UDP based Data Transfer Protocol over Fast Long Distance Network

Prof. Shraddha Patel[1], Prof. Kiran Acharya[2], Prof. Manoj Patel[3]
Department of Computer Engineering
[1, 2, 3] Alpha College of Engineering & Technology, Khatraj, India

Abstract- *many applications need to transfer bulk data, but their requirements are not completely the same. A basic and general requirement is high speed. Most applications require stable and reliable transfer. Traditional TCP performs very poorly over FLDnet. The main reason is its conservative congestion control mechanism like slow start and congestion avoidance and flow control mechanism UDP can transfer fast over FLDnet.UDP has no reliable transfer mechanism like sequence number, ACK and retransmission. Additionally, UDP makes no flow control and congestion control like TCP. So, it runs fast over FLDnet. But it's unreliable. If reliability is added into UDP and the advantage of fast is kept, it will satisfy the requirement of much e-Science application Based on UDP, several new transfer protocols with reliability are designed, such as Reliable Blast UDP (RBUDP), SABUL, UDP-based Data Transport (UDT), Tsunami .They are usually above the TCP and UDP and use UDP to transfer data and TCP to transfer control information.*

Keywords- transmission control protocol (TCP); Fast long Distance Network(FLDnet),RBDU Reliable Blast UDP,UDP – user datagram protocol,UDT-UDP based data transfer protocol,Transmission control protocol.

I. INTRODUCTION

TCP variants still cannot satisfy the requirement of bulk data transfer over FLDnet. Examples of large volumetric datasets involved in these applications include satellite weather data, astronomy observation data, and network monitoring data. In the past these data were usually stored in local storage and then were delivered or processed in a batch mode. Today they can be transferred to a remote site in real time and be processed there. Unfortunately, existing applications cannot automatically utilize the high speed networks when they move from the traditional Internet with relatively lower speed. TCP significantly underutilizes the bandwidth in high speed long distance networks. With this background, we need a new transport protocol to support these distributed data intensive applications.

So we want to design and develop a new protocol. This new protocol is expected to make use of the bandwidth much more efficiently than TCP, while allowing them both to share the network with fairness and friendliness.

II. UDP BASED PROTOCOL

Up to now, most researchers focus on modification on TCP. But, there are still some researchers try to design new protocols based on UDP. UDP is much simpler than TCP. It merely adds ports to identify individual application processes and a checksum to detect erroneous packets and simply discard them.UDP has no reliable transfer mechanism like sequence number, ACK and retransmission. Additionally, UDP makes no low control and congestion control like TCP. So, it runs fast over FLDnet. But it's unreliable. If reliability is added into UDP and the advantage of fast is kept, it will satisfy the requirement of many e-Science applications. There is no protocol directly modified on UDP known to the authors.

UDP based protocols usually above the TCP and UDP and use UDP to transfer data and TCP to transfer control information. Based on UDP, RBUDP adds simple ACK and retransmission mechanism to guarantee reliability. RBUDP firstly uses UDP to continually transfer all the data, the receiver keeps a tally of the packets that are received but gives no ACK until it receives the finish signal DONE. SABUL is an application level data transfer protocol for high bandwidth delay product networks. It use rate based congestion control that tunes the inter packet transmission time to achieve efficiency and fairness.UDT is more complex than RBUDP and it's similar to TCP. Based on UDP, in addition to add. For reliability; UDT receiver sends selective acknowledgment (SACK) at a fixed interval and sends negative acknowledgment (NAK) once a loss is detected to explicitly feedback packet loss. Reliability, UDT also adds congestion control and flow control mechanism. The basic principle of Tsunami is the same as RBUDP. Tsunami receiver periodically (every 50 blocks) makes a retransmission request and periodically calculates the current error rate and sends it to the sender. Second, Tsunami adds rate-based congestion control mechanism. The sender control the sending rate by adjusting the inter-packet delay according to the error rate received.

2.1 RBUDP

Based on UDP, R111 'DP adds **simple ACK and retransmission** mechanism to guarantee reliability. But it is different from TCP's ACK. In TCP, receiver sends an ACK to sender for each or every other received segment. This is too frequent. Sending ACK takes up handling time and the ACK packets also take up bandwidth. On other hand, in FLDnet, because the RTT is large, after the sender sends out all the packets allowed by window, it may has to wait for a long time to receive the ACK.While, RBUDP firstly uses UDP to continually transfer all the data, the receiver keeps a tally of the packets that are received but gives no ACK until it receives the finish signal DONE. Then, the receiver sends an ACK consisting of a bitmap tally of the received packets by TCP. The sender resends the missing packets and the process repeats until no more packets need to be retransmitted. Uses UDP for data traffic and TCP for signaling traffic. Estimates available bandwidth on the network using Iperf/app_perf (this requires user interaction i.e, NOT automated). Tries to send just below this rate in “blasts” to avoid losses (payload = RTT * Estimated BW). If losses do occur within a “blast”, TCP is used to exchange loss reports. Lost packets are recovered by retransmitting the lost packets in smaller “blasts”.

Advantages

Keeps the pipe as full as possible, Avoid TCP’s per-packet ack interaction, Provides analytical model- so performance is “predictable”.

Disadvantages

Sending rate needs to be adjusted by the user (no means of automatically adjusting sending rate in response to the dynamic network conditions) -Thus the solution is good *ONLY* in dedicated/QOS supported networks.

No flow control - a fast sender can flood a slow receiver. Offered solution is to use app_perf (modified Iperf developed by the authors to take into account the receiver bottleneck) for bandwidth estimation.

Another problem is that because the sender has to keep all the packets that have been sent for retransmission if needed, if the file size is bigger than the memory, it can’t be done**.**

2.2 UDT

UDT is more complex than RBUDP and it's similar to TCP. Based on UDP, in addition to add reliability, UDT also adds congestion control and flow control mechanisms. For reliability, UDT receiver sends selective acknowledgment (SACK) at a fixed interval and sends negative acknowledgment (NAK) once a loss is detected to explicitly feedback packet loss. For congestion control, UDT adopts DAIMD (AMID with decreasing increase) algorithm to adjust sending rate (not congestion window size like TCP). To differentiate congestion and error, UDT does not react to the first packet loss, while decreases the sending rates i f there is more than one packet loss in a congestion event**.** Additionally. UDT uses receiver-based packet pairs to estimate the link capacity. Like TCP, UDT also uses a low control window to limit the number of unacknowledged packets.

2.2.1 Configurable congestion control

CCC provide tool for fast implementation ,deployment and evaluation of new congestion control algorithms.CCC is one of the features included in the current UDT.CCC is written in C++ and it provides a set of control events handlers and parameters in a base C++ class. The UDT/CCC library is at the application level and it does not need root privilege to be installed. Meanwhile, it was specially developed to require very UDT/CCC is Udp based data transport library with configurable congestion control allow user to make use of new congestion control algorithm through simple few changes to the existing applications. Udt describes its design, implementation, and evaluation.

Two orthogonal elements
    The UDT protocol
    The UDT congestion control algorithm
Protocol design & implementation
    Functionality
    Efficiency
Congestion control algorithm
    Efficiency, fairness, friendliness, and stability

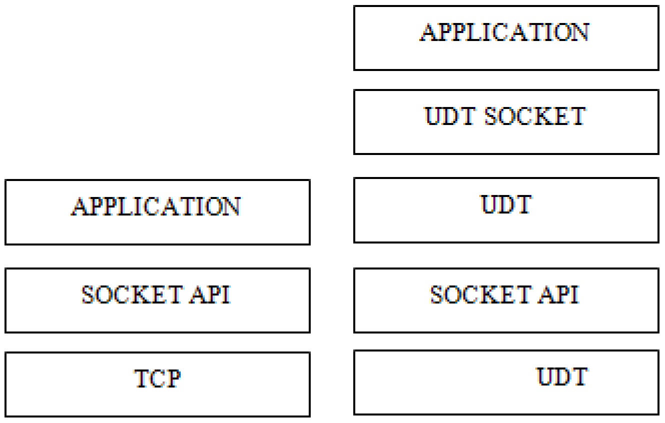
UDT protocol and the UDT congestion control algorithm.

- **The UDT protocol** can work with many different congestion control algorithms
- **Congestion control algorithm** can be used in other protocols. (For example, using it in TCP to form a new TCP variant.)
- **Protocol design and implementation** decides the functionalities of a protocol, and it also contributes partly to the efficiency

- **Congestion control algorithm** decides the characteristics of fairness, friendliness, stability, and part of efficiency.

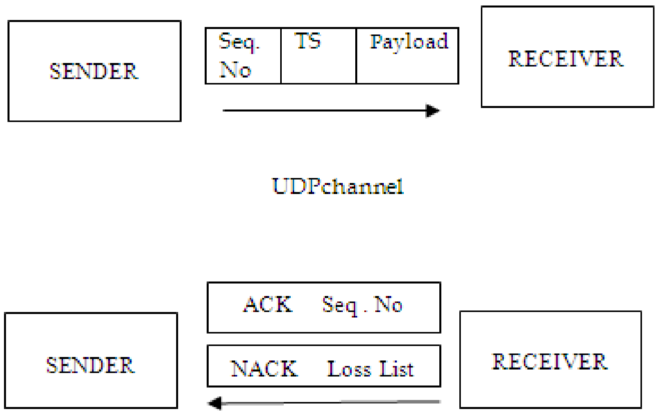
- Both of the two elements have an impact on the efficiency. The former affects the efficiency through the usage of the processor time, whereas the latter through the tuning of data sending rate.

2.3.2 UDT OVERVIEW



Applications make use of transport protocols such as TCP through a socket API. To use UDT, applications interact with the UDT socket API, while UDT uses system socket API to transfer data over UDP.

Protocol Architecture



This is an abstract view of a UDT instance. The timer is used to trigger various timeout events. In order to send user data to another UDT instance, a single UDP channel is set up. (Note that this channel only exists logically, as UDP is not connection-oriented.) Sender A sends user data in UDP packets to Receiver B. Each UDP packet is assigned a unique sequence number and a timestamp. Receiver B will periodically feedback acknowledgments and loss reports as well.

2.4 Functionalities

The reliability control is provided by sequencing and acknowledgment. Each UDT packet is assigned a unique increasing sequence number. The receiver will send back acknowledgments and loss reports according to packet arrival. Lost packets will be retransmitted.

The streaming service is implemented in buffer management.

UDT also has a series of mechanisms to set up, maintain, and tear down a UDT connection.

Finally, each UDT instance has both a sender and receiver, in order to provide duplex data transfer service.

- A UDT entity has two parts: the sender and the receiver

- The sender is responsible for data packet sending and the receiver is responsible for data packet receiving, control packet sending and receiving, and timer expiration detection.

- All data and control packets in both directions are transferred between a pair of UDP ports.

- The receiver maintains 4 self-clocked timers, which are queried after each time bounded UDP receiving.

- They are ACK, NAK,SYN, and EXP, for acknowledging, loss report, rate control, and timer out detection, respectively.

- ACK and SYN timers are constant and fixed in UDT, while NAK and EXP timers are updated during the runtime of UDT according to the RTT and current data transfer speed.

Efficiency Consideration

- Less packets
  - Timer-based acknowledging
- Less CPU time
  - Reduce per packet processing time
  - Reduce memory copy

- Reduce loss list processing time
- Light ACK vs. regular ACK

- Parallel processing
  - Threading architecture
- Less burst in processing
  - Evenly distribute the processing time

Application Programming Interface (API)

An application can make use of the UDT/CCC library in four ways. The library provides a set of C++ API that is very similar to the system socket API. Network programmers can learn it easily and use it in a similar way as using TCP sockets. In particular, applications can use the setsockopt/getsockopt method to set and configure a specific congestion control algorithm at run time.

UDT's congestion control algorithm.

- When we say "congestion control" here, we mean the mechanism to effectively utilize the network bandwidth. Another term, "flow control", is usually used to name the approach to prevent incoming traffic overwhelming the receiver. Congestion control is a more difficult problem.
- Congestion control can tune the data sending rate in two ways: limit the number of on-flight packets and set the inter-packet time. TCP uses the window-based approach, whereas UDT uses the rate-based approach.
- The most common congestion control algorithm is TCP's AIMD algorithm. We have already discussed AIMD.
- Another important factor in congestion control is how to indicate a network congestion. Most protocols use packet loss information, including both TCP and UDT. The increase in round trip time delay can also indicate congestion somewhere along the network.
- Congestion control vs. flow control
  - Congestion control: effectively utilize the network bandwidth
  - Flow control: prevent the receiver from being overwhelmed by incoming packets
- Window-based vs. rate-based
  - Window-based: tune the maximum number of on-flight packets (TCP)
  - Rate-based: tune the inter-packet sending time (UDT)
- AIMD: additive increases multiplicative decreases
- Feedback
  - Packet loss (Most TCP variants, UDT)
  - Delay (Vegas, FAST)

The CCC Interface

We identify four categories of configuration features to support configurable congestion control mechanisms. They are

1) Control Event Call backs

Seven basic call-back functions are defined in the base CCC class.Init and close, onACK, onLoss, on Timeout, onPktSent, onPktReceived, processCustomMsg.

2) Protocol Configuration

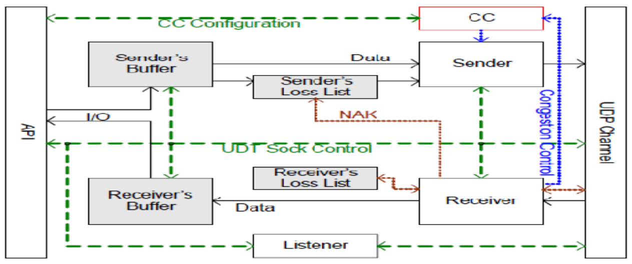
To accommodate certain control algorithms, some of the protocol behaviour has to be customized. For example; a control algorithm may be sensitive to the way that data packets are acknowledged. UDT/CCC provides necessary protocol configuration APIs for these purposes. Finally, UDT/CCC also allows users to modify the values of RTT (Round Trip Time) and RTO. A new congestion control class can choose to use either the RTT value provided by UDT, or its own calculated value. Similarly, the RTO value can also be redefined. There are other features of the UDT protocol that are either not related to congestion control or are helpful to most control algorithms. These features, such as selective acknowledgement (SACK) and robust reordering (RR) cannot be configured by CCC users, although some of the features can be configured through UDT interfaces.

3) Packet Extension

It is necessary to allow user-defined control packets for a configurable protocol stack.

4) Performance Monitoring

Protocol performance information supports the decisions and diagnosis of a control algorithm. The performance monitor provides information including the duration time since the connection was started, RTT, sending rate, receiving rate, loss rate, packet sending period, congestion window size, flow window size, number of ACKs, and number of NAKs.



Software Architecture

In UDT/CCC software architecture, UDT layer has five function components: the API module, the sender, the receiver, the listener, and the UDP channel, as well as four data components: sender's protocol buffer, receiver's protocol buffer, sender's loss list, andreceiver's loss list.

III. TSUNAMI

The basic principle of Tsunami is the same as RBUDP. Tsunami mainly makes two points of improvement on RBUDP.

First, Tsunami receiver does not wait for finishing of all the data transfer, but periodically (every 50 blocks) makes a retransmission request and periodically calculates the current error rate and sends it to the sender. Second, Tsunami adds rate-based congestion control mechanism. The sender control the sending rate by adjusting the inter-packet delay according to the error rate received. Additionally, if the number of packets that need to be retransmitted is too large, the sender will restart transmission of the file at the given block number.

IV.SABUL

SABUL is an application level data transfer protocol for high bandwidth delay product networks. It use rate based congestion control that tunes the inter packet transmission time to achieve efficiency and fairness. In order to remove fairness bias between flows with different network delay SABUL has been implemented as an open C++ Library. SABUL connection is unidirectional: data can be sent from one side to the other side.SABUL designed for reliability, high performance, fairness and stability.

SABUL use two connection: Control connection over TCP and DATA connection over UDP. Data is send from sender to receiver Over UDP channel while Control information containing current state of transfer is sent over TCP channel from receiver to sender.

Performance Characteristics:

Efficiency: Utilize bandwidth efficiently in high BDP environment.

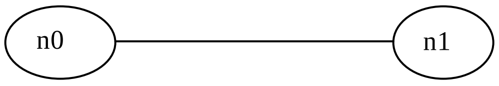
Intra Protocol fairness: SABUL is designed to be fair with other SABUL flows so that data intensive application like grid application can employ parallelism. Intra protocol fairness is examine by simulating SABUL flows with different initial sending rate and different rtts.

TCP Fairness: Designed to be friendly to TCP flow so that it can be safely deployable on public network.

Stability: Protocol is more stable if it has less oscillation throughput.

SIMULATION

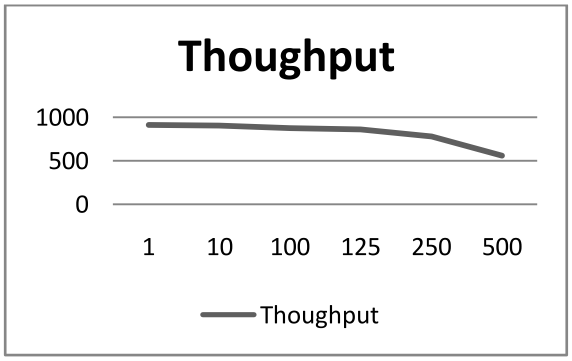
Simulation of UDT is performed in NS2. Figure shows the topology for simulation.



Simulation is setup for a point to point topology of two directly connected nodes as shown fig. Different simulation Scenario in ns2 for performance analysis. Scenario run for two directly connected node. Simulation use DropTail queue. Queue Size is set at least BDP.Link Capacity is 1Gbps.UDP packet size is 1500 bytes.

Measuring the efficiency of UDT proptocol

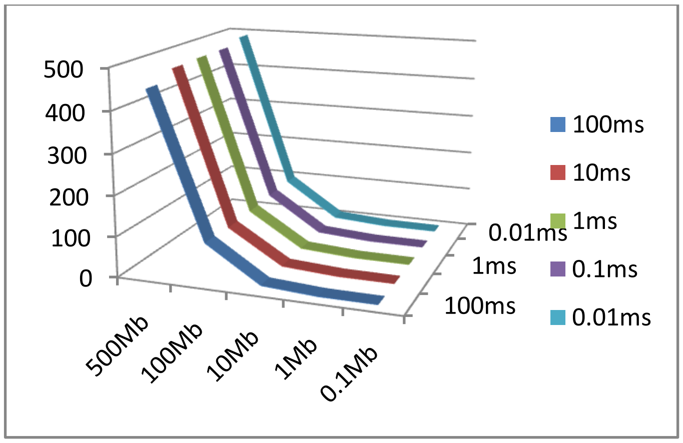
RTT Vs Throughput is shown in below figure.



UDT perform better than TCP in high RTT environment. As shown in fig. because of TCP AIMD algorithm throughput of TCP degrade as RTT increase.

With reference to topology as shown in figure we have Queue Size is 1000 and with varying RTT 100,10,1,0.1,0.01 and Bandwidth 500,100,10,1,0.1.

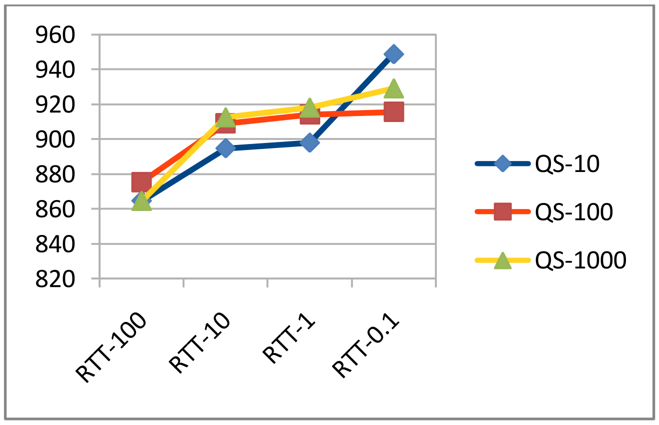
Bandwidth utilization of UDT flow with different link capacity and different RTT shown in figure.



Bandwidth utilization of UDT flow with different link capacity and different RTT in figure..Udt protocol has higher bandwidth utilization.

Measuring impact of queue size

If we set Bandwidth 1000Mbps and Queue Size 10,100,1000 with varying RTT 100,10,1,0.1.

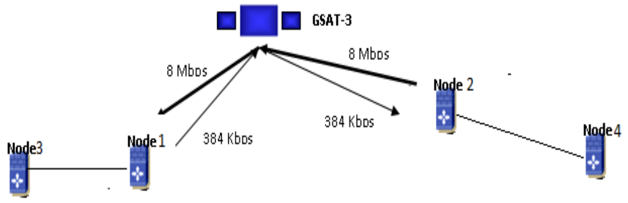


Result show that UDT can reach high bandwidth utilization with very small queue size. In 1Gbps ,10 ms RTT network, it only require a queue of 10 packets length to reach 85% of bandwidth utilization, and 100 packets length to reach 97% bandwidth utilization.

UDT can reach high bandwidth utilization with very small queue size.

Simulation of UDT

TOPOLOGY:



As shown in Figure, simulation topology utilizes GSAT-3 satellite configuration having maximum 8 Mbps forward link and 384 kbps return channel. The satellite is geo stationary. Our test set up is configured in mesh topology having 580-600 ms RTT and varying BER.

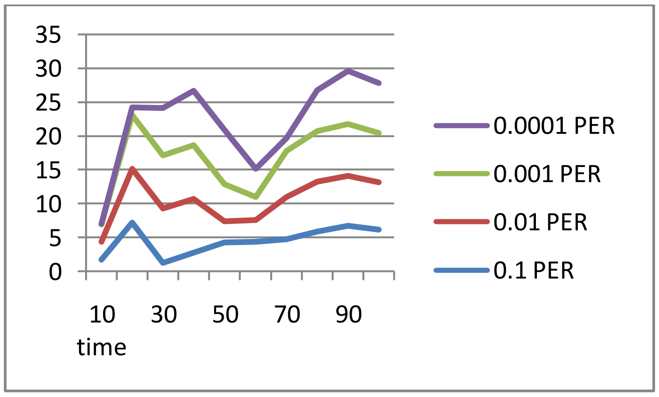
Duplex link:

n4-n2: 8 Mb Bandwidth, 2.5 ms Delay
n1-n3: Mb Bandwidth, 2.5 ms Delay

Simplex link:

n2-n0: 8 Mb Bandwidth, 590 ms Delay
n0-n2: 384Kb Bandwidth, 590 ms Delay
n0-n1: 8 Mb Bandwidth, 590 ms Delay
n1-n3: 384Kb Bandwidth, 590 ms Delay

Throughput of UDT flow VS Simulation time



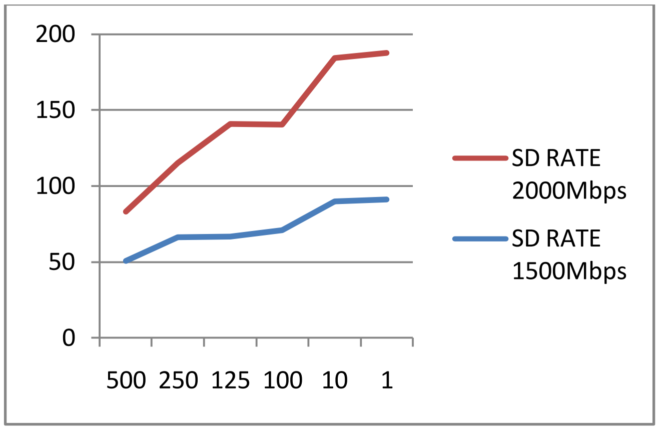
Observation:

Result shows that throughput of UDT flow against link error rate. As shown in fig. Throughput of UDT is decrease as link BER increase from 0.0001, 0.001, 0.01, and 0.1.UDT perform better in high RTT with high PER situation.UDT does not reduce its sending rate for any loss report, and increase link capacity. Result shows that throughput of single UDT flow against link error rate.BER set to 0.1, 0.01, 0.001, and 0.0001. As shown in graph throughput decrease as BER increase from 0.0001 to 0.1.UDT perform better with high BER.

RBUDP

Simulation is run for same topology as for UDT.Link efficiency is set to 1 Gbps with different packet size is set to 1500 bytes. Queue is DropTail queue. Queue size to least BDP. In RBUDP, sending rate needs to be adjusted by the user. RBUDP protocol can tries to send data packets just below the sending rate in 'blast' and the sending rate can be specified in advance by user. It sends data at user specified sending rate. Thus the protocol works only for private or QOS enabled network.

RTT Vs Throughput



RBUDP protocol can tries to send data packets just below the sending rate in 'blast' and the sending rate can be specified in advance by user.

It sends data at user specified sending rate.Thus the protocol works only for private or QOS enabled network.

V. CONCLUSION

This new protocol is expected to make use of the bandwidth much more efficiently than TCP, while allowing them both to share the network with fairness and friendliness.UDT also performing better in high RTT environment.

REFERENCES

[1] Y. H. Gu and R. L. Grossman, "UDT: UDP-based data transfer for highspeed wide area networks," Computer Networks, vol. 51, pp. 1777-1799,2007

[2] Xiuchao Wu, Mun Choon Chan, A. L. Ananda, and Chetan Ganjihal, A New High Speed Congestion Control Algorithm for Safely Ramping Up Bandwidth-greedy and Elastic Applications of the Internet, IEEE ICNP2009 conference.

[3] UDT open source project, http://udt.sf.net

[4] UDT: A Transport Protocol for Data Intensive Applications, Yunhong Gu and Robert L. Grossman,draft-gg-udt-02.txt.

[5] Performance Comparison of UDP-based Protocols Over Fast Long Distance Network. Yongmao Ren, Haina Tang, Jun Li and Hualin Qian.

[6] Configurable Congestion Control Feature of UDT Protocol. International Journal of Computer Science and Telecommunications [Volume 3, Issue 5, May 2012].

[7] Performance Evaluation of UDP-based High-speed Transport Protocols, Zhaojuan Yue, Yongmao Ren, Jun Li

[8] Supporting Configurable Congestion Control in Data Transport Services Yunhong Gu1 and Robert L. Grossman

[9] Experiences in Design and Implementation of a High Performance Transport Protocol Yunhong Gu, Xinwei Hong, and Robert L. Grossman.

[10] Optimizing UDP-based Protocol Implementations, Yunhong Gu and Robert L. Grossman.

[11] GridTCP: A transport layer data transfer protocol for satellite based Grid Computing Haresh S. Bhatt, Hitesh J Kotecha, VH Patel, K. Bandyopadhyay