A High Performance Based Flow Control Method for Data Transfer

D. Sharmila and V. Udhayakumar

ABSTRACT: This paper develops such a protocol, Performance Adaptive UDP (henceforth PA-UDP), which aims to dynamically and autonomously maximize performance under different systems. A novel delay-based rate throttling model is also demonstrated to be very accurate under diverse system latencies. Based on these models, we implemented a prototype under Linux, and the experimental results demonstrate that PA-UDP outperforms other existing high-speed protocols on commodity hardware in terms of throughput, packet loss, and CPU utilization. PA-UDP is efficient not only for high-speed research networks, but also for reliable high-performance bulk data transfer over dedicated local area networks where congestion and fairness are typically not a concern.

KEYWORDS: Network Congestion, Packet loss, TCP, UDP.

I. INTRODUCTION

A certain class of next generation science applications needs to be able to transfer increasingly large amounts of data between remote locations. Toward this goal, several new dedicated networks with bandwidths upward of 10 Gbps have emerged to facilitate bulk data transfer. The goal of our work is to present a protocol that can maximally utilize the bandwidth of these private links through a novel performance-based system flow control. As Multi giga bit speeds become more pervasive in dedicated LANs and WANs and as hard drives remain relatively stagnant in read and write speeds, it becomes increasingly important to address these issues inside of the data transfer protocol. We demonstrate a mathematical basis for the control algorithms we use, and we implement and benchmark our method against other commonly used applications and protocols. A new protocol is necessary, unfortunately, due to the fact that the de facto standard of network communication, TCP, has been found to be unsuitable for high-speed bulk transfer. It is difficult to configure TCP to saturate the bandwidth of these links due to several assumptions made during its creation. A second crucial shortcoming of TCP is its congestion window. To ensure in order, reliable delivery, both parties maintain a buffer the size of the congestion window and the sender sends a burst of packets.

D.Sharmila is a M.Tech Student and V.Udhayakumar is working as Assistant professor, both are from Department of Computer Science and Engineering, PRIST University Pondicherry, India, Emails: dhanarajansharmila@gmail.com, udhaya_kurinji@yahoo.com

The receiver then sends back positive acknowledgments (ACKs) in order to receive the next window. Using timeouts and logic, the sender decides which packets are lost in the window and resends them. This synchronization scheme ensures that the receiver receives all packets sent, in-order, and without duplicates; however, it can come at a price. On networks with high latencies, reliance on synchronous communication can severely stunt any attempt for high-bandwidth utilization because the protocol relies on latency-bound communication. With congestion window of 100 packets and a maximum segment size of 1,460 bytes (the difference between the MTU and TCP/IP header), a network with an infinite bandwidth and 10 ms round-trip time would only be able to achieve approximately 120 Mbps effective throughput. One could attempt to mitigate the latency bottleneck by letting cwin scale to the bandwidth-delay product (BW _ rtt) or by striping and parallelizing TCP streams, but there are also difficulties associated with these techniques. Regardless, illustrates the potentially deleterious effect of synchronous communication on high-latency channels.

II. TCP SOLUTIONS

Assuming a no-loss link, a window size of n packets would allow for 12:5n Mbps throughput. On real networks, however, it turns out that the Bandwidth-Delay Product (BDP) of the network is integral to the window size. As the name suggests, the BDP is simply the product of the bandwidth of the channel multiplied by the end-to-end delay of the hosts. In a sense, this is the amount of data present "on the line" at any given moment. A 10 Gbps channel with an RTT of 10 ms would need approximately a 12.5 Megabyte buffer on either end, because at any given time, 12.5 Megabytes would be on the line that potentially would need to be resent due to errors in the line or packet loss at the receiving end. Ideally, a channel could sustain maximum throughput by setting the BDP equal to the congestion window, but it can be difficult to determine these parameters accurately. Moreover, the TCP header field uses only 16 bits to specify window size. Therefore, unless the TCP protocol is rewritten at the kernel level, the largest usable window is 65 Kilobytes. Note that there are modifications to TCP that can increase the window size for large BDP networks. Efforts in this area also include dynamic windows, different acknowledgment procedures, and statistical measurements for channel parameters. Other TCP variants attempt to modify the congestion control algorithm to be more amenable to characteristics of high-speed networks. Still others look toward multiple TCP streams, like FTP, Grid FTP, and PTCP. Most employ a combination of these methods, including (but not limited to) High-Speed TCP, Scalable TCP, and FAST TCP. Many of the TCP-based algorithms are based in the transport layer, and thus, kernel

modification is usually necessary to implement them. Some also rely on specially configured routers. As a result, the widespread deployment of any of these algorithms would be a very daunting task. It would be ideal to be able to run a protocol on top of the two standard transport layer protocols, TCP and UDP, so that any computer could implement them. This would entail an application-level protocol which could combine the strengths of UDP and TCP and which could be applied universally to these types of networks.

III. FLOW CONTROL

Flow control, term very much concerned to our work, is the process of managing the rate of data transmission between two nodes to prevent a fast sender from outrunning a slow receiver. It provides a mechanism for the receiver to control the transmission speed, so that the receiving node is not overwhelmed with data from transmitting node. Flow control should be distinguished from congestion control is used for controlling the flow of data when congestion has actually occurred. Flow control is important because it is possible for a sending computer to transmit information at a faster rate than the destination computer can receive and process them. This can happen if the receiving computers have a heavy traffic load in comparison to the sending computer, or if the receiving computer has less processing power than the sending computer.

A. Types of Flow Control:

There are two types of flow control. They are

- Network Congestion
- Data buffer

Network Congestion:

A prevention mechanism that provides control over the quantity of data transmission that enters a device.

Data Buffer:

A prevention control mechanism that provides storage to contain data- bursts from other network devices, compensating for the variation of data transmission speeds.

IV. GOALS FOR HIGH SPPED DATA TRANSFER

Ideally, we would want a high-performing protocol suitable for a variety of high-speed, high-latency networks without much configuration necessary at the user level. Furthermore, we would like to see good performance on many types of hardware, including commodity hardware and disk systems. Understanding the interplay between these algorithms and the host properties is crucial. On high-speed, high-latency, congestion-free networks, a protocol should strive to accomplish two goals: to maximize good put by minimizing synchronous, latency-bound communication and to maximize the data rate according to the receiver's capacity. (Here, we define good put as the throughput of usable data, discounting any protocol headers or transport overhead. Latency-bound communication is one of the primary problems of TCP due to the positive acknowledgment congestion window mechanism. As previous solutions have shown, asynchronous communication is the key to achieving maximum good put. It is often the case that disk throughput is less than half of the network's potential and high-speed processing of packets greatly taxes the CPU. Due to this large discrepancy, it is critical that the data rate is set by the receiver's capacity.

A. Open Loop Flow Control

The open-loop flow control mechanism is characterized by having no feedback between the receiver and the transmitter. This simple means of control is widely used. The allocation of resources must be a "prior reservation" or "hop-to-hop" type. The Open Loop flow control has inherent problems with maximizing the utilization of network resources. Resource allocation is made at connection setup using a CAC (Connection Admission Control) and this allocation is made using information that is already "old news" during the lifetime of the connection.

B. Closed Loop Flow Control

The Closed Loop flow control mechanism is characterized by the ability of the network to report pending network congestion back to the transmitter. This information is then used by the transmitter in various ways to adapt its activity to existing network conditions.

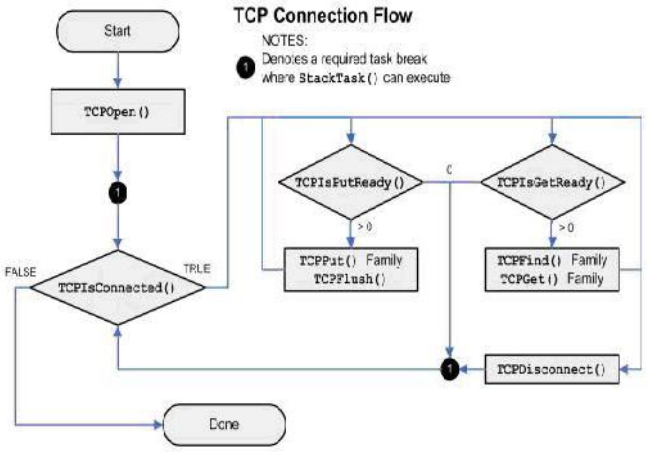

Fig: (b) closed loop flow control

C. Ethernet Flow Control

Ethernet is a specific computer network protocol. Flow control in Ethernet resides on the data link layer. A situation may arise where a sending station (computer) may be transmitting data faster than some other part of the network (including the receiving station) can accept it. The overwhelmed network element will send a PAUSE frame, which halts the transmission of the sender for a specified period of time. PAUSE is a flow control mechanism on full duplex Ethernet link segments defined by IEEE802.3x and uses MAC Control frames to carry the PAUSE commands. The MAC Control opcode for PAUSE is 0X0001 (hexadecimal). Only stations configured for full-duplex operation may send PAUSE frames. When a station wishes to send a PAUSE command, it sends the MAC Control frame to the 48-bit destination bridge group address of 01-80-C2-00-00-01. This particular address has been reserved for use in PAUSE frames. The use of a well-known address simplifies the flow control process by making it unnecessary for a station at one end of the link to discover and store the address of the station at the other end of the link. Another

advantage of using this multicast address arises from the use of flow control between network switches. The particular multicast address used is selected from a range of address which have been reserved by the IEEE 802.1D standard (which specifies the operation of switches). Normally, a frame with a multicast destination that is sent to a switch will be forwarded out all other ports of the switch. However, this range of multicast address is special and will not be forwarded by an 802.1D-compliant switch. Instead, frames sent to this address are understood by the switch to be frames meant to be acted upon within the switch. A PAUSE frame includes the period of pause time being requested, in the form of two byte unsigned integer (0 through 65535). This number is the requested duration of the pause. The pause time is measured in units of pause "quanta", where each unit is equal to 512 bit times.

V. ARCHITECTURE AND ALGORITHMS

First, we discuss a generic architecture which takes advantage of the considerations related in the previous section. In the next three sections, a real-life implementation is presented and its performance is analyzed and compared to other existing high-speed protocols.

A. Rate Control Algorithms

Certain system characteristics of the host receiving the file, an optimum rate can be calculated so that the receiver will not run out of memory during the transfer. Thus, a target rate can be negotiated at connection time. We propose a simple three-way handshake protocol where the first SYN packet from the sender asks for a rate. The sender may be restricted to 500 Mbps, for instance. The receiver then checks its system parameters r(disk), r(recv), and m, and either accepts the supplied rate, or throttles the rate down to the maximum allowed by the system. The following SYNACK packet would instruct the sender of a change, if any. Data could then be sent over the UDP socket at the target rate, with the receiver checking for lost packets and sending retransmission requests periodically over the TCP channel upon discovery of lost packets. The requests must be spaced out in time relative to the RTT of the channel, which can also be roughly measured during the initial handshake, so that multiple requests are not made for the same packet, while the packet has already been sent but not yet received. This is an example of a negative acknowledgment system, because the sender assumes that the packets were received correctly unless it receives data indicating otherwise. TCP should also be used for dynamic rate control. The disk throughput will vary over the course of a transfer, and as a consequence, should be monitored throughout. Rate adjustments can then proceed according to (6). To do this, disk activity, memory usage, and data rate must be monitored at specified time intervals.

B. Processing Packets

Several practical solutions exist to decrease CPU latency for receiving packets. Multithreading is an indispensable step to decouple other processes which have no sequential liability with one another. Minimizing I/O and system calls and appropriately using mutexes can contribute to overall efficiency. Thread priorities can often guarantee CPU attentiveness on certain kernel scheduler implementations. Also, libraries exist which guarantee high-performance, low-latency threads. Regardless of the measures mentioned above to curb latency, great care must be made to keep the CPU attentive to the receiving portion of the program. Even the resulting latencies from a single print statement in line with the receiving algorithm may cause the buildup and eventual overflow of the UDP buffer. Priority should be given to the receiving portion of the program given the limitations of the CPU. When the CPU cannot receive data as fast as they are sent, the kernel UDP buffer will overflow. Thus, a multithreaded program structure is mandated so that disk activity can be decoupled with the receiving algorithm. Given that disk activity and disk latencies are properly decoupled, appropriate scheduling priority is given to the receiving thread, and rate control is properly implemented, optimal transfer rates will be obtained given virtually any two host configurations. Reliable UDP works by assigning an ordered ID to each packet. In this way, the receiver knows when packets are missing and how to group and write the packets to disk. As stipulated previously, the receiver gets packets from the network and writes them to disk in parallel. Since most disks have written speeds well below that of a high-speed network, a growing buffer of data waiting to be written to disk will occur. It is therefore a priority to maximize disk performance. If data grams are received out of order, they can be dynamically rearranged from within the buffer, but a system waiting for a packet will have to halt disk activity at some point. In this scenario, we propose that when using PA-UDP, most of the time it is desirable from a performance standpoint to naively write packets to disk as they are received, regardless of order. The file can then be reordered afterward from a log detailing the order of ID reception. Note that this algorithm is only superior to in-order disk writing if there are not too many packets lost and written out of order. If the rate control of PA-UDP functions as it should, little packet loss should occur and this method should be optimal. Otherwise, it may be better to wait for incoming packets that have been lost before flushing a section of the buffer to disk.

VI. IMPLEMENTATION DETAILS

To verify the effectiveness of our proposed protocol, we have implemented PA-UDP according to the architecture. Written mostly in C for use in Linux and Unix environments, PA-UDP is a multithreaded application designed to be self-configuring with minimal human input. We have also included a parametric latency simulator, so we could test the effects of high latencies over a low-latency Gigabit LAN.

A. High Speed TCP

High speed TCP mechanisms deliver all of the familiar characteristics and benefits of TCP, but scale performance to hundreds of megabits per second for individual TCP. High speed TCP addresses issues in the “congestion avoidance” algorithm by specifying an approach where each individual high-speed TCP connection manages its congestion window as if it were an aggregate of multiple TCP connections. This allows a TCP window to expand more appropriately in a high-speed WAN environment, as well as to reduce its TCP window less disruptively when congestion is encountered. Second adjustment to the TCP algorithms which addresses the management and expansion of each connection’s TCP window when in the “slow start” phase. Rather than the old

method involving exponential expansion of the TCP window from a direct approach that expands the TCP window more expeditiously in the "slow start" phase. This approach accounts for the round-trip latency in the WAN and prevents TCP connections from stalling at low throughputs relative to overall available bandwidth for extended periods of time.

B. UDT

UDT is the data transport protocol proposed by this dissertation to support the distributed data intensive applications in wide area high-speed networks. UDT addresses the solution by investigating two orthogonal research problems: 1) the design and implementation of transport protocols with respect to throughput and CPU usage; and, 2) the Internet congestion control algorithm with respect to efficiency, fairness, and UDT uses packet-based sequencing to check packet loss and guarantee data reliability. It is specially designed for high-speed bulk data transfer by aiming to remove or reduce the overhead of memory copy, loss information processing, acknowledging, etc. The built-in (default) UDT congestion control algorithm is proposed to utilize high bandwidth efficiently and fairly. The UDT algorithm uses a loss-based AIMD mechanism. Bandwidth estimation technique is used to optimize its increase parameter dynamically. A random decrease factor is used to remove the negative effect of loss synchronization. UDT is not used to replace TCP on the Internet where the bottleneck bandwidth is relatively small and there are large amounts of multiplexed short life flows. However, when coexisting with TCP flows, UDT is designed not to occupy more bandwidth than TCP does unless the TCP flows fail to utilize their fair share due to TCP's efficiency problems in high bandwidth-delay product (BDP) environments. This design goal is due to the fact that TCP will still be used in these high BDP networks, and application that uses UDT may sometimes run on public networks.

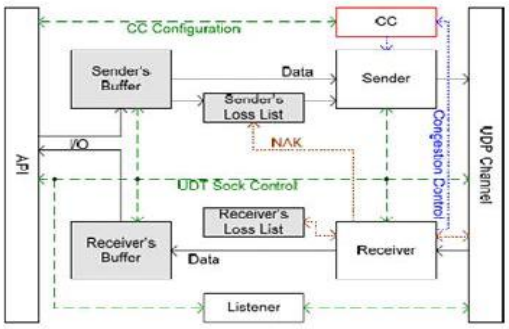

Fig: (b) UDT architecture

C. Hurricane

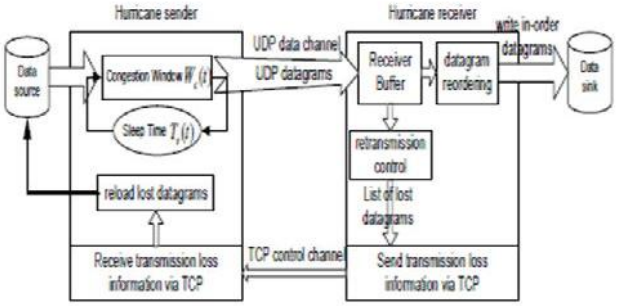

Fig: (C) Hurricane

Hurricane is based on the basic ideas of existing protocols but provided us a more convenient parameter tuning mechanism. By an explicit manual tuning of the parameters, we were able to achieve goodput rates. Hurricane is developed exclusively for high-speed file transfer on dedicated links. It does not attempt to be TCP friendly unlike some UDP-based protocols (UDT, for example) that are meant to be used over shared networks. Like other protocols in the same category, Hurricane employs a loose flow control mechanism and various (high) levels of persistent source rates. The design goal of Hurricane is to maximize link utilization without any expectation of sharing the channel. It is worth pointing out that in the design of Hurricane, we fix both window size and sleep time to achieve a flat source rate for a transport experiment with a specific bandwidth request. For transport protocols with more sophisticated rate control strategies such as the one we use later for stable streams, these control parameters are dynamically adjusted during one transport session. Hence, we associate these parameters here with time just for generalization purposes. Note that even with fixed window size and sleep time, the measured source rate is not likely to remain exactly the same over time due to the interactions between the kernel and processes. A Hurricane receiver accepts incoming UDP data grams, which are either written immediately to the local storage device if they arrived in order, or placed temporarily in a buffer for reordering otherwise. Whenever a control event is triggered, a sequential scanning is performed on the receiving buffer to check for a list of missing data grams. The datagram ID numbers on this list are grouped together and sent over a separate TCP channel to the Hurricane sender. The sender then reloads the missing datagram's into the sender congestion window for retransmission.

D. Tsunami

Our target networks for using Tsunami are typically not congested; the protocol did need some form of flow control in order to avoid exceeding the capability of the client to handle the incoming traffic. However, we still wanted to avoid collapsing the transmission rate in the presence of low-level packet loss. For this reason, we settled upon an adjustable error threshold. Packet loss above the threshold causes an exponential rise in the inter-packet delay; loss below the threshold causes an exponential decrease in the inter-packet delay until a target rate has been met. During a file transfer, the client has two running threads. The network thread handles all network communication, maintains the retransmission queue, and places blocks that are ready for disk storage into a ring

buffer. The disk thread simply moves blocks from the ring buffer to the destination file on disk. The server creates a single thread in response to each client connection that handles all disk and network activity. The client initiates a Tsunami session by connecting to the TCP port of the server. Upon connection, the server sends a small block of random data to the client. The client then xor's this random data with a shared secret, calculates an MD5 checksum, and transmits the result to the server. The server performs the same operation on the random data and verifies that the results are identical. We thus establish client authentication. After exchanging protocol revision codes, the client sends the name of the requested file to the server. If the server indicates that the file is available, the client sends its desired block size, target transfer rate, error threshold, and inter-packet delay scaling factors. The server responds with the length of the file, the agreed-upon block size, the number of block, and a timestamp. The client then creates a UDP port and transmits the port number. At this point, we are ready to transmit the file.

VII. DISK ACTIVITY

In the disk write threads, it is very important from a performance standpoint that writing is done synchronously with the kernel. File streams normally default to being buffered, but in our case, this can have adverse effects on CPU latencies. Normally, the kernel allocates as much space as necessary in unused RAM to allow for fast returns on disk writing operations. The RAM buffer is then asynchronously written to disk, depending on which algorithm is used, write-through, or write-back. We do not care if a system calls to write to disk halts thread activity, because disk activity is decoupled from data reception and halting will not affect the rate at which packets are received. Thus, it is not pertinent that a buffer be kept in unused RAM. In fact, if the transfer is large enough, eventually, this will cause a premature flushing of the kernel's disk buffer, which can introduce unacceptably high latencies across all threads. We found this to be the cause of many dropped packets even for file transfers having sizes less than the application buffers. Our solution was to force synchrony with repeated calls to sync. We employed two parallel threads to write to disk. Since part of the disk thread's job is to corral data together and do memory management, better efficiency can be achieved by having one thread do memory management, while the other is blocked by the hard disk and vice versa. A single-threaded solution would introduce a delay during memory management. Parallel disk threads remove this delay because execution is effectively pipelined. We found that the addition of a second thread significantly augmented disk performance. Since data may be written out of order due to packet loss, it is necessary to have a reordering algorithm which works to put the file in its proper order.

A. Retransmission and Rate Control

TCP is used for both retransmission requests and rate control. PA-UDP simply waits for a set period of time, and then, makes grouped retransmission requests if necessary. The retransmission packet structure is identical to Hurricane An array of integers is used, denoting datagram ID's that need to be retransmitted. The sender prioritizes these requests, locking down the UDP data flow with a mutex while sending the missed packets. It is not imperative that retransmission periods be calibrated except in cases where the sending buffer is small or there is a very large rtt. Care needs to be made to make sure that the rtt is not more than the retransmission wait period. If this is the case, requests will be sent multiple times before the sender can possibly resend them, resulting in duplicate packets. Setting the retransmission period at least five times higher than the rtt ensures that this will not happen while preserving the efficacy of the protocol. The retransmission period does directly influence the minimum size of the sending buffer, however. For instance, if a transfer is disk-to-disk and the sender does not have a requested packet in the application buffer, a seek time cost will incur when the disk is accessed nonsequentially for the packet. In this scenario, the retransmission request would considerably slow down the transfer during this time. This can be prevented by either increasing the application buffer or sufficiently lowering the retransmission sleep period. As outlined in, the rate control is computationally inexpensive. Global count variables are updated per received datagram and per written datagram. A profile is stored before and after a set sleep time. After the sleep time, the pertinent data can be constructed, including r (recv), r(disk), m, and f. These parameters are used in conjunction with and to update the sending rate accordingly. The request is sent over the TCP socket in the simple form "RATE: R," where R is an integer speed in megabits per second (Mbps). The sender receives the packet in the TCP monitoring thread and derives new sleep times from specifically; the equation used by the protocol is

$$t_d = \frac{L + \sqrt{L^2 - 4\beta LR}}{2R},$$

where R represents the newly requested rate. As per the algorithm in if the memory left is larger than the amount left to be transferred, the rate can be set at the allowed maximum.

B. 3-way Handshake

The "three-way handshake" happens thus. The originator (you, hopefully) sends an initial packet called a msg req(Control message) to establish communication .The Control Message consist of information about file name , file size in bytes, number of packets to be sent, sender ,destrination,port number. The destination then receives the control message, this message helps the receiver to configure its communication channel and sends a "SYN/ACK" . The originator then returns an "ACK" which acknowledges the packet the destination just sent him. The connection is now "OPEN" and ongoing communication between the originator and the destination are permitted until one of them issues a "FIN" packet, or a "RST" packet, or the connection times out. All the protocols of the Internet which need "connections" are built on the TCP protocol. The "three way handshake" establishes the communication. Much like you picking up your phone, getting a dial tone, dialing the number, hearing ringing, and then the other party saying "hello" or "mushi mushi."

VIII. Conclusion

In this paper are computationally inexpensive and can be added into existing protocols without much recoding as long as the protocol supports rate control via interpacket delay. Additionally, these techniques can be used to

maximize throughput for bulk transfer on Gigabit LANs, where disk performance is a limiting factor. Our preliminary results are very promising, With PA-UDP matching the predicted maximum performance. The prototype code for PA-UDP is available online at http://iweb.tntech.edu/hexb/pa-udp.tgz. In addition to low packet loss and high throughput, PAUDP has shown to be computationally efficient in terms of processing power per throughput. The adaptive nature of PA-UDP shows that it can scale computationally, given different hardware constraints. PA-UDP was tested against many other high-speed reliable UDP protocols, and also against BBCP, a high-speed TCP variant. Among all protocols tested, PA-UDP consistently outperformed the other protocols in CPU utilization efficiency.

REFERENCES

[1] N.S.V. Rao, W.R. Wing, S.M. Carter, and Q. Wu, "Ultrascience Net: Network Testbed for Large-Scale Science Applications," IEEE Comm. Magazine, vol. 43, no. 11, pp. S12-S17, Nov. 2005.

[2] X. Zheng, M. Veeraraghavan, N.S.V. Rao, Q. Wu, and M. Zhu, "CHEETAH: Circuit-Switched High-Speed End-to-End Transport Architecture Testbed," IEEE Comm. Magazine, vol. 43, no. 8, pp. 11-17, Aug. 2005.

[3] On-Demand Secure Circuits and Advance Reservation System, http://www.es.net/oscars, 2009.

[4] User Controlled LightPath Provisioning, http://phi.badlab.crc. ca/uclp, 2009.

[5] Enlightened Computing, www.enlightenedcomputing.org, 2009.

[6] Dynamic Resource Allocation via GMPLS Optical Networks, http://dragon.maxgigapop.net, 2009.

AUTHOR PROFILE

D.SHARMILA, is pursuing her M.Tech in Computer Science and Engineering in PRIST University, Puducherry, India

Mr.V.UDHAYA KUMAR M.TECH. Currently employed as Asst Professor in the Department of CSE., at PRIST University. Puducherry Campus.