P - 2586

PEER TO PEER FILE SHARING BASED ON NETWORK CODING

A PROJECT REPORT

Submitted by

JAYAPRADEEP.M.R.	71205205024
LOGESWARAN.A	71205205302
PRABHAKARAN.S	71205205303

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY,COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

28th, APRIL 2009

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**PEER TO PEER FILE SHARING BASED ON NETWORK CODING**" is the bonafide work of JAYAPRADEEP M R, LOGESWARAN A, PRABHAKARAN S who carried out the project work under my supervision.

[illegible]

SIGNATURE

Dr.S.Thangasamy

DEAN

Department of
Computer Science and Engineering

Kumaraguru College Of Technology
Coimbatore 641006

[illegible] 29/4/09

SIGNATURE

Ms.S.Sathyavathi M.E

SUPERVISOR

Department of Information Technology
Kumaraguru College Of Technology

Coimbatore-641006

The candidates with University Register Nos. **71205205024, 71205205302 & 71205205303** examined by us in the project viva-voce examination held

28th , APRIL 2009

[illegible] 28/4/09

INTERNAL EXAMINER

[illegible] 28/4/09

EXTERNAL EXAMINER

DECLARATION

We,

JAYAPRADEEP M R	Reg.No: 71205205024
LOGESWARAN A	Reg.No: 71205205302
PRABHAKARAN S	Reg.No: 71205205303

hereby declare that the project entitled **"Peer to peer file sharing based on network coding"**, submitted in partial fulfillment to Anna University as the project work of Bachelor of Technology (Information Technology) degree, is a record of original work done by us under the supervision and guidance of Department of Information Technology, Kumaraguru College of Technology,Coimbatore.

Place: Coimbatore

Date: 28th , APRIL 2009

[illegible] [illegible] [illegible]

[JAYAPRADEEP M R] [LOGESWARAN A] [PRABHAKARAN S]

Project Guided by

[illegible] 24/4/09

[Ms.S.Sathyavathi M.E]

ACKNOWLEDGEMENT

We express our sincere thanks to our Chairman **Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc., F.I.E.,** Vice Chairman **Dr.K. Arumugam B.E., M.S., M.I.E.,** Correspondent **Shri.M.Balasubramaniam and Joint Correspondent Dr.A.Selvakumar** for all their support and ray of strengthening hope extended. We are immensely grateful to our Principal **Dr.Joseph V.Thanikal M.E., Ph.D., PDF, CEPIT,** for his invaluable support to the outcome of this project.

We are deeply obliged to **Dr.S.Thangasamy,** Dean, Department of Computer Science and Engineering for his valuable guidance and useful suggestions during the course of this project.

We also extend our heartfelt thanks and gratitude to our project co-ordinator, **Ms.L.S Jayashree M.E., Ph.D.,** Asso.Prof., Department of Information Technology for providing us her support which really helped us.

We are indebted to our project guide and extend our gratitude towards **Ms.S.Sathyavathi M.E.,** Lecturer, Department of Information Technology for her helpful guidance and valuable support given to us throughout this project.

We thank the teaching and non-teaching staffs of our Department for providing us the technical support during the course of this project. We also thank all of our friends who helped us to complete this project successfully.

ABSTRACT

Network coding is a promising enhancement of routing to improve network throughput and provide high reliability. It allows a node to generate output messages by encoding its received messages. Peer-to-peer networks are a perfect place to apply network coding due to two reasons: the topology of a peer-to-peer network is constructed arbitrarily, thus it is easy to tailor the topology to facilitate network coding; the nodes in a peer-to-peer network are end hosts which can perform more complex operations such as decoding and encoding than simply storing and forwarding messages.

In this project, the system propose a scheme to apply network coding to peer-to-peer file sharing which employs a peer-to-peer network to distribute files resided in a web server or a file server. The scheme exploits a special type of network topology called combination network. It is proved that combination networks can achieve unbounded network coding gain measured by the ratio of network throughput with network coding to that without network coding. The scheme encodes a file into multiple messages and divides peers into multiple groups with each group responsible for relaying one of the messages.

The encoding scheme is designed to satisfy the property that any subset of the messages can be used to decode the original file as long as the size of the subset is sufficiently large. To meet this requirement, the system first define a deterministic linear network coding scheme which satisfies the desired property, then the system connect peers in the same group to flood

the corresponding message, and connect peers in different groups to distribute messages for decoding.

Moreover, the scheme can be readily extended to support topology awareness to further improve system performance in terms of throughput, reliability and link stress. The simulation results show that the new scheme can achieve 15%-20% higher throughput than the previous scheme which does not employ network coding. It achieves good reliability and robustness to link failure or churn.

CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

PPFEED	-Peer -To-Peer File Sharing Based on Network Coding
NS2	-Network Simulator Version 2
TCP	-Transmission Control Protocol
UDP	-User Datagram Protocol
MAC	-Message Authentication Code
NAM	-Network Animation
TCL	-Tool Command Language
OTCL	-Object Tool Command Language
RED	-Random Early Discard
CBR	-Constant Bit Rate
CBQ	-Constant Bit Queue

INTRODUCTION

1.INTRODUCTION

1.1 BACKGROUND STUDIES:

In the last several years, the Internet has witnessed tremendous increase of different types of web-based applications, ranging from web-based file sharing to video broadcasting/conferencing. Web-based applications have gained more and more interests due to the flexibility and easy accessibility. Many such applications involve one source (server) and multiple destinations (receivers). Due to lack of multicast support over the Internet, these applications suffer from the scalability problem, which limits the number of receivers involved.

Peer-to-peer is a new technology that can implement multicast at the application layer. By incorporating peer-to-peer technology into web-based applications, the scalability problem can be eliminated. In this project, the system consider applying peer-to-peer technology to file sharing services, for example, a web server or a file server holds a file that is requested by multiple clients (receivers). The topology of such an application can be represented by a multicast network. When peer-to-peer technology is used, the receivers help forward the file for each other besides receiving the file.

Network coding is another promising technology that is employed to improve system throughput and reliability. Multicast is a typical application which can benefit from network coding. In fact, Ahlswede et al. [1] has generally proved that if network coding is allowed in network nodes, then the capacity (i.e., the minimum of the max-flows between the source and every receiver, which is considered as the maximum rate the source can send messages to the receivers) can be achieved, in other words, network capacity can be fully utilized in multicast.

A multicast network can be modeled by a directed graph where each node has multiple incoming edges and outgoing edges. Then constructing a network coding scheme for a multicast network is equivalent to assigning a function to each edge which defines the mix operation for that edge. This function is called edge function. Suppose the edge is one of the outgoing edges of node, then its edge function takes the messages on the incoming edges of node as input and outputs a message to be sent on itself. A network coding scheme is the union of the edge functions of all the edges in the multicast network.

1.2 PROBLEM DEFINITION AND OBJECTIVES:

In the last several years, the Internet has witnessed treatment as increase of different types of web-based applications, ranging from web-based file sharing to video broadcasting/conferencing. Web-based applications have gained more and more interests due to the flexibility and easy accessibility. Many such applications involve one source and multiple destinations.

Due to lack of multicast support over the Internet, these applications suffer from the scalability problem, which limits the number of receivers involved. Peer-to-peer is a new technology that can implement multicast at the application layer. By incorporating, peer-to-peer technology into web-based applications, the scalability problem can be eliminated.

1.3 RELEVANT PROBLEM TO THE CURRENT SCENARIO:

A valid network coding scheme is a network coding scheme in which all the receivers can decode the original messages based on the messages they receive. Let messages be represented by symbols over a Galois field GF(q) where q is a prime number. Linear network coding refers to network coding schemes where the edge functions are linear functions over GF(q). If node w has e incoming edges {e1,e2,….ec} and one of the outgoing edges is e, then the edge function of edge e in a linear network coding scheme can be denoted by a vector Ve = {v1,v2,…vc}.

The message on edge e is generated by the linear function $M_e = v_1 M_{e_1} + v_2 M_{e_2} + \cdots + v_c M_{e_c}$, where M_{ei} represents the message transmitted on edge e_i. In a network with multicast capacity k, the source can transmit k messages simultaneously. If the system represent the k messages by a k dimensional vector over GF(q), the messages transmitted over edge e can be represented by the product of vector and another k dimensional vector Ve. Vector Ve is called the edge vector of edge e. The system has Ve = $v_1 V'_{e_1} + v_2 V'_{e_2} + \cdots + v_c V'_{e_c}$

A valid linear network coding scheme is a linear network coding scheme where the rank of the matrix composed by the edge vectors of the receiver's incoming edges is equal to the dimension of, which is in this case. In recent years, there has been some work on linear network coding in the literature. Li, et al. [2] showed that linear network coding over a finite field is sufficient to achieve multicast capacity. Kotter, et al. gave an algebraic characterization for a linear network coding scheme in [3].

They also gave an upper bound on the field size and a polynomial time algorithm to verify the validity of a network coding scheme. Ho, et al. presented a random linear network coding approach in which nodes generate edge vectors randomly. Clearly, the linear network coding scheme generated by this approach is not always valid. In [6], Lun, et al. proposed a distributed algorithm to find a subgraph of the original topology such that the link cost can be minimized without sacrificing multicast capacity.

LITERATURE REVIEW

2. LITERATURE REVIEW

With reference to the random network coding, Jaggi, et al. proposed a polynomial deterministic algorithm in [14] which can construct deterministic linear network coding schemes for multicast networks. Peer-to-peer (overlay) networks are a perfect place to apply network coding due to two reasons: the topology of a peer-to-peer network is constructed arbitrarily. It is easy to tailor the topology to facilitate network coding; the nodes in a peer-to-peer network are end hosts which can perform more complex operations, such as decoding and encoding, than simply storing and forwarding messages.

In [8], linear network coding was applied to application layer multicast, in which a rudimentary mesh graph is first constructed, and on top of it a rudimentary tree is formed. Then a multicast graph is constructed, which is a subgraph of the rudimentary mesh and a supergraph of the rudimentary tree. The multicast graph constructed this way is 2-redundant, which means that each receiver has two disjoint paths to the source. By taking advantage of the 2-redundancy property of the multicast graph, a light-weight algorithm generates a sequence of 2-dimensional transformation vectors which are linearly independent. These vectors are assigned to the edges as their edge functions.

However, the project did not discuss how to process dynamic joining or leaving of peers, while dynamic membership is a common phenomenon in peer-to-peer networks. Moreover, the 2-redundancy property limits the minimum cut of the multicast graph, which in turn limits network throughput. In [15], random network coding was applied to content distribution, in which nodes encode their received messages with random coefficients. Compared to deterministic network coding, random network coding is inherently distributed.

In random network coding, nodes can determine the edge functions of its outgoing edges independently by generating random coefficients for the edge functions. The advantage of random network coding is that there is no control overhead to construct and maintain a linear coding scheme among nodes. However, the edge vectors of a receiver's incoming edges may not be linearly independent. In other words, a receiver may not recover the original messages even it receives or more messages (here is the multicast capacity of the multicast network).

To reduce the probability of failing to decode messages, it is required to encode over a very large field. Another drawback of random network coding is the increased data traffic. As there is no deterministic path for data delivery, all the nodes take part in relaying the data to the receivers even if it is not necessary. As a result, the same message may be transmitted through the same link multiple times. In this project, the system aim at providing an efficient and reliable file sharing service over peer-to-peer networks by utilizing network coding. The system call it Peer-to-Peer File sharing based on network coding, or PPFEED for short. The system utilizes a special type of network with a regular topology called combination network.

It was demonstrated in [12] that when the network size increases, this type of network can achieve unbounded network coding gain measured by the ratio of network throughput with network coding to that without network coding. The basic idea of PPFEED is to construct an overlay network over the source and the receivers such that it can be decomposed into multiple combination networks. Compared to [8], the approach can accommodate dynamic membership and construct a much simpler overlay network topology in different values.

Compared to [15], the network coding scheme is deterministic, which means that the validity of the coding scheme is guaranteed. The data traffic is then minimized so that the same messages are transmitted through an overlay link at most once. Also, system reliability is improved dramatically with little overhead. In addition, PPFEED can be extended to support topology awareness.

DETAILS OF METHODOLOGY EMPLOYED

3. METHODOLOGY EMPLOYED

3.1 COMBINATION NETWORK:

A combination network is a multicast network with a regular topology. The topology of a combination network is a regular graph which contains three types of nodes: source node, relay node and receiver node. A combination network contains a source node which generates messages, relay nodes which receive messages from the source node and relay them to the receiver nodes, and receiver nodes which receive messages from the relay nodes. There are links connecting the source node to the _ relay nodes respectively.

For every node out of the relay nodes, there are links connecting them to a receiver node, different combinations, and receiver nodes. Fig. 1 shows a combination network for n=4 and k=2, usually denoted as a C^2_4 combination network.

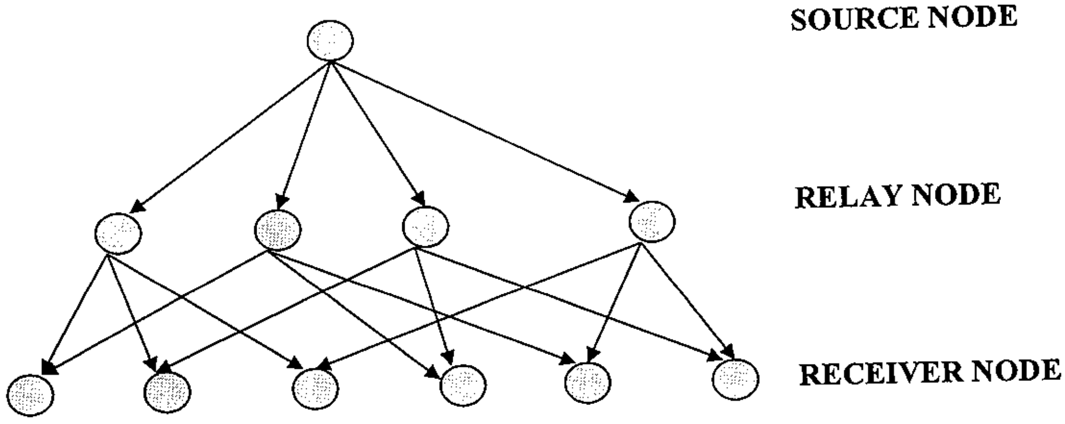


Fig 1: Combination network

Combination networks have good performance with respect to network coding gain. It was proved that network coding gain (i.e., the ratio of throughput with network coding to that without network coding) is unbounded when both n and k approach infinity. For a C^k_n combination network, the multicast capacity is k as each receiver node has k disjoint paths to the source node. This implies that to achieve the multicast capacity in a combination network, the minimum subgraph to deploy network coding is the ntire topology of the combination network.

3.2 LINEAR NETWORK CODING:

The system has discussed the good properties of combination networks. The next issue is how to design a valid linear network coding scheme on a combination network. Before the system go into details of the linear network coding scheme construction, the system first determine an appropriate value for k, which is the multicast capacity of a combination network. On one hand, given the number of relay nodes n, the network coding gain reaches its maximum when k=n/2.

On the other hand, k should not be too large because a large will increase the link stress and overhead. Hence, as a tradeoff, in this project the system only consider combination networks with n=4, k=2 or n=6, k=3 where the network coding gain is 1.5 and 2, respectively. As mentioned earlier, constructing a valid linear network coding scheme is equivalent to assigning each edge an edge vector such that the vectors of the incoming edges of a receiver are linearly independent.

3.3 HARDWARE REQUIREMENTS:

PROCESSOR	:	PENTIUM IV 2.26GHz
RAM	:	256 MB
HARD DISK	:	80 GB
MONITOR	:	SVGA COLOR 15"
NETWORK CARD	:	1MBPS

3.4 SOFTWARE REQUIREMENTS:

OPERATING SYSTEM	:	LINUX 8.0 PCQ
SCRIPT LANGUAGE	:	TCL
ANIMATOR	:	NAM
OUTPUT FILE	:	Transcript file
GRAPHICAL TOOL	:	X - GRAPH

3.5 SOFTWARE DESCRIPTION:

NS2

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkely written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks.

Overview

NS is an event driven network simulator developed at UC Berkeley that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and VBR, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations.

The NS project is now a part of the VINT project that develops tools for simulation results display, analysis and converters that convert network topologies generated by well-known generators to NS formats. Currently, NS (version 2) written in C++ and OTcl (Tcl script language with Object-oriented extensions developed at MIT) is available. This document talks briefly about the basic structure of NS, and explains in detail how to use NS mostly by giving examples.

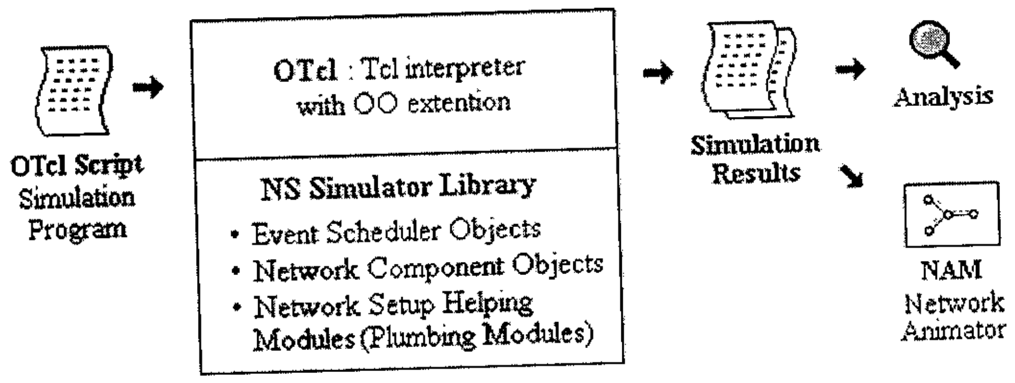


Figure 2. Simplified User's View of NS

As shown in Figure 1, in a simplified user's view, NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object). In other words, to use NS, you program in OTcl script language.

To setup and run a simulation network, a user should write an OTcl script that initiates an event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler. The term "plumbing" is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the "neighbor" pointer of an object to the address of an appropriate object. When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object. This may sound like complicated job, but the plumbing OTcl modules actually make the job very easy. The power of NS comes from this plumbing.

Another major component of NS beside network objects is the event scheduler. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event.

Network components communicate with one another passing packets, however this does not consume actual simulation time. All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet. For example, a network switch component that simulates a switch with 20 microseconds of switching delay issues an event for a packet to be switched to the scheduler as an event 20 microsecond later. The scheduler after 20 microsecond dequeues the event and fires it to the switch component, which then passes the packet to an appropriate output link component.

Another use of an event scheduler is timer. For example, TCP needs a timer to keep track of a packet transmission time out for retransmission (transmission of a packet with the same TCP packet number but different NS packet ID). Timers use event schedulers in a similar manner that delay does.

The only difference is that timer measures a time value associated with a packet and does an appropriate action related to that packet after a certain time goes by, and does not simulate a delay.

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object.

The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTcl. Likewise, an object (not in the data path) can be entirely implemented in OTcl. Figure 2 shows an object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

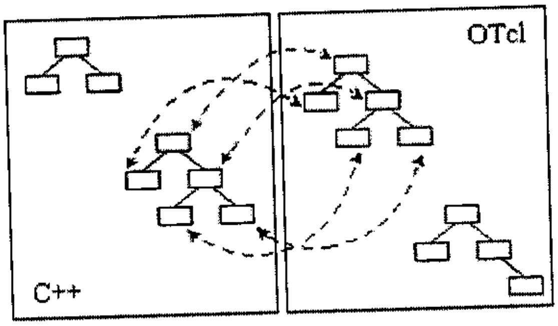


Figure 3. C++ and OTcl: The Duality

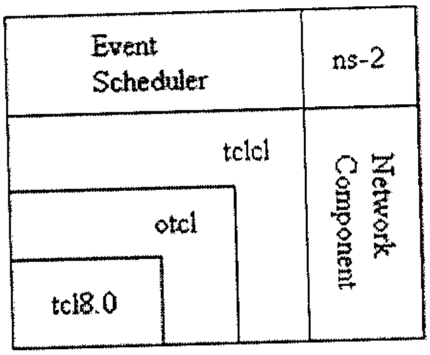


Figure 4. Architectural View of NS

Figure 3 shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is a OO extended Tcl interpreter with network simulator libraries.

As shown in Figure 1, when a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input Tcl (or more specifically, OTcl) script. The data can be used for simulation analysis (two simulation result analysis examples are presented in later sections) or as an input to a graphical simulation display tool called Network Animator (NAM) that is developed as a part of VINT project. NAM has a nice graphical user interface similar to that of a CD player (play, fast forward, rewind, pause and so on), and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis.

OTcl: The User Language

NS is basically an OTcl interpreter with network simulation object libraries. It is very useful to know how to program in OTcl to use NS. This section shows an example Tcl and OTcl script, from which one can get the basic idea of programming in OTcl.

Example 1 is a general Tcl script that shows how to create a procedure and call it, how to assign values to variables, and how to make a loop. Knowing that OTcl is Object-orieneted extension of Tcl, it is obvious that all Tcl commands work on OTcl - the relationship between Tcl and Otcl is just same as C and C++. To run this script you should download ex-tcl.tcl, and type "ns ex-tcl.tcl" at your shell prompt - the command "ns" starts the NS (an OTcl interpreter). You will also get the same results if you type "tcl ex-tcl.tcl", if tcl8.0 is installed in your machine.

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}

# Calling the "test" procedure created above
test
```

Example 1. A Sample Tcl Script

In Tcl, the keyword **proc** is used to define a procedure, followed by an procedure name and arguments in curly brackets. The keyword **set** is used to assign a value to a variable. [**expr** ...] is to make the interpreter calculate the value of expression within the bracket after the keyword. One thing to note is that to get the value assigned to a variable, **$** is used with the variable name. The keyword **puts** prints out the following string within double quotation marks. The following shows the result of Example 1.

```
k < 5, pow = 1.0
k < 5, pow = 1120.0
k < 5, pow = 1254400.0
k < 5, pow = 1404928000.0
k < 5, pow = 1573519360000.0
k >= 5, mod = 0
k >= 5, mod = 4
k >= 5, mod = 0
k >= 5, mod = 0
k >= 5, mod = 4
```

Network Components:

Compound network components shown below a partial OTcl class hierarchy of NS, which will help understanding the basic network components..

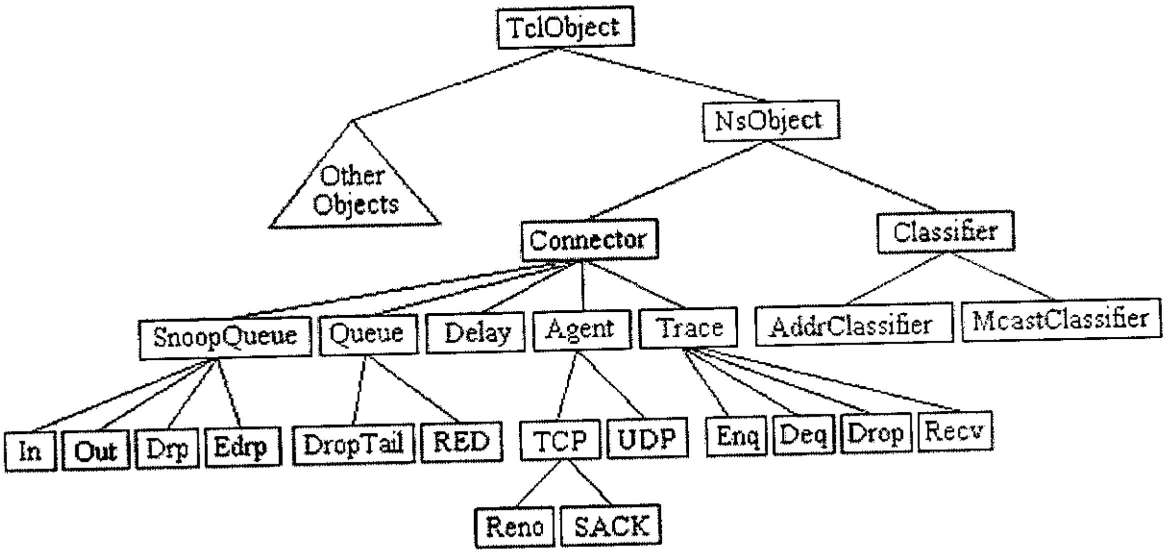


Figure 5: Class Hierarchy (Partial)

The root of the hierarchy is the TclObject class that is the superclass of all OTcl library objects (scheduler, network components, timers and the other objects including NAM related ones). As an ancestor class of TclObject, NsObject class is the superclass of all basic network component objects that handle packets, which may compose compound network objects such as nodes and links. The basic network components are further divided into two subclasses, Connector and Classifier, based on the number of the possible output data paths. The basic network objects that have only one output data path are under the Connector class, and switching objects that have possible multiple output data paths are under the Classifier class.

Node and Routing:

A node is a compound object composed of a node entry object and classifiers as shown in Figure 5. There are two types of nodes in NS. A unicast node has an address classifier that does unicast routing and a port classifier. A multicast node, in addition, has a classifier that classify multicast packets from unicast packets and a multicast classifier that performs multicast routing.

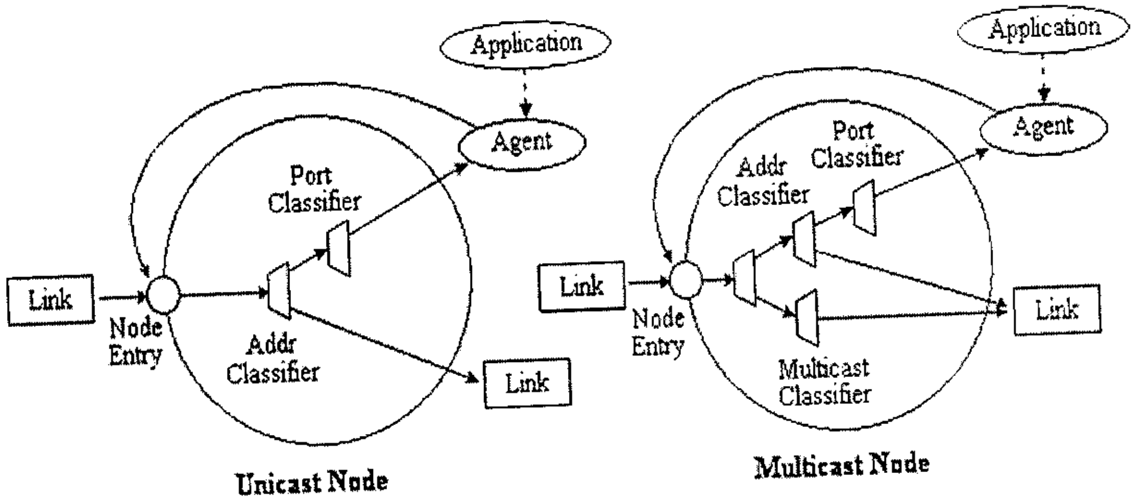


Figure 6: Node (Unicast and Multicast)

In NS, Unicast nodes are the default nodes. To create Multicast nodes the user must explicitly notify in the input OTcl script, right after creating a scheduler object, that all the nodes that will be created are multicast nodes.

After specifying the node type, the user can also select a specific routing protocol other than using a default one.

- **Unicast**

  - $ns rtproto type

  - type: Static, Session, DV, cost, multi-path

- **Multicast**

  - $ns multicast (right after set $ns [new Scheduler])

  -$ns mrtproto type

  - type: CtrMcast, DM, ST, BST

Link

A link is another major compound object in NS. When a user creates a link using a duplex-link member function of a Simulator object, two simplex links in both directions are created as shown in Figure 6.

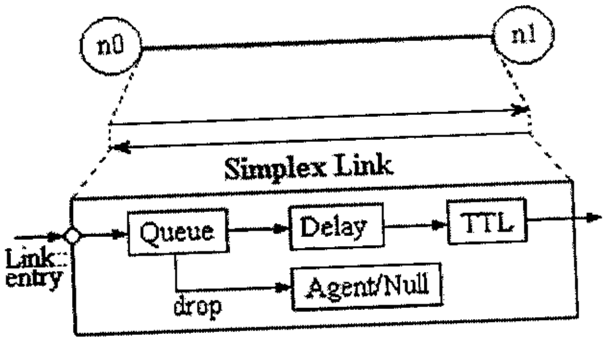


Figure 7. Link

One thing to note is that an output queue of a node is actually implemented as a part of simplex link object. Packets dequeued from a queue are passed to the Delay object that simulates the link delay, and packets dropped at a queue are sent to a Null Agent and are freed there. Finally, the TTL object calculates Time To Live parameters for each packet received and updates the TTL field of the packet.

Tracing:

In NS, network activities are traced around simplex links. If the simulator is directed to trace network activities (specified using $ns trace-all file or $ns namtrace-all file), the links created after the command will have the following trace objects inserted as shown in Figure 7. Users can also specifically create a trace object of type type between the given src and dst nodes using the create-trace {type file src dst} command.

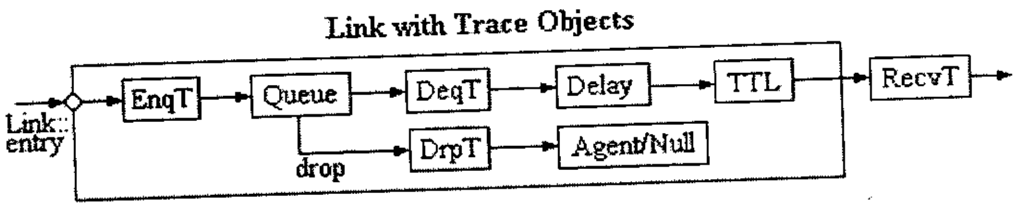


Figure 8: Inserting Trace Objects

When each inserted trace object (i.e. EnqT, DeqT, DrpT and RecvT) receives a packet, it writes to the specified trace file without consuming any simulation time, and passes the packet to the next network object.

Queue Monitor:

Basically, tracing objects are designed to record packet arrival time at which they are located. Although a user gets enough information from the trace, he or she might be interested in what is going on inside a specific output queue. For example, a user interested in RED queue behavior may want to measure the dynamics of average queue size and current queue size of a specific RED queue (i.e. need for queue monitoring). Queue monitoring can be achieved using queue monitor objects and snoop queue objects as shown in Figure 8.

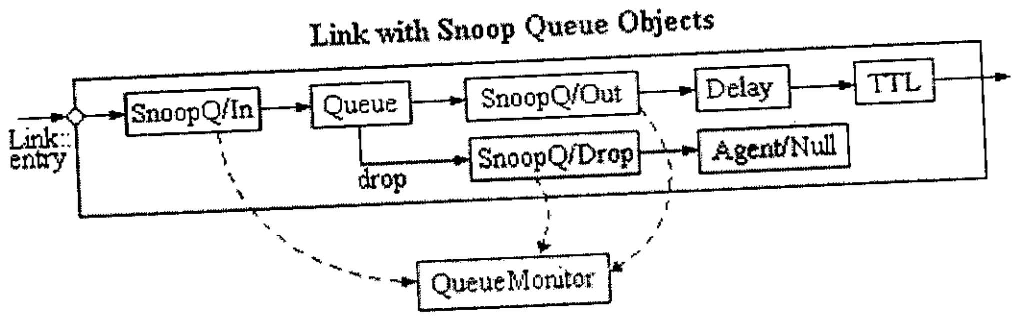


Figure 9. Monitoring Queue

When a packet arrives, a snoop queue object notifies the queue monitor object of this event. The queue monitor using this information monitors the queue. A RED queue monitoring example is shown in the RED Queue Monitor Example section. Note that snoop queue objects can be used in parallel with tracing objects even though it is not shown in the above figure.

Packet Flow Example:

Until now, the two most important network components (node and link) were examined. Figure 9 shows internals of an example simulation network setup and packet flow. The network consist of two nodes (n0 and n1) of which the network addresses are 0 and 1 respectively. A TCP agent attached to n0 using port 0 communicates with a TCP sink object attached to n1 port 0. Finally, an FTP application (or traffic source) is attached to the TCP agent, asking to send some amount of data.

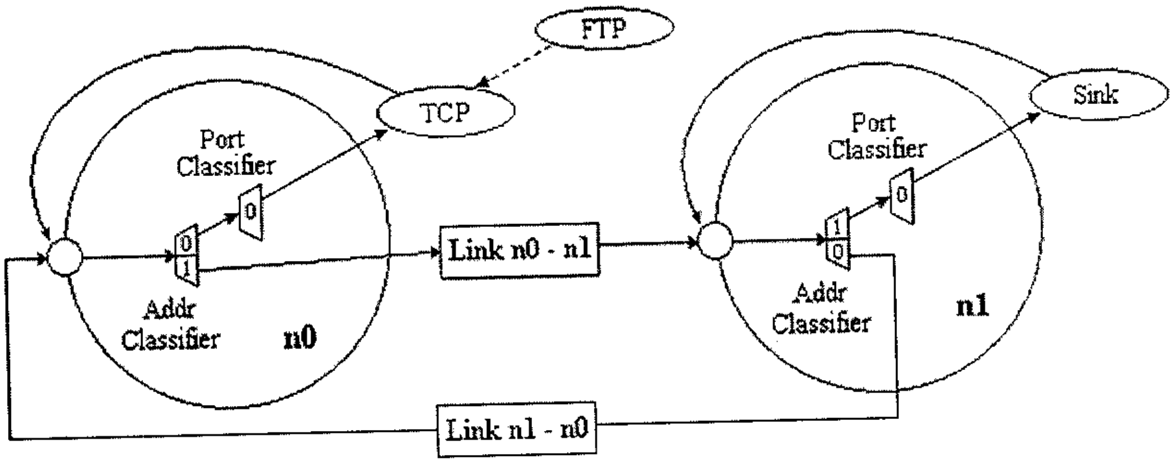


Figure 10: Packet Flow Example

Note that the above figure does not show the exact behavior of a FTP over TCP. It only shows the detailed internals of simulation network setup and a packet flow.

PROJECT DESCRIPTION

4. PROJECT DESCRIPTION

The objective of the project is to distribute the file contained in the server to its associated destination node groups in peer to peer networks. Peers interested in the file form an overlay network through which the file is distributed. The construction of the overlay network is based on the idea of combination networks. Similar to the combination networks, there are two system parameters in the overlay network. The server encodes the file into different messages using the linear coding scheme given in the previous section. Thus any messages out of the messages can be used to decode the original file. The peers are divided into disjoint groups and each group is assigned a unique *group ID*. Each group of peers is responsible for relaying one of the encoded messages. To accelerate the distribution of the messages, peers in the same group are connected by an unstructured overlay network according to some loose rules.

Each peer in a group is connected to at least other peers which are in different groups respectively. The entire overlay network can be considered as a union of multiple combination networks (or subgraphs of combination networks). Different combination networks may overlap. A peer that is a relay node in a combination network may be a receiver node in another combination network. Each peer is a receiver node in one combination network. The server is always the source node in any combination network.

There are three groups of peers colored in black, blue and red, respectively. The colored links represent corresponding messages with arrows pointing to the transmission directions. We can see that each peer has at least two links with different colors pointed to itself. All the peers are able to decode the received messages.

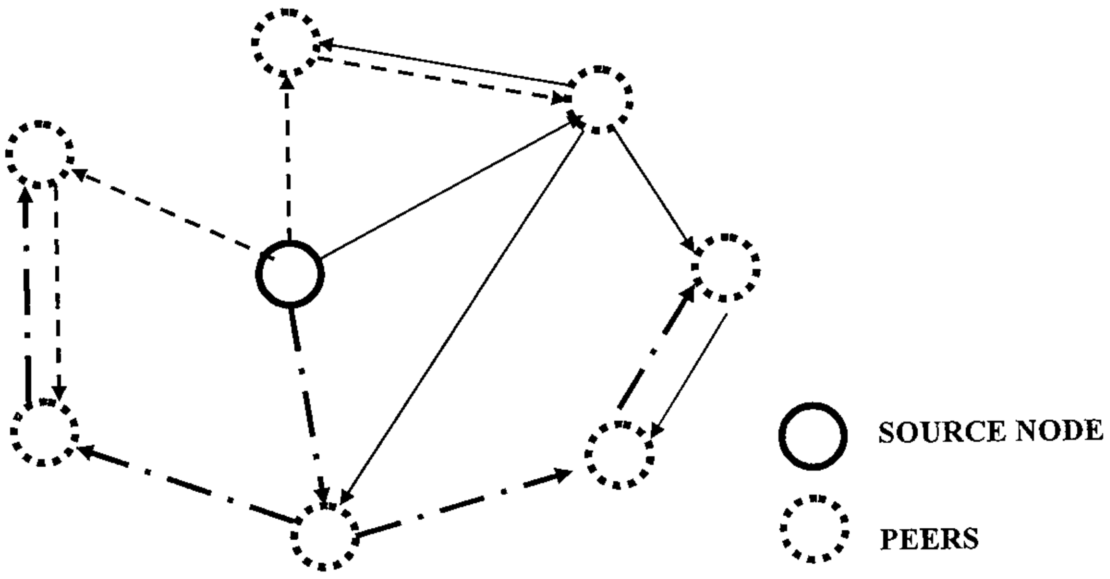


Fig 11: An example overlay network constructed by PPFEED.

Overlay links are added to the system on demand as new peers join the system. There are two key issues in the overlay network construction: first, how to connect the peers in the same group; second, how to connect the peers in different groups. The first issue involves how to distribute the messages in a scalable way. Here adopt some loose rules similar to the unstructured peer-to-peer network for a newly joined peer, it is responded by a random list of the existing peers, and the new peer creates connections with these peers.

The reason is three folds. First, it is resilient to peer join. A new peer can be connected to any existing peer. Second, the overlay construction and maintenance overhead is low. There is no constraint on the overlay topology as long as it is connected. If some peers leave the system, it is convenient for the rest of the peers to reconnect through the server. Third, it is easy to flood messages on the resulting overlay network.

The second issue involves how the peers receive other messages to decode the messages. As any different messages can be used to decode, we only need to find peers in different groups. Since there are eligible groups, we first connect the peer to random peers in different groups. Then it performs a local topology adjustment to find the peers such that the average latency between the peer and peers is minimized.

4.1 Peer Joining

Assume that the server is well-known whose IP address is known to all the peers by some address translation service such as DNS. When a peer wants to retrieve a file hosted by the server, it initiates a join process by sending a JOIN request to the server. The server keeps track of the number of peers in each group and maintains a partial list of existing peers in the group and their IP addresses.

The purpose of maintaining a partial list instead of a full list is to achieve a balance between scalability and efficiency. On one hand, when the network size is large, it is resource-consuming to maintain a full list of peers in the server. On the other hand, if the list is too short, it may cause much longer latency for new peers to join when the peers in the list crash. Note that although the server is responsible for bootstrapping the peers, it will not be the bottleneck of the system since once the peer receives the list, it communicates with other peers for topology construction and data dissemination.

When the server receives a peer's join request, it assigns the peer to a group such that the numbers of peers in different groups are balanced. Then the server sends the list of peers of that group to the joining peer and updates the number of peers in that group. After receiving the list of peers, the new peer will contact them and create overlay links with them. These peers are called intra-neighbors of the new peer because they are within the same group. In contrast, the neighbors that are in different groups are called inter-neighbors. The new peer asks one of its intra-neighbors, which is picked randomly, to provide a list of its inter-neighbors. The new peer then takes the list of peers as its inter-neighbors.

The topology of the peer-to-peer network can be considered as a combination of multiple unstructured peer-to-peer networks, each of which is composed of the peers within the same group. The topology within one group is arbitrary as long as it is connected. The only constraint is on the edges between different groups. It is required that each peer is connected to at least peers in different groups respectively.

As the overlay topology is formed by always connecting the new peers to a random list of existing peers in the network, the overlay links between peers may not be good with respect to latency or link stress. To alleviate the performance degradation, we optimize the overlay topology by a process called local topology adjustment. The basic idea is to replace the direct neighbors by peers with better performance through local search. Each peer sends periodic QUERY messages with a search radius. Once it finds a peer with shorter latency than the current neighbor, it switches its neighbor to the new peer. The local topology adjustment is done periodically such that the peer can find the best neighbor gradually and the overlap topology can adapt to the dynamic joins/leaves.

4.2 Peer Leaving

There are two types of peer leaving: friendly or abruptly. Friendly leaving means that the leaving peer initiates a leaving process so that the system is aware of its leaving and can make necessary updates accordingly. Abruptly leaving means that the leaving peer leaves the system without any notification, mainly due to link crash or computer crash. For the friendly leaving, the leaving peer will initiate a leaving process by sending LEAVE messages to both of its intra neighbors and inter-neighbors. The leaving of the peer may impair the connectivity between peers in the same group. To rebuild the connectivity, the intra-neighbors will initiate the join process with the group ID in the join request.

After receiving a join request with a designated group ID, the server acts as one of the intra-neighbors of the peer temporarily guarantee the connectivity. The topology is further adjusted by the local topology adjustment process. Here the server acts as a connection "hub" for the peers that were connected to the leaving peer. This may increase the data forwarding burden on the server temporarily. Nevertheless, the situation will be improved after the local topology adjustment process is done. Moreover, it can achieve strong robustness with little control overhead. For example, it can handle concurrent peer leavings. The connectivity is rebuilt after the server receives all the join requests from the intra-neighbors of the leaving peers.

After receiving the LEAVE message from the leaving peer, the inter-neighbors will ask one of its intra-neighbors for a new inter-neighbor to replace the leaving one. The leaving peer will also send a LEAVE message to the server. This LEAVE message will make the server update the number of peers in the group. For the abruptly leaving, peers send HELLO messages to its neighbors periodically and maintain a HELLO timer for each neighbor. Receiving a HELLO message triggers a reset of the corresponding HELLO timer.

The neighbors detect the abruptly leaving by the timeout of the HELLO timer. Similarly, intra neighbors initiate the join process after detecting the sudden leave. Inter-neighbors ask their intra-neighbors for a replacement of the left peer. Moreover, one of the neighbors is chosen to send a LEAVE message to the server on behalf of the abruptly leaving peer so that the server can update the number of the peers in the group. To minimize the selection overhead, choose the inter-neighbor whose group ID is next to that of the leaving peer to perform this task.

4.3 Data Dissemination

Before sending out the file, the server needs to encode the file. The encoding is over a Galois field. The file is divided into multiple blocks with each block represented by a symbol. The first blocks are encoded and then the second blocks, and so on. In the case that there are not enough blocks to encode, the file is padded with zero string. Assuming the field size each blocks are encoded into different messages using the linear coding scheme.

After encoding, the server sends the encoded messages to the peers in the groups respectively. The group ID and the encoding function form a one-to-one mapping. Peers can learn which messages they receive based on the sender of the messages: if the sender is the server, the messages correspond to the group ID of the peer itself; if the sender is a peer, the messages correspond to the group ID of the sender.

The peers forward all the messages they receive based on the following rules:

Rule 1. If the message comes from the server, the peer forwards it to all its intra-neighbors and inter-neighbors.

Rule 2. If the message comes from one of its intra-neighbors, the peer forwards it to other intra-neighbors except for the sender.

Rule 3. If the message comes from one of its inter-neighbors, the peer does nothing.

Peers forward messages in a push style, which means that the messages are forwarded under the three rules as soon as they arrive at a peer. A peer decodes the messages right after it receives different messages. Thus data dissemination is fast and simple.

PERFORMANCE EVALUATION

5. PERFORMANCE EVALUATION

The developed system of this project, evaluate the performance of network coding on peer to peer network file traversal through simulations. The system compares the scheme with a peer-to-peer multicast system. The previous scheme first constructs an overlay mesh spanning over all the peers. The overlay mesh is a richer connected graph which satisfies some desirable performance properties.

The multicast tree is a spanning tree on top of the mesh and is constructed on demand of the source peer. The system chooses Narada as the comparison counterpart because it is a representative overlay multicast scheme without network coding so that the system can evaluate the benefit network coding brings. The simulation adopts following three performance metrics:

Throughput:

Throughput is defined as the service the system provides in one time unit. Here the system let different systems transmit the same file, thus throughput can be simply represented by the time consumed by the transmission. The shorter the time consumed, the higher the throughput. The system starts transmitting the file from time. Then the consumed time is the time when the peers finish receiving the file, denoted by finish time.

Reliability:

This performance metric is used to evaluate the ability of the system to handle errors. The system uses the number of retransmissions to characterize this ability. A system with higher reliability will have a smaller number of retransmissions, and thus higher throughput.

Link stress:

Link stress is defined as the number of copies of the same message transmitted through the same link. It is a performance metric that only applies to an overlay network due to the mismatch between the overlay network and the physical network. The system uses it to evaluate the effectiveness of the topology awareness improvement and the efficiency of the system. The system studies the performance of the system in three different configurations:

(i) ***Baseline configuration:*** In this configuration, the overlay links are constructed randomly. The file is sent after the overlay network is formed.

(ii) ***Topology awareness configuration:*** In this configuration, the overlay links are constructed by taking into consideration of topology mismatch. The file is sent after the overlay network is formed.

(iii) ***Dynamic peer join/leave configuration:*** In this configuration, the overlay links are constructed randomly. Peers join and leave the system during the file transmission.

CONCLUSION

6. CONCLUSION

In this project, the system have proposed a peer-to-peer file sharing scheme based on network coding, PPFEED. The scheme can serve as a peer-to-peer middleware created within the web services framework for web-based file sharing applications. Compared to other file sharing schemes, the advantages of the scheme can be summarized as follows.

(a) ***Scalability:*** Files are distributed through a peer-to-peer network. With the increase of the network size, the total available bandwidth also increases.

(b) ***Efficiency:*** The linear network coding scheme is deterministic and easy to implement. There is no requirement for peers to collaborate to construct the linear coding scheme on demand.

All the peers need is the mapping between the group ID and the encoding function. And this mapping does not change with time. Compared to random network coding, the receiver can always recover the original messages after receiving k different messages and the data dissemination is more efficient as data messages are transmitted through the same overlay link at most once.

(c) ***Reliability:*** The redundant links can greatly improve the reliability of the system with little overhead.

(d) ***Resilience:*** Churn is a common problem in overlay networks. By adding redundant links, the negative effect of churn is alleviated.

(e) ***Topology awareness:*** Simulation results show that the proposed topology clustering scheme can greatly reduce link stress and improve throughput.

FUTURE ENHANCEMENTS

7. FUTURE ENHANCEMENT

The future work includes how to optimize the system under unstable network status such as cone4gestion. Mobile peer-to-peer systems have recently got in the limelight of the research community that is striving to build efficient and effective mobile content addressable networks. Along this line of research, a new peer-to-peer (P2P) file sharing protocol suited to mobile ad hoc networks (MANET). The main ingredients are network coding and mobility assisted data propagation.

APPENDICES

APPENDIX

SOURCE CODE:

1.PEER TO PEER OVERLAY NETWORK:

```
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)          30                   ;# number of mobilenodes
set val(rp)          AODV                  ;# routing protocol
set val(x)              1330
set val(y)              850
# Initialize Global Variables
set ns_          [new Simulator]
set tracefd     [open peer.tr w]
$ns_ trace-all $tracefd
set namtrace [open peer.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)
$ns_ color 0 blue
# Create channel #1 and #2
set chan_1_ [new $val(chan)]
set chan_2_ [new $val(chan)]
```

```
# Create node(0) "attached" to channel #1
# configure node, please note the change below.
$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 OFF \
                 -channel $chan_1_
 # node_(1) can also be created with the same configuration, or with a different
 # channel specified.
 # Uncomment below two lines will create node_(1) with a different channel.
 $ns_ node-config \           channel $chan_2_
 set node_(0) [$ns_ node]
 set node_(1) [$ns_ node]
 set node_(2) [$ns_ node]
 set node_(3) [$ns_ node]
 set node_(4) [$ns_ node]
 set node_(5) [$ns_ node]
 set node_(6) [$ns_ node]
 set node_(7) [$ns_ node]
 set node_(8) [$ns_ node]
 set node_(9) [$ns_ node]
 set node_(10) [$ns_ node]
set node_(11) [$ns_ node]
set node_(12) [$ns_ node]
set node_(13) [$ns_ node]
set node_(14) [$ns_ node]
set node_(15) [$ns_ node]
set node_(16) [$ns_ node]
set node_(17) [$ns_ node]
set node_(18) [$ns_ node]
set node_(19) [$ns_ node]
set node_(20) [$ns_ node]
set node_(21) [$ns_ node]
set node_(22) [$ns_ node]
set node_(23) [$ns_ node]
set node_(24) [$ns_ node]
set node_(25) [$ns_ node]
set node_(26) [$ns_ node]
set node_(27) [$ns_ node]
set node_(28) [$ns_ node]
set node_(29) [$ns_ node]
$node_(0) label "SOURCE"
$node_(1) label "DISTRIBUTOR-1"
$node_(2) label "DISTRIBUTOR-2"
$node_(3) label "DISTRIBUTOR-3"
$node_(4) label "DISTRIBUTOR-4"
$node_(5) label "R-1"
$node_(6) label "R-2"
$node_(7) label "R-3"
$node_(8) label "R-4"
$node_(9) label "R-5"
$node_(10) label "R-6"
$node_(11) label "R-7"
```

```
$node_(12) label "R-8"
$node_(13) label "R-9"
$node_(14) label "R-10"
$node_(15) label "R-11"
$node_(16) label "R-12"
for {set i 0} {$i < $val(nn)} {incr i} {
        $ns_ initial_node_pos $node_($i) 35
}
# Provide initial (X,Y, for now Z=0) co-ordinates for mobilenodes
$node_(0) set X_ 535.0
$node_(0) set Y_ 700.0
$node_(0) set Z_ 10.0
$node_(1) set X_ 85.0
$node_(1) set Y_ 550.0
$node_(1) set Z_ 10.0
$node_(2) set X_ 385.0
$node_(2) set Y_ 550.0
$node_(2) set Z_ 10.0
$node_(3) set X_ 685.0
$node_(3) set Y_ 550.0
$node_(3) set Z_ 10.0
$node_(4) set X_ 985.0
$node_(4) set Y_ 550.0
$node_(4) set Z_ 10.0
$node_(5) set X_ 10.0
$node_(5) set Y_ 350.0
$node_(5) set Z_ 10.0
$node_(6) set X_ 85.0
$node_(6) set Y_ 350.0
$node_(6) set Z_ 10.0
$node_(7) set X_ 160.0
```

```
$node_(7) set Y_ 350.0
$node_(7) set Z_ 10.0
$node_(8) set X_ 310.0
$node_(8) set Y_ 350.0
$node_(8) set Z_ 10.0
$node_(9) set X_ 385.0
$node_(9) set Y_ 350.0
$node_(9) set Z_ 10.0
$node_(10) set X_ 460.0
$node_(10) set Y_ 350.0
$node_(10) set Z_ 10.0
$node_(11) set X_ 610.0
$node_(11) set Y_ 350.0
$node_(11) set Z_ 10.0
$node_(12) set X_ 685.0
$node_(12) set Y_ 350.0
$node_(12) set Z_ 10.0
$node_(13) set X_ 760.0
$node_(13) set Y_ 350.0
$node_(13) set Z_ 10.0
$node_(14) set X_ 910.0
$node_(14) set Y_ 350.0
$node_(14) set Z_ 10.0
$node_(15) set X_ 985.0
$node_(15) set Y_ 350.0
$node_(15) set Z_ 10.0
$node_(16) set X_ 1060.0
$node_(16) set Y_ 350.0
$node_(16) set Z_ 10.0
$node_(17) set X_ 1125.0
$node_(17) set Y_ 670.0
```

```
$node_(17) set Z_ 10.0
$node_(18) set X_ 945.0
$node_(18) set Y_ 740.0
$node_(18) set Z_ 10.0
$node_(19) set X_ 750.0
$node_(19) set Y_ 760.0
$node_(19) set Z_ 10.0
$node_(20) set X_ 120.0
$node_(20) set Y_ 830.0
$node_(20) set Z_ 10.0
$node_(21) set X_ 880.0
$node_(21) set Y_ 820.0
$node_(21) set Z_ 10.0
$node_(22) set X_ 40.0
$node_(22) set Y_ 705.0
$node_(22) set Z_ 10.0
$node_(23) set X_ 310.0
$node_(23) set Y_ 760.0
$node_(23) set Z_ 10.0
$node_(24) set X_ 215.0
$node_(24) set Y_ 700.0
$node_(24) set Z_ 10.0
$node_(25) set X_ 86.0
$node_(25) set Y_ 190.0
$node_(25) set Z_ 10.0
$node_(26) set X_ 280.0
$node_(26) set Y_ 250.0
$node_(26) set Z_ 10.0
$node_(27) set X_ 530.0
$node_(27) set Y_ 240.0
$node_(27) set Z_ 10.0
```

```
$node_(28) set X_ 770.0
$node_(28) set Y_ 220.0
$node_(28) set Z_ 10.0
$node_(29) set X_ 1030.0
$node_(29) set Y_ 210.0
$node_(29) set Z_ 10.0
# Now produce some simple node movements
$ns_ at 0 "$node_(0) setdest 735.0 700.0 20.0"
$ns_ at 0 "$node_(1) setdest 285.0 550.0 20.0"
$ns_ at 0 "$node_(2) setdest 585.0 550.0 20.0"
$ns_ at 0 "$node_(3) setdest 885.0 550.0 20.0"
$ns_ at 0 "$node_(4) setdest 1185.0 550.0 20.0"
$ns_ at 0 "$node_(5) setdest 210.0 350.0 20.0"
$ns_ at 0 "$node_(6) setdest 285.0 350.0 20.0"
$ns_ at 0 "$node_(7) setdest 360.0 350.0 20.0"
$ns_ at 0 "$node_(8) setdest 510.0 350.0 20.0"
$ns_ at 0 "$node_(9) setdest 585.0 350.0 20.0"
$ns_ at 0 "$node_(10) setdest 660.0 350.0 20.0"
$ns_ at 0 "$node_(11) setdest 810.0 350.0 20.0"
$ns_ at 0 "$node_(12) setdest 885.0 350.0 20.0"
$ns_ at 0 "$node_(13) setdest 960.0 350.0 20.0"
$ns_ at 0 "$node_(14) setdest 1110.0 350.0 20.0"
$ns_ at 0 "$node_(15) setdest 1185.0 350.0 20.0"
$ns_ at 0 "$node_(16) setdest 1260.0 350.0 20.0"
ns_ at 0 "$node_(17) setdest 1325.0 670.0 20.0"
$ns_ at 0 "$node_(18) setdest 1145.0 740.0 20.0"
$ns_ at 0 "$node_(19) setdest 950.0 760.0 20.0"
$ns_ at 0 "$node_(20) setdest 320.0 830.0 20.0"
$ns_ at 0 "$node_(21) setdest 1080.0 820.0 20.0"
$ns_ at 0 "$node_(22) setdest 240.0 705.0 20.0"
$ns_ at 0 "$node_(23) setdest 510.0 760.0 20.0"
```

```
$ns_ at 0 "$node_(24) setdest 415.0 700.0 20.0"
$ns_ at 0 "$node_(25) setdest 286.0 190.0 20.0"
$ns_ at 0 "$node_(26) setdest 480.0 250.0 20.0"
$ns_ at 0 "$node_(27) setdest 730.0 240.0 20.0"
$ns_ at 0 "$node_(28) setdest 970.0 220.0 20.0"
$ns_ at 0 "$node_(29) setdest 1230.0 210.0 20.0"
#SETTING UP THE CONNECTIONS
set tcp0 [new Agent/TCP]
set sink1 [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp0
$ns_ attach-agent $node_(1) $sink1
$ns_ connect $tcp0 $sink1
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns_ at 0.9 "$ftp0 start"
$ns_ at 1.1 "$ftp0 stop"
set tcp1 [new Agent/TCP]
set sink2 [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp1
$ns_ attach-agent $node_(2) $sink2
$ns_ connect $tcp1 $sink2
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns_ at 1.0 "$ftp1 start"
$ns_ at 1.2 "$ftp1 stop"
set tcp2 [new Agent/TCP]
set sink3 [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp2
$ns_ attach-agent $node_(3) $sink3
$ns_ connect $tcp2 $sink3
set ftp2 [new Application/FTP]
```

```
$ftp2 attach-agent $tcp2
$ns_ at 0.8 "$ftp2 start"
$ns_ at 1.0 "$ftp2 stop"
set tcp3 [new Agent/TCP]
set sink4 [new Agent/TCPSink]
$ns_ attach-agent $node_(0) $tcp3
$ns_ attach-agent $node_(4) $sink4
$ns_ connect $tcp3 $sink4
set ftp3 [new Application/FTP]
$ftp3 attach-agent $tcp3
$ns_ at 0.9 "$ftp3 start"
$ns_ at 1.2 "$ftp3 stop"
set tcp4 [new Agent/TCP]
set sink5 [new Agent/TCPSink]
$ns_ attach-agent $node_(1) $tcp4
$ns_ attach-agent $node_(5) $sink5
$ns_ connect $tcp4 $sink5
set ftp4 [new Application/FTP]
$ftp4 attach-agent $tcp4
$ns_ at 1.51 "$ftp4 start"
$ns_ at 2.0 "$ftp4 stop"
set tcp5 [new Agent/TCP]
set sink6 [new Agent/TCPSink]
$ns_ attach-agent $node_(1) $tcp5
$ns_ attach-agent $node_(6) $sink6
$ns_ connect $tcp5 $sink6
set ftp5 [new Application/FTP]
$ftp5 attach-agent $tcp5
$ns_ at 1.52 "$ftp5 start"
$ns_ at 2.0 "$ftp5 stop"
```

```
set tcp6 [new Agent/TCP]
set sink7 [new Agent/TCPSink]
$ns_ attach-agent $node_(1) $tcp6
$ns_ attach-agent $node_(7) $sink7
$ns_ connect $tcp6 $sink7
set ftp6 [new Application/FTP]
$ftp6 attach-agent $tcp6
$ns_ at 1.5 "$ftp6 start"
$ns_ at 2.0 "$ftp6 stop"
set tcp7 [new Agent/TCP]
set sink8 [new Agent/TCPSink]
$ns_ attach-agent $node_(2) $tcp7
$ns_ attach-agent $node_(8) $sink8
$ns_ connect $tcp7 $sink8
set ftp7 [new Application/FTP]
$ftp7 attach-agent $tcp7
$ns_ at 2.0 "$ftp7 start"
$ns_ at 2.5 "$ftp7 stop"
set tcp8 [new Agent/TCP]
set sink9 [new Agent/TCPSink]
$ns_ attach-agent $node_(2) $tcp8
$ns_ attach-agent $node_(9) $sink9
$ns_ connect $tcp8 $sink9
set ftp8 [new Application/FTP]
$ftp8 attach-agent $tcp8
$ns_ at 2.0 "$ftp8 start"
$ns_ at 2.5 "$ftp8 stop"
set tcp9 [new Agent/TCP]
set sink10 [new Agent/TCPSink]
$ns_ attach-agent $node_(2) $tcp9
$ns_ attach-agent $node_(10) $sink10
```

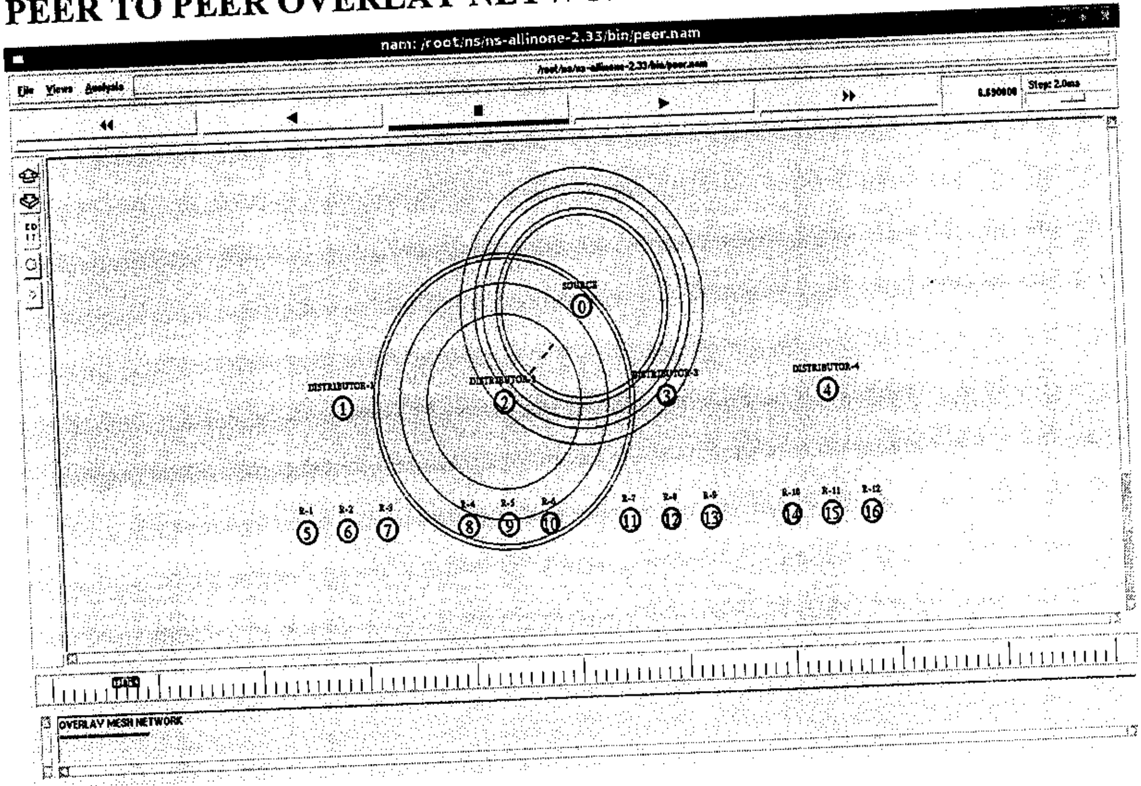
```
$ns_ connect $tcp9 $sink10
set ftp9 [new Application/FTP]
$ftp9 attach-agent $tcp9
$ns_ at 1.9 "$ftp9 start"
$ns_ at 2.5 "$ftp9 stop"
set tcp10 [new Agent/TCP]
set sink11 [new Agent/TCPSink]
$ns_ attach-agent $node_(3) $tcp10
$ns_ attach-agent $node_(11) $sink11
$ns_ connect $tcp10 $sink11
set ftp10 [new Application/FTP]
$ftp10 attach-agent $tcp10
$ns_ at 2.51 "$ftp10 start"
$ns_ at 3.0 "$ftp10 stop"
set tcp11 [new Agent/TCP]
set sink12 [new Agent/TCPSink]
$ns_ attach-agent $node_(3) $tcp11
$ns_ attach-agent $node_(12) $sink12
$ns_ connect $tcp11 $sink12
set ftp11 [new Application/FTP]
$ftp11 attach-agent $tcp11
$ns_ at 2.53 "$ftp11 start"
$ns_ at 3.0 "$ftp11 stop"
set tcp12 [new Agent/TCP]
set sink13 [new Agent/TCPSink]
$ns_ attach-agent $node_(3) $tcp12
$ns_ attach-agent $node_(13) $sink13
$ns_ connect $tcp12 $sink13
set ftp12 [new Application/FTP]
$ftp12 attach-agent $tcp12
$ns_ at 2.55 "$ftp12 start"
```

```
$ns_ at 3.0 "$ftp12 stop"
set tcp13 [new Agent/TCP]
set sink14 [new Agent/TCPSink]
$ns_ attach-agent $node_(4) $tcp13
$ns_ attach-agent $node_(14) $sink14
$ns_ connect $tcp13 $sink14
set ftp13 [new Application/FTP]
$ftp13 attach-agent $tcp13
$ns_ at 3.01 "$ftp13 start"
$ns_ at 3.5 "$ftp13 stop"
set tcp14 [new Agent/TCP]
set sink15 [new Agent/TCPSink]
$ns_ attach-agent $node_(4) $tcp14
$ns_ attach-agent $node_(15) $sink15
$ns_ connect $tcp14 $sink15
set ftp14 [new Application/FTP]
$ftp14 attach-agent $tcp14
$ns_ at 3.02 "$ftp14 start"
$ns_ at 3.5 "$ftp14 stop"
set tcp15 [new Agent/TCP]
set sink16 [new Agent/TCPSink]
$ns_ attach-agent $node_(4) $tcp15
$ns_ attach-agent $node_(16) $sink16
$ns_ connect $tcp15 $sink16
set ftp15 [new Application/FTP]
$ftp15 attach-agent $tcp15
$ns_ at 3.0 "$ftp15 start"
$ns_ at 3.5 "$ftp15 stop"
######################
$ns_ at 0.5 "$ns_ trace-annotate \"OVERLAY MESH NETWORK \""
$ns_ at 0.5 "$ns_ trace-annotate \"************************** \""
```

```
$ns_ at 1.0 "$ns_ trace-annotate \"SOURCE SEND FILES TO ALL DISTRIBUTORS
\""
# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
   $ns_ at 10.0 "$node_($i) reset";
}
$ns_ at 10.02 "stop"
$ns_ at 10.03 "puts \"NS EXITING...\" ; $ns_ halt"
proc stop {} {
 global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
          puts "running nam..."
          exec nam peer &
 }
 puts "Starting Simulation..."
 $ns_ run
```

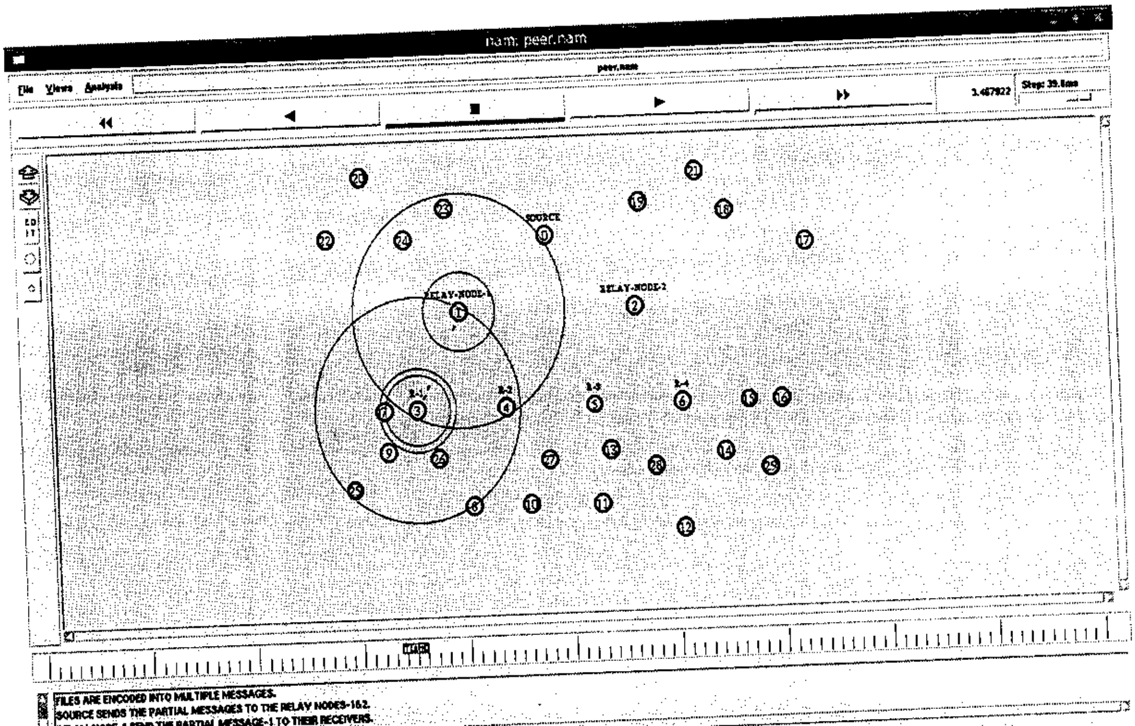
SCREENSOTS:

PEER TO PEER OVERLAY NETWORK:



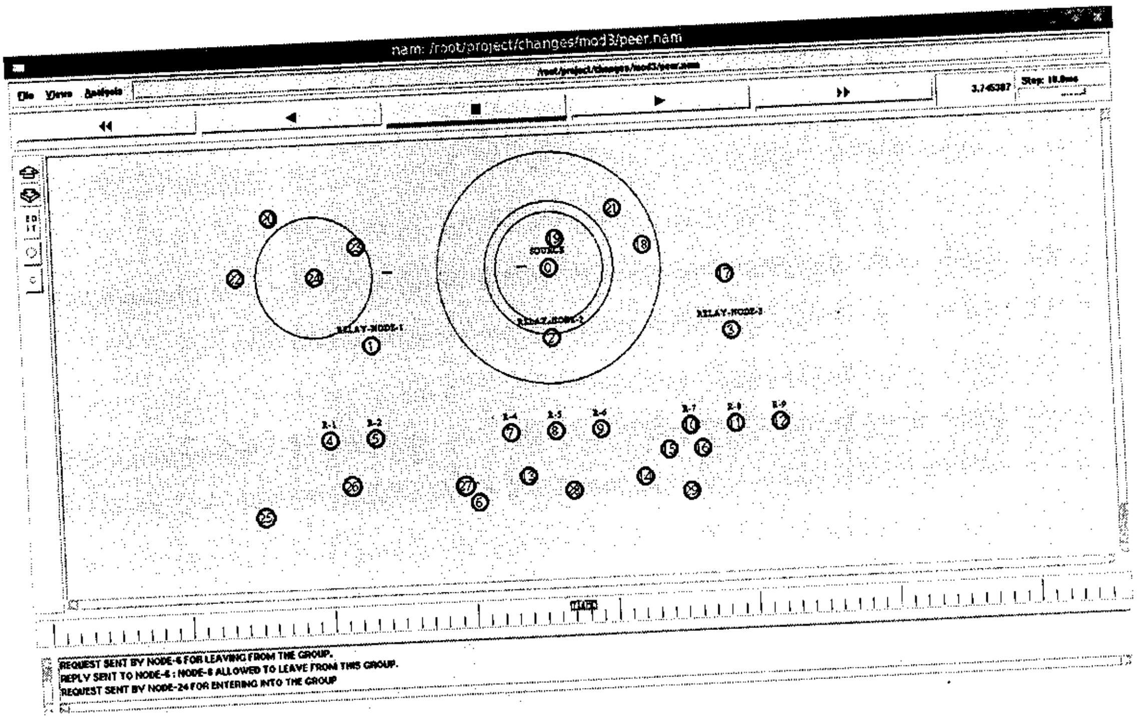
OVERLAY MESH NETWORK

COMBINATION NETWORK:



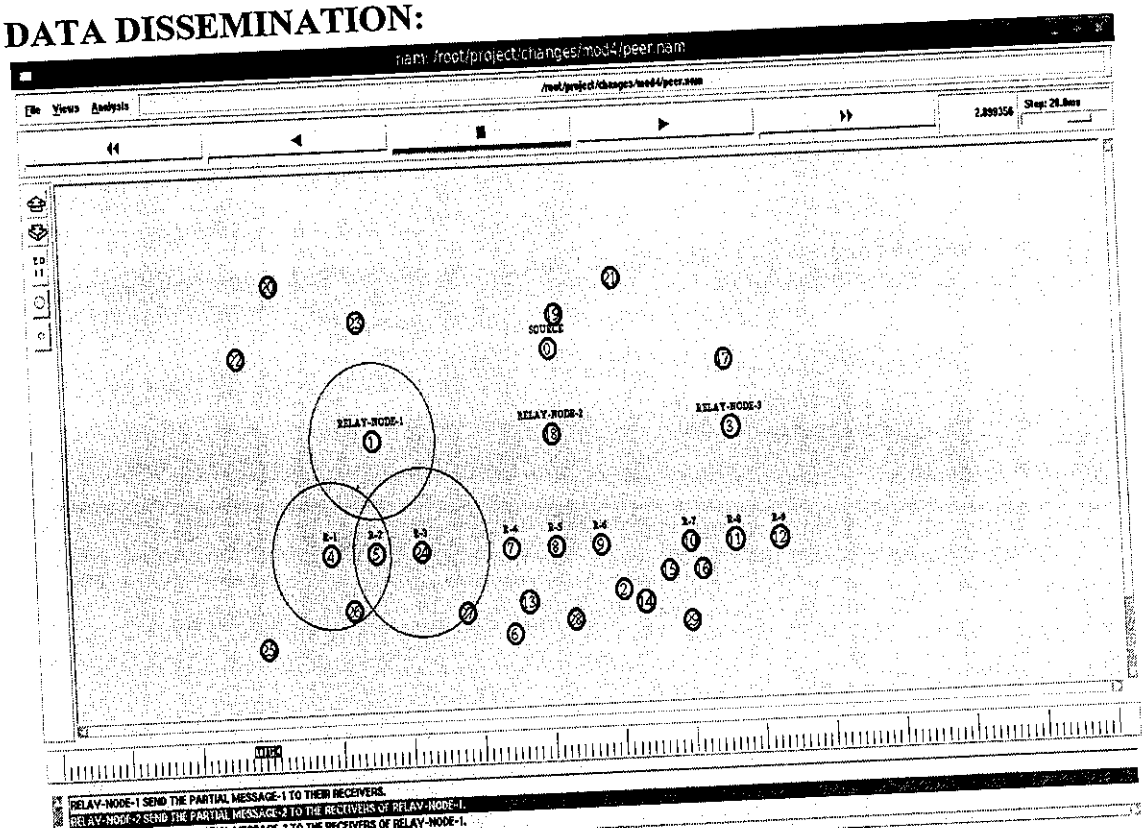
FILES ARE ENCODED INTO MULTIPLE MESSAGES.
SOURCE SENDS THE PARTIAL MESSAGES TO THE RELAY NODES-1&2.
[illegible] NODE-1 SEND THE PARTIAL MESSAGE-1 TO THEIR RECEIVERS.

PEER LEAVING/JOINING IN COMBINATION NETWORK:



```
REQUEST SENT BY NODE-6 FOR LEAVING FROM THE GROUP.
REPLY SENT TO NODE-6 : NODE-6 ALLOWED TO LEAVE FROM THIS GROUP.
REQUEST SENT BY NODE-24 FOR ENTERING INTO THE GROUP
```

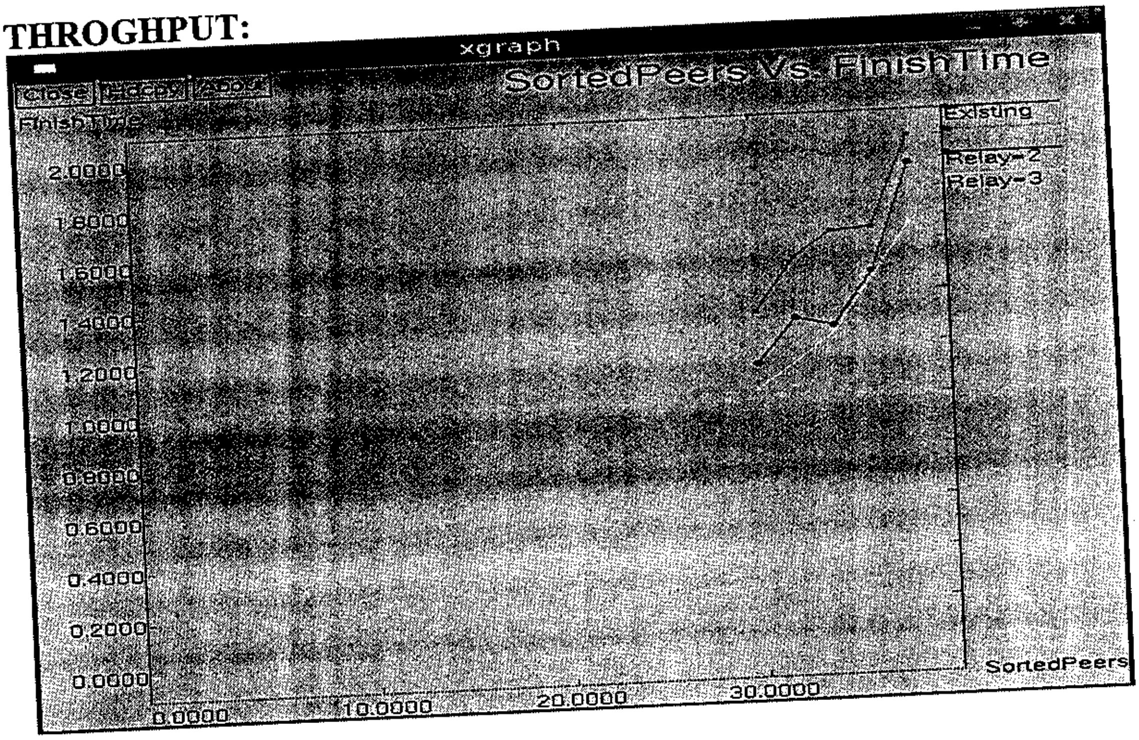
DATA DISSEMINATION:



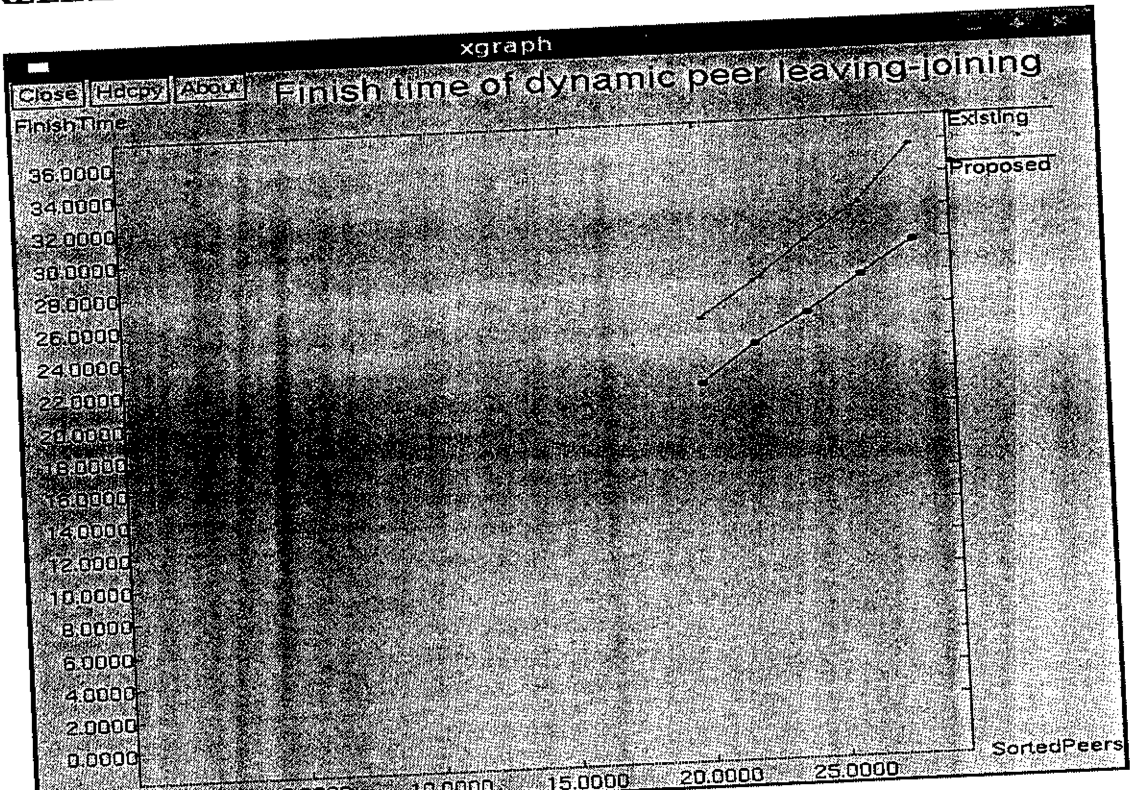
```
RELAY-NODE-1 SEND THE PARTIAL MESSAGE-1 TO THEIR RECEIVERS.
RELAY-NODE-2 SEND THE PARTIAL MESSAGE-2 TO THE RECEIVERS OF RELAY-NODE-1.
```

PERFORMANCE EVALUATION:

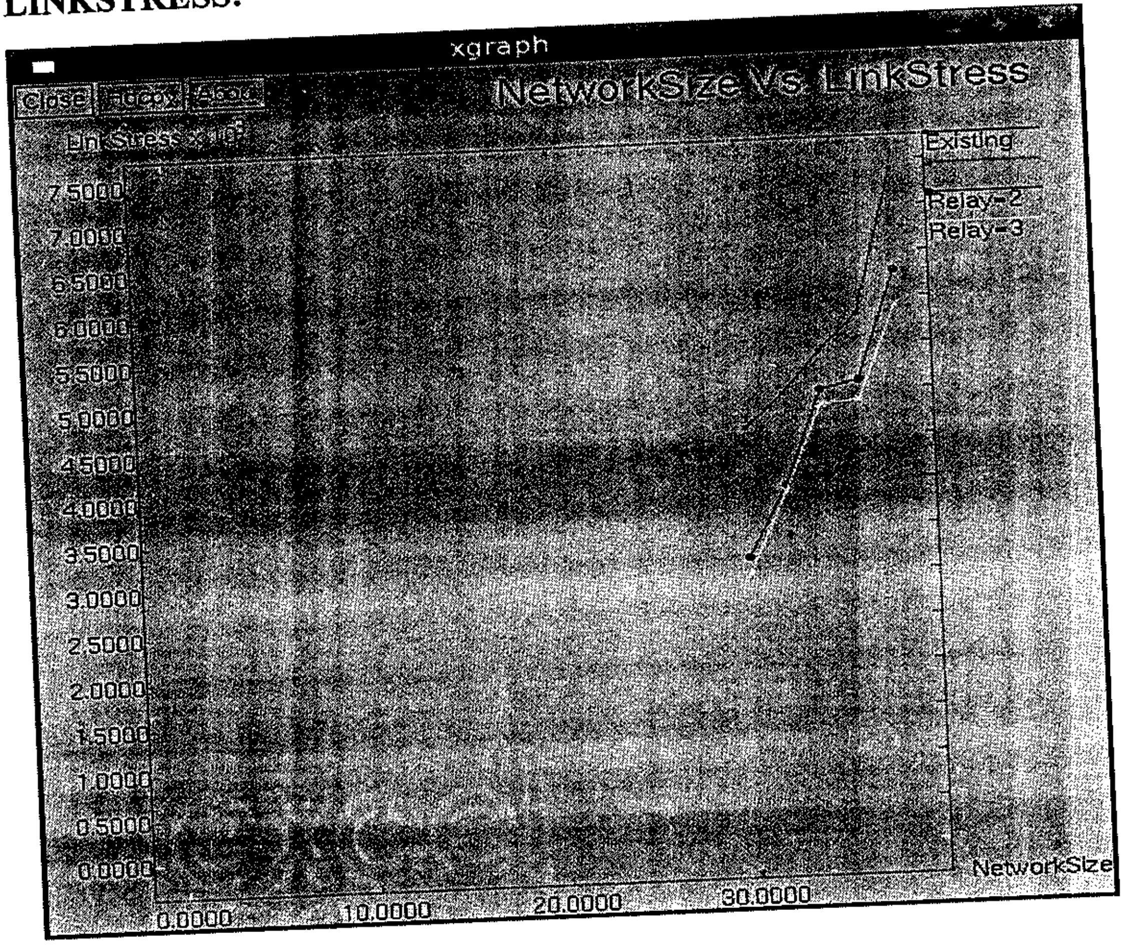
THROGHPUT:



RELIABILITY:



LINKSTRESS:


xgraph
Close
Hdcpy
About
NetworkSize Vs. LinkStress
LinkStress x 10³
Existing
Relay-2
Relay-3
7.5000
7.0000
6.5000
6.0000
5.5000
5.0000
4.5000
4.0000
3.5000
3.0000
2.5000
2.0000
1.5000
1.0000
0.5000
0.0000
NetworkSize
0.0000
10.0000
20.0000
30.0000

REFERENCES

REFERENCES

[1] R. Ahlswede, N. Cai, S.-Y. R. Li and R.W. Yeung, "Network information flow," IEEE Trans. Information Theory, vol. 46, 2000, pp. 1204-1216.

[2] S.-Y.R. Li, R.W. Yeung and N. Cai, "Linear network coding," IEEE Trans.Information Theory, vol. 49, 2003, pp. 371-381.

[3] R. Koetter and M. Medard, "An algebraic approach to network coding," IEEE/ACM Trans. Networking, vol. 11, no. 5, 2003, pp. 782-795.

[4] T. Ho, M. Medard, J. Shi, M. Effros and D.R. Karger, "On randomized network coding," Proc. Annual Allerton Conference on Communication, Control, and Computing, 2003.

[5] T. Ho, M. Medard, R. Koetter, D. Karger, M. Effros, J. Shi and B. Leong, "A random linear network coding approach to multicast," IEEE Trans. Information Theory, vol. 52, 2006, pp. 4413- 4430.

[6] D.S. Lun, N. Ratnakar, R. Koetter, M. Medard, E. Ahmed and H. Lee, "Achieving minimum-cost multicast: a decentralized approach based on network coding," Proc. IEEE INFOCOM 2005, Mar. 2005.

[7] D. S. Lun, M. Medard, T. Ho and R. Koetter, "Network coding with a cost criterion," Proc. 2004 International Symposium on Information Theory and its Applications (ISITA 2004), Oct. 2004.

[8] Y. Zhu, B.C. Li and J. Guo, "Multicast with network coding in application layer overlay networks," IEEE Journal on Selected Areas in Communication, Sep. 2004.

[9] A.G. Dimakis, P.B. Godfrey,M.J.Wainwright and K. Ramchandran, "Network coding for distributed storage systems," Proc. IEEE INFOCOM 2007, May. 2007.

[10] M. Kim, C.W. Ahn, M. Medard and M. Effros, "On minimizing network coding resources: An evolutionary approach," Proc. NetCod, 2006.

[11] K. Bhattad, N. Ratnakar, R. Koetter and K.R. Narayanan, "Minimal network coding for multicast," Proc. IEEE International Symposium on Information Theory , 2005.

[12] C.K. Ngai and R.W. Yeung, "Network coding gain of combination networks," IEEE Information Theory Workshop, Oct. 2004, pp. 283-287.

[13] Y.H. Chu, S.G. Rao, S. Seshan and H. Zhang, "A case for end system multicast," IEEE Journal on Selected Areas in Communication, Special Issue on Networking Support for Multicast, vol. 20, no. 8, 2002.

[14] S. Jaggi, P. Sanders, P.A. Chou, M. Effros, S. Egner, K. Jain and L. Tolhuizen, "Polynomial time algorithms for multicast network code construction," IEEE Trans. Information Theory, vol 51, no. 6, June 2005, pp. 1973-1982.

[15] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," IEEE INFOCOM 2005, Miami, FL, March, 2005.

[16] S. Ratnasamy, M. Handley, R. Karp and S. Shenker, "A scalable content addressable network," Proc. ACM SIGCOMM, 2001, pp. 149-160.

[17] Gnutella Protocol Development, the gnutella v0.6 protocol. Available: http://rfc-gnutella.sourceforge.net/developer/index.html. 2003.

[18] S. Ratnasamy, M. Handley, R. M. Karp and S. Shenker, "Topologically aware overlay construction and server selection," IEEE INFOCOM 2002, New York, NY, June, 2002.

[19] http://www.cc.gatech.edu/projects/gtitm/.