, wrdata, 0, c1data1.Length);
            Array.Copy(c1data2, 0, wrdata, c1data1.Length, c1data2.Length);
            Array.Copy(c1data3, 0, wrdata, c1data1.Length + c1data2.Length,
c1data3.Length);
            Array.Copy(c1data4, 0, wrdata, c1data1.Length + c1data2.Length +
c1data3.Length, c1data4.Length);
            Array.Copy(c1data5, 0, wrdata, c1data1.Length + c1data2.Length +
c1data3.Length + c1data4.Length, c1data5.Length);
            Array.Copy(c1data6, 0, wrdata, c1data1.Length + c1data2.Length +
c1data3.Length + c1data4.Length + c1data5.Length, c1data6.Length);
            Array.Copy(c1data7, 0, wrdata, c1data1.Length + c1data2.Length +
c1data3.Length + c1data4.Length + c1data5.Length + c1data6.Length,
c1data7.Length);
            Array.Copy(c1data8, 0, wrdata, c1data1.Length + c1data2.Length +
c1data3.Length + c1data4.Length + c1data5.Length + c1data6.Length +
c1data7.Length, c1data8.Length);
            Array.Copy(c1data9, 0, wrdata, c1data1.Length + c1data2.Length +
c1data3.Length + c1data4.Length + c1data5.Length + c1data6.Length +
c1data7.Length + c1data8.Length, c1data9.Length);
            Array.Copy(c1data10, 0, wrdata, c1data1.Length + c1data2.Length +
c1data3.Length + c1data4.Length + c1data5.Length + c1data6.Length +
c1data7.Length + c1data8.Length + c1data9.Length, c1data10.Length);

            int fileNameLen = BitConverter.ToInt32(wrdata, 0);
            string fileName = Encoding.ASCII.GetString(wrdata, 4, fileNameLen);

            BinaryWriter bWrite = new BinaryWriter(File.Open(receivedPath +
"/MS Client 1/" + fileName, FileMode.Append)); ;
            bWrite.Write(wrdata, 4 + fileNameLen, receivedBytesLen - 4 -
fileNameLen);
            bWrite.Close();
```

```
        }
      }
    }
    private void client2()
    {
      if (cseg2 == 0)
      {
        cseg2++;
        c2data1 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c2data1, 0, receivedBytesLen - 3);

      }
      else if (cseg2 == 1)
      {
        cseg2++;
        c2data2 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c2data2, 0, receivedBytesLen - 3);

      }
      else if (cseg2 == 2)
      {
        cseg2++;
        c2data3 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c2data3, 0, receivedBytesLen - 3);

      }
      else if (cseg2 == 3)
      {
        cseg2++;
        c2data4 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c2data4, 0, receivedBytesLen - 3);
```

```
}
else if (cseg2 == 4)
{
   cseg2++;
   c2data5 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, c2data5, 0, receivedBytesLen - 3);

}
else if (cseg2 == 5)
{
   cseg2++;
   c2data6 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, c2data6, 0, receivedBytesLen - 3);

}
else if (cseg2 == 6)
{
   cseg2++;
   c2data7 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, c2data7, 0, receivedBytesLen - 3);

}
else if (cseg2 == 7)
{
   cseg2++;
   c2data8 = new byte[receivedBytesLen - 3];
   Array.Copy(clientData, 3, c2data8, 0, receivedBytesLen - 3);

}
else if (cseg2 == 8)
{
   cseg2++;
```

```
        c2data9 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c2data9, 0, receivedBytesLen - 3);

    }
    else if (cseg2 == 9)
    {
        cseg2++;
        c2data10 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c2data10, 0, receivedBytesLen - 3);

    }
}
private void client3()
{
    if (cseg3 == 0)
    {
        cseg3++;
        c3data1 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c3data1, 0, receivedBytesLen - 3);

    }
    else if (cseg3 == 1)
    {
        cseg3++;
        c3data2 = new byte[receivedBytesLen - 3];
        Array.Copy(clientData, 3, c3data2, 0, receivedBytesLen - 3);

    }
    else if (cseg3 == 2)
    {
        cseg3++;
        c3data3 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c3data3, 0, receivedBytesLen - 3);

}
else if (cseg3 == 3)
{
    cseg3++;
    c3data4 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c3data4, 0, receivedBytesLen - 3);

}
else if (cseg3 == 4)
{
    cseg3++;
    c3data5 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c3data5, 0, receivedBytesLen - 3);

}
else if (cseg3 == 5)
{
    cseg3++;
    c3data6 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c3data6, 0, receivedBytesLen - 3);

}
else if (cseg3 == 6)
{
    cseg3++;
    c3data7 = new byte[receivedBytesLen - 3];
    Array.Copy(clientData, 3, c3data7, 0, receivedBytesLen - 3);

}
else if (cseg3 == 7)
```

```
        {
          cseg3++;
          c3data8 = new byte[receivedBytesLen - 3];
          Array.Copy(clientData, 3, c3data8, 0, receivedBytesLen - 3);

        }
        else if (cseg3 == 8)
        {
          cseg3++;
          c3data9 = new byte[receivedBytesLen - 3];
          Array.Copy(clientData, 3, c3data9, 0, receivedBytesLen - 3);

        }
        else if (cseg3 == 9)
        {
          cseg3++;
          c3data10 = new byte[receivedBytesLen - 3];
          Array.Copy(clientData, 3, c3data10, 0, receivedBytesLen - 3);

        }
      }
      public void send()
      {
        try
        {
          IPAddress[] ipAddress = Dns.GetHostAddresses("192.168.1.2");
          IPEndPoint ipEnd = new IPEndPoint(ipAddress[0], 5657);
          Socket clientSock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.IP);

          byte[] fileNameByte = Encoding.ASCII.GetBytes(fileDes);
          clientSock.Connect(ipEnd);
```

```
            clientSock.Send(clientData);
            clientSock.Close();
            StartServer();
          }

          catch (Exception ex)
          {
            if (ex.Message == "A connection attempt failed because the connected
party did not properly respond after a period of time, or established connection
failed because connected host has failed to respond")
            {
              //lblError.Text = "";
              //lblError.Text = "No Such System Available Try other IP";
            }
            else
            {
              if (ex.Message == "No connection could be made because the target
machine actively refused it")
              {
                //lblError.Text = "";
                //lblError.Text = "File Sending fail. Because server not running.";
              }
              else
              {
                //lblError.Text = "";
                //lblError.Text = "File Sending fail." + ex.Message;
              }
            }
          }
        }
      }
    }
```

A.2 SCREEN SHOTS

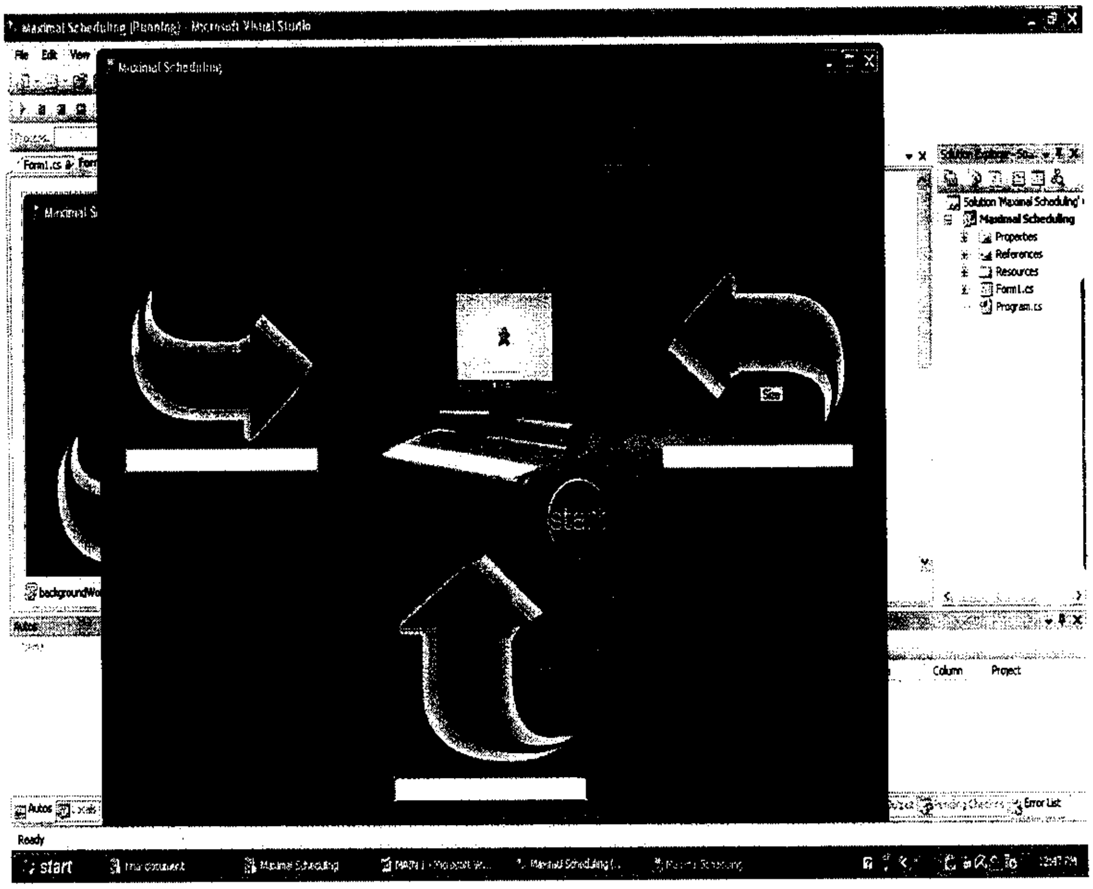


Figure A.1 Maximal scheduling main screen

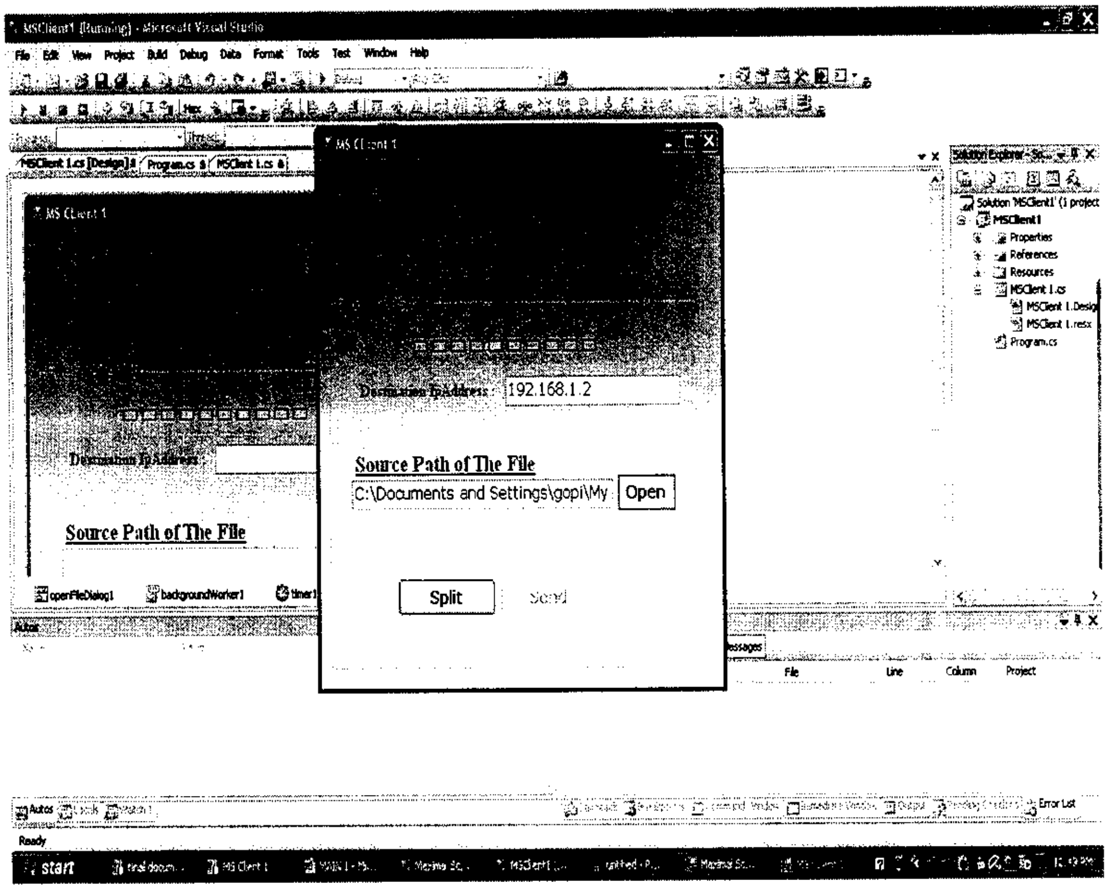


Figure A.2 Node 1 initiating data transfer

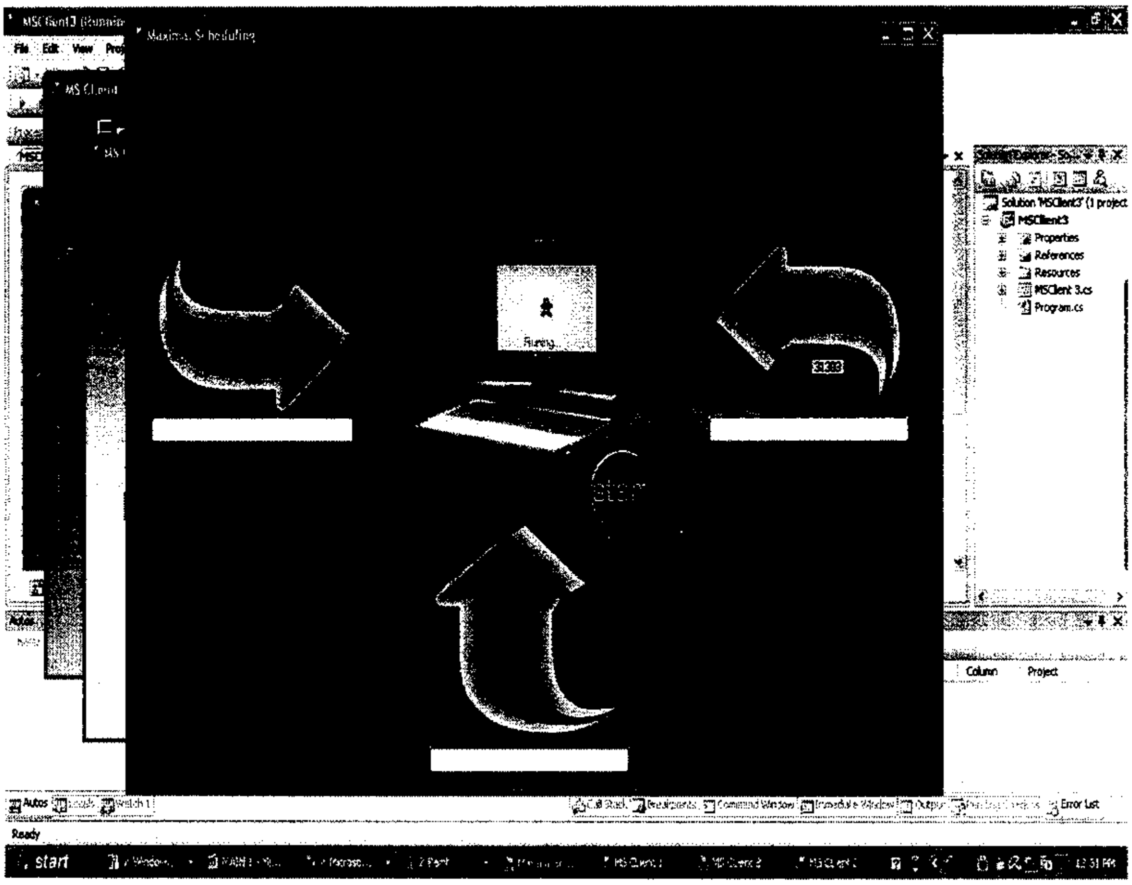


Figure A.3 Setting data size after splitting

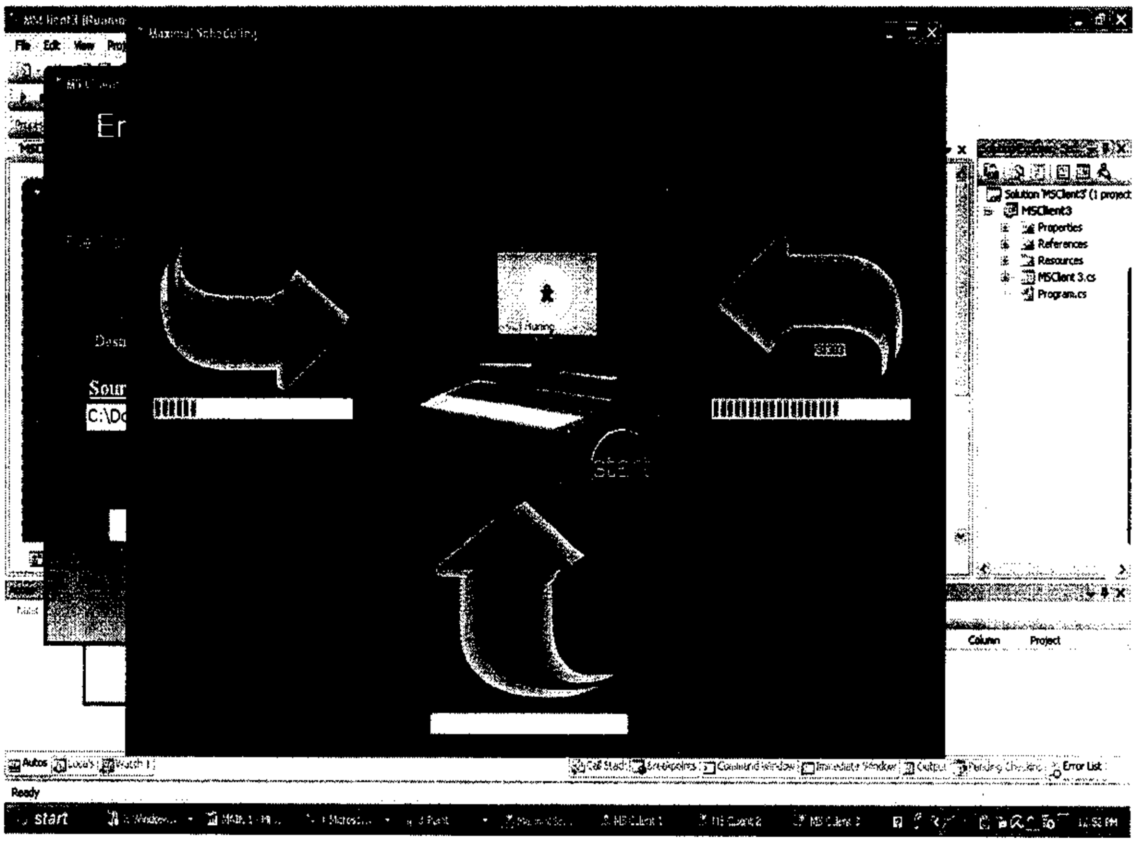


Figure A.4 scheduling the data transfer

REFERENCES

1 Aparna.K, (2010), 'Performance Comparison of MANET (Mobile Ad hoc Network) Protocols (ODMRP with AMRIS and MAODV)' International Journal of Computer Applications (0975 – 8887), Vol.1, No. 10, pp. 42-46.

2 Baburaj.E , Vasudevan.V (2007) 'Multicast Routing Using On-Demand Multicast Routing Protocol in Jade Agents' , International Journal of Soft Computing 2, pp. 411-416.

3 Bagrodia.R, Gerla.M, Hsu.J, Lee.S.J, Su.W, (2000), 'A performance comparison study of Adhoc Wireless Multicast protocols', Proc.of the 19th Annual Joint Conf. Of the IEEE Computer and Communications Societies' , pp.565-574.

4 Bommaiah.E, Liu.M, McAuley.A, Talpade.R, (1998), 'AMRoute: Adhoc Multicast Routing Protocol, Internet draft, draft-talpade-manet-amroute-00.txt'.

5 Chaporkar .P, Kar .K, Luo .X, and Sarkar .S,(2008)' Throughput and fairnessguarantees through maximal scheduling in wireless networks' IEEE Trans. on Information Theory, Vol. 54, No. 2, pp. 572-594.

6 Erciyes. K, Dagdeviren. O, Cokuslu. D, (2006),' modelling and simulation of wireless sensor and mobile ad hoc Networks', Proceedings of the International Conference on Modeling and Simulation, No. B101, pp.28-30.

7 Gerla.M , Hsu.J, Lee.S.J, Su.W,(2002), 'On-Demand Multicast Routing Protocol (ODMRP) for Ad hoc Networks', -ietf-manetodmrp-02.txt.

8 Narsimha.G, Venugopal Reddy.A, Sarma.S.S.V.N, (2007) 'The Effective Multicasting Routing Protocol in Wireless Mobile Adhoc Network', IEEE Computer Society,04196208,Proceedings of the Sixth International Conference on Networking(ICN'07).

9 Neely M.J,(2009) 'Delay Analysis for Maximal Scheduling with Flow Control in Wireless Networks with Bursty Traffic', IEEE Transactions On Networking, Vol. 17, No. 4, pp. 1146-1159.

10 Wu .X, Srikant .R, and Perkins J.R,(2007) ' Scheduling efficiency of distributed greedy scheduling algorithms in wireless networks' IEEE Transactions on Mobile Computing.

11 Yudhvir Singh, Yogesh Chaba , Monika Jain, Prabha Rani, (2010), 'Performance Evaluation of On-Demand MulticastingRouting Protocols in Moble Adhoc Networks', IEEE Computer Society, pp. 298-301.

12 Zhao.Y, Xu.L, Shi.M, (2003), 'On-Demand Multicast Routing Protocol-Multipoint Relay (ODMRP-MRP) in Mobile Ad-hoc Network', proceedings of ICCT, pp. 1295-13.