Characteristics of 10Gbps Optical Networks and Transport Protocols

Project Report

In

The Department of Computer Science

by

Yixin Wu

B.S., Soochow University, 2001

December, 2008
Acknowledgements

I would like to thank many people who made this project possible. First, I would like to thank my advisor Dr. Seung-Jong Park for encouraging and inspiring me throughout my research. I also like to thank him for giving me this great opportunity to do this cutting edge research. I would also like to acknowledge him for his invaluable feedback, without which this project would never be possible.

I would like to thank my committee members for their support for this project. I would also like to thank my fellow students Suman Kumar, Yasser Ahmed Mohammed, and Cheng Cui for constant inspiring and motivating me. I would like to thank Andrei Hutanu for informing me the possibility of using UDT and some profiling tools. I would also like to thank Dr. Gabrielle Allen for all the advice. This project wouldn’t have been completed without their help.

I want to thank CCT and LONI for letting me utilize their precious network resources to complete my work. Finally, I would like to thank all my friends and family for their support.
# Table of Contents

Acknowledgements ..........................................................................................................................2  
List of Figures .............................................................................................................................4  
Abstract .....................................................................................................................................5  
1. Introduction ............................................................................................................................7  
2. TCP Variants ...........................................................................................................................9  
   2.1 TCP ....................................................................................................................................9  
   2.2 High Speed TCP (HSTCP) ...............................................................................................11  
   2.3 Scalable TCP ..................................................................................................................11  
   2.4 Hamilton TCP (H-TCP) ..................................................................................................12  
   2.5 FAST TCP .....................................................................................................................13  
   2.6 BIC and CUBIC .............................................................................................................13  
3. Experiment Setup ....................................................................................................................15  
4. Results and analysis ................................................................................................................18  
   4.1 Throughput of TCP variants over different packet loss rate ........................................18  
   4.2 RTT Fairness of TCP variants on 10Gbps back-to-back connection ..............................20  
   4.3 Throughput and congestion window measured on Enlightened testbed ....................22  
   4.4 UDP performance on Enlightened testbed ...................................................................23  
   4.5 Performance comparison of UDT and TCPs on a non optimized 10Gbps link ..........25  
5. Conclusion .............................................................................................................................28  
References ..................................................................................................................................30  
Vita ............................................................................................................................................32
List of Figures

Figure 2-1 TCP/IP model........................................................................................................9
Figure 2-2 Problem of AIMD over high speed networks ....................................................10
Figure 3-1 Back-to-back Connection with Emulated Delay .............................................15
Figure 3-2 architecture of Enlightened testbed.................................................................16
Figure 4-1 Performance of TCPs in different packet loss rate from $10^{-5}$.......................19
Figure 4-2 Performance of TCPs in different packet loss rate from $10^{-9}$.......................19
Figure 4-3 Fairness of TCPs in a low speed network scenario .......................................21
Figure 4-4 RTT Fairness of TCPs in a high speed network scenario ...............................22
Figure 4-5 TCP throughput and congestion window on Enlightened testbed...............23
Figure 4-6 Enlightened UDP throughput from Baton Rouge to Los Angeles ...............24
Figure 4-7 UDP throughput from Los Angeles to Baton Rouge .....................................25
Figure 4-8 Throughput comparison of TCP and UDT ..................................................26
Abstract

The Additive Increase Multiplicative Decrease (AIMD) mechanism of the Transport Control Protocol (TCP) congestion control worked very well on low speed networks over the past decades. But as the emergence of high speed networks, such as 10Gbps optical networks, AIMD of TCP cannot make the best use of this huge bandwidth because of slow additive increase and dramatic multiplicative decrease.

A lot of efforts have been put to this issue. TCP variants, such as BIC, CUBIC, FAST TCP, HSTCP, STCP, Scalable TCP and HTCP have been proposed by the network research community based on different philosophy. They were proved by experiments to outperform traditional TCP in case of throughput in high speed networks. But their fairness behaved differently in different scenarios.

Besides TCP, some application level protocols based on UDP also showed some advantages in high speed networks. UDT is one of the most popular application level protocols. It provides much more flexibilities when used to transfer bulk data in high speed networks.

In this work, we aim to perform the following experiments:

- Compare the performance of TCP variants in different packet loss rate scenarios.
- Compare the RTT fairness of TCP variants in a valid 10Gbps optical network.
- Reveal some characteristics of the realistic optical network.
Compare the performance of UDP based application level protocol UDT and TCP in a common scenario.

Explore the shortcomings of back-to-back 10Gbps connections and realistic 10Gbps optical networks, and show a possible solution which is our current work.
1. Introduction

Nowadays, there have been several research initiatives which developed and deployed high speed networks over major research institutions. These national or regional networks, such as NLR (National Lambda Rail) [1], INTERNET2 [2], LONI (Louisiana Optical Networks Initiative) [3], etc., provide bandwidths greater than 10Gbps. Meanwhile, many new data intensive distributed applications, which require high speed networks, have emerged, such as high definition class broadcast and distributed visualization. In HD class broadcast, it demands at least 1.5Gbps $(1920\text{pixels/row})\times 1080\text{(rows/frame)} \times 3\text{(bytes/pixel)} \times 30\text{(frames/second)} \times 8\text{(bits/byte)}$ bandwidth.

TCP/IP has been a very good choice for low bandwidth network. Due to the wide use of TCP/IP networks and the increment of bandwidth, it has been studied and improved over the last twenty five years to provide higher link utilization and fairness among competing flows. It is well known that the mechanism of Additive Increase Multiplicative Decrease (AIMD) congestion control algorithm is not suitable for high speed networks. Additive increase is too conservative to reach peak throughput quickly and multiplicative decrease is too drastic to recover from packet loss events. In order to overcome the poor performance of TCP on high-speed long-distance networks [4], [5], a number of congestion control algorithms have been proposed, such as HSTCP [4], STCP [7], BIC TCP [8], CUBIC [9], HTCP [10] and Fast TCP [6]. And for cluster environment, some specific protocols were developed for low latency and high speed, such as infiniband, myrinet, etc. Besides TCP, some reliable protocols based on UDP were also developed to facilitate the high speed of networks while providing inter and intra protocol fairness, such as UDT, RBUDP, etc.

Although 10Gbps optical networks have been used for some time, characteristics of 10Gbps networks are still not revealed, which makes the TCP congestion control algorithms more or less biased. In this article, we first have a brief description of all TCP
related congestion control algorithms. In section 3, experiment setup and related tools are discussed. Experiments results are presented and analyzed in section 4. Except TCP, some other methods, which are used to transfer data efficiently and fairly, are also discussed. Conclusion will be drawn in section 5.
2. TCP Variants

2.1 TCP

Figure 2.1 shows the TCP/IP model, in which the transport layer is the fourth layer. The widely used transport protocols used in this layer are TCP (Transfer Control Protocol) and UDP (User Datagram Protocol). TCP is a connection-oriented protocol provides reliable data transfer, whereas UDP is a connectionless unreliable protocol.

![TCP/IP model](image)

**Figure 2-1 TCP/IP model**

The main challenge of TCP is to provide efficiency and fairness simultaneously. In low bandwidth networks, AIMD is used for this purpose. The AIMD works as follows during different phases:

- **Slow start**: \( \text{cwnd} = \text{cwnd} + 1, \text{when cwnd} < \text{threshold} \)
- **Congestion avoidance**: \( \text{cwnd} = \text{cwnd} + 1/\text{cwnd}, \text{when cwnd} > \text{threshold} \)
- **Triple duplicate ACK**: \( \text{threshold} = \text{cwnd}/2, \text{cwnd} = \text{threshold} \)
- **Timeout**: \( \text{threshold} = \text{cwnd}/2, \text{cwnd} = 1; \)
Although AIMD works perfectly on low speed networks, it has a big problem for high speed networks. In figure 2.2, in order to achieve a steady state throughput of 10Gbps it requires a packet drop rate at most once 1 2/3 hours.

**Figure 2-2 Problem of AIMD over high speed networks**

In fact, the throughput of a TCP flow can be approximately by the following formula [11]:

\[
T = \frac{S}{R \sqrt{\frac{2p}{3} + t_{RTO} \left( \frac{3p}{8} \right) p(1 + 32p^2)}}
\]

$S$ is TCP segment size, $p$ is the loss rate, $R$ is RTT, $t_{RTO}$ is the TCP timeout.

From this formula, TCP will perform worse in longer fatter network. Since TCP throughput depends on RTT, short RTT TCP flows will dominate long RTT flows, which is called RTT fairness problems.

Since its bad performance on long fat pipe, several TCPs are proposed recently for high speed networks, such as BIC, CUBIC, FAST TCP, HSTCP, H-TCP, and STCP. We will give a brief description for all those TCP variants.
2.2 High Speed TCP (HSTCP)

HSTCP was developed by Sally Floyd. It improves the loss recovery time of standard TCP by changing the AIMD algorithms. The modified AIMD only works in higher congestion window. When the congestion window is lower than a given threshold, it works the same as standard TCP, which provides inter protocol fairness in low speed networks.

The modified AIMD works as follows:

Slow start: cwnd = cwnd + 1

ACK: cwnd = cwnd + a(cwnd)/cwnd;

Drop: cwnd = (1 – b(cwnd))*cwnd;

where a(cwnd) and b(cwnd) depends on the current value of congestion window. The range of a(cwnd) could be from 1 to 73 packets, and b(cwnd) from 0.5 to 0.09.

However, the fairness issue was not explored thoroughly in paper. Actually HSTCP behaves like parallel TCP. When there is no packet loss, the total congestion window of parallel TCP will increase more than 1 packet. When there is a loss, not all TCP flows have packet lost, which lead to the percentage of congestion window decrease less than 50%.

For 1500-byte packet size and 100ms RTT, HSTCP can achieve a steady throughput of 10Gbps with a packet loss rate at most once every 12 seconds.

2.3 Scalable TCP

Scalable TCP was proposed by Tom Kelly. Like HSTCP, it still aims at the AIMD algorithms of standard TCP to achieve shorter loss recovery time. Packet loss
recovery time of a standard TCP is proportional to the congestion window size and RTT whereas for STCP it is only proportional to RTT.

To make STCP not so aggressive in low speed networks, it behaves the same as standard TCP until the congestion window size reaches a threshold. And the congestion avoidance phase is modified as follows:

**ACK:** \( cwnd = cwnd + 0.01 \)

**Detect Congestion:** \( cwnd = cwnd - 0.125 \times cwnd \)

The value of 0.01, 0.125 and threshold window size can be changed. Actually STCP uses MIMD algorithm, which makes STCP not so fair when competing with a standard TCP in high speed networks. And MIMD cannot reach a steady state. The big advantage of STCP is high link usage.

### 2.4 Hamilton TCP (H-TCP)

H-TCP is a minor modification to standard TCP just like STCP and HSTCP. H-TCP changes increase parameter \( a(cwnd) \) according to the time between successive congestion events and the ratio of the observed minimum RTT to maximum RTT. This strategy overcomes the problem of “RTT unfairness”, which occurs to most of the TCP variants.

At a loss event, H-TCP adopts an adaptive back-off strategy and reduces its congestion window based on the minimum to maximum observed RTT ratio.

H-TCP congestion control algorithm works as follows.

**ACK:**

\[
\alpha \left\{ \begin{array}{ll}
1 & \Delta \leq \Delta^L \\
1 + 10(\Delta - \Delta^L) + \left(\frac{\Delta - \Delta^L}{2}\right)^2 & \Delta > \Delta^L \\
\end{array} \right.
\]

\[
\alpha \leftarrow 2(1 - \beta) \alpha.
\]

Packet Drop:
where alpha is the AI parameter, beta is the MD parameter, delta is the time since last packet loss, delta L is a threshold value, B(k) is the throughput before last packet loss, B(k+1) is the throughput before the current packet loss.

2.5 FAST TCP

FAST TCP is an alternative congestion control algorithm built on TCP Vegas. It aims at flow level properties, such as stable equilibrium, well-defined fairness, and high link utilization. It only requires sender side modification.

FAST TCP uses queuing delay in addition to packet loss as a sign of congestion. Standard TCP is not stable in high bandwidth delay product networks. But FAST TCP can eliminate packet level oscillations by properly choose the equation and feedback mechanism. Since it is already commercialized, the details of the congestion control algorithm are unknown. According the paper, under normal network conditions, FAST periodically updates the congestion window w based on the average RTT as follows.

\[
\beta \left( k + 1 \right) \leftarrow \begin{cases} 
0.5 & \frac{B^-(k+1)-B^-(k)}{B^-(k)} > 0.2 \\
\frac{RTT_{\text{max}}}{RTT_{\text{min}}} & \text{otherwise.}
\end{cases}
\]

where gamma is a constant between 0 and 1, RTT is the current average RTT, baseRTT is the minimum RTT observed so far, and alpha is a protocol parameter that determines the total number of packets queued in routers in equilibrium along the flow’s path. In the prototype, the window updated period is 20ms.

2.6 BIC and CUBIC
BIC and CUBIC was proposed by Networking Research Lab of NCSU to improve TCP performance in high-speed networks. A unique feature of BIC is its congestion window growth functions.

In BIC, congestion control is viewed as a search problem where the algorithm tries to find an appropriate window (target window) to make it efficient and fair. BIC uses binary search between the current window and maximum window to approach the target window. This binary search makes window increase to be logarithmic, and it increases faster when the target window is far from the current window, but slower when the current window is closer to the target window. When the current window exceeds the target window, BIC uses “slow start” to find the next maximum window – the congestion window before a packet was dropped.

The binary search technique is combined with additive increase to make the network change less dramatic. This additive increase ($S_{\text{max}}$) allows BIC to increase window faster when there are lots of available bandwidth. BIC still use multiplicative decrease after a packet loss.

CUBIC is an enhanced version of BIC. It simplifies the BIC window control and improves its TCP-friendliness and RTT-fairness while keeping the major strengths of BIC (scalability and stability). The window growth function of CUBIC is a cubic function as follows:

$$W_{\text{cubic}} = C(t - K)^3 + W_{\text{max}}$$

Where $C$ is a scaling factor, $t$ is the elapsed time since last window reduction, $W_{\text{max}}$ is the window size just before the last window reduction, and $K = \frac{3\sqrt[3]{W_{\text{max}}\beta}}{C}$, where $\beta$ is a constant multiplicative decrease factor for window reduction at packet loss event.

According to [9], CUBIC ensures intra-protocol fairness among competing flows of the same protocol and offers a good RTT fairness property.
3. Experiment Setup

Figure 3.1 shows the experiment setup. Two servers are connected back-to-back by an optical fiber cable. Both servers have the same hardware and software configurations. The computers contain dual 2.6-GHz AMD Opteron 252 processors, running on a 800MHz front side bus (FSB), having 8GB RAM, and hosting Chelsio N210-10Gbps Ethernet Adapter on the PCI-X 133Mhz/64-bit slot. From software perspective, servers are installed Ubuntu Linux with 2.6.17 kernel, which was recompiled to support Chelsio N210-10Gbps Ethernet Adapter and TCP congestion control algorithms.

For back-to-back connection, it is a very simple scenario without noticeable delay, full bandwidth, no packet loss. There is not much research can be done in such a scenario. But with software network emulator, we are able to emulate a lot more scenarios to compare the performance of different TCP variants.

Network emulators are used to emulate realistic network environments by setting various interesting network parameters such as delay, packet loss, etc. There are two categories of network emulators -- one is software emulator and the other is hardware emulator. Hardware network emulators provide more accurate parameters control and more functionality with much higher price (around $25,000 each). One example is anuesystems’ XGEM 10 Gigabit Ethernet Multi-Profile Emulator which can handle LAN PHY rate of 10.3125Gbps for optical and/or copper CX4 interfaces.

Instead of using very expensive hardware emulators, we use open source software emulators. There are three widely used software network emulators on UNIX or Linux.
platforms: Dummynet [12], NistNet [13] and NetEm [14]. They have similar designs. Dummynet is a standard portion of FreeBSD and implemented as part of the packet filtering mechanism whereas NistNet is implemented as a kernel module of Linux. NetEm is an enhancement of the traffic control facilities of Linux based on quality of service (QoS) and differentiated services facilities in the Linux kernel. Many functions in NistNet are reused in NetEm. Both dummynet and NetEm operate on outgoing packets; however NistNet operates on incoming packets. NistNet was adopted based on the following reasons. First, dummynet is part of FreeBSD and it is not available in Linux. Second, sender side will take more system resources than receiver side, which makes sender side be a potential bottleneck when transferring a 10Gbps TCP steam. Third, further results showed that NetEm has some instability and larger overhead compared to NistNet. At the time NistNet was used, it didn’t support some features needed in this experiment. So NistNet was modified to support a couple of extra functions to support more scenarios.

We also had the opportunity to perform some experiments on a realistic 10Gbps optical network – Enlightened testbed [15]. Enlightened testbed, funded by NSF, was a nationwide 10Gbps bandwidth testbed running over NLR and Louisiana Optical Network Initiative (LONI), connected via four all-photonic Calient switches, all using GMPLS control plane technologies. Figure 3.2 is the topology of the Enlightened testbed.

![Figure 3-2 Architecture of Enlightened testbed](image)

At the time, we can only access computers in LSU and Caltech.
In this report, our experiments focus on the performance of bulk data transfer. Iperf [16] and nuttcp [17] are both widely used network performance measurement tools. They also allow the tuning of various parameters and UDP characteristics.

Compared with TCP, UDP are also used to transfer data, especially for real time data. For transferring data reliably, applications using UDP have to provide a reliable mechanism. UDT is a library supporting UDP reliable data transfer. Plus, it doesn’t need any super user account to tune Linux TCP/IP stack. It can be optimized by application parameters. It also provides application level implementation of TCP congestion control algorithms. The performance of UDT and TCPs is also compared.
4. Results and analysis

In this section, results and analysis of experiments are discussed. Additional experiments, which the previous graduate student in our lab didn’t finish, were performed. We first talk about the results got from back-to-back connection by modifying the source code of emulator. Then results from Enlightened tested are discussed.

4.1 Throughput of TCP variants over different packet loss rate

Figure 4.1 is the result we can get without modifying NistNet [19]. But the problem is $10^{-5}$ is not a practical packet loss rate, let alone $10^{-3}$. In optical network, the packet loss rate is from $10^{-8}$ to $10^{-11}$.

NistNet only support a 16 bit linear congruential generator to generate a random 16 bit integer. If the packet loss is p, NistNet will use the following random number generator to get an integer and compare with p*0xFFFF. Packet will be dropped if the random number is less than p*0xFFFF, otherwise, packet will be put into a queue.

```c
//16 bit Linear Congruential Generator
long myrandom(void)
{
    static int dirty=2;
    dirty *= 69069;
    dirty += 85;
    return dirty;
}

//48 bit Linear Congruential Generator
long myrandom_48(void)
{
    static long dirty=2;
    dirty *= 44485709377909;
    dirty += 11863279;
    return dirty;
}
```

For a 16 bit linear congruential generator, it can support up to $2^{16}$ (around $1.5\times10^{-5}$) packet loss rate. But for 48 bit linear congruential generator [18], it can support $2^{48}$ (around $3.5\times10^{-15}$) packet loss rate. On 32 bit CPU computer, long is 4 bytes. And for 64 bit, it is 8 bytes.
From figure 4.2, if there is no congestion, which means the packet loss rate will be less than $10^{-8}$, all TCP variants except TCP Reno behave almost the same. The throughput of TCP Reno is only 8.7% less than other TCPs. In moderate congested network, where packet loss rate is from $10^{-8}$ to $10^{-4}$, scalable TCP is the best because of its MIMD
algorithm. In heavily congested network, all TCPs behave badly. Packet loss based TCPs will encounter more 3 duplicate ACKs or time out, which leads to congestion window decrease.

4.2 RTT Fairness of TCP variants on 10Gbps back-to-back connection

There are two kinds of fairness of TCP congestion control algorithms – intra protocol and inter protocol fairness. Intra protocol fairness is between the same congestion control algorithms. And inter protocol fairness is between different TCPs. Jain’s fairness index is used to measure the fairness of TCPs.

\[
f(x_1, x_2, x_3, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2}
\]

Since our scenario is a back-to-back connection, only intra protocol fairness can be measured. The reason is TCP congestion control algorithms are implemented in Linux kernel, and it is impossible to pick different TCPs for different flows. So we can only measure RTT fairness of TCPs. But NistNet does not provide two flows competing queue mechanism. NistNet source code was modified to support sharing FIFO queue.

First, we want to make sure that RTT fairness measured in back-to-back scenario is valid in low speed scenario by comparing with the results other researchers got. Then RTT fairness in high speed scenario will be more credible.

Figure 4.3 is the result of Jain’s fairness index. In this scenario, MTU was 1500 bytes; bottleneck bandwidth was 400Mbps; router queue was 1333 packets. Two competing TCP flows emulated by iperf had been running for 2 hours. The delay of one flow was set as 162ms. And the delay of another flow was set separately as 42ms, 82ms, and 162ms just as [20]. From figure 4.3, CUBIC is the best in case of RTT fairness because of its linear RTT fairness. HTCP is a little bit better than Reno and much better than the other TCPs. HTCP behaves as Reno in low speed network. When the delays of both flows are equal, all TCPs are very fair. The result it almost the same as [20], which means that back-to-back scenario is a valid scenario.
Since back-to-back scenario is a valid scenario, we transferred the scenario to high speed network with settings as follows.

- Linux TCP/IP stack was optimized.
- MTU= 9000 bytes.
- Bottleneck bandwidth = 5Gbps.
- Queue size = 1000 packets, which means 9M bytes.
- The delay of one flow was set as 162ms, and the other as 42ms, 82ms, 162ms.
- Iperf run one hour.

Figure 4.4 is the fairness index measured. From this figure, CUBIC is still the best because of its linear RTT fairness. HTCP is still a little better than Reno. Scalable is the worst. But highspeed TCP shows some weird curve because of its changing AI and MD parameters and it is not fair even the delay of two flows are the same. Other TCPs are fair when the delays of both flows are equal.
Figure 4-4 RTT Fairness of TCPs in a high speed network scenario

4.3 Throughput and congestion window measured on Enlightened testbed

Since we had the access to Enlightened testbed, we also measured the TCP throughput and congestion window by running TCP on different sites. We run iperf mostly between Los Angeles and Baton Rouge because we have one our own 10Gbps NIC connected to Enlightened testbed and we only had the access (and sudo account) to machines in Los Angeles. The sudo account made us optimize the Linux TCP/IP stack in machines of Los Angeles. Because the connection was exclusively shared by several parts, we can only perform some very simple experiments and were unable to analyze some of the problems.
From figure 4.5, the throughput is not symmetric on two directions. Possible reason could be that the bottleneck was at the sender side. The sender side CPU percentage was always 100%, and both sides used different 10Gbps NICs.

From figure 4.5, the throughput is not necessary related to the congestion window as expected. The throughput measurement tool (iperf or nuttcp) are limited. They can’t catch accurately the traffic in every single second. Although the congestion window didn’t change when transferring TCP flow from Baton Rouge to LA, the throughput did has a little fluctuation. Since the 10Gbps link was exclusively used, there is no bottleneck or queue on network, which means there was no congestion. The congestion window did fluctuate a little bit when transferring TCP flows from LA to Baton Rouge while the throughput didn’t change. Possible reasons could be that the Linux OS in LA was older than in Baton Rouge, and it might have a different implementation of TCP Reno. Since there was no congestion on the Enlightened testbed, it does not mean much to discuss congestion control algorithms.

4.4 UDP performance on Enlightened testbed

UDP is a connectionless, non reliable transport layer protocol. It is mostly used to transfer time-sensitive applications without reliability such as real time video conference. In such scenario, packets delayed longer than some time will be useless and dropped. And UDP's connectionless feature is also useful for servers that respond large amount of small queries from different clients.

UDP performance was also measured on the Enlightened testbed. The results are shown in figure 4.6 and 4.7. From those figures, the performance is totally different in two directions – from Baton Rouge to Los Angeles and from Los Angeles to Baton Rouge. One reason might be the difference between hardware, such as CPU, Memory,
and 10Gbps NICs. In our experiment, the sender side is the bottleneck. When UDP sending rate was increased, sender side will take more and more CPU percentage until 100%. Since the link between Baton Rouge and Los Angeles was a dedicated link, packet lost can only caused by endpoint or physical random loss instead of queuing loss.

According to figure 4.6, different time can get different results. But they show somehow the similar trends. When UDP sending rate was increased, more packet was lost. Before the UDP sending rate was close to 5Gbps, there are still $10^{-8}$ to $10^{-7}$ packet loss rate, which could be considered as physical random loss. But the data measured on April 28th shows no packet loss, which is very rare.

![Figure 4-6 Enlightened UDP throughput from Baton Rouge to Los Angeles](image)

From Figure 4.7, it shows something dramatically. It might be caused by some other applications on the machine in Los Angeles taking too much CPU clocks, because we had the control of the machines in Baton Rouge while cannot use machines in Los Angeles exclusively. The physical random loss rate also range from $10^{-8}$ to $10^{-7}$. When UDP sending rate was increased over its sending capability, the loss rate increased quickly.
4.5 Performance comparison of UDT and TCPs on a non optimized 10Gbps link

UDT is an application level, reliable and connection-oriented data transport protocol. It uses UDP to transfer data and protocol control information. It is specially designed for high-speed bulk data transfer.

UDT supports a large variety of congestion control algorithms. Actually all kinds of TCP congestion control algorithms can be implemented in UDT, which is just congestion control above UDP. Since congestion control can be implemented in user space, it can support run time congestion control algorithms. And each UDT flow can choose different congestion control algorithms, which is not possible for TCP.

The default UDT congestion control algorithm is a loss-based AIMD mechanism with bandwidth estimation technique used to optimize its increase parameter. And a random decreased factor is used to remove the negative effect of loss synchronization.

In all, according to [21], UDT has three major outstanding aspects.
- UDT is at the application level, which promotes a much better deployment method than in-kernel protocols including XCP and TCP variants.
- UDT comes with an efficient and fair congestion control algorithm, which makes it a better approach than the other UDP-based protocols that either have no, or very limited congestion control abilities.
- UDT itself is also a protocol framework with configurable congestion control, which makes it a research tool for the evaluation of new congestion control algorithms.

The reason why we want to compare the performance of UDT and TCP is sometimes it is really difficult or impossible to get a sudo account to optimize the Linux TCP/IP stack. For example, it is impossible to optimize TCP/IP stack in some clusters because other applications’ TCP flows will be impacted. In such scenario, we might be able to use UDT to outperform TCPs.

In this scenario, we didn’t optimize Linux TCP/IP stack except changing MTU to 9000 bytes to support jumbo frame. This is exactly the case for LONI clusters. In bluedawg, ducky and zeke, the head nodes all have a 10Gbps NIC without any optimization of TCP/IP stack. And packet loss rate was set to $10^{-7}$. We also used NistNet to emulate different delay scenarios. UDT was tuned in 16ms delay scenario and used for other delay scenarios. Without application level tuning, the performance of UDT was not so good. We used nuttcp to transfer TCP flows and the data transfer tool in UDT package to transfer UDT flows for 300 seconds. The performance comparison result is shown in figure 4.8.

![Throughput Comparison of TCP and UDT](image)

**Figure 4-8 Throughput comparison of TCP and UDT**

According to figure 4.8, UDT outperforms TCP much without Linux TCP/IP tuning. When there is no delay enforced on back-to-back connection, the throughput of UDT and
TCP are almost the same – 3.3Gbps. But after delay was enforced, the performance of TCP decreased fast. Performance of TCP variants is not shown here because they had almost the same throughput (difference is less than 1Mbps) when delay is enforced. The reason is the send and receive buffer. And TCPs without system tuning failed to utilize the bandwidth. The UDT throughput in 32ms delay is higher than 3ms and 8ms, which might be caused by application tuning of UDT. UDT parameters should be tuned according to the different delays. The big advantage of UDT is to improve bandwidth efficiency when no sudo account can be used to tune TCP/IP stack for bulk data transfer. But it does not provide automatic performance tuning mechanism.
5. Conclusion

In this report, we described the performance of TCPs, UDP and UDT in different scenarios to show the characteristics of 10Gbps optical networks.

By changing the packet loss rate enforced by NistNet, all TCPs performed very well in very low packet loss rate scenarios (less than $10^{-7}$). This is the characteristic of most 10Gbps optical networks. When packet loss rate increased, different TCPs behaved different. But it doesn’t mean the higher, the better. From RTT fairness result, different TCPs have different fairness index. And CUBIC is the best in terms of RTT fairness. Scalable is the best in terms of throughput in high packet loss rate scenarios, while it is the worst of RTT fairness. Since the two main goals of TCP are fairness and efficiency. It is hard to tell which one is the best. Some TCPs can outperform others in specific scenarios in terms of throughput and fairness.

UDP experiment was also performed to measure the characteristics of realistic 10Gbps optical networks. It also showed the normal packet loss rate of optical networks are around $10^{-7}$. And two directions of transmission behaved differently because of different end systems.

Finally, the throughput of UDT and TCPs are compared in raw Linux TCP/IP scenario. The result showed UDT can outperform TCPs when delay was enforced. Even for small delay as 3ms, the performance of TCP decreased dramatically because of the small default sending and receiving buffer size. In some cases, it will be very nice to use UDT when sudo account can’t be assigned. UDT also provides a lot more flexibility than TCP does.

In this article, we didn’t compare the performance of delay based TCP variants, such as TCP Vegas, FAST TCP. They should perform better when packet loss rate increases. The main reason is there is no implementation on Linux kernel. And FAST TCP is not accessible because of its commercialization.

From our experiments, back-to-back connection scenario seemed too simple to be widely used. Realistic optical networks as Enlightened testbed were exclusively shared by so many researchers and seemed difficult to access. From our experience, Enlightened testbed had to be reserved before we started the experiment and network administrators had to be involved to configure the control plane. Some automatic reservation tools could also be used for reservations, although there is no standard and easy tool for this. In all, realistic 10Gbps optical networks are not easily accessed for most of the users even for network researchers. So some 10Gbps optical network testbed are much needed for easy access and use. This is our current task – to build a reconfigurable 10Gbps optical
network testbed. It can support multiple users and provide different topologies according to users’ requirement. And different experiments can run independently at the same time. Such a testbed can provide more complicate scenario than back-to-back scenario and much easier access to common researchers than realistic optical network testbed. And it also provides the flexibility to scale to serve more researchers simultaneous.
References


[16] ”Iperf 1.7.0 - The TCP/UDP Bandwidth Measurement Tool” http://dast.nlanr.net/projects/Iperf/


Vita

Yixin Wu was born in January, 1979, in Danyang, People’s Republic of China. In 1997, he was admitted with exemption of college entrance examination to SooChow University, P.R.China. Yixin graduated with Bachelor of Engineering (with honors) from SooChow University in July 2001. Then he became a graduate student of Shanghai Jiaotong University, P.R.China in September 2001. In spring of 2004, he graduated with the degree of master in engineering from Shanghai Jiaotong University. He worked as an embedded software engineer in Fulhwa Microelectronics Corporation before he began his graduate studies as a doctoral student in the Department of Computer Science, Louisiana State University (LSU). In LSU, he studied and performed research in the area of network under the guidance of Dr. Seung-Jong Park. He worked on TCP congestion control algorithms for high speed 10Gbps optical networks.