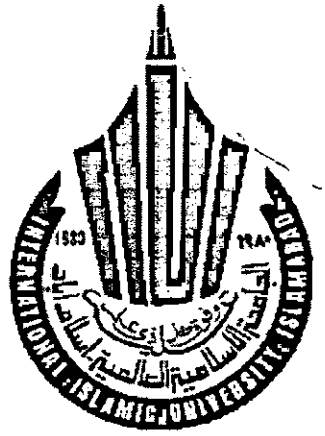
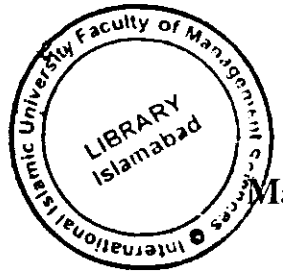
CONGESTION CONTROLLED ADAPTIVE MULTICAST IN SATELLITE BASED NETWORKS





Submitted by

Waqas Paracha

Malik Nasir Mahmood

Supervised by

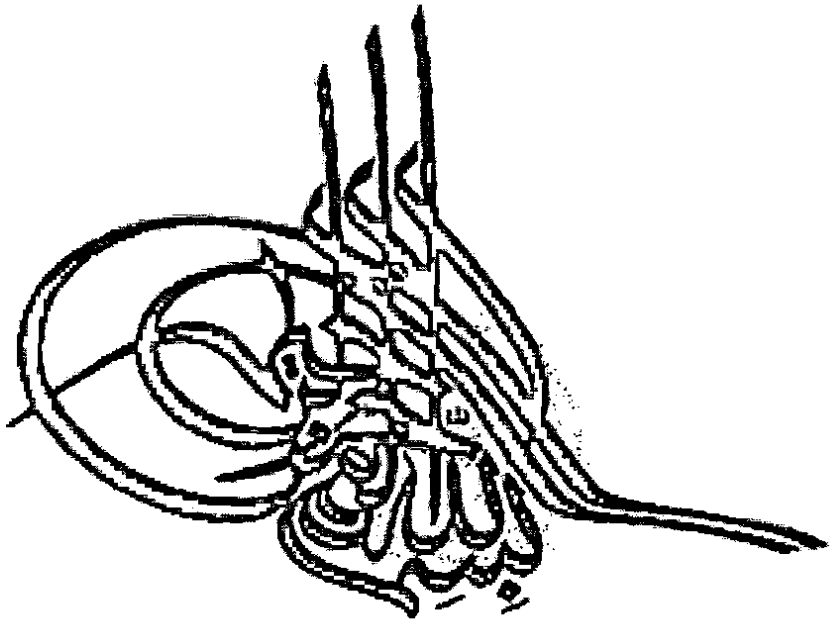
Dr. S. Tauseef-ur-Rehman

Department of Computer Science

Faculty of Applied Sciences

International Islamic University, Islamabad

2004



In the name of ALMIGHTY ALLAH,

The most Beneficent, the most

Merciful.

Department of Computer Science
Faculty of Applied Sciences
International Islamic University, Islamabad.

13 rd September, 2004

Acc. No. (DIS) T-1070

Final Approval

It is certified that we have read the thesis, titled "CONGESTION CONTROLLED ADAPTIVE MULTICAST IN SATELLITE BASED NETWORKS" submitted by *Waqas Paracha* under university Reg. No. **166-CS/MS/03** and *Malik Nasir Mahmood* under university reg. No. **48-CS/MS/01**. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad, for the Degree of **Master of Science**.

Committee

External Examiner
Dr. Abdus Sattar
Consultant, Multimedia Courseware Design Project
Department of Computer Science,
Allama Iqbal Open University, Islamabad.

Internal Examiners
Dr. Sikander Hayat Khial
Head, Department of Computer Science,
International Islamic University, Islamabad.

Supervisor
Dr. S. Tauseef-ur-Rehman
Head, Department of Telecommunication Engineering
& Computer Engineering,
International Islamic University, Islamabad.

Dedication

Dedicated to our families

&

our kind supervisor

Dr. S. Tauseef-ur-Rehman

who have helped us through out our study career.

A dissertation submitted to the

Department of Computer Science

Faculty of Applied Sciences

International Islamic University, Islamabad

as a partial fulfillment of the requirements

for the award of the degree of

Master of Science

Declaration

We hereby declare that this software, neither as a whole nor as a part thereof has been copied out from any source. It is further declared that we have developed this software entirely on the basis of our personal efforts made under the sincere guidance of our teachers. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

Waqas Paracha
166-CS/MS/03

Malik Nasir Mahmood
48-CS/MS/01

Acknowledgements

All praise to the Almighty Allah, the most Merciful, the most Gracious, without whose help and blessings, we would have been entirely unable to complete the project.

Thanks to our Parents who helped us during most difficult times and it is due to their unexplainable care and love that we are at this position today.

Thanks to our project supervisor **Dr. S. Tauseef-ur-Rehman,** whose sincere efforts helped us to complete this project successfully?

Acknowledgement is also due to our families for their help in this project.

Waqas Paracha

Malik Nasir Mahmood

Project in Brief

Project Title:	CONGESTION CONTROLLED ADAPTIVE MULTICAST IN SATELLITE BASED NETWORKS
Objective:	To design and implement an efficient and congestion controlled protocol for adaptive multicast for Satellite based networks.
Undertaken By:	**Waqas Paracha** **Malik Nasir Mahmood**
Supervised By:	**Dr. Tauseef-ur-Rehman** Head, Department of Telecommunication Engineering & Computer Engineering International Islamic University, Islamabad.
Technologies Used:	Ns-2.26, Tcl, C++, Kate editor
System Used:	Pentium® IV
Operating System Used:	Redhat Linux 9.0
Date Started:	16th February, 2004
Date Completed:	13rd September, 2004

Abstract

The use of contention-based MAC protocols combined with hidden terminal problems make multi-hop wireless ad hoc networks much more sensitive to load and congestion than wired networks or even wireless cellular networks. In such an environment, we argue that multicast reliability cannot be achieved solely by retransmission of lost packets as is typically done in wired networks with protocols such as SRM. We contend that in order to achieve reliable multicast delivery in such networks, besides reliability mechanisms, we must also consider jointly two components: reliability and congestion control. In this project, we propose a congestion controlled adaptive multicast transport protocol and show that congestion control alone can significantly improve reliable packet delivery in ad hoc networks when compared to traditional "wired" reliable multicast protocols alone do not lead to reliability, as is the case in wired networks. To achieve complete reliability, two complementary steps are required: congestion control and reliable delivery. In this project, we focus on the congestion control facet.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

Multicast routing in wireless ad hoc networks has gain considerable interest in recent years. The main benefit of multicasting is the significant reduction of network load gained when packets need to be transmitted to a group of nodes. Congestion control (at the network level) is vital in multicast since the most important form of congestion control in the Internet. Moreover, overload control is essential in wireless networks where scarce bandwidth is the norm.

Unicast congestion control performs rate regulation at the source side in order to track the network bandwidth available to the unique receiver. In a multicast environment, the capacities of the different receivers are heterogeneous, so that several capacities can now possibly be used as a target to perform rate regulation. Congestion avoidance in the whole multicast tree requires that the source adapts the flow rate to the lowest capacity receiver, thus penalizing all the other receivers that might take advantage of a higher transmission rate.

1.1 Congestion Control

A congestion control system typically monitors various factors like CPU occupancy, link occupancy and messaging delay. Based on these factors it takes a decision if the system is overloaded. If the system is overloaded, it initiates actions to reduce the load by asking front end processors to reject traffic. The throttling of traffic will reduce the load but it there will be a certain time delay before which the monitored variables like CPU and Link occupancy show downward trend. Congestion control systems are designed to take this into account by spacing out congestion control actions. If the system continues to be overloaded, subsequent congestion control actions can further increase the traffic throttling. If the traffic load is just right, the system maintains current traffic throttling actions. If the system gets under loaded, the traffic throttling is reduced.

1.1.1 Congestion Control Basics

A system is said to be congested if it is being offered more traffic than its rated capacity. Most of the time, the system overload is due to too many active users. System

maintenance and repair actions can also lead to system congestion. Whatever be the cause of overload, it will manifest as depletion of resources that are critical to the operation of the system. These resources can be CPU, free buffers, link bandwidth etc. Resource crunch will lead to lengthening of various queues for these resources. Due to lengthening of queues the response time of the system to external events will increase beyond permissible limits. . Resource crunch will lead to lengthening of various queues for these resources. Due to lengthening of queues the response time of the system to external events will increase beyond permissible limits. For example, in Xeon system overload will lead to increase in dial tone delay.

1.1.2 Impact of Overload on System Performance

Increase in response time will lead to application level timeouts. This will further worsen the situation because applications will needlessly resend messages on timeouts, causing further congestion. If this condition continues the system might reach a condition where it can service no users. Thus in absence of any congestion control, the system will perform much below its rated capacity, leave alone handling the excess traffic. Refer to the figures below for a comparison of systems load handling capability with and without congestion control. (The system load is represented in Busy Hour Call Attempts, i.e. BHCA. The system has a rated capacity of 5000 BHCA).

1.1.3 Congestion Control Mechanism

The main objective of congestion control is to keep the system running pretty close to its rated capacity, even when faced with extreme overload. This is achieved by restricting users that are allowed service. The idea is to give satisfactory service to a small percentage of users rather than give highly degraded service to all the users. The users that were given service will leave the system after completion of service. This will reduce the load on the system. Now a different set of users can be given service. Thus in a phased manner, all users will get some service from the system. An important requirement for the above mentioned scheme to work is that user blocking be done by the system should not overload main processors in the system. This is achieved by asking front end processors in the system to block the excess traffic. Thus the main processors do not even see the rejected traffic. They have to work only on the traffic that has been

accepted, thus the main processors will be able to handle traffic pretty close to the rated capacity. See graphs below

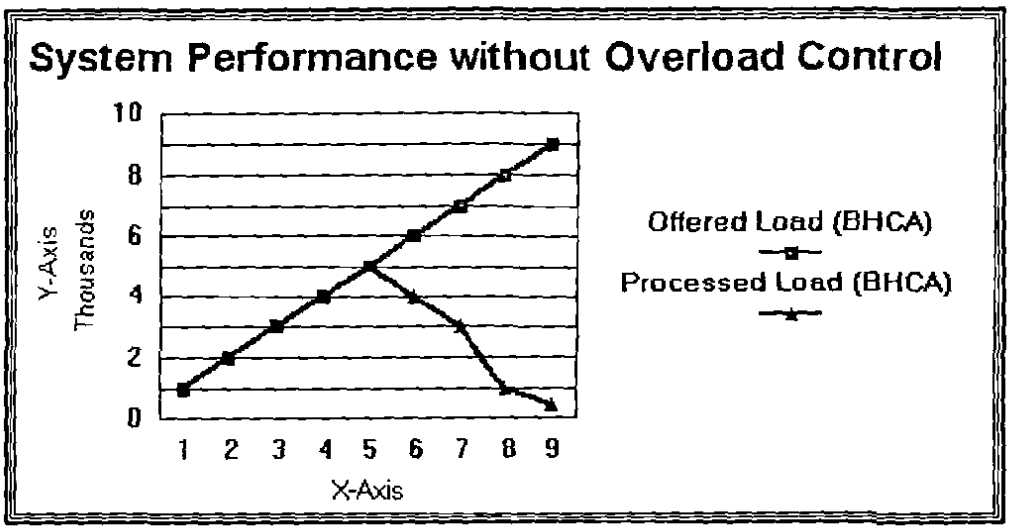


FIG 1: SYSTEM PERFORMANCE WITHOUT OVERLOAD CONTROL

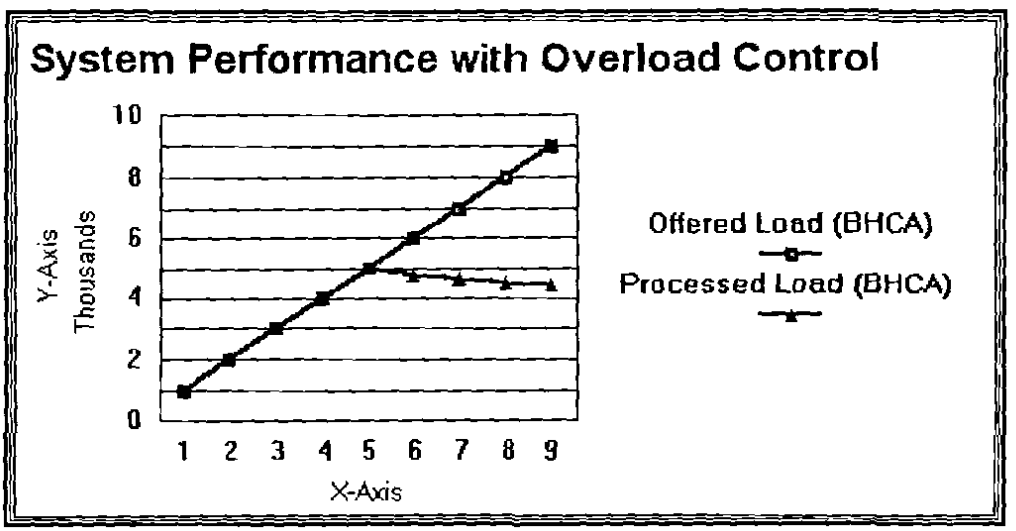


FIG 2: SYSTEM PERFORMANCE WITH OVERLOAD CONTROL

1.1.4 Congestion Control Triggers

The following types of congestion triggers are covered:

- CPU Occupancy Triggers
- Link Utilization Triggers

- Queue Length Triggers
- Protocol Congestion Triggers
- Resource Occupancy Triggers

1.1.4.1 CPU Occupancy Triggers

Most Real-time systems should be designed to operate with average CPU occupancy of below 40%. This leaves room for addition of more features that might require more CPU time than the current software. This also leaves headroom for sudden increase in CPU requirement due to fault handling and maintenance actions. Increase in CPU utilization is a very strong indicator of increasing load on the system. Systems should be designed to report congestion triggers when CPU utilization over a few minute interval exceeds 60 to 70%. The CPU overload abatement trigger should be reported when the CPU utilization falls below 50%. As you can see we need to have a good mechanism for measuring CPU utilization. Unfortunately, measuring CPU utilization can only be approximate. We discuss two algorithms commonly used for CPU utilization measurement:

- CPU Occupancy Sampling
- Lowest Priority Task Scheduling

CPU Occupancy Sampling

This technique uses the periodic timer interrupt to obtain an estimate of CPU utilization. Typically the timer interrupt is executed with a 10 ms periodicity. Whenever the interrupt is scheduled, it increments a "CPU active" counter whenever it finds that a task is active at that instant. A "CPU inactive" counter is incremented when no task is found to be active at that time. An estimate of CPU occupancy can be obtained by using the values of these counters. This technique only gives a very rough estimate of the CPU utilization. For most systems this technique suffices. However, the biggest drawback of this algorithm is that it does not work when some task is being scheduled periodically.

Lowest Priority Task Scheduling

This technique works by designing a task that is supposed to run at the lowest possible priority in the system. This task is designed to be always ready to run. Since all useful application tasks have a priority higher than this task, the operating system will

schedule this task only when it finds that no other task is in active state. A good estimate of CPU utilization can be obtained by keeping track of the time between successive scheduling of this task, In most cases this technique works better than the sampling based technique. However this algorithm does not take into account the CPU time spent in interrupt handling. Thus adjustment factors need to be applied if a system spends a lot of time executing interrupt service routines.

1.1.4.2 Link Utilization Triggers

Link Utilization needs to be carefully monitored when the link bandwidth is limited. If the utilization of links exceeds a 50-60%, a link congestion trigger should be reported. Link congestion abatement condition should be reported when link congestion goes below 30-40%. Link utilization can be calculated by keeping track of number of bytes of data transmitted in a given period. Dividing this number by the maximum number of bytes that could have been transmitted gives an idea of link occupancy. For example, consider a link with a capacity of 80 Kbps (i.e. 10 K Bytes per second). If during a 10 second interval 25 KB data is transmitted, this represents 25% link occupancy as a total of 100 KB data can be transmitted if this link is fully loaded.

1.1.4.3 Queue Length Triggers

When a system encounters congestion for a particular resource, the queues for the entities waiting for the resource will increase in length as more and more entities have to wait for service of the resource. This applies universally to all types of resources. For example, consider the following queuing scenarios:

- When the link occupancy is high, transmit protocol queues will be backlogged.
- When the CPU occupancy is very high, the ready to run queue maintained by the operating system will increase in length.
- Also high CPU occupancy can lead to backlogging of receive protocol queues if the task involved in protocol processing is not getting scheduled.

Backlogged queues lead to delays. Queuing Theory provides a very simple equation describing the relationship between resource occupancy and the delay involved in obtaining that resource.

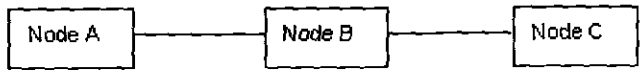
$$T = \frac{\rho}{\lambda(1-\rho)}$$

Here T is the average delay for the resource, the arrival rate is given by lambda and occupancy is given by p. As occupancy approaches 1 (i.e. 100%), the denominator approaches 0, thus the total delay experienced will race towards infinity.

The above equation underscores the importance of generating congestion control triggers in time. Whenever the queue length exceeds a threshold, a congestion trigger should be immediately generated. This will give the congestion control software sometime to initiate action and bring the resource occupancy down. If the congestion trigger is delayed, the system might reach pretty close to 100 % resource occupancy, thus leading to terrible service to all users of the system.

1.1.4.4 Protocol Congestion Triggers

Many protocols like SS7 have inbuilt support for congestion control. Whenever a congestion condition is detected, these protocols generate a congestion indication. This indication can be passed to congestion control software for further action. Protocol congestion triggers can help in detecting end-to-end congestion in the system.



When a protocol congestion condition is detected, congestion control software will have to further isolate the exact cause of congestion. Sometimes the congestion control software will have to work in tandem with the fault handling software to isolate the real cause of congestion. (Keep in mind that a link with high bit error rate will also lead to protocol congestion) Consider a case where node A and C are communicating via a node B. If the protocol handler at A generates a protocol congestion trigger towards C, the congestion could be present at any of the following places:

- Node A CPU is overloaded and is not processing packets received from Node B.
- Link between Node A and Node B is congested.
- Node B CPU is overloaded and it is delaying routing of packets between A and C.
- Link between Node B and C is congested.
- Node C CPU is overloaded and is not processing packets from Node B.

You can see from the above description that isolating the protocol congestion trigger can be daunting task even in a simple network shown above. Imagine the complexity of isolating congestion source in a big network with hundreds of nodes.

1.1.4.5 Buffer Occupancy Triggers

Just like any other resource, a heavily loaded system may run out of buffer space. This can happen due to memory leaks in the software. In a stable system, buffer congestion might be reported when it faces congestion. This happens because a large number of buffers are queued up waiting for resources.

Just like all other congestion triggers, an onset and abatement level is defined for buffer congestion. When the number of free buffers depletes below a certain threshold a congestion trigger is reported to the congestion control handler.

1.1.4.6 Resource Occupancy Triggers

We have covered basic computing resources like CPU, link bandwidth and memory buffers. Most embedded and real-time systems deal will other resources like timeslots, DSP processors, terminals, frequencies etc. Congestion triggers might be generated when the system is running short on any of these resources. For example, a switching system might be designed to report a congestion condition when it is running low on number of free timeslots.

1.2 Multicast Communication

Multicast communications is certainly an efficient mean of supporting group-oriented applications. In such an environment we show that error control mechanisms alone do not lead to reliability as in the case in wired networks. In this type of communication data is transmitted only to selected members/groups from a no. of receiving groups.

1.2.1 Adaptive Multicast

In adaptive multicast the multicast transmission changes with the changes in the network's condition i.e. Network congestion occurs.

1.2.2 Reliable Multicast

Reliable multicast for wired networks has been a very active area of research. One may consider applying them to MANETs (Mobile Ad hoc Networks). We argue that the

design choices underlying wired reliable multicast transport protocols are not adequate for MANETs environments. Ad hoc network protocol must handle node mobility and are extremely sensitive to network load and congestion because of hidden terminal problem. While this protocol may work well in stable network with low mobility and low failure rate, its performance will likely degrade dynamic MANETs where topology changes are frequent.

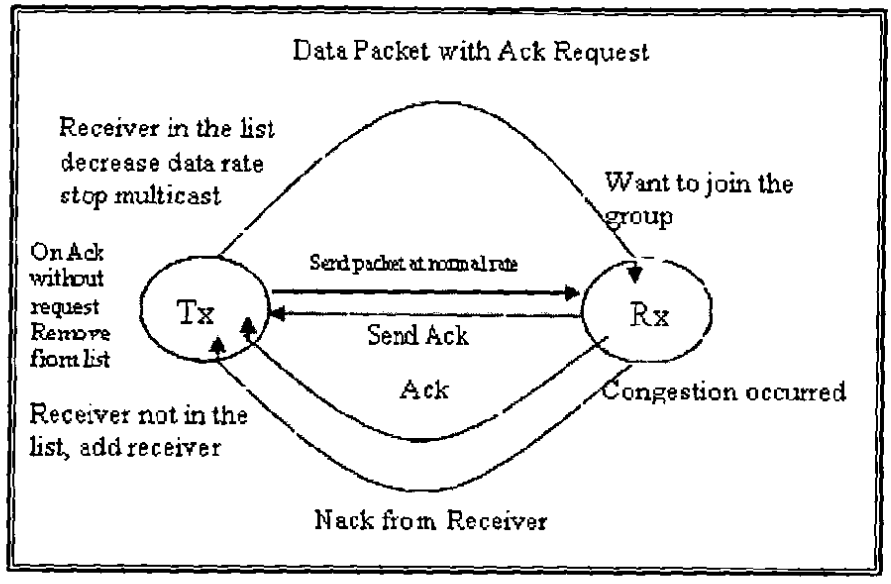


FIG 3: STATE TRANSITION DIAGRAM

1.3 Mobile Ad hoc Networks

Recent advantages in portable computing devices and wireless technology have revamped the concept of mobile packet radio networks into what is called MOBILE ADHOC NETWORKS.

A mobile ad hoc network is defined as a multi-hop wireless network where all network components are capable of movements.

The technology of Mobile Ad hoc Networking is somewhat synonymous with Mobile Packet Radio Networking (a term coined via during early military research in the 70's and 80's), Mobile Mesh Networking (a term that appeared in an article in The Economist regarding the structure of future military networks) and Mobile, Multicast Wireless Networking.

There is current and future need for dynamic Ad hoc networking technology. The emerging field of mobile and nomadic computing, with its current emphasis on mobile IP operation, should gradually broaden and require highly-adaptive mobile networking technology to effectively manage Multicast, Ad hoc network clusters that can operate autonomously or, more than likely, be attached at some point(s) to the fixed Internet.

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

The most important part in developing any research project is Literature review. In order to understand or develop any research project we should have deep knowledge about it. We should know what research has already been done, what comments have been made by the pioneers of this field about this project and whether it is feasible to start work on it or not. For example to start a work on a network protocol, extensive literature survey about routing mechanisms and multicast communication, is needed. We should have the knowledge about the following things.

2.1 Computer Networks

With the invention of computers, new paradigms in computing begin. Computers were used to solve different problems and perform tasks for their users but soon it became obvious that a single computer, no matter how fast it was, needed to interact with other computers for different reasons [1]. Therefore computers were connected with each other through different media and at different speeds. Different protocols were written to support these interconnections.

2.1.1 Wireless Networks

Wireless networks utilize radio waves and/or microwaves to maintain communication channels between computers. Wireless networking is a more modern alternative to wired networking that relies on copper and/or fiber optic cabling between network devices.

A wireless network offers advantages and disadvantages compared to a wired network. Advantages of wireless include mobility and elimination of unsightly cables. Disadvantages of wireless include the potential for radio interference due to weather, other wireless devices, or obstructions like walls.

Wireless is rapidly gaining in popularity for both home and business networking. Wireless technology continues to improve, and the cost of wireless products continues to decrease. Popular wireless local area networking (WLAN) products conform to the 802.11 "Wi-Fi" standards.

2.1.1.1 802.11 Standards

Wireless networking hardware requires the use of underlying technology that deals with radio frequencies as well as data transmission. The most widely used standard is 802.11 produced by the Institute of Electrical and Electronic Engineers (IEEE). This is a standard defining all aspects of Radio Frequency Wireless networking.

2.2 Routing Protocols

Routing protocols are the mechanisms by which different entities in a node communicate with each other. Routing protocols handle the communication of the nodes with each other. Routing protocols provide different services to the nodes. Once two nodes have routing protocols running on them, which are interoperable with each other then they can communicate easily with each other. The design of routing protocols is not easy. Many factors have to be taken care of to design a routing protocol.

2.2.1 Classification of Routing Protocols

Routing protocols can be classified [1][2] into categories depending on their properties.

- Centralized vs. Distributed.
- Static vs. Adaptive
- Reactive vs. Proactive

2.2.1.1 Centralized vs. Distributed

One way to categorize the routing protocols is to divide them into centralized and distributed algorithms. In centralized algorithms, all route choices are made at a central node, while in distributed algorithms, the computation of routes is shared among the network nodes.

2.2.1.2 Static vs. Adaptive

Another classification of routing protocols relates to whether they change routes in response to the traffic input patterns. In static algorithms, the route used by source-destination pairs is fixed regardless of traffic conditions. It can only change in response to a node or a link failure. This type of algorithm cannot achieve high throughput under a broad variety of traffic input patterns as explained in [1][2]. Most major packet networks

use some form of adaptive routing where routes used to route between source-destination pairs may change in response to congestion [1][2].

2.2.1.3 Proactive vs. Reactive

A third classification that is more related to ad hoc networks is to classify the routing algorithms as either proactive or reactive. Proactive protocols attempt to continuously evaluate the routes within the network, so that when a packet needs to be forwarded, the route is already known and can be used immediately. The family of Distance-Vector protocols is an example of a proactive scheme. Reactive protocols, on the other hand, invoke a route determination procedure on demand only. If a route is unknown, the source node initiates a search to find one, which tends to cause a traffic surge as the query is propagated through the network. Nodes that receive the query and have a route to the requested destination respond to the query. In general, reactive protocols are primarily interested in finding any route to a destination, not necessarily the optimal route. Data sent in networks using reactive protocols do tend to suffer a delay during the search for a route. Under highly dynamic link conditions, reactive protocols are expected to generate less overhead and provide more reliable routing than proactive routing, but at the cost of finding the optimal route.

The family of classical flooding algorithms belongs to the reactive group. Proactive schemes have the advantage that when a route is needed, the delay before actual packets can be sent is very small. On the other hand, proactive schemes need time to converge to a steady state. This can cause problems if the topology is changing frequently.

2.3 Types of Mobile Networking Routing Agents

The Wireless Network essentially consists of the Mobile Node at the core, with additional supporting features that allows simulations of multi-hop ad-hoc networks, wireless LANs etc. The Mobile Node object is a split object. A Mobile Node thus is the basic Node object with added functionalities of a wireless and mobile node like ability to move within a given topology, ability to receive and transmit signals to and from a wireless channel etc. A major difference between them, though, is that a Mobile Node is not connected by means of Links to other nodes or Mobile nodes. The four different Ad

hoc currently implemented for mobile networking are **DSDV, AODV, TORA** and **DSR** brief description of each is given below[3].

2.3.1 Destination Sequenced Distance Vector Protocol (DSDV)

In this routing protocol routing messages are exchanged between neighboring mobile nodes (i.e. mobile nodes that are within range of one another). Routing updates may be triggered or routine. Updates are triggered in case a routing information from one of the neighbors forces a change in the routing table. A packet for which the route to its destination is not known is cached while routing queries are sent out. The pkts are cached until route-replies are received from the destination. There is a maximum buffer size for caching the pkts waiting for routing information beyond which pkts are dropped. All packets destined for the mobile node are routed directly by the address dmux to its port dmux. The port dmux hands the packets to the respective destination agents. A port number of 255 is used to attach routing agent in mobile nodes. The mobile nodes also use a default-target in their classifier (or address demux). In the event a target is not found for the destination in the classifier (which happens when the destination of the packet is not the mobile node itself), the pkts are handed to the default-target which is the routing agent. The routing agent assigns the next hop for the packet and sends it down to the link layer.

2.3.2 On-Demand Distance Vector Routing (AODV)

AODV is a combination of both DSR and DSDV protocols. It has the basic route-discovery and route-maintenance of DSR and uses the hop-by-hop routing, sequence numbers and beacons of DSDV. The node that wants to know a route to a given destination generates a ROUTE REQUEST. The route request is forwarded by intermediate nodes that also create a reverse route for itself from the destination. When the request reaches a node with route to destination it generates a ROUTE REPLY containing the number of hops requires to reach destination. All nodes that participate in forwarding this reply to the source node creates a forward route to destination. This state created from each node from source to destination is a hop-by-hop state and not the entire route as is done in source routing.

2.3.2 Temporally-Ordered Routing Algorithm (TORA)

TORA is a distributed routing protocol based on "link reversal" algorithm. At every node a separate copy of TORA is run for every destination. When a node needs a route to a given destination it broadcasts a QUERY message containing the address of the destination for which it requires a route. This packet travels through the network until it reaches the destination or an intermediate node that has a route to the destination node. This recipient node then broadcasts an UPDATE packet listing its height w.r.t the destination. As this node propagates through the network each node updates its height to a value greater than the height of the neighbor from which it receives the UPDATE. This results in a series of directed links from the node that originated the QUERY to the destination node. If a node discovers a particular destination to be unreachable it sets a local maximum value of height for that destination. Incase the node cannot find any neighbor having finite height w.r.t. this destination it attempts to find a new route. In case of network partition, the node broadcasts a CLEAR message that resets all routing states and removes invalid routes from the network. TORA operates on top of IMEP (Internet MANET Encapsulation Protocol) that provides reliable delivery of route-messages and informs the routing protocol of any changes of the links to its neighbors. IMEP tries to aggregate IMEP and TORA messages into a single packet (called block) in order to reduce overhead. For link-status sensing and maintaining a list of neighbor nodes, IMEP sends out periodic BEACON messages which are answered by each node that hears it by a HELLO reply message.

2.3.2 Dynamic Source Routing (DSR)

The DSR agent checks every data packet for source-route information. It forwards the packet as per the routing information. Incase it does not find routing information in the packet, it provides the source route, if route is known, or caches the packet and sends out route queries if route to destination is not known. Routing queries, always triggered by a data packet with no route to its destination, are initially broadcast to all neighbors. Route-replies are send back either by intermediate nodes or the destination node, to the source, if it can find routing info for the destination in the route-query. It hands over all

packets destined to itself to the port dmux. In SRNode the port number 255 points to a null agent since the packet has already been processed by the routing agent.

2.4 Networking Components in a Mobile Node

The network stack for a mobile node consists of a link layer(LL), an ARP module connected to LL, an interface priority queue(IFq), a MAC layer(MAC), a network interface(netIF), all connected to the channel. These network components are created and plumbed together in OTcl. The relevant MobileNode method add-interface() is shown below:

```
#
# The following setups up link layer, MAC layer, network interface
# and physical layer structures for the mobile node.
#
Node/MobileNode instproc add-interface { channel pmodel
lltype mactype qtype qlen iftype anttype } {
$self instvar arptable_ nifs_
$self instvar netif_ mac_ ifq_ ll_
global ns_ MacTrace opt
set t $nifs_
incr nifs_
set netif_($t) [new $iftype] ;# net-interface
set mac_($t) [new $mactype] ;# mac layer
set ifq_($t) [new $qtype] ;# interface queue
set ll_($t) [new $lltype] ;# link layer
set ant_($t) [new $anttype]
#
# Local Variables
#
set nullAgent_ [$ns_ set nullAgent_]
set netif $netif_($t)
set mac $mac_($t)
set ifq $ifq_($t)
set ll $ll_($t)
```

```
#
# Initialize ARP table only once.
#
if { $arptable_ == "" } {
set arptable_ [new ARPTable $self $mac]
set drpT [cmu-trace Drop "IFQ" $self]
$arptable_ drop-target $drpT
}
#
# Link Layer
#
$ll arptable $arptable_
$ll mac $mac
$ll up-target [$self entry]
$ll down-target $ifq
#
# Interface Queue
#
$ifq target $mac
$ifq set qlim_ $qlen
set drpT [cmu-trace Drop "IFQ" $self]
$ifq drop-target $drpT
#
# Mac Layer
#
$mac netif $netif
$mac up-target $ll
$mac down-target $netif
$mac nodes $opt(nn)
#
# Network Interface
#
$netif channel $channel
```

```
$netif up-target $mac
$netif propagation $pmodel ;# Propagation Model
$netif node $self ;# Bind node <---> interface
$netif antenna $ant_($t) ;# attach antenna
#
# Physical Channel
#
$channel addif $netif ;# add to list of interfaces
# ============================================================
# Setting up trace objects
if { $MacTrace == "ON" } {
#
# Trace RTS/CTS/ACK Packets
#
set rcvT [cmu-trace Recv "MAC" $self]
$mac log-target $rcvT
#
# Trace Sent Packets
#
set sndT [cmu-trace Send "MAC" $self]
$sndT target [$mac sendtarget]
$mac sendtarget $sndT
#
# Trace Received Packets
#
set rcvT [cmu-trace Recv "MAC" $self]
$rcvT target [$mac recvtarget]
$mac recvtarget $rcvT
#
# Trace Dropped Packets
#
set drpT [cmu-trace Drop "MAC" $self]
$mac drop-target $drpT
```

```
} else {
$mac log-target [$ns_ set nullAgent_]
$mac drop-target [$ns_ set nullAgent_]
}
# ==========================================================
$self addif $netif
}
```

Each component is briefly described here

2.4.1 Link Layer

The LL used by mobile node has an ARP module connected to it which resolves all IP to hardware (Mac) address conversions. Normally for all outgoing (into the channel) packets, the packets are handed down to the LL by the Routing Agent. The LL hands down packets to the interface queue. For all incoming packets (out of the channel), the MAC layer hands up packets to the LL which is then handed off at the node_entry_ point [3].

2.4.2 ARP

The Address Resolution Protocol (implemented in BSD style) module receives queries from Link layer. If ARP has the hardware address for destination, it writes it into the MAC header of the packet. Otherwise it broadcasts an ARP query, and caches the packet temporarily. For each unknown destination hardware address, there is a buffer for a single packet. Incase additional packets to the same destination is sent to ARP, the earlier buffered packet is dropped. Once the hardware address of a packet's next hop is known, the packet is inserted into the interface queue.

2.4.3 Interface Queue

The class PriQueue is implemented as a priority queue which gives priority to routing protocol packets, inserting them at the head of the queue. It supports running a filter over all packets in the queue and removes those with a specified destination address.

2.4.4 Mac Layer

The IEEE 802.11 distributed coordination function (DCF) Mac protocol has been implemented by CMU. It uses a RTS/CTS/DATA/ACK pattern for all unicast packets and simply sends out DATA for all broadcast packets. The implementation uses both physical and virtual carrier sense.

2.4.5 Tap Agents

Agents that subclass themselves as class Tap can register themselves with the MAC object using method installTap(). If the particular Mac protocol permits it, the tap will promiscuously be given all packets received by the MAC layer, before address filtering is done.

2.4.6 Network Interfaces

The Network Interphase layer serves as a hardware interface which is used by mobile node to access the channel. The wireless shared media interface is implemented as class Phy/WirelessPhy. This interface subject to collisions and the radio propagation model receives packets transmitted by other node interfaces to the channel. The interface stamps each transmitted packet with the meta-data related to the transmitting interface like the transmission power, wavelength etc. This meta-data in pkt header is used by the propagation model in receiving network interface to determine if the packet has minimum power to be received and/or captured and/or detected (carrier sense) by the receiving node. The model approximates the DSSS radio interface (LucentWaveLan direct-sequence spread-spectrum).

2.4.7 Radio Propagation Model

It uses Friss-space attenuation (1/r2) at near distances and an approximation to Two Ray Ground (1/r4) at far distances. The approximation assumes specular reflection off a flat ground plane.

2.4.8 Antenna

An omni-directional antenna having unity gain is used by mobile nodes.

2.5 Generation of node-movement and traffic-connection for wireless scenarios

Normally for large topologies, the node movement and traffic connection patterns are defined in separate files for convenience. These movement and traffic files may be generated using CMU's movement- and connection-generators. In this section we shall describe both separately.

2.5.1 Mobile Node Movement

Each node is assigned a starting position. The information regarding number of hops between the nodes is fed to the central object "GOD". Next each node is a speed and a direction to move to. Setdest arguments are in the following manner.

./setdest -n <num_of_nodes> -p <pausetime> -s <maxspeed> -t <simtime>
-x <maxx> -y <maxy> > <outdir>/<scenario-file>

2.5.2 Mobile IP

The Mobile IP scenario consists of Home-Agents(HA) and Foreign-Agents(FA) and have Mobile-Hosts(MH) moving between their HA and FAs [3].

The HA and FA nodes are defined as MobileNode/MIPBS having a registering agent (regagent_) that sends beacon out to the mobile nodes, sets up encapsulator and decapsulator, as required and replies to solicitations from MHs. The MH nodes are defined as MobileNode/MIPMH which too have a regagent_ that receives and responds to beacons and sends out solicitations to HA or FAs. ***Fig 2*** illustrates the schematic of a MobileNode/MIPBS node. The MobileNode/MIPMH node is very similar to this except for the fact that it does not have any encapsulator or decapsulator. As for the SRNode version of a MH, it does not have the hierarchical classifiers and the RA agent forms the entry point of the node. See ***Fig 3*** for model of a SRNode.

The MobileNode/MIPBS node routinely broadcasts beacon or advertisement messages out to MHs. A solicitation from a mobile node generates an ad that is send directly to the requesting MH. The address of the base-station sending out beacon is heard by MH and is used as the COA (care-of-address) of the MH. Thus as the MH

moves from its native to foreign domains, its COA changes. Upon receiving reg_request (as reply to ads) from a mobile host the base-station checks to see if it is the HA for the MH. If not, it sets up its decapsulator and forwards the reg_request towards the HA of the MH. In case the base-station is the HA for the requesting MH but the COA does not match its own, it sets up an encapsulator and sends reg-request-reply back to the COA (address of the FA) who has forwarded the reg_request to it. So now all packets destined to the MH reaching the HA would be tunneled through the encapsulator which encapsulates the IP pkthdr with a IPinIP hdr, now destined to the COA instead of MH. The FA's decapsulator receives this packet, removes the encapsulation and sends it to the MH.

Source: The *ns* Manual

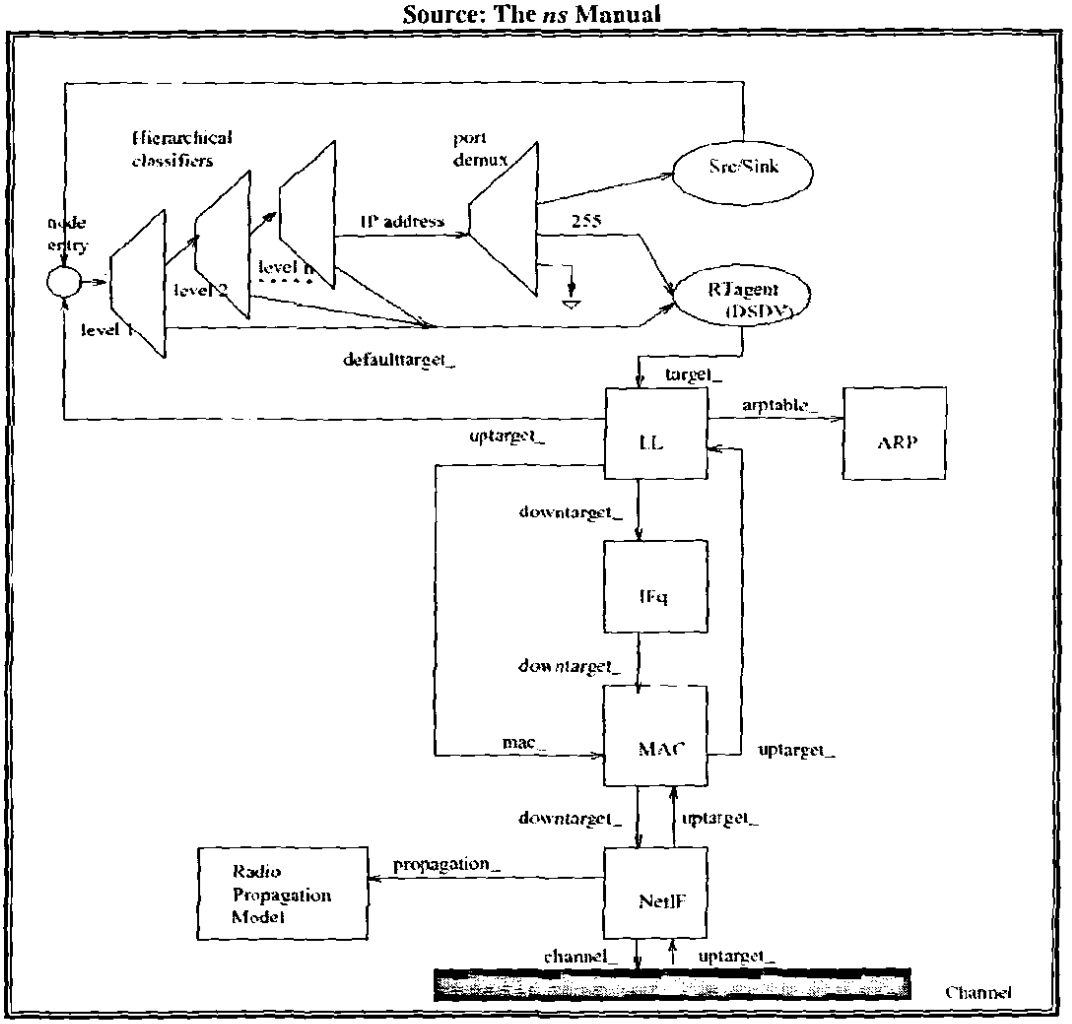


FIG 4: SCHEMATIC OF A BASE STATION NODE

Source: The *ns* Manual

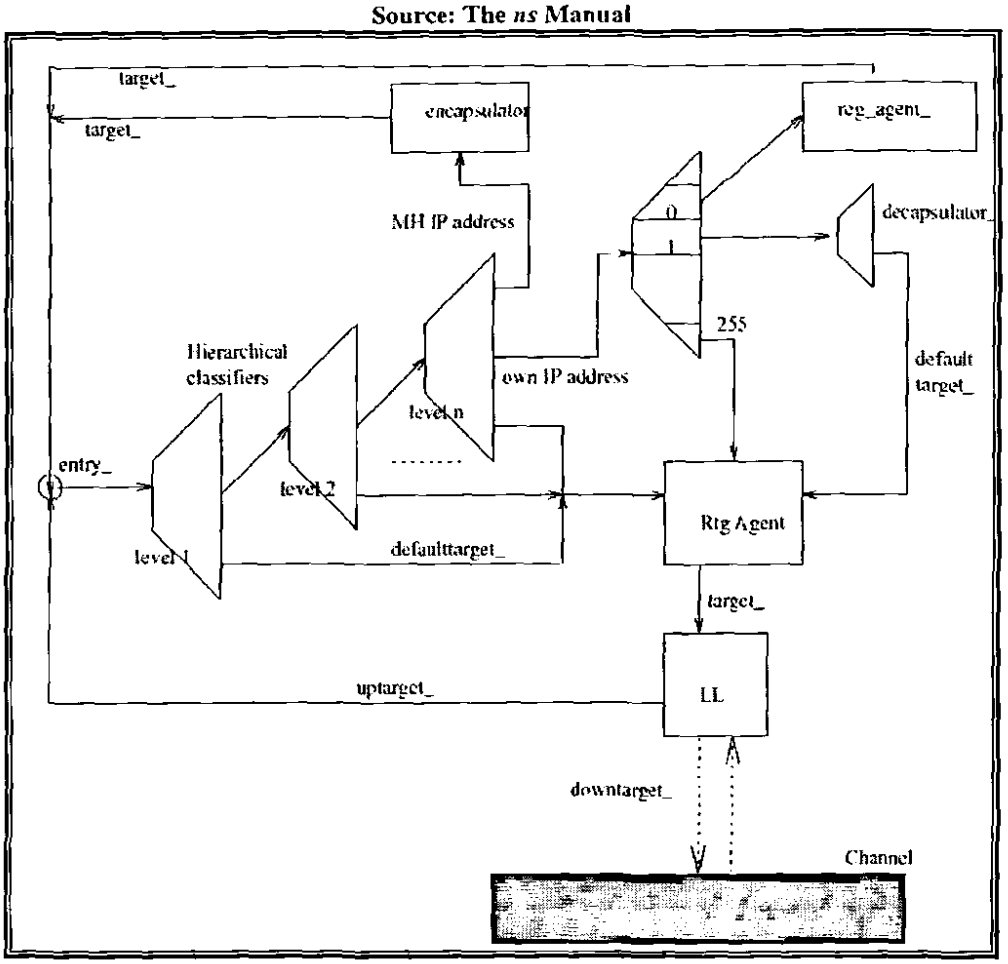


FIG 5: SCHEMATIC OF A WIRELESS MOBILEIP BASE STATION NODE

If the COA matches that of the HA, it just removes the encapsulator it might have set up (when its mobile host was roaming into foreign networks) and sends the reply directly back to the MH, as the MH have now returned to its native domain.

The mobile host sends out solicitations if it does not hear any ads from the base-stations. Upon receiving ads, it changes its COA to the address of the HA/FA it has heard the ad from, and replies back to the COA with a request for registration (reg-request). Initially the MH maybe in the range of the HA and receives all pkts directly from its COA which is HA in this case. Eventually as the MH moves out of range of its HA and into the a foreign domain of a FA, the MH's COA changes from its HA to that of the FA. The HA now sets up an encapsulator and tunnels all pkts destined for MH towards the

FA. The FA decapsulates the pkts and hands them over to the MH. The data from MH destined for the wired world is always routed towards its current COA.

CHAPTER 3

PROBLEM DOMAIN

3. PROBLEM DOMAIN

Reliable multicast transport protocols for wired networks have been a very active research area and the focus of the IETF Reliable Multicast Research Group (RMRG). We hypothesize that the design choices underlying wired reliable multicast transport protocols are not adequate for wireless ad hoc network environments. Ad hoc networks are extremely sensitive to network load and congestion, even more so than wired shared-medium networks, such as Ethernet, because of the hidden terminal problem. However, although reliable multicast is considered one of the fundamental technologies for enabling key applications in ad hoc networks, research in the area is minimal. A couple of exceptions are the Anonymous Gossip (AG) protocol and the work on reliable broadcast exhibits high recovery delay and will likely degrade in dynamic ad hoc network scenarios where topology changes are frequent. In fact, the earlier simulation results indicate that the protocol does not perform well in the presence of high node mobility. Moreover the congestion control likely to fail in the earlier techniques. Thus the two important features of reliable multicasting (Reliability and congestion control) are missing in existing system.

3.1 Project Scope

A simple rate-based reactive congestion control multicast protocol will be designed for wireless ad hoc networks. As a reactive protocol it will transmit data packets at a rate specified by the application traffic until congestion arises. Congestion will be indicated through negative acknowledgements (NACKs) sent back to the source by the multicast receivers that have not received a certain number of consecutive packets. Once the source is informed of congestion, it will enter the congestion control phase. The source would multicast new "targeted" data packets, with each new data packet instructing a specific receiver to reply with an acknowledgement (ACK). Packets will be multicasted to targeted receivers one at a time. The source then waits for an ACK from the targeted receiver or a timeout before moving to the next receiver. When all ACKs are received from the target receivers (an indication that the network is no longer congested), the source will exit the congestion control phase and will revert to the initial application

traffic rate. Thus, during the congestion control phase, the source will clock its sending rate based on the targeted receiver.

3.2 Objectives

The objective of this work is to control congestion in adaptive multicast with the following features:

3.2.1 Congestion Control

The Protocol will be able to control congestion on all nodes.

3.2.2 Quick Convergence

The Protocol will be able to converge quickly after changes in the network load and topology.

3.2.3 Low Processing

The Protocol will need low processing power for its operation.

3.2.4 Efficient Operation

The algorithm will work very efficiently.

3.2.5 Reduced Multicast Delivery Rate

The Protocol reduces Multicast delivery rate to what is acceptable to a set of slower, more congested receivers.

3.3 Proposed Solution

An important requirement for the above mentioned scheme to work is that user blocking be done by the system should not overload main processors in the system. This can be achieved by asking front end processors in the system to block the excess traffic. Thus the main processors do not even see the rejected traffic. They have to work only on the traffic that has been accepted, thus the main processors will be able to handle traffic pretty close to the rated capacity.

A simple rate-based reactive congestion control multicast protocol will be designed for wireless ad hoc networks. As a reactive protocol it will transmit data packets at a rate specified by the application traffic until congestion arises. Congestion will be indicated through negative acknowledgements (NACKs) sent back to the source by the multicast receivers that have not received a certain number of consecutive packets. Once the source is informed of congestion, it will enter the congestion control phase. The

source would multicast new "targeted" data packets, with each new data packet instructing a specific receiver to reply with an acknowledgement (ACK). Packets will be multicasted to targeted receivers one at a time. The source then waits for an ACK from the targeted receiver or a timeout before moving to the next receiver. When all ACKs are received from the target receivers (an indication that the network is no longer congested), the source will exit the congestion control phase and will revert to the initial application traffic rate. Thus, during the congestion control phase, the source will clock its sending rate based on the targeted receiver.

CHAPTER 4

DESIGNING

4. DESIGNING

Our Protocol is designed and the simulation environment chosen for it is the NS simulator. Now let us discuss the design of NS simulator.

4.1 Simulation Environment

NS is an object-oriented simulator. Developed at UC Berkeley it simulates different networks, implements network protocols such as TCP, UDP, traffic source behavior such as FTP, Telnet, CBR, routing algorithms such as Dijkstra and more. It also implements multicasting and some MAC layer protocols. It uses a unique split programming concept for providing its functionality. The two languages which ns uses are C++ and OTcl (an extension of the popular Tcl scripting language). Correspondingly there are two class hierarchies in ns.

- Compiled Hierarchy
- Interpreted Hierarchy

Both these hierarchies are closely related with each other. OTcl is the object oriented version of Tcl language and it has the same relation- ship with Tcl as C++ has with c.

Source: ***ns by examples***

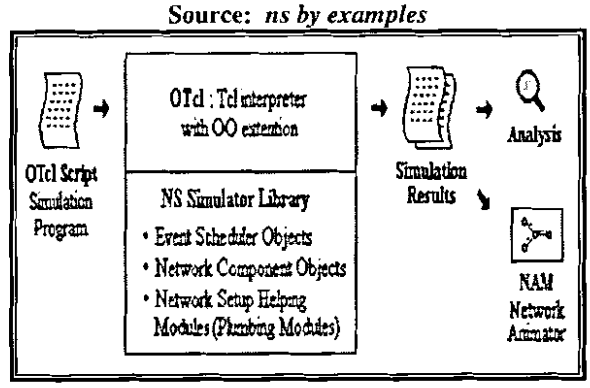


FIG 6: NS ARCHITECTURE

As shown in figure 5 contains an object oriented Tcl (OTcl) interpreter that has a simulation event scheduler and network component object and network setup libraries. NS is written in two languages C++ and OTcl. This is mainly done for efficiency reasons.

The event scheduler and the basic network objects are written and compiled in C++. Through a unique binding mechanism the compiled objects of C++ are made available to the OTcl interpreter.

Source: The *ns* Manual

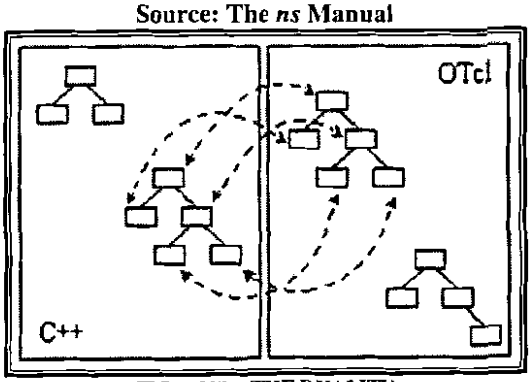


FIG 7: NS -- THE DUALITY

We can say that NS is a Object Oriented Tcl interpreter with network simulator libraries. NS has a well thought architecture. This was particularly demonstrated when the wireless/satellite support was added to NS seamlessly.

NS contains a graphical simulation display tool called NAM. NAM is used extensively to graphically visualize different simulations. It is a very capable tool and can present information such as throughput and number of packet drops at each link, although accurate simulation analysis cannot be done with that data.

Source: The *ns* Manual

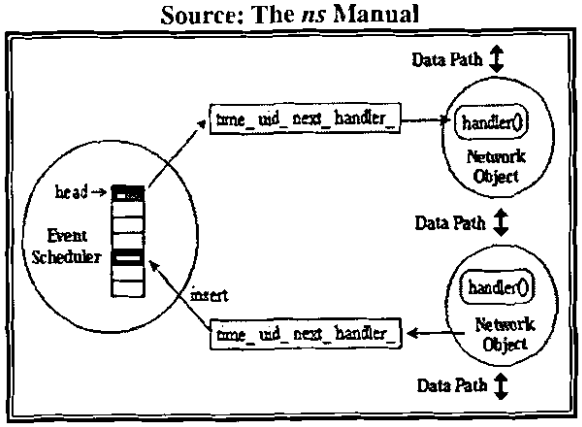


FIG 8: INTERNAL ARCHITECTURE

NS contains a discrete event scheduler. The main users of an event scheduler are network components that simulate packet-handling delay or that need timers. Above figure shows the NS event scheduler in action. Packets are send from one network object to another by using send (Packet* p) {target_->recv(p)}; for the sender and recv (Packet* , Handler* h = 0) for the receiver.

One very significant addition to the NS simulator is that of a real-time scheduler allowing NS to integrate in a real life LAN and introduce packets into it. This opens up some very exciting possibilities.

A simple NS simulation may consist of 2 nodes and a duplex link between them. A time interval may be set using a scheduler after which the nodes may start communication with each other. A stop time may also be specified. This whole simulation can be written in about 10 lines of OTCL script.

Source: ns Tutorial

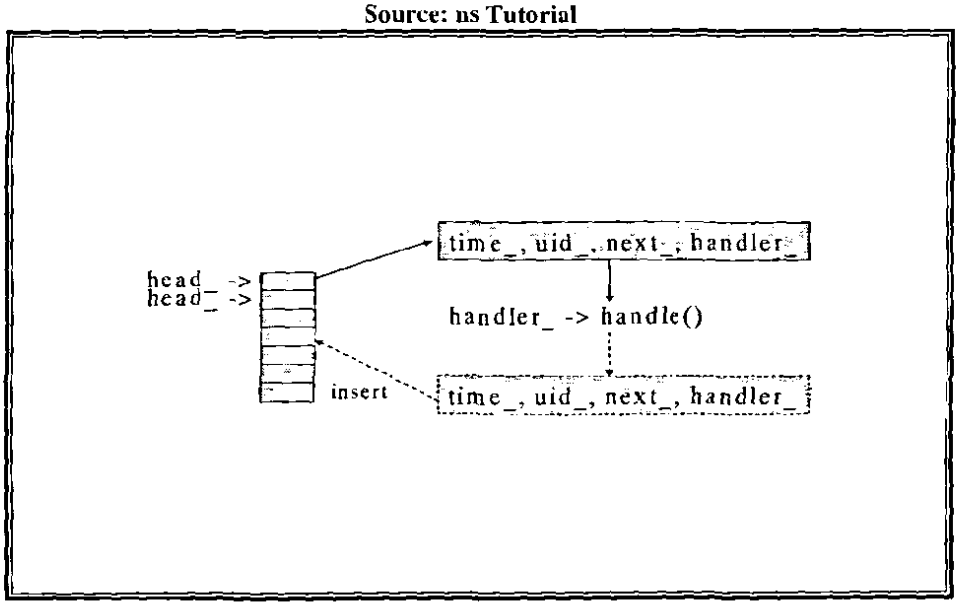


FIG 9: NS EVENT SCHEDULING

NS is a discrete event scheduler. It is not based on real-time. It has its own virtual time which is different from the virtual time. As soon as one event is executed, immediately the virtual time is advanced to next event time. The scheduler of NS is also explained in the above figure.

Source: The *ns* Manual

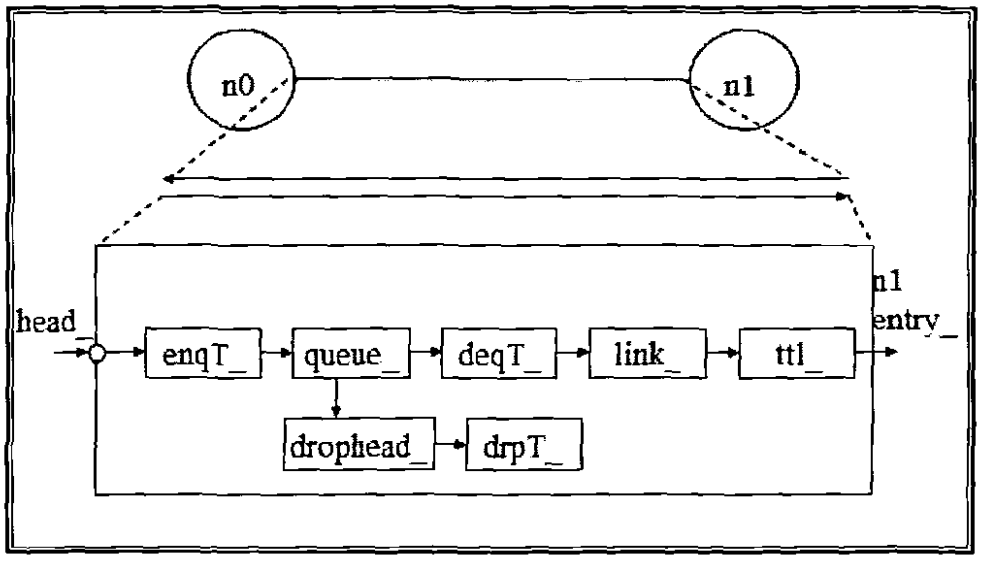


FIG 10: INTERNAL OBJECT STRUCTURE

As shown in figure, we can trace the queue with special trace objects. The queue is composed of several objects. The queue starts with the head_ object. At the other end of the queue the entry_ object of n1 object processes the packet.

Source: The *ns* Manual

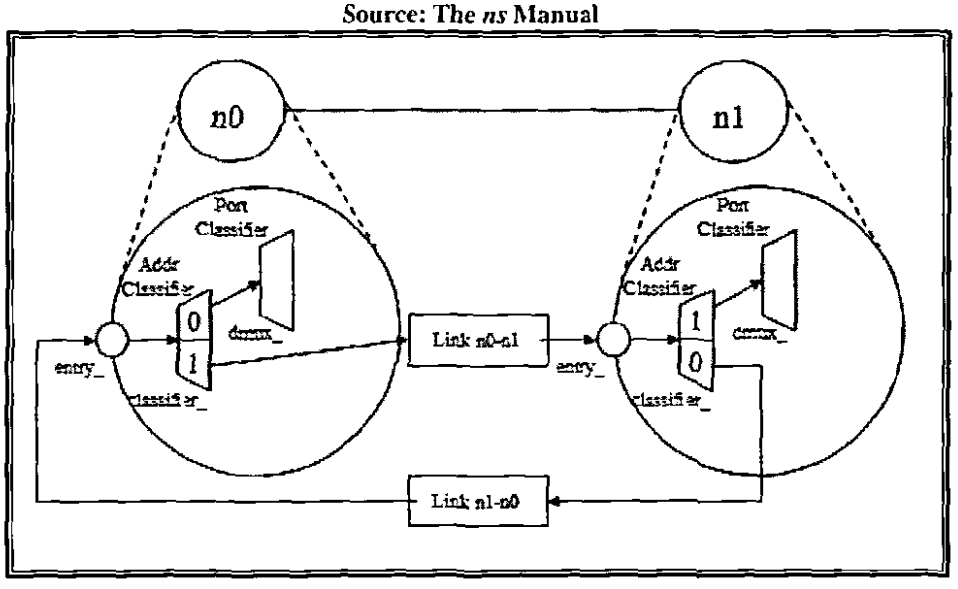


FIG 11: INTERNAL NODE ARCHITECTURE

As shown in above figure the architecture of the two Q nodes is described. The two nodes have address classifiers, port classifiers and link between them. The address classifier is used to check that whether the packet is intended for this Q node or not. If

yes then it sends the packet for further processing to the port classifier whose object name is dmux_. The dmux_ checks that to which port the packet may be send for further processing. The link structure between the two nodes is already described.

Source: The *ns* Manual

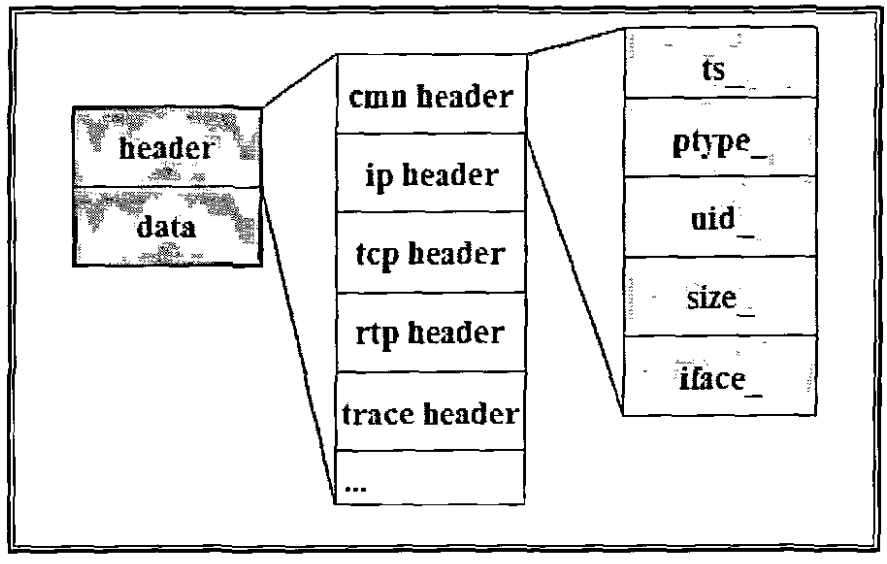


FIG 12: NS PACKET STRUCTURE

NS packet header is defined in the above picture. There is a common header which is necessary in every packet. It is followed by the IP header and TCP header which are the normal headers used in TCP/IP stack. Then rtp header and trace header come followed by our custom headers. These customer headers include the header we designed: the Q-Routing header.

Source: The *ns* Manual

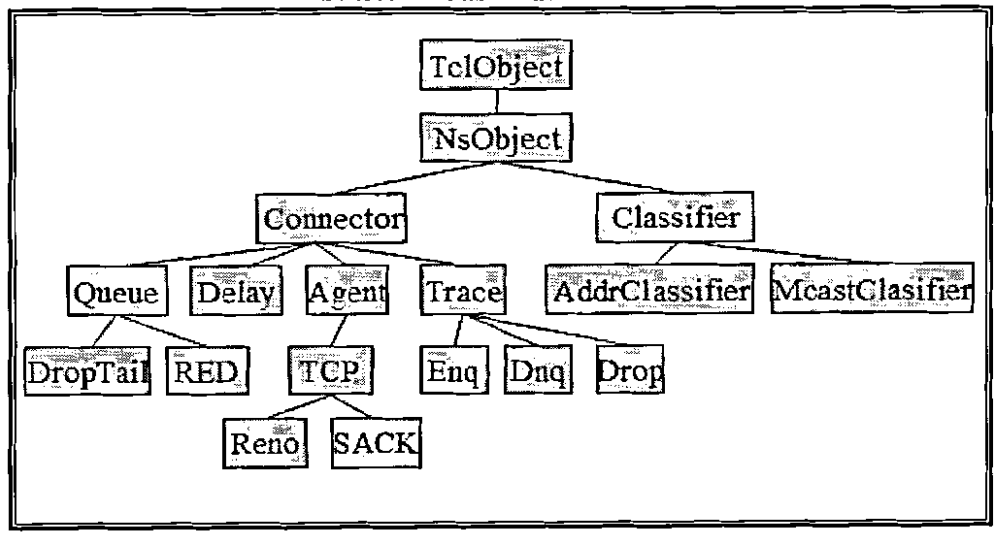


FIG 13: NS CLASS HIERARCHY

Partial NS class hierarchy is shown in the above figure. As shown TclObject is at the top of the hierarchy immediately followed by the NsObject class.

Source: The *ns* Manual

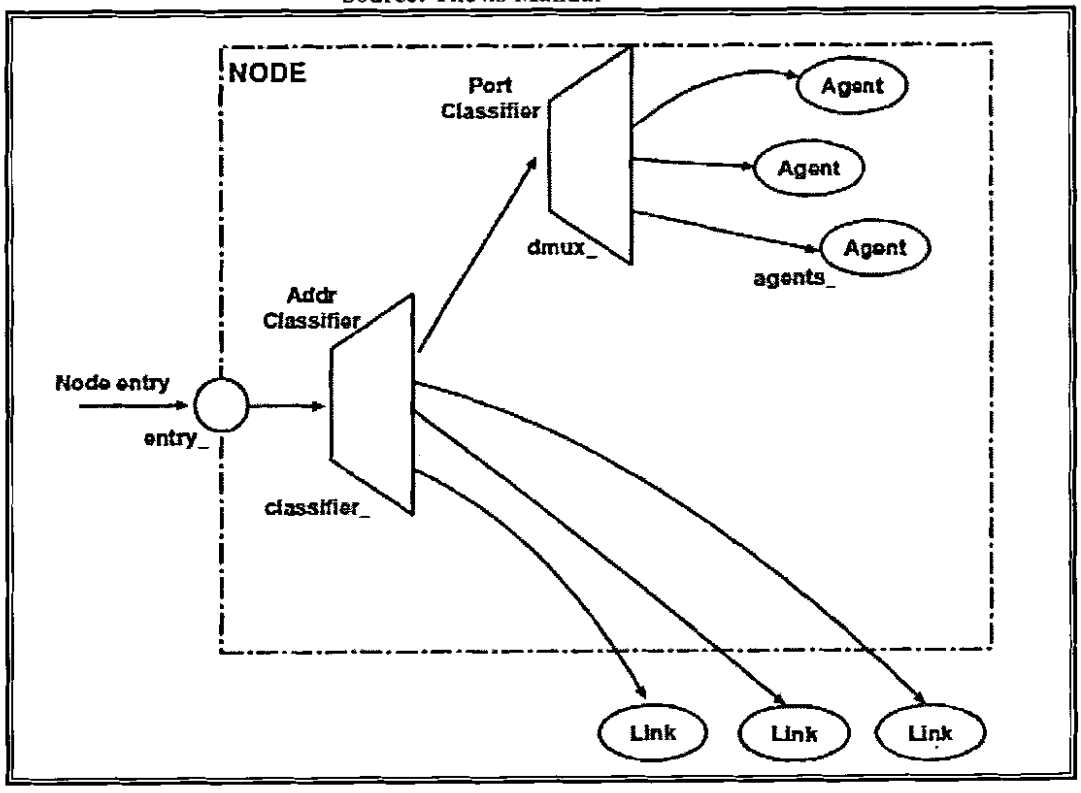


FIG 14: NODE CLASSIFIERS

CHAPTER 5

IMPLIMENTATION

AND RESULTS

5. IMPLIMENTATION AND RESULTS

This chapter describes the development phase and the testing of the embedded operating system kernel.

5.1 Development of the MM-App Framework

Many modifications in NS were made to implement the MM-App framework. They are analyzed one by one. First the changes in the ns-default file are analyzed.

5.1.1 Modifications in NS to implement Protocol

Ns simulator was modified to implement Protocol. A new udp-mm protocol was added. The protocol was based on congestion control. A new packet header entry was placed in ns/tcl/lib/ns-packet.tcl. Similarly for addition in the c++ source a new entry was placed in the ns/common/packet.h. In two other files (.cc and .h) the main source of the algorithm was written. The entries in the ns-packet.tcl and packet.h were made to make the protocol visible in the c++ and the Tcl namespace. The Makefile had to be modified also to ensure that the compiler and linker know about the new written files.

Once the required changes were done, make command was issued in the main ns folder. This resulted in the parsing of the Makefile and the recompilation of the NS simulator. Once this step was done successfully the udp-mm protocol classes became visible in the Tcl and c++ namespace. Sample simulations were run to check to correct installation of the algorithm and they worked perfectly.

5.1.1.1 ns-default.tcl

This file stores the default settings for different protocols. We also add the entry for the udp-mm protocol in this file.

```
#
# Application
#
Application/Telnet set interval_ 1.0
Application/MmApp set rate0_ 0.3mb
Application/MmApp set rate1_ 0.6mb
Application/MmApp set rate2_ 0.9mb
Application/MmApp set rate3_ 1.2mb
```

```
Application/MmApp set rate4_ 1.5mb
Application/MmApp set pktsize_ 1000
Application/MmApp set random_ false
```

5.1.1.2 ns-packet.tcl

In this file the packet name of the new packet which the udp-mm will use is given. It also includes descriptions of other packets already included in NS. The new packet should be added to this file to make it visible in the Tcl namespace.

```
foreach prot {
        AODV
        ARP
        aSRM
        Common
        CtrMcast
        Diffusion
        Encap
        Flags
        HttpInval
        IMEP
        IP
        IPinIP
        IVS
        LDP
        LL
        mcastCtrl
        MFTP
        MPLS
        Mac
        Message
        MIP
        Ping
        PGM
```

```
        PGM_SPM
        PGM_NAK
        RAP
        RTP
        Resv
        rtProtoDV
        rtProtoLS
        SR
        Src_rt
        SRM
        SRMEXT
        Snoop
        TCP
        TCPA
        TFRC
        TFRC_ACK
        TORA
        GAF
        UMP
        Pushback
        NV
        Multimedia
} {
        add-packet-header $prot
}
```

5.1.1.3 packet.h

In ns/common/packet.h entries were made to add the packet name in the c++ namespace.

```
enum packet_t {
        PT_TCP,
        PT_UDP,
```

PT_CBR,
PT_AUDIO,
PT_VIDEO,
PT_ACK,
PT_START,
PT_STOP,
PT_PRUNE,
PT_GRAFT,
PT_GRAFTACK,
PT_JOIN,
PT_ASSERT,
PT_MESSAGE,
PT_RTCP,
PT_RTP,
PT_RTPROTO_DV,
PT_CtrMcast_Encap,
PT_CtrMcast_Decap,
PT_SRM,

/* simple signaling messages */
PT_REQUEST,
PT_ACCEPT,
PT_CONFIRM,
PT_TEARDOWN,
PT_LIVE, **// packet from live network**
PT_REJECT,
PT_TELNET, **// not needed: telnet use TCP**
PT_FTP,
PT_PARETO,
PT_EXP,
PT_INVAL,
PT_HTTP,

```
/* new encapsulator */
PT_ENCAPSULATED,
PT_MFTP,

/* CMU/Monarch's extensions */
PT_ARP,
PT_MAC,
PT_TORA,
PT_DSR,
PT_AODV,
PT_IMEP,

/* RAP packets */
PT_RAP_DATA,
PT_RAP_ACK,
PT_TFRC,
PT_TFRC_ACK,
PT_PING,

/* Diffusion packets - Chalermek */
PT_DIFF,

/* LinkState routing update packets */
PT_RTPROTO_LS,

/*MPLS LDP header */
PT_LDP,

/* GAF packet */
PT_GAF,

/* ReadAudio traffic */
PT_REALAUDIO,
```

```
        /* Pushback Messages */
        PT_PUSHBACK,

#ifdef HAVE_STL
        // Pragmatic General Multicast
        PT_PGM,
#endif //STL

        // LMS packets
        PT_LMS,
        PT_LMS_SETUP,

        // insert new packet types here
        PT_Mutimedia,
        PT_NTYPE // This MUST be the LAST one
};

class p_info {
public:
        p_info() {
                name_[PT_TCP]= "tcp";
                name_[PT_UDP]= "udp";
                name_[PT_CBR]= "cbr";
                name_[PT_AUDIO]= "audio";
                name_[PT_VIDEO]= "video";
                name_[PT_ACK]= "ack";
                name_[PT_START]= "start";
                name_[PT_STOP]= "stop";
                name_[PT_PRUNE]= "prune";
                name_[PT_GRAFT]= "graft";
                name_[PT_GRAFTACK]= "graftAck";
                name_[PT_JOIN]= "join";
```

```
name_[PT_ASSERT]= "assert";
name_[PT_MESSAGE]= "message";
name_[PT_RTCP]= "rtcp";
name_[PT_RTP]= "rtp";
name_[PT_RTPROTO_DV]= "rtProtoDV";
name_[PT_CtrMcast_Encap]= "CtrMcast_Encap";
name_[PT_CtrMcast_Decap]= "CtrMcast_Decap";
name_[PT_SRM]= "SRM";

name_[PT_REQUEST]= "sa_req";
name_[PT_ACCEPT]= "sa_accept";
name_[PT_CONFIRM]= "sa_conf";
name_[PT_TEARDOWN]= "sa_teardown";
name_[PT_LIVE]= "live";
name_[PT_REJECT]= "sa_reject";

name_[PT_TELNET]= "telnet";
name_[PT_FTP]= "ftp";
name_[PT_PARETO]= "pareto";
name_[PT_EXP]= "exp";
name_[PT_INVAL]= "httpInval";
name_[PT_HTTP]= "http";
name_[PT_ENCAPSULATED]= "encap";
name_[PT_MFTP]= "mftp";
name_[PT_ARP]= "ARP";
name_[PT_MAC]= "MAC";
name_[PT_TORA]= "TORA";
name_[PT_DSR]= "DSR";
name_[PT_AODV]= "AODV";
name_[PT_IMEP]= "IMEP";

name_[PT_RAP_DATA] = "rap_data";
name_[PT_RAP_ACK] = "rap_ack";
```

```
        name_[PT_TFRC]= "tcpFriend";
        name_[PT_TFRC_ACK]= "tcpFriendCtl";
        name_[PT_PING]="ping";

        /* For diffusion : Chalermek */
        name_[PT_DIFF] = "diffusion";

        // Link state routing updates
        name_[PT_RTPROTO_LS] = "rtProtoLS";

        // MPLS LDP packets
        name_[PT_LDP] = "LDP";

        // for GAF
        name_[PT_GAF] = "gaf";

        // RealAudio packets
        name_[PT_REALAUDIO] = "ra";

        //pushback
        name_[PT_PUSHBACK] = "pushback";

#ifdef HAVE_STL
        // for PGM
        name_[PT_PGM] = "PGM";

#endif //STL
        // LMS entries
        name_[PT_LMS]="LMS";
        name_[PT_LMS_SETUP]="LMS_SETUP";
        name_[PT_Mutimedia]="Multimedia";
        name_[PT_NTYPE]= "undefined";
```

```
        }
        const char* name(packet_t p) const {
                if ( p <= PT_NTYPE ) return name_[p];
                return 0;
        }
        static bool data_packet(packet_t type) {
                return ( (type) == PT_TCP || \
                        (type) == PT_TELNET || \
                        (type) == PT_CBR || \
                        (type) == PT_AUDIO || \
                        (type) == PT_VIDEO || \
                        (type) == PT_ACK \
                        );
        }
private:
        static char* name_[PT_NTYPE+1];
};
extern p_info packet_info; /* map PT_* to string name */
//extern char* p_info::name_[];

#define DATA_PACKET(type) ( (type) == PT_TCP ||\
                (type) == PT_TELNET ||\
                (type) == PT_CBR ||\
                (type) == PT_AUDIO ||\
                (type) == PT_VIDEO ||\
                (type) == PT_ACK\
                )
```

5.1.1.4 agent.h

To make the new application and agent running with your NS distribution, you should add two methods to "Agent" class as public. In the "command" member function of the "MmApp" class, there defined an "attach-agent" OTcl command. When this command is received from OTcl, "MmApp" tries to attach itself to the underlying agent.

Before the attachment, it invokes the "supportMM()" member function of the underlying agent to check if the underlying agent support for multimedia transmission (i.e. can it carry data from application to application), and invokes "enableMM()" if it does. Even though these two member functions are defined in the "UdpMmAgnet" class, it is not defined in its base ancestor "Agent" class, and the two member functions of the general "Agent" class are called. Therefore, when trying to compile the code, it will give an error message. Inserting the two methods as public member functions of the "Agent" class (in "agent.h") as follows will solve this problem.

```
class EventTrace;
class Agent : public Connector {
 public:
        Agent(packet_t pktType);
        virtual ~Agent();
        void recv(Packet*, Handler*);

        //Added for edrop tracing - ratul
        void recvOnly(Packet *) {};

        void send(Packet* p, Handler* h) { target_->recv(p, h); }
        virtual void timeout(int tno);

        virtual void sendmsg(int sz, AppData*, const char* flags = 0);
        virtual void send(int sz, AppData *data) { sendmsg(sz, data, 0); }
        virtual void sendto(int sz, AppData*, const char* flags, nsaddr_t dst);

        virtual int supportMM() { return 0;}
        virtual void enableMM() {}
        virtual void sendmsg(int nbytes, const char *flags = 0);
        virtual void send(int nbytes) { sendmsg(nbytes); }
```

```
virtual void sendto(int nbytes, const char* flags, nsaddr_t dst);
virtual void connect(nsaddr_t dst);
virtual void close();
virtual void listen();
virtual void attachApp(Application* app);
virtual int& size() { return size_; }
inline nsaddr_t& addr() { return here_.addr_; }
inline nsaddr_t& port() { return here_.port_; }
inline nsaddr_t& daddr() { return dst_.addr_; }
inline nsaddr_t& dport() { return dst_.port_; }
void set_pkttype(packet_t pkttype) { type_ = pkttype; }
inline packet_t get_pkttype() { return type_; }
```

5.1.1.5 app.h

You also need to add an additional member function "recv_msg(int nbytes, const char *msg)" to "Application" class as shown in Figure 26 (b). This member function, which was included in the "Application" class in the old versions of NS (ns-2.1b4a for sure), is removed from the class in the latest versions (ns-2.1b8a for sure). Our multimedia application was initially written for the ns-2.1.b4a, and therefore requires "Application::recv_msg()" member function for ns-2.1b8a or later versions.

```
class Application : public Process {
public:
        Application();
        virtual void send(int nbytes);
        virtual void recv(int nbytes);
        virtual void recv_msg(int nbytes, const char *msg = 0) {};
        virtual void resume();
#endif
```

5.2 Design and Implementation

For the application, we decided to take the CBR implementation and modify it to have the "five level media scaling" feature. We examined the C++ class hierarchy, and decided to name the class of this application as "MmApp" and implement as a child class of "Application". The matching OTcl hierarchy name is "Application/MmApp". The sender and receiver behavior of the application is implemented together in "MmApp". For the modified UDP agent that supports "MmApp", we decided to name it "UdpMmAgent" and implement it as a child class of "UdpAgent". The matching OTcl hierarchy name is "Agent/UDP/UDPmm"

5.2.1 MmApp Header

For the application level communication, we decided to define a header of which the structure named in C++ "hdr_mm". Whenever the application has information to transmit, it will hand it to "UdpMmAgent" in the "hdr_mm" structure format. Then, "UdpMmAgent" allocates one or more packets (depending on the simulated data packet size) and writes the data to the multimedia header of each packet. Figure 23 shows the header definition, in which a structure for the header and a header class object, "MultimediaHeaderClass" that should be derived from "PacketHeaderCalss" is defined. In defining this class, the OTcl name for the header ("PacketHeader/Multimedia") and the size of the header structure defined are presented. Notice that bind_offset() must be called in the constructor of this class.

We also add lines to packet.h and ns-packet.tcl as shown in Figure 24 (a) and (b) to add our "Mulltimedia" header to the header stack. At this point, the header creation process is finished, and "UdpMmAgent" will access the new header via hdr_mm::access() member function.

```
#include "timer-handler.h"
#include "packet.h"
#include "app.h"
#include "udp-mm.h"
```

```
// This is used for receiver's received packet accounting
struct pkt_accounting {
    int last_seq;       // sequence number of last received MM pkt
    int last_scale;     // rate (0-4) of last acked
    int lost_pkts;      // number of lost pkts since last ack
    int recv_pkts;      // number of received pkts since last ack
    double rtt;         // round trip time
};
class MmApp;

// Sender uses this timer to
// schedule next app data packet transmission time
class SendTimer : public TimerHandler {
 public:
        SendTimer(MmApp* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        MmApp* t_;
};

// Reciver uses this timer to schedule
// next ack packet transmission time
class AckTimer : public TimerHandler {
 public:
        AckTimer(MmApp* t) : TimerHandler(), t_(t) {}
        inline virtual void expire(Event*);
 protected:
        MmApp* t_;
};
// Mulitmedia Application Class Definition
class MmApp : public Application {
 public:
        MmApp();
```

```
        void send_mm_pkt();   // called by SendTimer:expire (Sender)
        void send_ack_pkt();  // called by AckTimer:expire (Receiver)
        void send_nack_pkt(); // called by AckTimer:expire (Receiver)
protected:
        int command(int argc, const char*const* argv);
        void start();    // Start sending data packets (Sender)
        void stop();     // Stop sending data packets (Sender)
private:
        void init();
        inline double next_snd_time();                          // (Sender)
        virtual void recv_msg(int nbytes, const char *msg = 0); // (Sender/Receiver)
        void set_scale(const hdr_mm *mh_buf);                   // (Sender)
        void adjust_scale(void);                                // (Receiver)
        void account_recv_pkt(const hdr_mm *mh_buf);            // (Receiver)
        void init_recv_pkt_accounting();                        // (Receiver)
        double rate[5];          // Transmission rates associated to scale values
        double interval_;        // Application data packet transmission interval
        int pktsize_;            // Application data packet size
        int random_;             // If 1 add randomness to the interval
        int running_;            // If 1 application is running
        int seq_;                // Application data packet sequence number
        int scale_;              // Media scale parameter
        pkt_accounting p_accnt;
        SendTimer snd_timer_;    // SendTimer
        AckTimer ack_timer_;     // AckTimer
        AckTimer nack_timer_;    // NackTimer
};
```

5.2.2 MmApp Sender

The sender uses a timer for scheduling the next application data packet transmission. We defined the "SendTimer" class derived from the "TimerHandler" class, and wrote its "expire()" member function to call "send_mm_pkt()" member function of the "MmApp" object. Then, we included an instance of this object as a private member of

"MmApp" object referred to as "snd_timer_". Figure 25 shows the "SendTimer" implementation.

Before setting this timer, "MmApp" re-calculates the next transmission time using the transmission rate associated with the current scale value and the size of application data packet that is given in the input simulation script (or use the default size). The "MmApp" sender, when an ACK application packet arrives from the receiver side, updates the scale parameter.

5.2.3 MmApp Receiver

The receiver uses a timer, named "ack_timer_", to schedule the next application ACK packet transmission of which the interval is equal to the mean RTT. When receiving an application data packet from the sender, the receiver counts the number of received packets and also counts the number of lost packets using the packet sequence number. When the "ack_timer_" expires, it invokes the "send_ack_pkt" member function of "MmApp", which adjusts the scale value looking at the received and lost packet accounting information, resets the received and lost count to 0, and sends an ACK packet with the adjusted scale value. Note that the receiver doesn't have any connection establishment or closing methods. Therefore, starting from the first packet arrival, the receiver periodically sends ACK packets and never stops (this is a bug).

```
==========================================================
#include "random.h"
#include "mm-app.h"
// MmApp OTcl linkage class
static class MmAppClass : public TclClass {
 public:
  MmAppClass() : TclClass("Application/MmApp") {}
  TclObject* create(int, const char*const*) {
    return (new MmApp);
  }
} class_app_mm;
```

```
// When snd_timer_ expires call MmApp:send_mm_pkt()
void SendTimer::expire(Event*)
{
 t_->send_mm_pkt();
}
// When ack_timer_ expires call MmApp:send_ack_pkt()
void AckTimer::expire(Event*)
{
 t_->send_ack_pkt();
 t_->send_nack_pkt();
}
// Constructor (also initialize instances of timers)
MmApp::MmApp() : running_(0), snd_timer_(this), ack_timer_(this), nack_timer_(this)
{
 bind_bw("rate0_", &rate[0]);
 bind_bw("rate1_", &rate[1]);
 bind_bw("rate2_", &rate[2]);
 bind_bw("rate3_", &rate[3]);
 bind_bw("rate4_", &rate[4]);
 bind("pktsize_", &pktsize_);
 bind_bool("random_", &random_);
}
// OTcl command interpreter
int MmApp::command(int argc, const char*const* argv)
{
 Tcl& tcl = Tcl::instance();
 if (argc == 3) {
  if (strcmp(argv[1], "attach-agent") == 0) {
   agent_ = (Agent*) TclObject::lookup(argv[2]);
   if (agent_ == 0) {
       tcl.resultf("no such agent %s", argv[2]);
       return(TCL_ERROR);
   }
```

```
    // Make sure the underlying agent support MM
    if(agent_->supportMM()) {
        agent_->enableMM();
    }
    else {
        tcl.resultf("agent \"%s\" does not support MM Application", argv[2]);
        return(TCL_ERROR);
    }
    agent_->attachApp(this);
    return(TCL_OK);
   }
  }
  return (Application::command(argc, argv));
}
void MmApp::init()
{
  scale_ = 0; // Start at minimum rate
  seq_ = 0;  // MM sequence number (start from 0)
  interval_ = (double)(pktsize_ << 3)/(double)rate[scale_];
}
void MmApp::start()
{
  init();
  running_ = 1;
  send_mm_pkt();
}
void MmApp::stop()
{
  running_ = 0;
}
```

```
// Send application data packet
void MmApp::send_mm_pkt()
{
 hdr_mm mh_buf;

 if (running_) {
  // The below info is passed to UDPmm agent, which will write it
  // to MM header after packet creation.
  mh_buf.ack = 0;                          // This is a MM packet
  mh_buf.nack = 1;                         // This is negative ack
  mh_buf.seq = seq_++;                     // MM sequece number
  mh_buf.nbytes = pktsize_;                // Size of MM packet (NOT UDP packet size)
  mh_buf.time = Scheduler::instance().clock(); // Current time
  mh_buf.scale = scale_;                   // Current scale value
  agent_->sendmsg(pktsize_, (char*) &mh_buf); // send to UDP

  // Reschedule the send_pkt timer
  double next_time_ = next_snd_time();
  if(next_time_ > 0) snd_timer_.resched(next_time_);
 }
}
// Schedule next data packet transmission time
double MmApp::next_snd_time()
{
 // Recompute interval in case rate or size chages
 interval_ = (double)(pktsize_ << 3)/(double)rate[scale_];
 double next_time_ = interval_;
 if(random_)
  next_time_ += interval_ * Random::uniform(-0.5, 0.5);
 return next_time_;
}
// Receive message from underlying agent
void MmApp::recv_msg(int nbytes, const char *msg)
```

```
{
 if(msg) {
  hdr_mm* mh_buf = (hdr_mm*) msg;
  if(mh_buf->ack == 1) {
   // If received packet is ACK packet
   set_scale(mh_buf);
   send_nack_pkt();
  }
  else {
   // If received packet is MM packet
   account_recv_pkt(mh_buf);
   if(mh_buf->seq != 0) send_ack_pkt();
  }
 }
}
// Sender sets its scale to what reciver notifies
void MmApp::set_scale(const hdr_mm *mh_buf)
{
 scale_ = mh_buf->scale;
}

void MmApp::account_recv_pkt(const hdr_mm *mh_buf)
{
 double local_time = Scheduler::instance().clock();
 // Calculate RTT
 if(mh_buf->seq == 0) {
  init_recv_pkt_accounting();
  p_accnt.rtt = (local_time - mh_buf->time);//2*
 }
 else
  p_accnt.rtt = 0.9 * p_accnt.rtt + 0.1 * 2*(local_time - mh_buf->time);
 // Count Received packets and Calculate Packet Loss
 p_accnt.recv_pkts ++;
```

```
 p_accnt.lost_pkts += (mh_buf->seq - p_accnt.last_seq - 1);
 p_accnt.last_seq = mh_buf->seq;
}
void MmApp::init_recv_pkt_accounting()
{
 p_accnt.last_seq = -1;
 p_accnt.last_scale = 0;
 p_accnt.lost_pkts = 0;
 p_accnt.recv_pkts = 0;
}
void MmApp::send_ack_pkt(void)
{
 double local_time = Scheduler::instance().clock();
 adjust_scale();
 // Send ack message
 hdr_mm ack_buf;
 ack_buf.ack = 1; // This packet is ack packet
 ack_buf.time = local_time;
 ack_buf.nbytes = 50; // Ack packet size is 40 Bytes
 ack_buf.scale = p_accnt.last_scale;
 agent_->sendmsg(ack_buf.nbytes, (char*) &ack_buf);
 // Schedul next ACK time
 ack_timer_.resched(p_accnt.rtt);
}
//Code for Nack
void MmApp::send_nack_pkt(void)
{
 double local_time = Scheduler::instance().clock();
 adjust_scale();
 // send ack message
 hdr_mm nack_buf;
 nack_buf.ack = 1; // this packet is nack packet
 nack_buf.time = local_time;
```

```
 nack_buf.nbytes = 40; // Nack packet size is 40 Bytes
 nack_buf.scale = p_accnt.last_scale;
 agent_->sendmsg(nack_buf.nbytes, (char*) &nack_buf);
 // Schedul next ACK time
 nack_timer_.resched(p_accnt.rtt);
}
void MmApp::adjust_scale(void)
{
 if(p_accnt.recv_pkts > 0) {
  if(p_accnt.lost_pkts > 0)
   p_accnt.last_scale = (int)(p_accnt.last_scale / 2);
  else {
   p_accnt.last_scale++;
   if(p_accnt.last_scale > 4) p_accnt.last_scale = 4;
  }
 }
 p_accnt.recv_pkts = 0;
 p_accnt.lost_pkts = 0;
}
```

==========

5.2.4 UdpMmAgent

The "UdpMmAgent" is modified from the "UdpAgent" to have the following additional features: (1) writing to the sending data packet MM header the information received from a "MmApp" (or reading information from the received data packet MM header and handing it to the "MmApp"), (2) segmentation and re-assembly ("UdpAgent" only implements segmentation), and (3) setting the priority bit (IPv6) to 15 (max priority) for the "MmApp" packets.

```
#include "udp-mm.h"
#include "rtp.h"
#include "random.h"
#include <string.h>
```

```
int hdr_mm::offset_;
// Mulitmedia Header Class
static class MultimediaHeaderClass : public PacketHeaderClass {
public:
        MultimediaHeaderClass() : PacketHeaderClass("PacketHeader/Multimedia",
                                                    sizeof(hdr_mm)) {
                bind_offset(&hdr_mm::offset_);
        }
} class_mmhdr;
// UdpMmAgent OTcl linkage class
static class UdpMmAgentClass : public TclClass {
public:
        UdpMmAgentClass() : TclClass("Agent/UDP/UDPmm") {}
        TclObject* create(int, const char*const*) {
                return (new UdpMmAgent());
        }
} class_udpmm_agent;
// Constructor (with no arg)
UdpMmAgent::UdpMmAgent() : UdpAgent()
{
        support_mm_ = 0;
        asm_info.seq = -1;
}
UdpMmAgent::UdpMmAgent(packet_t type) : UdpAgent(type)
{
        support_mm_ = 0;
        asm_info.seq = -1;
}
// Add Support of Multimedia Application to UdpAgent::sendmsg
void UdpMmAgent::sendmsg(int nbytes, const char* flags)
{
        Packet *p;
        int n, remain;
```

```
//Calculate the number of packets required to send the data
if (size_) {
        n = (nbytes/size_ + (nbytes%size_ ? 1 : 0));
        remain = nbytes%size_;
}
else
        printf("Error: UDPmm size = 0\n");

if (nbytes == -1) {
        printf("Error:  sendmsg() for UDPmm should not be -1\n");
        return;
}
double local_time =Scheduler::instance().dock();
while (n-- > 0) {
        p = allocpkt(); //Allocate Packet memory to the pointer P
        if (n==0 && remain>0)
                hdr_cmn::access(p)->size() = remain;
        else
                hdr_cmn::access(p)->size() = size_;
        hdr_rtp* rh = hdr_rtp::access(p);
        rh->flags() = 0;
        rh->seqno() = ++seqno_;
        hdr_cmn::access(p)->timestamp() = (u_int32_t)(SAMPLERATE*local_time);
        // to eliminate recv to use MM fields for non MM packets
        hdr_mm* mh = hdr_mm::access(p);
        mh->ack = 0;
        mh->seq = 0;
        mh->nbytes = 0;
        mh->time = 0;
        mh->scale = 0;
        // mm udp packets are distinguished by setting the ip
        // priority bit to 15 (Max Priority).
```

```
            if(support_mm_) {
                hdr_ip* ih = hdr_ip::access(p);
                ih->prio_ = 15;
                if(flags) // MM Seq Num is passed as flags
                    memcpy(mh, flags, sizeof(hdr_mm));
            }
            if (flags && (0 ==strcmp(flags, "NEW_BURST")))
                rh->flags() |= RTP_M;
            target_->recv(p);
        }
        idle();
}
// Support Packet Re-Assembly and Multimedia Application
void UdpMmAgent::recv(Packet* p, Handler*)
{
        hdr_ip* ih = hdr_ip::access(p);
        int bytes_to_deliver = hdr_cmn::access(p)->size();
        // If it is a MM packet (data or ack)
        if(ih->prio_ == 15) {
            if(app_) {  // if MM Application exists
                // re-assemble MM Application packet if segmented
                hdr_mm* mh = hdr_mm::access(p);
                if(mh->seq == asm_info.seq)
                    asm_info.rbytes += hdr_cmn::access(p)->size();
                else {
                    asm_info.seq = mh->seq;
                    asm_info.tbytes = mh->nbytes;
                    asm_info.rbytes = hdr_cmn::access(p)->size();
                }
                // If fully reassembled, pass the packet to application
                if(asm_info.tbytes == asm_info.rbytes) {
                    hdr_mm mh_buf;
                    memcpy(&mh_buf, mh, sizeof(hdr_mm));
```

```
                        app_->recv_msg(mh_buf.nbytes, (char*) &mh_buf);
                    }
                }
                Packet::free(p);
        }
        // If it is a normal data packet (not MM data or ack packet)
        else {
                if (app_) app_->recv(bytes_to_deliver);
                Packet::free(p);
        }
}
```

==========

5.3 Testing

After development and implementation the protocol was tested with different traffic scenarios and topologies. The protocol was implemented and NS simulator was recompiled and linked. After linking the following command was issued on the command line to check the correct installation of the udp-mm protocol.

==========

```
#
set val(chan)           Channel/WirelessChannel;    #Channel Type
set val(prop)           Propagation/TwoRayGround;   # radio-propagation model
set val(netif)          Phy/WirelessPhy;            # network interface type
set val(mac)            Mac/802_11;                 # MAC type
set val(ifq)            Queue/DropTail/PriQueue;    # interface queue type
set val(ll)             LL;                         # link layer type
set val(ant)            Antenna/OmniAntenna;        # antenna model
set val(ifqlen)         50;                         # max packet in ifq
set val(nn)             10;                         # number of nodes
set val(rp)             AODV;                       # routing protocol
set val(x)              500
set val(y)              500
```

```
# Antenna height and location
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 10.0
Antenna/OmniAntenna set Gt_ 1.0
Antenna/OmniAntenna set Gr_ 1.0

# Initialize Global Variables
set ns_ [new Simulator]
#$ns_ multicast

#Color Queue
$ns_ color 0 blue
$ns_ color 1 blue
$ns_ color 2 chocolate
$ns_ color 3 red
$ns_ color 4 brown
$ns_ color 5 tan
$ns_ color 6 gold
$ns_ color 7 black

set tracefd     [open wireless_out.tr w]
$ns_ trace-all $tracefd
set namtrace [open wireless_out.nam w]
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)
# set up topography object
set topo       [new Topography]

$topo load_flatgrid $val(x) $val(y)

# Create God
create-god $val(nn)
# Create channel #1 and #2
```

```
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]

# Configure Node
$ns_ node-config -adhocRouting $val(rp) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace ON \
                 -channel $chan_1_

#Node Colors
set node_(0) [$ns_ node]
$node_(0) color "green"
set node_(1) [$ns_ node]
$node_(1) color "red"
set node_(2) [$ns_ node]
$node_(2) color "blue"
set node_(3) [$ns_ node]
$node_(3) color "brown"
set node_(4) [$ns_ node]
$node_(4) color "tan"
set node_(5) [$ns_ node]
$node_(5) color "green"
set node_(6) [$ns_ node]
```

```
$node_(6) color "blue"
set node_(7) [$ns_ node]
$node_(7) color "red"
set node_(8) [$ns_ node]
$node_(8) color "gold"
set node_(9) [$ns_ node]
$node_(9) color "green"
#
# Provide initial (X,Y, for now Z=0) co-ordinates for nodes
#
$node_(0) set X_ 5.0
$node_(0) set Y_ 2.0
$node_(0) set Z_ 0.0

$node_(1) set X_ 216.0
$node_(1) set Y_ 42.0
$node_(1) set Z_ 0.0

$node_(2) set X_ 447.0
$node_(2) set Y_ 79.0
$node_(2) set Z_ 0.0

$node_(3) set X_ 493.0
$node_(3) set Y_ 30.0
$node_(3) set Z_ 0.0

$node_(4) set X_ 55.0
$node_(4) set Y_ 35.0
$node_(4) set Z_ 0.0

$node_(5) set X_ 334.0
$node_(5) set Y_ 62.0
$node_(5) set Z_ 0.0
```

```
$node_(6) set X_ 204.0
$node_(6) set Y_ 92.0
$node_(6) set Z_ 0.0

$node_(7) set X_ 254.0
$node_(7) set Y_ 72.0
$node_(7) set Z_ 0.0

$node_(8) set X_ 405.0
$node_(8) set Y_ 59.0
$node_(8) set Z_ 0.0

$node_(9) set X_ 145.0
$node_(9) set Y_ 22.0
$node_(9) set Z_ 0.0

for {set i 0} {$i < $val(nn)} {incr i} {
        $ns_ initial_node_pos $node_($i) 20
}

# Nodes then starts to move away from initial position
$ns_ at 0.1 "$node_(1) setdest 154.0 210.0 30.0"
$ns_ at 0.1 "$node_(2) setdest 118.0 177.0 30.0"
$ns_ at 0.1 "$node_(3) setdest 150.0 134.0 30.0"
$ns_ at 0.1 "$node_(7) setdest 31.0 256.0 30.0"
$ns_ at 0.1 "$node_(9) setdest 67.0 295.0 30.0"
```

5.3.1 Testing with Multimedia Application

Given is the source code of a file which contains the implementation of a network topology with multimedia transmission between the nodes.

```
==========================================================
#Declare an MM agent for Transmitor node_(0)
set udp_s [new Agent/UDP/UDPmm]
$ns_ attach-agent $node_(0) $udp_s
#Set packet size
$udp_s set packetSize_ 1000
#Declare an MM agent for Receiving node_(1)
set udp_r [new Agent/UDP/UDPmm]
$ns_ attach-agent $node_(1) $udp_r
#Setup a MM Application for node_(0)
set mmapp_s [new Application/MmApp]
$mmapp_s attach-agent $udp_s
#Set packet size
$mmapp_s set pktsize_ 1000
$mmapp_s set random_ false
$ns_ connect $udp_s $udp_r
#Tell node when to start transmission
$ns_ at 1.0 "$mmapp_s start"
#*---------------------------------------*
#Declare an MM agent for Transmitor node_(0)
set udp1_s [new Agent/UDP/UDPmm]
$ns_ attach-agent $node_(0) $udp1_s
#Set packet size
$udp1_s set packetSize_ 1000
#Declare an MM agent for Receiving node_(2)
set udp1_r [new Agent/UDP/UDPmm]
```

```
$ns_ attach-agent $node_(2) $udp1_r
#Setup a MM Application for node_(0)
set mmapp1_s [new Application/MmApp]
$mmapp1_s attach-agent $udp1_s
#Set packet size
$mmapp1_s set pktsize_ 1000
$mmapp1_s set random_ false
$ns_ connect $udp1_s $udp1_r
#Tell node when to start transmission
$ns_ at 1.4 "$mmapp1_s start"
#*--------------------------------------*
#Declare an MM agent for Transmitor node_(0)
set udp2_s [new Agent/UDP/UDPmm]
$ns_ attach-agent $node_(0) $udp2_s
#Declare an MM agent for Receiving node_(3)
set udp2_r [new Agent/UDP/UDPmm]
$ns_ attach-agent $node_(3) $udp2_r
#Setup a MM Application for node_(0)
set mmapp2_s [new Application/MmApp]
$mmapp2_s attach-agent $udp2_s
#Set packet size
$mmapp2_s set pktsize_ 1000
$mmapp2_s set random_ false
$ns_ connect $udp2_s $udp2_r
#Tell node when to start transmission
$ns_ at 1.6 "$mmapp2_s start"
#*--------------------------------------*
#Declare an MM agent for Transmitor node_(4)
set udp3_s [new Agent/UDP/UDPmm]
$ns_ attach-agent $node_(4) $udp3_s
```

#Setup a MM Application for node_(4)

set mmapp3_s [new Application/MmApp]

$mmapp3_s attach-agent $udp3_s

#Set packet size

$mmapp3_s set pktsize_ 1000

$mmapp3_s set random_ false

$ns_ connect $udp3_s $udp2_r

#Tell node when to start transmission

$ns_ at 2.0 "$mmapp3_s start"

$ns_ at 2.017 "$mmapp1_s stop"

#*----------*

#Declare an MM agent for Transmitor node_(4)

set udp4_s [new Agent/UDP/UDPmm]

$ns_ attach-agent $node_(5) $udp4_s

#Setup a MM Application for node_(4)

set mmapp4_s [new Application/MmApp]

$mmapp4_s attach-agent $udp4_s

#Set packet size

$mmapp4_s set pktsize_ 1000

$mmapp4_s set random_ false

$ns_ connect $udp4_s $udp2_r

#Tell node when to start transmission

$ns_ at 2.5 "$mmapp4_s start"

#*----------*

#Declare an MM agent for Transmitor node_(4)

set udp5_s [new Agent/UDP/UDPmm]

$ns_ attach-agent $node_(6) $udp5_s

#Declare an MM agent for Receiving node_(7)

set udp5_r [new Agent/UDP/UDPmm]

$ns_ attach-agent $node_(7) $udp5_r

#Setup a MM Application for node_(4)

set mmapp5_s [new Application/MmApp]

$mmapp5_s attach-agent $udp5_s

#Set packet size

$mmapp5_s set pktsize_ 1000

$mmapp5_s set random_ false

$ns_ connect $udp5_s Sudp5_r

#Tell node when to start transmission

$ns_ at 2.0 "$mmapp5_s start"

#*----------*

#Declare an MM agent for Transmitor node_(4)

set udp6_s [new Agent/UDP/UDPmm]

$ns_ attach-agent $node_(8) $udp6_s

#Declare an MM agent for Receiving node_(9)

set udp6_r [new Agent/UDP/UDPmm]

$ns_ attach-agent $node_(9) $udp6_r

#Setup a MM Application for node_(8)

set mmapp6_s [new Application/MmApp]

$mmapp6_s attach-agent $udp6_s

#Set packet size

$mmapp6_s set pktsize_ 1000

$mmapp6_s set random_ false

$ns_ connect $udp6_s $udp6_r

#Tell node when to start transmission

$ns_ at 2.0 "$mmapp6_s start"

#*----------*

```
#
# Tell nodes when the simulation ends
#
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at 5.0 "$node_($i) reset";
}

$ns_ at 2.7 "stop"
$ns_ at 2.71 "puts \"NS EXITING...\" ; $ns_ halt"

proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
}
puts "Starting Simulation..."
$ns_ run
```

5.3.2 Comparison with SRM and UDP

In our testing we compared the protocol with SRM and UDP under different scenarios the results shows that our protocol performs better than both SRM and UDP. The graphical output for those tests is discussed below.

5.3.2.1 Packet Delivery vs. Traffic Rate

We observe that under light load (500ms) SRM achieves near perfect reliability. However, as the traffic rate increases beyond 300ms of interdeparture time, SRM performance starts to suffer and drops at 30%. This is due to high contention the network experience as traffic rate and network load grows.

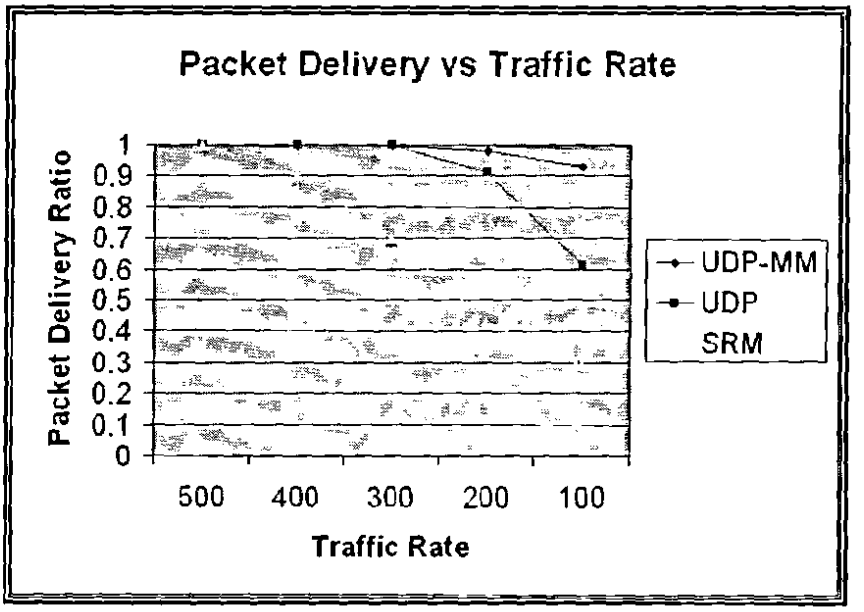


FIG 15: PACKET DELIVERY VS. TRAFFIC RATE

5.3.2.2 Control Overhead vs. Traffic rate

We observe that as the traffic rate increases, SRM control overhead increase.

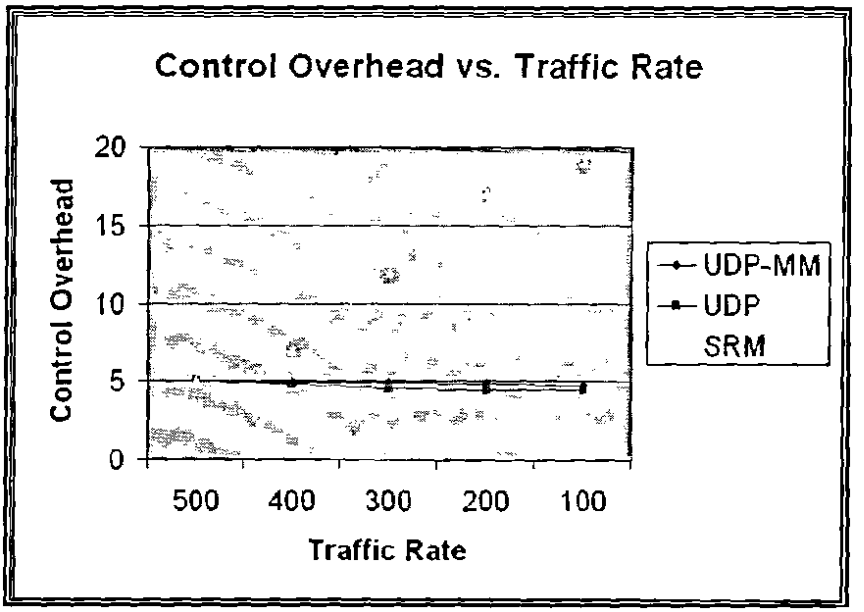


FIG 16: CONTROL OVERHEAD VS TRAFFIC RATE

5.3.2.3 End-To-End Delay vs. Traffic rate

The increase in traffic rate increases the End-To-End delay both in case of SRM and UDP.

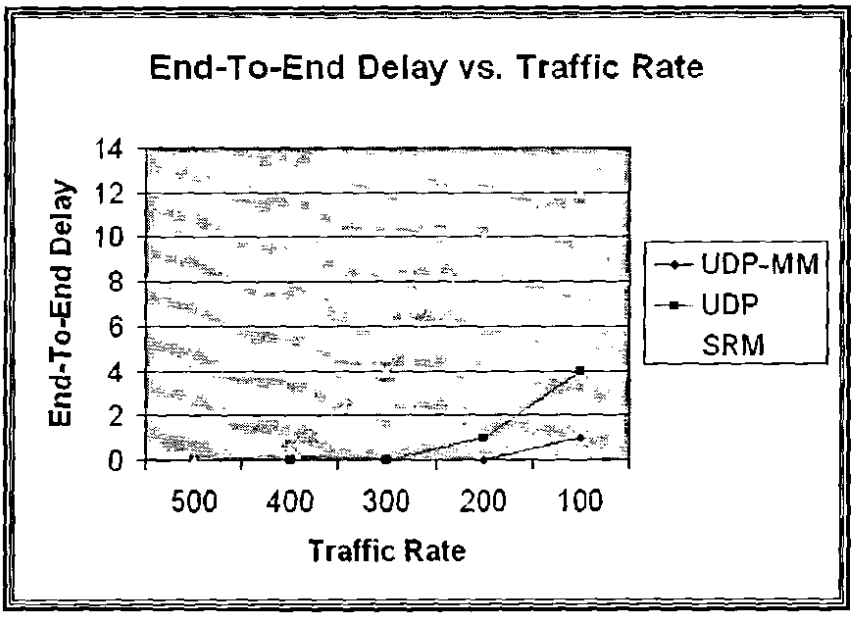


FIG 17: END-TO-END DELAY VS. TRAFFIC RATE

5.3.2.4 Total Data Packet Received vs. Traffic Rate

With the increase in the traffic rate the total data packet received by SRM decreases.

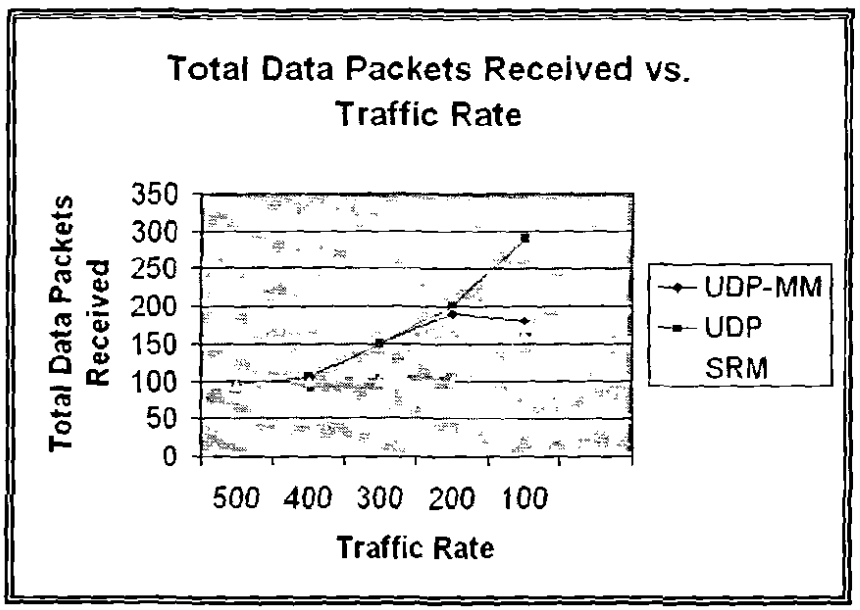


FIG 18: TOTAL DATA PACKET RECEIVED VS. TRAFFIC RATE

5.3.2.5 Packet Delivery Ratio vs. No. of Receivers

We observe that varying no of multicast receivers has no impact of delivering data packets when using UDP. Our protocol experiences approximately same packet delivery ratio as that of UDP. However same is not true for SRM as it performs worse as the no. of receivers grow.

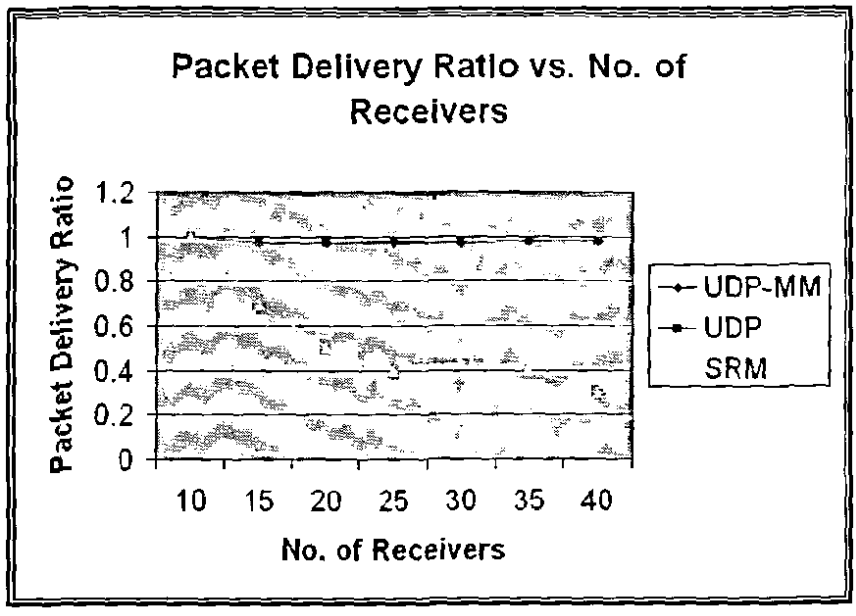


FIG 19: PACKET DELIVERY RATIO VS. NO. OF RECEIVERS

5.3.2.6 Control Overhead vs. No. of Receivers

Control overhead also increases for SRM as the no. of receivers increases. This results in network congestion, packet drop and poor network performance because it invokes further control message transmission.

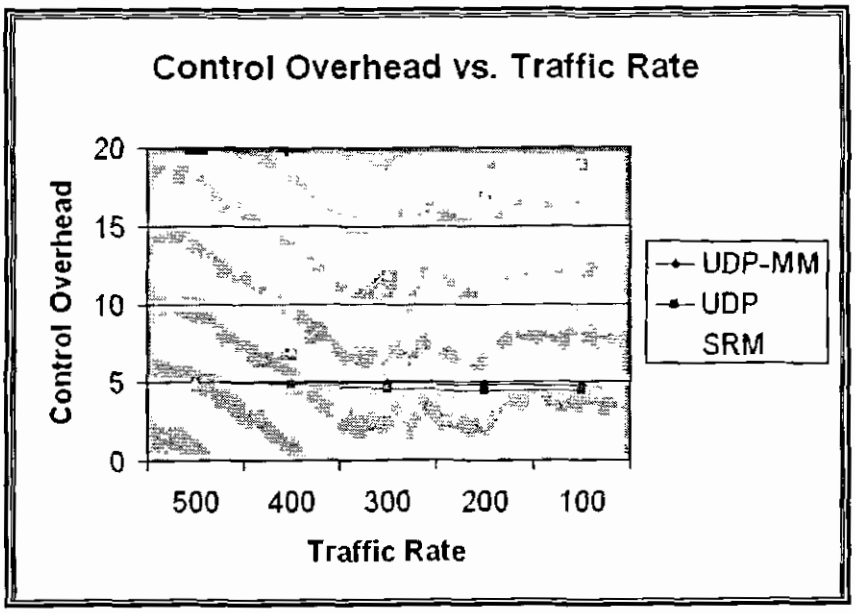


FIG 20: CONTROL OVERHEAD VS NO. OF RECEIVERS

5.3.2.7 Control Overhead vs. Mobility

SRM displays highest control overhead with mobility involved.

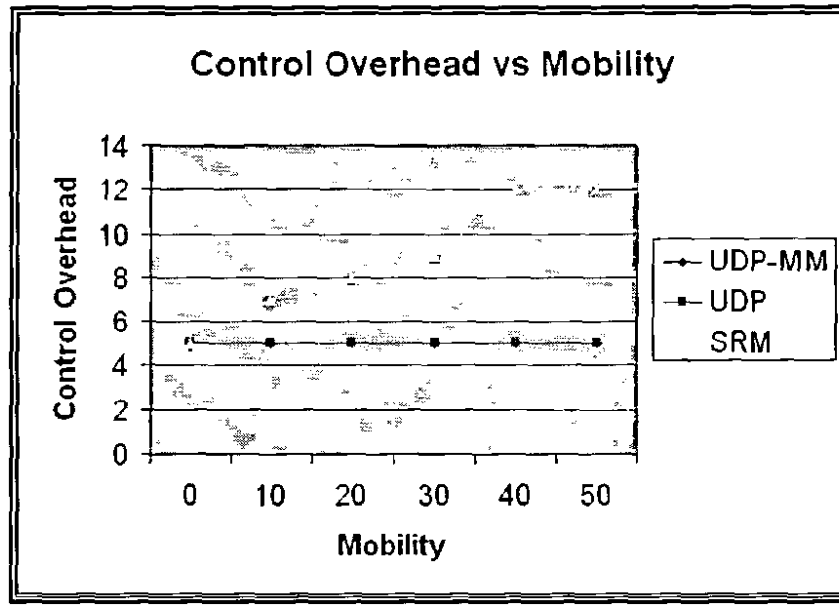


FIG 21: CONTROL OVERHEAD VS MOBILITY

5.3.3 Testing with Complex Topology

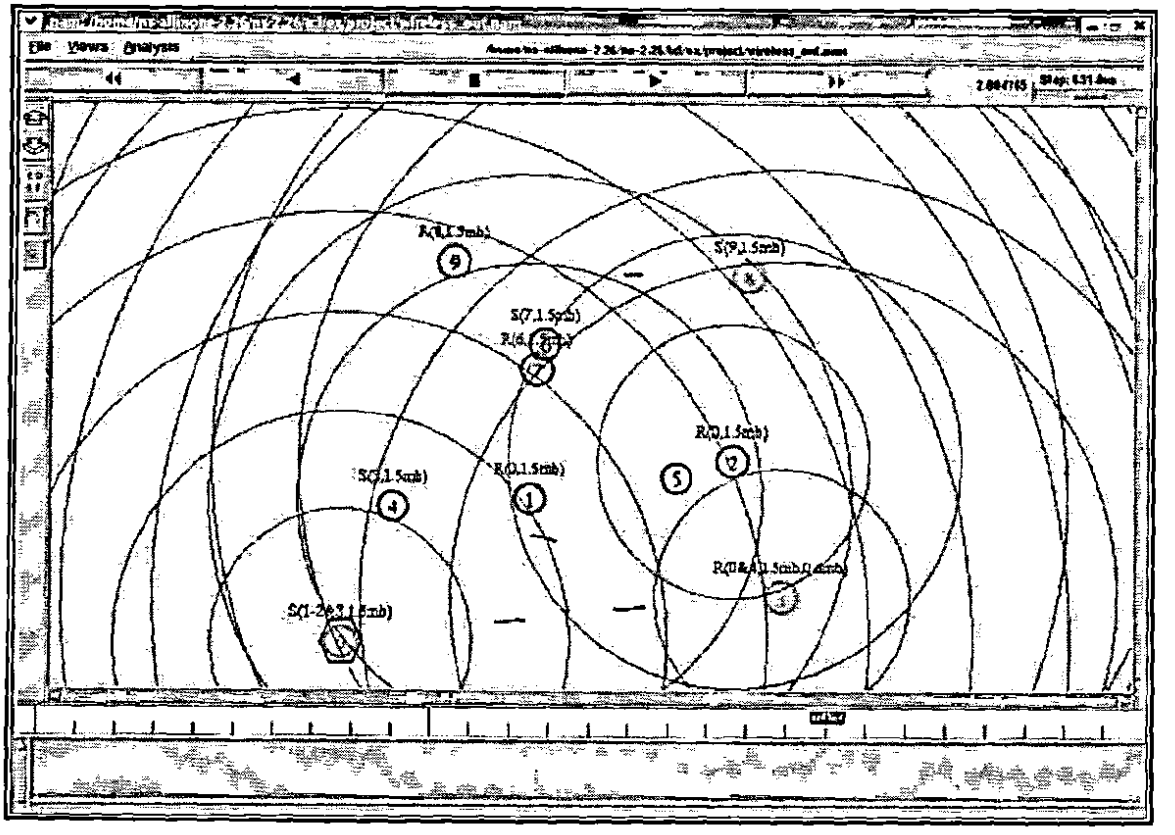


FIG 22: TESTING WITH COMPLEX TOPOLOGY

In the above figure another topology is created. This topology is a complex one consisting of many nodes which are interconnected with each other through wireless links. The performance was better then SRM and UDP.

5.3.4 Testing with Multicasting

The topology given is that of nodes with multicasting support enabled. As expected the results were very encouraging.

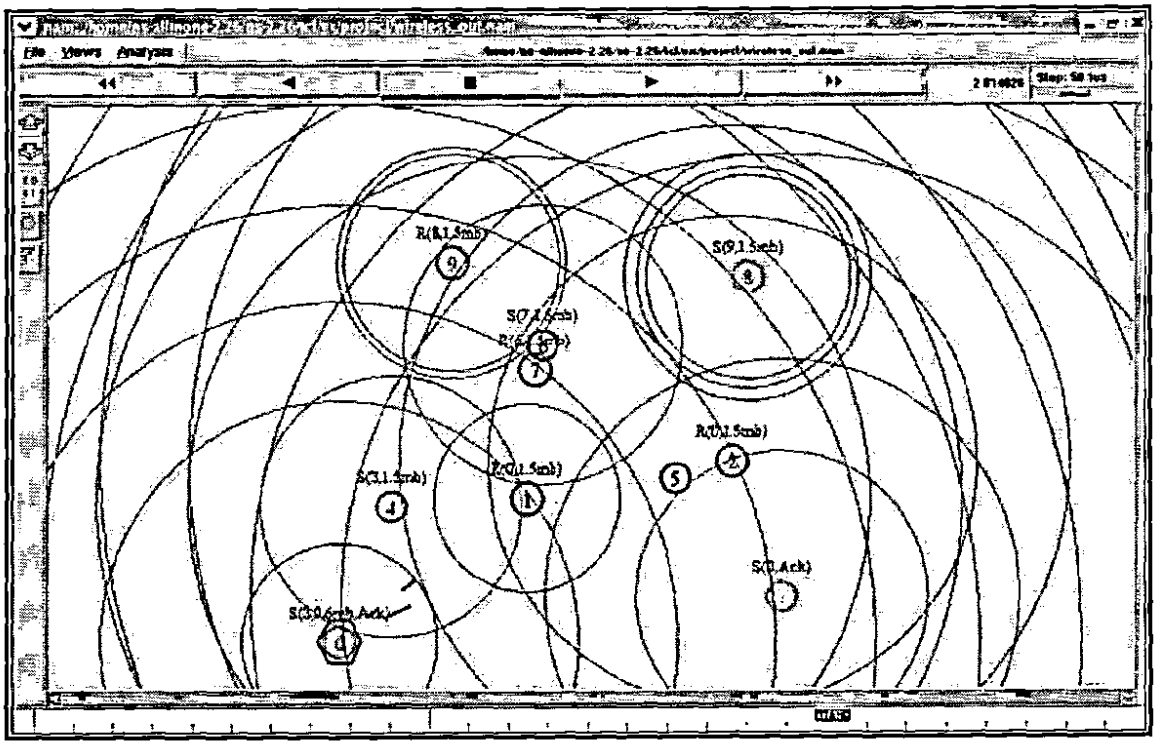


FIG 23: TESTING WITH MULTICASTING

5.3.5 Testing with Trace Files

The file given below is a trace file. It has the standard architecture of ns. The fields denote packet type, *protocol used, arrival time* etc. The trace file contained the complete correct information of the network simulation.

```
+ -t 1.106666667 -s 0 -d -1 -p udp -e 1000 -c 2 -a 0 -i 10 -k AGT
- -t 1.106666667 -s 0 -d -1 -p udp -e 1000 -c 2 -a 0 -i 10 -k AGT
h -t 1.106666667 -s 0 -d -1 -p udp -e 1000 -c 2 -a 0 -i 10 -k AGT -R 3333.48 -D 0.33
+ -t 1.120810887 -s 1 -d -1 -p message -e 32 -c 2 -a 0 -i 11 -k RTR
- -t 1.120810887 -s 1 -d -1 -p message -e 32 -c 2 -a 0 -i 11 -k RTR
h -t 1.120810887 -s 1 -d -1 -p message -e 32 -c 2 -a 0 -i 11 -k RTR -R 3333.48 -D 0.33
+ -t 1.121165887 -s 1 -d -1 -p message -e 84 -c 2 -a 0 -i 11 -k MAC
- -t 1.121165887 -s 1 -d -1 -p message -e 84 -c 2 -a 0 -i 11 -k MAC
h -t 1.121165887 -s 1 -d -1 -p message -e 84 -c 2 -a 0 -i 11 -k MAC -R 3333.48 -D 0.33
+ -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 6 -k RTR
- -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 6 -k RTR
h -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 6 -k RTR
+ -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 7 -k RTR
- -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 7 -k RTR
h -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 7 -k RTR
d -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 6 -k IFQ
+ -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 8 -k RTR
- -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 8 -k RTR
h -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 8 -k RTR
d -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 7 -k IFQ
+ -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 9 -k RTR
- -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 9 -k RTR
h -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 9 -k RTR
d -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 8 -k RTR
d -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 9 -k RTR
+ -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 10 -k RTR
- -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 10 -k RTR
h -t 1.121863650 -s 0 -d 1 -p udp -e 1040 -c 2 -a 0 -i 10 -k RTR
```

FIG 24: TRACE FILE OUTPUT PATTERN

CHAPTER 6

CONCLUSION AND FUTURE WORK

6 Conclusion and Future Work

6.1 Conclusion

We proposed udp-mm, a congestion control multicast transport protocol designed for ad hoc networks. Through simulation, we showed that by performing congestion control, udp-mm adapts well to the various network conditions observed in terms of achieving high packet delivery ratio and low end-to-end latency. We also demonstrated that SRM (designed to provide reliable delivery in wired networks) performed worse than udp-mm. The increase in the SRM control messages results in network congestion, packet drops, and poor network performance because it invokes further control message transmissions. Other metrics (not shown) in the number of receivers experiments are similar to those reported in Providing congestion control, we believe, is the first step in achieving reliable multicast in ad hoc networks.

6.2 Future Work

From the results in this paper, we note that udp-mm still does not guarantee 100% reliability. In general, some of the losses are unavoidable, as they may be caused by network by mobility or hidden terminal effects. Some form of end-to-end reliable retransmission mechanism specifically designed for wireless ad hoc networks is needed in conjunction with congestion control to achieve perfect reliability. The research can be continued in this direction.

APPENDIX-A

NS SIMULATOR ON LINUX

Appendix-A NS Simulator on Linux

Following steps were performed to run NS simulator on Linux.

A.1 Linux Installation

Redhat Linux 9.0 was used. The system installation went smoothly. Custom Installation was selected. Disk Druid window was loaded. The drive partitions were reformatted. Almost all the hardware was detected flawlessly by Linux and and after other settings the installation started. Redhat Linux 9.0 is now on 3 cds with a huge amount of software. The Tcl/Tk package is also installed which is needed by ns. The installation took almost 1:30 hours after which the system rebooted and the linux started.

A.2 NS Installation

First of all install the ns-allinone package. In ns-allinone-xxx\ns-2.26 folder the main ns package resides.

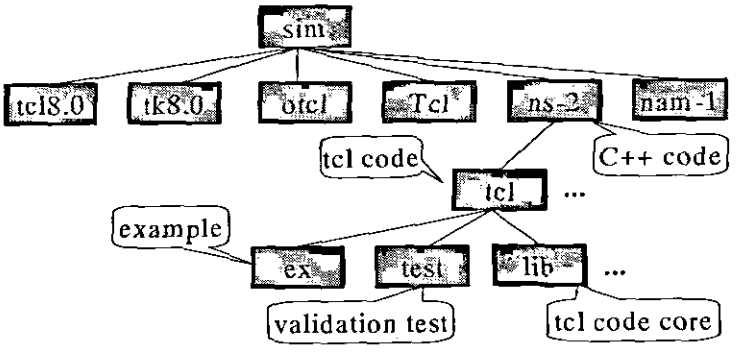


FIG 25: NS DIRECTORY STRUCTURE

Enter into the main folder of ns and type. /configure. This will result in the configuration of the NS simulator. Messages will be displayed and important information will be conveyed. Once this process is over without any errors, cursor reappears on the command line. Now type make. This will compile and install the ns-allinone package

which contains Tcl, Otcl, Tclcl, xgraph etc. This is a lengthy process and requires a fast computer.

```
[root@localhost ns-2.26]# ./configure
loading cache ./config.cache
No .configure file found in current directory
Continuing with default options...
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
checking build system type... i686-pc-linux-gnu
checking for gcc... (cached) gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for c++... (cached) c++
checking whether the C++ compiler (c++  ) works... yes
checking whether the C++ compiler (c++  ) is a cross-compiler... no
checking whether we are using GNU C++... (cached) yes
checking whether c++ accepts -g... (cached) yes
checking how to run the C preprocessor... (cached) gcc -E
checking for ANSI C header files... (cached) yes
checking for string.h... (cached) yes
checking for main in -lXbsd... (cached) no
checking for socket in -lsocket... (cached) no
checking for gethostbyname in -lnsl... (cached) yes
checking for dcgettext in -lintl... (cached) no
checking for getnodebyname in -ldnet_stub... (cached) no
checking that c++ can handle -O2... yes
checking if STL works without any namespace... yes
checking will use STL... yes
checking for tcl.h... -I../include
checking for libtcl8.3... -L../lib -ltcl8.3
checking for init.tcl... ../lib/tcl8.3
checking for tclsh8.3.2... (cached) ../bin/tclsh8.3
checking for tk.h... -I../include
checking for libtk8.3... -L../lib -ltk8.3
checking for tk.tcl... ../lib/tk8.3
checking for otcl.h... -I../otcl-1.0a8
checking for libotcl1.0a8... -L../otcl-1.0a8 -lotcl
checking for tclcl.h... -I../tclcl-1.0b13
checking for libtclcl... -L../tclcl-1.0b13 -ltclcl
checking for tcl2c++... ../tclcl-1.0b13
checking for X11 header files
checking for X11 library archive
checking for XOpenDisplay in -lX11... (cached) no
checking for libXext.a
checking for libtcldbg... no
checking dmalloc... not requested with --with-dmalloc
[root@localhost ns-2.26]#
```

FIG 26: NS PACKAGE COMPILATION

APPENDIX-B

GLOSSARY OF TERMS

Appendix-B Glossary of Terms

This appendix contains terminology that is related to computer networks

AODV: The Ad hoc On Demand Distance Vector routing algorithm is a routing protocol designed for ad hoc mobile networks.

DSDV: Destination Sequenced Distance Vector Protocol

DSR: Dynamic Source Routing

TORA: Temporally-Ordered Routing Algorithm

ARP: The Address Resolution Protocol implemented in BSD style

Bandwidth: Total link capacity of a link to carry information (typically bits)

Channel: The physical medium is divided into logical channel, allowing possibly shared uses of the medium. Channels may be made available by subdividing the medium into distinct time slots, distinct spectral bands, or de-correlated coding sequences.

DropTail: DropTail is a queue. Just a *FIFO* queue.

DSDV: Destination Sequenced Distance Vector Protocol

Flooding: The process of delivering data or control messages to every node within the data network.

Host: Any node that is not a router.

Link: A communication facility or medium over which nodes can communicate at the link layer.

Loop free: A path taken by a packet never transits the same intermediate node twice before arrival at the destination.

Next hop: A neighbour, which has been designed to forward packets along the way to a particular destination.

Neighbor: A node that is within transmitter range from another node on the same channel.

Node: A device that implements IP.

Node ID: Unique identifier that identifies a particular node.

Neighbour-node: A node that is one hop away.

Retransmission: Transmission of a packet with the same TCP packet number but different NS packet ID.

Router: A node that forwards IP packets not explicitly addressed to itself. In case of ad hoc networks, all nodes are at least unicast routers.

Routing table: The table where the routing protocols keep routing information for various destinations. This information can include the next hop and the number of hops to the destination.

Scalability: A protocol is scalable if it is applicable to large as well as small populations.

Throughput: The amount of data from a source to a destination processed by the protocol for which throughput is to be measured for instance, IP, TCP, or the MAC protocol.

BIBLIOGRAPHY AND REFERENCES

BIBLIOGRAPHY AND REFERENCES

[1] Tanenbaum, A. S., Computer Networks. 3rd Ed. Prentice Hall PTR, Upper addle River, New Jersey, U.S.A, 1996.

[2] William Stalling, Modern Computer Networks: Third Edition, Prentice Hall, SA, 1997.

[3] Ns Manual

[4] Fairness Evaluation Experiments for Multicast Congestion Control Protocols by Karim Seada and Ahmed Helmy

[5] An Adaptive Protocol for Reliable Multicast in Mobile Multi-hop Radio Networks by Sandeep K. S. Gupta and Pradip K. Srimani

[6] Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks by Jorjeta G. Jetcheva and David B. Johnson

[7] Neighbour supporting Ad hoc multicast routing protocols by Seungjoon lee and Chongkwon Kim

[8] Multicast Routing Issues in Ad Hoc Networks by Katia Obraczka Gene Tsudik

[9] Bandwidth-Efficient Multicast Routing for Multihop, Ad-Hoc Wireless Networks by Tatsuya Suda, Omochika Ozaki and Jaime Bae Kim

[10] www.isi.edu/nsnam

[11] http://www.isi.edu/nsnam/ns/ns-documentation

[12] http://www.lam-mpi.org/

[13] AMRoute: Ad hoc Multicast Routing Protocol by Mingyan Liu, Rajesh R. Talpade, Anthony McAuley and Ethendranath Bommaiah

[14] The essential guide to wireless communications applications by Andy Dornan Second Edition.

Nasir Mahmood

From: 'John Impagliazzo' [inroads@Mail1.Hofstra.edu]
Sent: Thursday, September 30, 2004 3:39 PM
To: nasir.mehmood@millips.com.pk; stauseef@yahoo.com; wparacha786@yahoo.com
Subject: inroads - 2004 December

Greetings,

I am pleased to inform you that your article entitled
-- CONGESTION CONTROLLED ADAPTIVE MULTICAST IN WIRELESS NETWORK-- is accepted for publication in inroads-the SIGCSE Bulletin. The printed article will contain editorial modifications that will not alter the content or meaning of your work. I am scheduling your work for publication in the 2004 December issue (Volume 36, Number 4) of ACM inroads.

You should know that you would retain copyright ownership of your work, which will become part of the ACM Digital Library. Because of this, I ask you to pay particular attention to the Abstract and to the Keywords, as these two aspects of the publication will be accessible and searchable by the public. Therefore, please ensure that the Abstract (between 50 and 100 words) accurately eflects your work. Also, if you have not yet done so, please include at least three but not more than five keywords reflecting the nature of your article. Do not resend your article; I will patch these two parts into your existing work where necessary. If your work already includes one or both of these parts, I will overwrite them with your updated version.

To ensure publication in a timely manner, kindly REPLY to this email and respond to the following FOUR items.

(1) You do accept publication in inroads.

(2) You did not publish this article else where.

(3) You are including a text version of the Abstract in the body of this email, if it is different from the one appearing in your work. If you require no change, respond below by stating <Same>.
NEW ABSTRACT

(4) You are including a text version of the Keywords in the body of this email, if they are different to the ones appearing in your work. If you require no change, respond below by stating <Same>.
NEW KEYWORDS

NOTE:
I cannot publish your work until you submit to me the complete postal address of each author.

Thank you for your interest in SIGCSE.
John Impagliazzo
Editor-in-Chief
inroads

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
John Impagliazzo, Ph.D. // Editor-in-Chief, inroads Dept of Computer Science // 103 Hofstra University Hempstead, New York 11549-1030 USA
Tel: +1-516-463-6774 // Fax: +1-516-463-5790 // <inroads@Hofstra.edu>
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

CONGESTION CONTROLLED ADAPTIVE MULTICAST IN WIRELESS NETWORK

Waqas Paracha, Malik Nasir Mahmood, Tauseef-ur-Rehman
Faculty of Applied Sciences, International Islamic University, Islamabad, Pakistan

Abstract

The use of MAC protocol combined with hidden terminal problems make wireless network much more sensitive to load and congestion than wired networks. In such an environment we say, that multicast reliability can not be achieved by retransmission of lost packets as it is typically done in wired networks using protocol such as SRM. We suggest that in order to attain reliable multicast delivery in such networks, beside reliability mechanisms, we must also consider two components: reliability and congestion control. In this paper we purpose CCAM congestion controlled adaptive multicast protocol and show that congestion control alone can sufficiently improve reliable packet delivery in wireless networks with compared to traditional wired networks. The proposed protocol also achieves congestion control through high multicast efficiency and low communication overhead.

1. Introduction

An ad-hoc network is a collection of wireless mobile nodes, which form a temporary network without relying on the existing network infrastructure or centralized administration [1]. Example applications include military battlefield scenarios, civilian disaster relief, emergency rescue, and exploration (e.g., hostile environments like space, under water, etc.) operations. The types of scenarios ad hoc networks target make group-oriented applications, such as teleconferencing, multimedia data transmission and data dissemination services, one of the primary classes of applications. Multicast communication is certainly an efficient means of supporting group-oriented applications. In such an environment, we show that error control mechanisms alone do not lead to reliability, as is the case in wired networks. To achieve complete reliability, two complementary steps are required: congestion control and reliable delivery. In this paper, we focus on the congestion control facet.

The remainder of this paper is organized as follows. Section 2 examines the motivation behind our research. Section 3 follows by describing the operations of CCAM, a rate-based congestion controlled multicast transport protocol. Simulation methodology and results are reported in Sections 4 and 5, respectively. Section 6 presents our concluding remarks.

2. Motivation

Reliable multicast transport protocols for wired networks have been a very active research area and the focus of the IETF Reliable Multicast Research Group (RMRG) [9]. Several reliable multicast protocols have been proposed. We hypothesize that the design choices underlying wired reliable multicast transport protocols are not adequate for wireless ad hoc network environments. Ad hoc networks are extremely sensitive to network load and congestion, even more so than wired shared-medium networks, such as Ethernet, because of the hidden terminal problem [8]. However, although reliable multicast is considered one of the fundamental technologies for enabling key applications in ad hoc networks, research in the area is minimal. A couple of exceptions are the Anonymous Gossip (AG) protocol [2] and the work on reliable broadcast by Pagani and Rossi [7] The former exhibits high recovery delay while the latter will likely degrade in dynamic ad hoc network scenarios where topology changes are frequent. In fact, simulation results reported in [7] indicate that the protocol does not perform well in the presence of high node mobility. In our research, we investigate the effects of congestion control on reliable multicast communication in mobile ad hoc networks. We propose CCAM (Congestion-controlled Adaptive Multicast), a rate-based congestion control multicast transport protocol that adapts the transmission rate on the basis of congestion in the network. We compare its performance against plain, unreliable UDP and SRM (Scalable Reliable Multicast) [3] protocol in our comparative study as an example protocol that employs error recovery, but not congestion control, to achieve reliable delivery.

3. Congestion Controlled Adaptive Multicast (CCAM)

CCAM is a simple rate-based reactive congestion control multicast protocol designed for wireless ad hoc networks. As a reactive protocol, CCAM transmits data packets at a rate specified by the application traffic until congestion arises. Congestion is indicated through negative acknowledgements (NACKs) sent back to the source by the multicast receivers that that experience congestion at one instance. Once the source is informed of congestion, it enters the congestion control phase. The source multicasts new "targeted" data packets, with each new data packet instructing a specific receiver to reply with an acknowledgement (ACK). Packets are multicasted to targeted receivers one at a time. The source then waits for an ACK from the targeted receiver or a timeout before moving to the next receiver. When all ACKs are received from the target receivers (an indication that the network is no longer congested), the source exits the congestion control phase and reverts to the initial application traffic rate. Thus, during the congestion control phase, the source clocks its sending rate based on the targeted receiver. A receiver unicasts a NACK to the source when it determines from the packet sequence numbers that it has not received N consecutive data packets from the source. In our experiments, N is chosen to be three since we do not want to perform congestion control for sporadic, occasional losses, such as the ones resulting from wireless medium transmission errors. Figure 1 depicts CCAM's source state transition diagram.

3.1 Detailed Description

Each multicast source *maintains* a Receiver List. Every time a source receives a NACK from a receiver that is not in the Receiver List, the receiver is added to the list. Nodes in *the Receiver List are removed from* the list when the source hears an ACK from them without a request of Ack. In addition, the source keeps track of the end-to-end latency between itself and *each receiver that sent* NACKs (through a timestamp). The source starts multicasting data packets at the rate specified by the application. Upon reception of NACKs, the source adds the receivers that sent the NACKs to the Receiver List and enters the congestion control phase. During congestion control phase, the source selects a receiver and multicasts the next data packet with an indication in the header that instructs the target receiver to reply with an ACK. If the ACK is received from the receiver before a timeout (based on the measured end-to-end delay between the receiver and the source), the source presumes that receiver is no longer experiencing congestion. Next node in the Receiver List is chosen for the next ACK reception. When all receivers in the Receiver List ACK the source, the source assumes the network is no longer congested and exits the congestion control phase reverting to the initial application traffic rate. If a timeout occurs before the ACK reception, the current target receiver is again transmitted the data packet with an Ack request and the next receiver is selected. Thus, the source clocks its sending rate based on the response from the receivers. A receiver unicasts a NACK to the source when it determines from the packet sequence numbers that it has not received N consecutive data packets from the source. In our experiments, N is chosen to be three since we do not want to perform congestion control for sporadic, occasional losses, such as the ones resulting from wireless medium transmission errors. Figure 1 depicts CCAM's source state transition diagram.

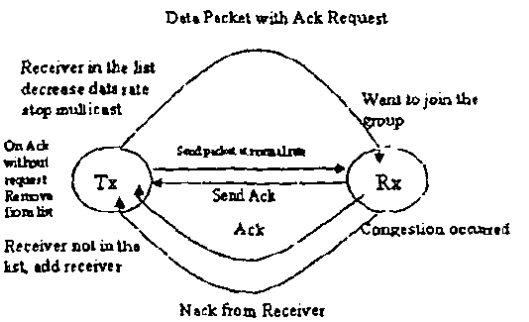


Figure 1. CCAM state transition diagram

3.2 Observations

The proposed scheme reacts to every NACK sent to the source. As an alternate scheme, the receiver experiencing the longest latency can be used to trigger congestion control. CCAM does not attempt to multicast and/or suppress NACKs (as in SRM for example). Multicasting and suppressing NACKs has proven to work well in wired networks because packet losses are correlated; that is, packet losses observed by upstream nodes are also experienced by downstream nodes [3]. However, in ad hoc networks, losses are generally not correlated

since there can be multiple paths to reach a destination. This is especially true in the case of using a mesh-structured protocol (e.g., On-Demand Multicast Routing Protocol [5], Core-Assisted Mesh Protocol [4]) as the underlying multicast routing protocol. It is important to note that although using N consecutive packet losses as a criteria for sending NACKs applies well when high reliability is the goal, other alternatives are also possible. For instance, if an application operates on a less stringent reliability constraint, the receivers may opt to send NACKs only if the packet loss rate reaches a certain threshold or if the delay becomes intolerable. Other schemes may also apply. Thus, the triggering of NACKs and congestion control mechanism can be geared towards the deployed application. Finally, it should be pointed out that CCAM reduces the multicast delivery rate to what is acceptable to a set of slower, more congested receivers. This is appropriate in applications where all parties must have approximately the same quality of information to operate properly (e.g., audio and video resolution). In other applications, it may be more advantageous to selectively reduce the rate and degrade the video quality on congested paths by using layered encoding and selective dropping.

4. Simulation Methodology

To analyze the performance of CCAM, we compare CCAM against UDP and SRM running on top of the Ad hoc On- Demand Distance Vector Protocol (AODV) [5]. UDP is chosen as it provides the basic multicast support without guaranteeing reliability; therefore, any reliable multicast algorithm should demonstrate improvements over UDP. SRM is elected because it employs error-, but not congestion control, to achieve reliable delivery. AODV is selected for this study as it is shown to perform well in ad hoc networks. However, other ad hoc multicast routing protocols can also be used.

The performance metrics we examine are packet delivery ratio, control overhead, end-to-end delay and total number of data packets received by multicast receivers. The packet delivery ratio is defined as the number of data packets received by the multicast members over the number of data packets the members are supposed to receive. This metric measures the effectiveness of a protocol (or how reliable it is). Control overhead is defined as the total number of all packets (data and control) transmitted by all the nodes in the routing and transport layer protocols over the number of data packets received by the multicast receivers. It is used to assess protocol efficiency. End-to-end delay measures the data packet latency from the source to the destination and evaluates the protocol's timeliness.

4.1 Simulation Platform and Parameters

We use the NS2 simulator [10] for performance evaluation. Tables 1 through Table 3 list the parameters that are constant throughout our simulation. Parameters that vary are detailed in the appropriate sections.

Table 1. Simulation Environment Parameters

Parameter	Value
Terrain	1500 m x 1500 m
Number of nodes	10
Node placement	Random
Multicast protocol	AODV
MAC protocol	IEEE 802.11 DCF
Bandwidth	2Mb/s
Propagation	Two-ray ground reflection
Maximum radio range	500 m

Table 2. SRM Parameters

Parameter	Value
C_1, C_2	2
D1, D2	1
Max repair request retry	4

Table 3. CCAM Parameters

Parameter	Value
N	3

5. Result and Analysis

We present the performance of CCAM by comparing against UDP and SRM multicast under various ad hoc network scenarios by varying the traffic rate, number of receivers and mobility. As previously mentioned, the goal comparing a wired reliable multicast transport (i.e., SRM) to CCAM is to show that congestion control alone outperform the use of pure error control techniques without congestion control) in ad hoc network scenarios achieving reliable delivery.

5.1 Traffic Rate

In the traffic rate experiments, we randomly choose multicast sources and ten multicast receivers. All are stationary in these

experiments. We vary the application driven" data packet interdeparture time at source from 500 ms (2 packets per second) to 100 ms (10 packets per second). This is the maximum sending rate, i.e., the rate sources send if no congestion is experienced. During congestion, CCAM lowers the sending rate based on its congestion control algorithm.

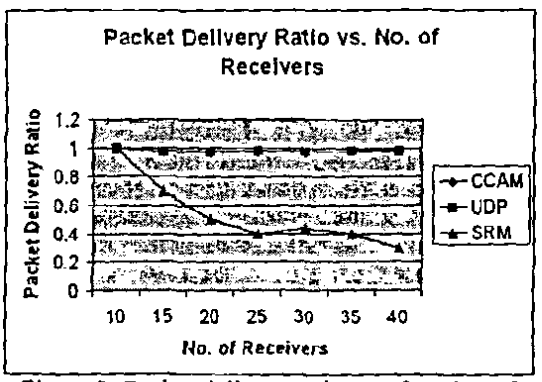


Figure 2. Packet delivery ratio as a function of traffic rate

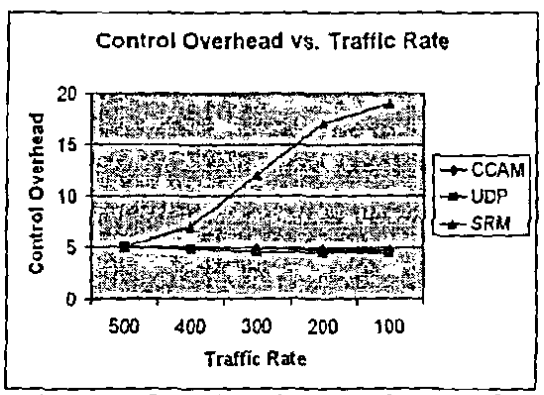


Figure 3. Control overhead as a function of traffic rate

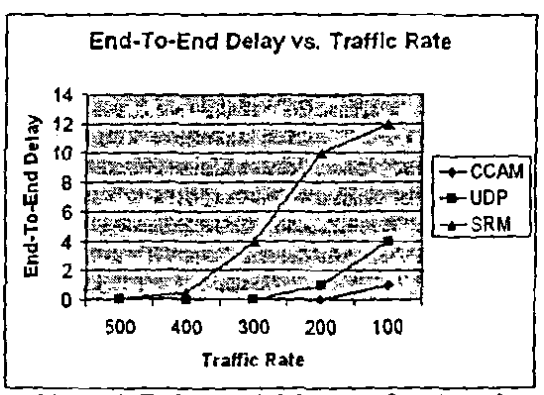


Figure 4. End-to-End delay as a function of traffic rate

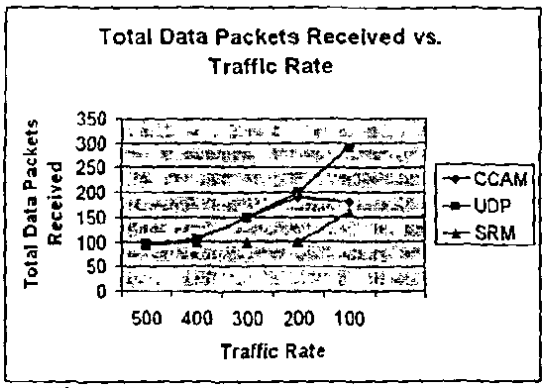


Figure 5. Total data packets received as a function of traffic rate

We observe from Figure 2 that under light load (500 ms), SRM achieves near perfect reliability. However, as the traffic rate increases beyond 300 ms of interdeparture time, SRM performance starts to suffer and dramatically drops to approximately 30% packet delivery ratio. This degradation is due to the high contention the network experiences as the traffic rate and network load grow; high contention results in increased packet losses. This behavior is well known for CSMA protocols [8], which IEEE 802.11 is based on in the broadcast mode. Surprisingly, SRM performs even worse than UDP (which provides no reliability at all). As the network load increases, the packet loss grows and more SRM control messages are sent for recovery, as evident from Figure 3 (CCAM and UDP exhibit approximately the same overhead, thus the CCAM curve is hidden behind the UDP curve). SRM's poor performance is due to the fact that it attempts to recover the dropped packets through repair request and repair. The repair request and repair messages only add fuel to the fire; they simply contribute to more network congestion and therefore are counterproductive as the network is already saturated. Further analysis into the simulation statistics and trace files reveals that there are more packet drops from the queue maintained at each node as traffic rate increases under UDP and especially for SRM. CCAM, on the other hand, shows resiliency to the network load by achieving a packet delivery ratio of 94% or higher under all traffic loads examined. CCAM's robustness is due to congestion and rate control. Figure 5 shows that CCAM achieves a throughput performance that is between UDP and SRM. CCAM also outperforms both UDP and SRM in latency (Figure 4). SRM achieves the worst delay because of its lost packet retransmissions. Although neither UDP nor

CCAM retransmit lost packets, CCAM still provides better delay performance. As UDP does not adjust its sending rate, the queue size at each node is larger than in CCAM, which leads to higher delay.

5.2 No. of Receivers

In these experiments, five sources are randomly chosen while the number of receivers is varied between 10 and 40. The packet interdeparture time is fixed at 500 ms for each source. Mobility is not considered. The results are shown below.

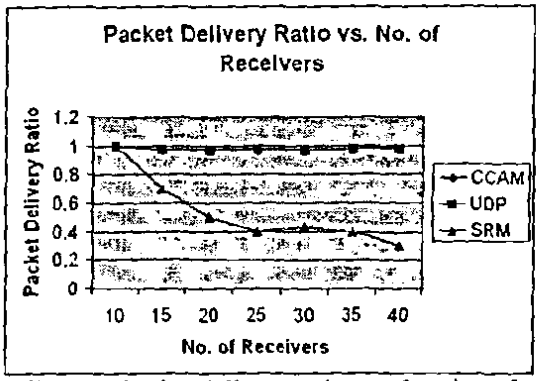


Figure 6. Packet delivery ratio as a function of no. of receivers

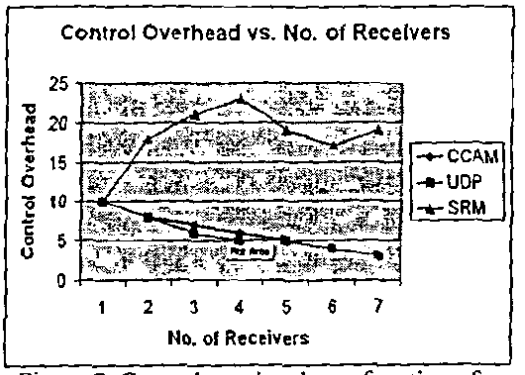


Figure 7. Control overhead as a function of no. of receivers

We observe that varying the number of multicast receivers has no impact on delivering data packets when using UDP. This finding is consistent with [5] where it was found that the performance of AODV under UDP is not affected by the multicast group size in the network due to the mesh created by AODV; the mesh allows for alternate routes, which leads to better reliability.

CCAM experiences approximately the same packet delivery ratio as UDP under all the metrics being investigated. Under this scenario, packets drops are rare, as evident by the high packet delivery ratio of UDP. Since the receivers in CCAM transmit NACKs to the source only if N consecutive packet loss is determined, this scenario has little impact on CCAM. However, the same is not true for SRM. We see that SRM performs worse as the number of receivers grows. As we have more receivers in the network, there is a higher probability that some data packets are dropped before reaching the destinations. Such condition is detrimental to SRM since it must use more control messages to perform recovery. The increase in the SRM control messages results in network congestion, packet drops, and poor network performance because it invokes further control message transmissions. Other metrics (not shown) in the number of receivers experiments are similar to those reported in section 4.

5.3 Mobility

In the mobility experiments, we randomly choose five sources and ten receivers, with a packet interdeparture rate of 500 ms. we vary the mobility speed from 0 m/s to 50 m/s. Based on [5], we expect that mobility would have little impact on UDP performance when using AODV because of the redundant transmission of the forwarding group mash topology. The results confirm our expectations.

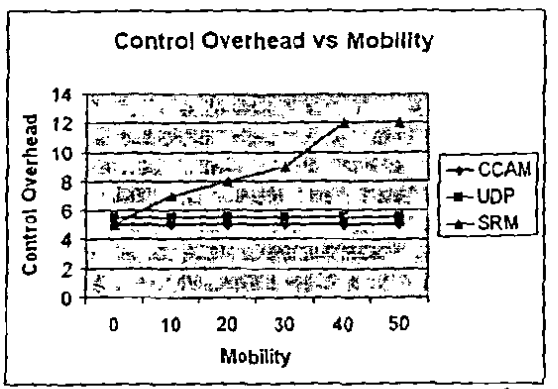


Figure 6. Control overhead as a function of mobility speed

Mobility, similar to the number of receivers experiments, has little bearing on CCAM and UDP. In addition to demonstrating worse packet delivery ratio (not shown), SRM displays the highest control overhead. CCAM, however, competes favorably with UDP.

8 Conclusion and Future Work

We proposed CCAM, a congestion control multicast transport protocol designed for ad hoc networks. Through simulation, we showed that by performing congestion control, CCAM adapts well to the various network conditions observed in terms of achieving high packet delivery ratio and low end-to-end latency. We also demonstrated that SRM (designed to provide reliable delivery in wired networks) performed worse than CCAM (and even UDP) in terms of packet delivery ratio, control overhead and end-to-end delay when deployed in ad hoc networks. The major offender is the SRM extra load introduced by the control packets needed to recover from losses. Providing congestion control, we believe, is the first step in achieving reliable multicast in ad hoc networks. From the results in this paper, we note that CCAM still does not guarantee 100% reliability. In general, some of the losses are unavoidable, as they may be caused by network by mobility or hidden terminal effects. Some form of end-to-end reliable retransmission mechanism specifically designed for wireless ad hoc networks is needed in conjunction with congestion control to achieve perfect reliability. We are currently continuing research in this direction.

9 Reference

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song, "PARSEC: A parallel simulation environment for complex system," IEEE Computer Magazine, vol. 31, no. 10, Oct.1998, pp. 77-85.

[2] R. Chandra, V. Ramasubramanian, and K.P. Birman, "Anonymous Gossip: Improving Multicast Reliability in Mobile Ad-Hoc Networks," Proceedings of IEEE ICDCS 2001, Mesa, AZ, Apr. 2001, pp. 275-283.

[3] S. Floyd, V. Jacobson, S. McCanne, C. Liu, and L. Zhang, "A reliable multicast framework for lightweight sessions and application-level framing," IEEE/ACM Transactions on Networking, vol. 5, no. 6, Dec. 1997, pp. 784-803.

[4] J.J. Garcia-Luna-Aceves and E.L. Madruga, "The Core-Assisted Mesh Protocol, "IEEE Journal on Selected Areas in Communications, vol. 17, no. 8, Aug. 1999, pp. 1380-1394.

[5] E. M. Royer and C. E. Perkin, "Multicast Operations of the *Ad-hoc On-Demand Distance Vector Routing* Protocol. Proceedings of the 5th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom), pages 207-218,Seattle, WA, August 1999.

[6] K. Obraczka, "Multicast Transport Mechanisms: A *Survey and Taxonomy*," *IEEE Communications* Magazine, vol. 36, no. 1, Jan. 1998, pp. 94-102.

[7] E. Pagani and G.P. Rossi, "Providing Reliable and Fault Tolerant Broadcast Delivery in Mobile Ad-Hoc Networks," ACM/Baltzer Mobile Networks and Applications, vol. 4, no. 3, Aug. 1999, pp. 175-192.

[8] A.S. Tanenbaum, Computer Networks, Third Edition, Prentice Hall, New Jersey, 1996.

[9] http://www.ietf.org/html.charters/rmt-charter.html.

[10] http:// www.isi.edu\nsnam/ns/index.html