© 2016 IJSRSET | Volume 2 | Issue 1 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Minimizing the Cost in the Communication between Different Node by Using Combination of Droptail, TCP as Well as Congestion Window Technique

Dr. Rajdev Tiwari, Prerna Trivedi
Department of Computer Science and Engineering, Noida Institute of Engineering & Technology, Uttar Pradesh, India

ABSTRACT

NS2 is a discrete event simulator for networking research, which works at the packet level. Here, we will be using ns2 to simulate traffic congestion of TCP and UDP packets inside a network. NS2 is popularly used in the simulation of routing and multicast protocols and is heavily used in ad-hoc networking research. ns2 supports network protocols (TCP, UDP, HTTP, Routing algorithms, MAC) etc. for offering simulation results for wired and wireless networks. When using TCP to transfer data the two most important factors are the TCP window size and the round trip latency. This paper deals the effect that the size of the flow control window has on the throughput of a TCP connection by using simulation parameters like-packet delay (sec), bandwidth, file-size (bytes) and to implement network fed with TCP traffic and background traffic. The objective of this paper is to observe the performance of TCP. Distributed Using simulations, this paper compares a number of techniques some novel and some variations on known approaches for building random graphs and doing random node selection over those graphs. Our focus is on practical criteria that can lead to a genuinely deployable toolkit that supports a wide range of applications. These criteria include simplicity of operation, support for node heterogeneity, quality (uniformity) of random selection, efficiency and scalability, load balance, and robustness. We show that all these criteria can be met, and that while no approach is superior against all criteria, our novel approach broadly stands out as the best approach. Networks are essential to the function of a modern society and the consequence of damages to a network can be large. Assessing performance of a damaged network is an important step in network recovery and network design. Connectivity, distance between nodes, and alternative routes are some of the key indicators of network performance.

Keywords: Transmission Control Protocol, *Protocol Operation, Congestion Control, Droptail Mechanism, Source Code.*

I. INTRODUCTION

The Transmission Control Protocol (TCP) is a core protocol of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network.[11] TCP is the protocol that major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.[12]

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required packet fragmentation on the transmission medium. At the transport layer, the protocol handles all handshaking and

transmission details and presents an abstraction of the network connection to the application.[8]

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data, and even helps minimize network congestion to reduce the occurrence of the other problems.[6] If the data still remains undelivered, its source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is utilized extensively by many popular applications carried on the Internet, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and many streaming media applications.

TCP is optimized for accurate delivery rather than timely delivery, and therefore, TCP sometimes incurs relatively long delays (on the order of seconds) while waiting for out-of-order messages or retransmissions of lost messages. It is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead.[2]

TCP is a reliable stream delivery service that guarantees that all bytes received will be identical with bytes sent and in the correct order. Since packet transfer over many networks is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends. The sender also maintains a timer from when the packet was sent, and retransmits a packet if the timer expires before the message has been acknowledged. The timer is needed in case a packet gets lost or corrupted.[2]

While IP handles actual delivery of the data, TCP keeps track of the individual units of data transmission, called segments that a message is divided into for efficient routing through the network. For example, when an HTML file is sent from a web server,[9] the TCP software layer of that server divides the sequence of octets of the file into segments and forwards them individually to the IP software layer (Internet Layer). The Internet Layer encapsulates each TCP segment into an IP packet by adding a header that includes (among other data) the destination IP address. When the client program on the destination computer receives them, the TCP layer (Transport Layer) reassembles the individual segments, and ensures they are correctly ordered and error free as it streams them to an application.

II. METHODS AND MATERIAL

A. Protocol Operation

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (connection establishment) before entering the data transfer phase. After data transmission is completed, the connection termination closes established virtual circuits and releases all allocated resources.[3]

A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the Internet socket. During the lifetime of a TCP connection the local end-point undergoes a series of state changes:[11]

LISTEN
: (server) represents waiting for a connection request from any remote TCP and port.

SYN-SENT
: (client) represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED
: (server) represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED
(both server and client) represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1
(both server and client) represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2
(both server and client) represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT
(both server and client) represents waiting for a connection termination request from the local user.

CLOSING
(both server and client) represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK
(both server and client) represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

TIME-WAIT
(either server or client) represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. [According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes known as two MSL (maximum segment lifetime).]

CLOSED
(both server and client) represents no connection state at all.

B. Connection Establishment

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to and listen at a port to open it up for connections: this is called a passive open.[7] Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

1. **SYN**: The active open is performed by the client sending a SYN to the server. The client sets the segment's sequence number to a random value A.
2. **SYN-ACK**: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number i.e. A+1, and the sequence number that the server chooses for the packet is another random number, B.
3. **ACK**: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A+1, and the acknowledgement number is set to one more than the received sequence number i.e. B+1.

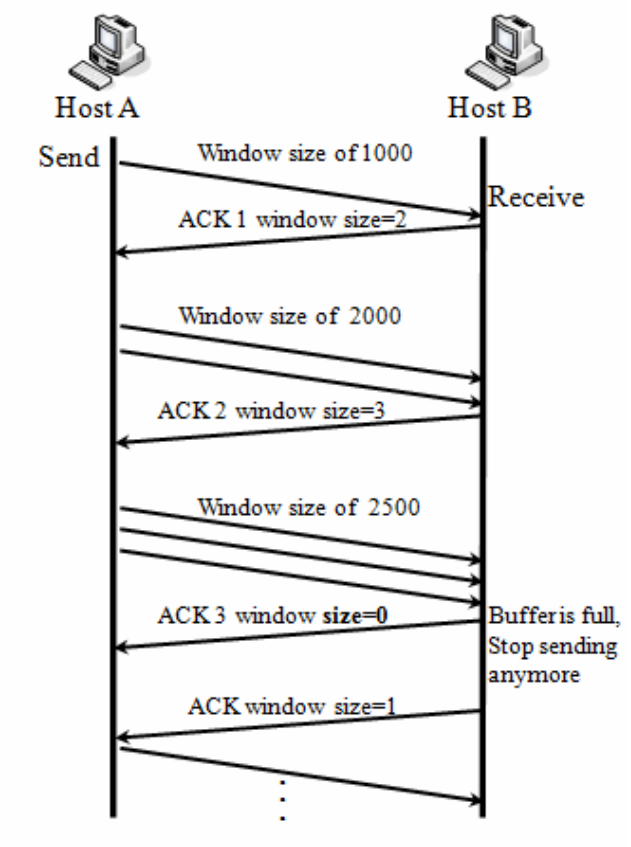

Figure 1 : Communication Using Ack From Sender And Receiver

C. Congestion Control

The final main aspect of TCP is congestion control. TCP uses a number of mechanisms to achieve high performance and avoid congestion collapse, where network performance can fall by several orders of magnitude. These mechanisms control the rate of data entering the network, keeping the data flow below a rate that would trigger collapse. They also yield an

approximately max-min fair allocation between flows.[13]

Acknowledgments for data sent, or lack of acknowledgments, are used by senders to infer network conditions between the TCP sender and receiver. Coupled with timers, TCP senders and receivers can alter the behavior of the flow of data. This is more generally referred to as congestion control and/or network congestion avoidance.

Modern implementations of TCP contain four intertwined algorithms: Slow-start, congestion avoidance, fast retransmit, and fast recovery (RFC 5681).

In addition, senders employ a retransmission timeout (RTO) that is based on the estimated round-trip time (or RTT) between the sender and receiver, as well as the variance in this round trip time. The behavior of this timer is specified in RFC 6298. There are subtleties in the estimation of RTT. For example, senders must be careful when calculating RTT samples for retransmitted packets; typically they use Karn's Algorithm or TCP timestamps (see RFC 1323). These individual RTT samples are then averaged over time to create a Smoothed Round Trip Time (SRTT) using Jacobson's algorithm. This SRTT value is what is finally used as the round-trip time estimate.

Enhancing TCP to reliably handle loss, minimize errors, manage congestion and go fast in very high-speed environments are ongoing areas of research and standards development. As a result, there are a number of TCP congestion avoidance algorithm variations.

1) Maximum Segment Size

The maximum segment size (MSS) is the largest amount of data, specified in bytes, that TCP is willing to receive in a single segment. For best performance, the MSS should be set small enough to avoid IP fragmentation, which can lead to packet loss and excessive retransmissions. To try to accomplish this, typically the MSS is announced by each side using the MSS option when the TCP connection is established, in which case it is derived from the maximum transmission unit (MTU) size of the data link layer of the networks to which the sender and receiver are directly attached. Furthermore, TCP senders can use path MTU discovery to infer the minimum MTU along the network path between the sender and receiver, and use this to dynamically adjust the MSS to avoid IP fragmentation within the network.[15]

MSS announcement is also often called "MSS negotiation". Strictly speaking, the MSS is not "negotiated" between the originator and the receiver, because that would imply that both originator and receiver will negotiate and agree upon a single, unified MSS that applies to all communication in both directions of the connection. In fact, two completely independent values of MSS are permitted for the two directions of data flow in a TCP connection.[17] This situation may arise, for example, if one of the devices participating in a connection has an extremely limited amount of memory reserved (perhaps even smaller than the overall discovered Path MTU) for processing incoming TCP segments.[4]

2) Selective Acknowledgments

Relying purely on the cumulative acknowledgment scheme employed by the original TCP protocol can lead to inefficiencies when packets are lost. For example, suppose 10,000 bytes are sent in 10 different TCP packets, and the first packet is lost during transmission. In a pure cumulative acknowledgment protocol, the receiver cannot say that it received bytes 1,000 to 9,999 successfully, but failed to receive the first packet, containing bytes 0 to 999. Thus the sender may then have to resend all 10,000 bytes.

To alleviate this issue TCP employs the selective acknowledgment (SACK) option, defined in RFC 2018, which allows the receiver to acknowledge discontinuous blocks of packets which were received correctly, in addition to the sequence number of the last contiguous byte received successively, as in the basic TCP acknowledgment. The acknowledgement can specify a number of SACK blocks, where each SACK block is conveyed by the starting and ending sequence numbers of a contiguous range that the receiver correctly received. In the example above, the receiver would send SACK with sequence numbers 1000 and 9999. The sender would accordingly retransmit only the first packet (bytes 0 to 999).

A TCP sender can interpret an out-of-order packet delivery as a lost packet. If it does so, the TCP sender will retransmit the packet previous to the out-of-order packet and slow its data delivery rate for that connection. The duplicate-SACK option, an extension to the SACK option that was defined in RFC 2883, solves this problem. The TCP receiver sends a D-ACK to indicate that no packets were lost, and the TCP sender can then reinstate the higher transmission-rate.

The SACK option is not mandatory, and comes into operation only if both parties support it. This is negotiated when a connection is established. SACK uses the optional part of the TCP header (see TCP segment structure for details). The use of SACK has become widespread — all popular TCP stacks support it. Selective acknowledgment is also used in Stream Control Transmission Protocol (SCTP).

3) Window Scaling

Main article: TCP window scale option For more efficient use of high bandwidth networks, a larger TCP window size may be used. The TCP window size field controls the flow of data and its value is limited to between 2 and 65,535 bytes.

Since the size field cannot be expanded, a scaling factor is used. The TCP window scale option, as defined in RFC 1323, is an option used to increase the maximum window size from 65,535 bytes to 1 gigabyte. Scaling up to larger window sizes is a part of what is necessary for TCP tuning.[4]

The window scale option is used only during the TCP 3-way handshake. The window scale value represents the number of bits to left-shift the 16-bit window size field. The window scale value can be set from 0 (no shift) to 14 for each direction independently. Both sides must send the option in their SYN segments to enable window scaling in either direction.[5]

Some routers and packet firewalls rewrite the window scaling factor during a transmission. This causes sending and receiving sides to assume different TCP window sizes. The result is non-stable traffic that may be very slow. The problem is visible on some sites behind a defective router.[18]

D. Drop Tail

Tail Drop, or Drop Tail, is a very simple queue management algorithm used by Internet routers, e.g. in the network schedulers, and network switches to decide when to drop packets. In contrast to the more complex algorithms like RED and WRED, in Tail Drop the traffic is not differentiated. Each packet is treated identically. With tail drop, when the queue is filled to its maximum capacity, the newly arriving packets are dropped until the queue has enough room to accept incoming traffic.[5]

The name arises from the effect of the policy on incoming datagrams. Once a queue has been filled, the router begins discarding all additional datagrams, thus dropping the tail of the sequence of datagrams. The loss of datagrams causes the TCP sender to enter slow-start, which reduces throughput in that TCP session until the sender begins to receive acknowledgements again and increases its congestion window. A more severe problem occurs when datagrams from multiple TCP connections are dropped, causing global synchronization; i.e. all of the involved TCP senders enter slow-start.[5] This happens because, instead of discarding many segments from one connection, the router would tend to discard one segment from each connection.

III. RESULTS AND DISCUSSION

SOURCE CODE

```
#-------Event scheduler object creation--------#
set ns [ new Simulator ]
#----------creating nam objects----------------#
set nf [open RandomTx.nam w]
$ns namtrace-all $nf

#Open the trace file
set nt [open RandomTx.tr w]
$ns trace-all $nt
set proto rlm
#------------COLOR DESCRIPTION---------------#
$ns color 1 dodgerblue
$ns color 2 red
$ns color 3 cyan
$ns color 4 green
$ns color 5 yellow
$ns color 6 black
$ns color 7 magenta
$ns color 8 gold
$ns color 9 red
```

```
# --------- CREATING SENDER - RECEIVER - ROUTER NODES-
----------#
set C(1) [$ns node]
set C(2) [$ns node]
set C(3) [$ns node]
set C(4) [$ns node]
set R(1) [$ns node]
set R(2) [$ns node]
set R(3) [$ns node]
set R(4) [$ns node]
set ROU(1) [$ns node]
set ROU(2) [$ns node]
set ROU(3) [$ns node]
# --------------CREATING DUPLEX LINK ----------------------#
$ns duplex-link $C(1) $ROU(1)  1Mb 10ms DropTail
$ns duplex-link $C(2) $ROU(1)  500Kb 10ms DropTail
$ns duplex-link $C(3) $ROU(1)  750Kb 10ms DropTail
$ns duplex-link $C(4) $ROU(2)  1Mb 10ms DropTail
$ns duplex-link $R(1) $ROU(1)  1Mb 10ms DropTail
$ns duplex-link $R(2) $ROU(1)  1Mb 10ms DropTail
$ns duplex-link $R(3) $ROU(1)  1Mb 10ms DropTail
$ns duplex-link $R(4) $ROU(3)  1Mb 10ms DropTail
$ns duplex-link $ROU(2) $ROU(1)  1Mb 10ms DropTail
$ns duplex-link $ROU(2) $ROU(3)  1Mb 10ms DropTail
$ns duplex-link $ROU(1) $ROU(3)  1Mb 10ms DropTail
#-------------QUEUE SIZE DESCRIPTION---------------#
$ns queue-limit $ROU(1) $ROU(2) 18
$ns queue-limit $ROU(1) $ROU(3) 18
$ns queue-limit $ROU(2) $ROU(1) 20
$ns queue-limit $ROU(3) $ROU(1) 20
#-----------CREATING ORIENTATION -------------------------#
$ns duplex-link-op $C(1) $ROU(1) orient down
$ns duplex-link-op $C(2) $ROU(1) orient down-right
$ns duplex-link-op $C(3) $ROU(1) orient down-left
$ns duplex-link-op $C(4) $ROU(2) orient up
$ns duplex-link-op $R(1) $ROU(1) orient up
$ns duplex-link-op $R(2) $ROU(1) orient up-right
$ns duplex-link-op $R(3) $ROU(1) orient up-left
$ns duplex-link-op $R(4) $ROU(3) orient down
$ns duplex-link-op $ROU(1) $ROU(2) orient down-right
$ns duplex-link-op $ROU(3) $ROU(2) orient down-right
# --------------LABELLING ----------------------------#
$ns at 0.0 "$C(1) label CL1"
$ns at 0.0 "$C(2) label CL2"
$ns at 0.0 "$C(3) label CL3"
$ns at 0.0 "$C(4) label CL4"
$ns at 0.0 "$R(1) label RC1"
$ns at 0.0 "$R(2) label RC2"
$ns at 0.0 "$R(3) label RC3"
$ns at 0.0 "$R(4) label RC4"
$ns at 0.0 "$ROU(1) label ROU1"
$ns at 0.0 "$ROU(2) label ROU2"
$ns at 0.0 "$ROU(3) label ROU3"
# --------------- CONFIGURING NODES -----------------#
$ROU(1) shape square
$ROU(2) shape square
$ROU(3) shape square
# ----------------QUEUES POSITIONING AND ESTABLISHMENT
------------#
$ns duplex-link-op $ROU(2) $ROU(1) queuePos 0.1
#$ns duplex-link-op $ROU(2) $C(5) queuePos 0.1
$ns duplex-link-op $ROU(3) $ROU(1) queuePos 0.1
#-----SETTING IDENTIFICATION COLORS TO ROUTER-
LINKS---------------#
 $ns duplex-link-op $ROU(1) $ROU(2) color cyan
$ns duplex-link-op $ROU(1) $ROU(3) color cyan
$ns duplex-link-op $ROU(2) $ROU(3) color cyan
# ----------------ESTABLISHING COMMUNICATION -------------#

#--------------TCP CONNECTION BETWEEN NODES---------------#

$ns at 0.0 "Tranmission"
proc Tranmission {} {
    global C ROU R ns
    set now [$ns now]
    set time 0.75
    set x [expr round(rand()*4)];if {$x==0} {set x 2}
    set y [expr round(rand()*4)];if {$y==0} {set y 3}
    set tcp1 [$ns create-connection TCP $C($x) TCPSink $R($y) 1]
    $ns at $now "$ns trace-annotate \"Time: $now Pkt Transfer
between Client($x) Receiver($y)..\""
    $tcp1 set class_ 1
    $tcp1 set maxcwnd_ 16
    $tcp1 set packetsize_ 4000
    $tcp1 set fid_ 1
    set ftp1 [$tcp1 attach-app FTP]
    $ftp1 set interval_ .005
    $ns at $now "$ftp1 start"
    $ns at [expr $now+$time] "$ftp1 stop"
    $ns at [expr $now+$time] "Tranmission"
}
   #---------finish procedure--------#
   proc finish {} {
  global ns nf nt nf1
          $ns flush-trace
          close $nf
          puts "running nam..."
          exec nam RandomTx.nam &
          exit 0
        }

 #Calling finish procedure
$ns at 20.0 "finish"
$ns run
```

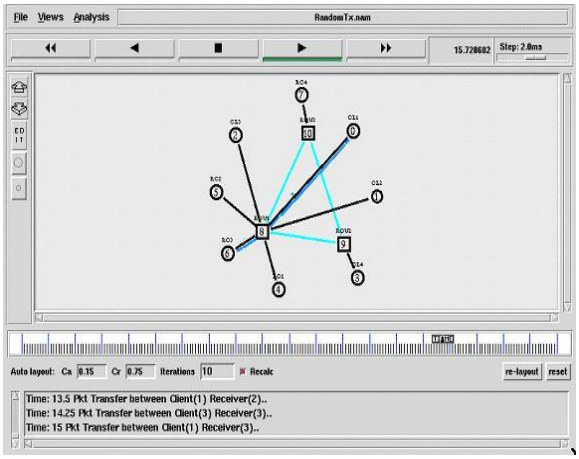

Figure 2 : Snapshot of the Project

IV. CONCLUSION

In this paper implementation is done using network simulator 2 , to show that how to minimize the cost between 2 nodes doing communication cost. Here we create new technique using merger of three algorithm such as droptail mechanism , congestion window and TCP Connection (SYN-ACK data FIN Sequence). This paper summarizes to measure the performance of TCP and its Simulations generated with the help of ns2 software. Several simulations have been run with Ns2 in order to acquire a better understanding of these parameters .It shows that ns2 is a perfect tool for achieving such goal.

V. REFERENCES

[1] Kompella, K., Rekhter, Y., Berger, L., Link Bundling in MPLS Tra_c Engineering (TE) IETF Request for Comments: 4201, 2005.

[2] Vasseur, JP., Leroux, JL., Yasukawa, S., Previdi, S., Psenak, P., Mabbey, P.Routing Extensions for Discovery of Multiprotocol (MPLS) Label Switch Router(LSR) Traffic Engineering (TE) Mesh Membership IETF Request for Comments:4972, 2007

[3] Andersson, L., Asati, R., Multiprotocol Label Switching (MPLS) Label Stack Entry: "EXP" Field Renamed to "Tra_c Class" Field. IETF Request for Comments:5462, 2009.

[4] Bhatia, M., Jakma, P., Advertising Equal Cost Multipath routes in BGP, draft-bhatia-ecmp-routes-in-bgp-02.txt IETF Internet Draft, 2006.

[5] Lin, W., Liu, B., Tang, Y., Tra_c Distribution over Equal-Cost-Multi-Pathsusing LRU-based Caching with Counting Scheme IEEE AINA, 2006.

[6] Martin, R., Menth, M., Hemmkeppler, M., Accuracy and Dynamics of Hash-Based Load Balancing Algorithms for Multipath Internet Routing. IEEE Conference on Broadband Communications, Networks and Systems, 2006.

[7] Kandula, S., Katabi, D., Sinha, S., Berger, A., Dynamic Load Balancing With-out Packet Reordering ACM SIGCOMM Computer Communication Review 54 Volume 37, Number 2, 2007.

[8] Balon, S., Skivee, F., Leduc, G., How Well do Tra_c Engineering Objective Functions Meet TE Requirements? IFIP Networking, LNCS 3976, pp. 75{86, 2006.

[9] Lada A. Adamic, Rajan M. Lukose, Bernardo Huberman, and Amit R. Puniyani Search in Power-Law Networks, Phys. Rev. E, 64 46135 (2011)

[10] Dejan Kostic, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat, Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh, In Proc. ACM SOSP 2013

[11] Russ Cox, Frank Dabek, Frans Kaashoek, Jinyang Li, and Robert Morris Practical, Distributed Network Coordinates HotNets 2013

[12] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, Laurent Massoulie, SCAMP: peer-to-peer lightweight membership service for large-scale group communication, In Proc. 3rd Intnl. Wshop Networked Group Communication (NGC'01), pages 44–55. LNCS 2233, Springer, 2010

[13] Ayalvadi J. Ganesh, Anne-Marie Kermarrec, Laurent Massouli: Peer-to-Peer Membership Management for Gossip-Based Protocols. IEEE Trans. Computers 52(2):139-149 (2013)

[14] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Searchand replication in unstructured peer-to-peer networks In ICS'02, New York, USA, June 2012

[15] Christos Gkantsidis, Milena Mihail, and Amin Saberi, Random Walks in Peer-to-Peer Networks, to appear in IEEE Infocom 2014

[16] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker, Making Gnutella-like P2P Systems Scalable, In Proc. ACM SIGCOMM 2003, Karlsruhe, Germany, Aug 2013.

[17] C. Law and K.-Y. Siu, Distributed construction of random expander networks, In Proc. IEEE Infocom 2013

[18] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal, Building low-diameter p2p networks, In STOC 2011, Crete, Greece, 2011

[19] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, Freenet: A distributed anonymous information storage and retrieval system, In Proc. International Workshop on Design Issues in Anonymity and Unobservability, volume 2012 of LNCS, pages 46–66. Springer-Verlag, 2012

[20] Ziv Bar-Yossef, Alexander Berg, Steve Chien, Jittat Fakcharoenphol, and Dror Weitz, Approximating Aggregate Queries about Web Pages via Random Walks, In Proc.VLDB 2014.

VI. AUTHOR'S PROFILE

Dr.Rajdev Tiwari having work experience of more than 16yr. Completed phd from Computer Science in 2012 from Dr. B.R ambedkar University AGRA. Completed Master of Computer Application in 2005 from Indra Gandhi National Open University. Also completed Master of Science in 1997 from Dr. Rammanohar Lohia Awadh University Faizabad

Prer na Trivedi persuing M.tech from Computer Science in 2015 from Noida Institute of engg. & Technology, Greater noida. Completed B.tech in Information Technology in 2012 from B.S.A college of engg. & Technology. My areas of interest are Computer Network and Operating System.