Teraflows over Gigabit WANs with UDT

Robert L. Grossman[a,b], Yunhong Gu[a] and Xinwei Hong[a]
Antony Antony[c], Johan Blom[c], Freek Dijkstra[c], and Cees de Laat[c]

[a]National Center for Data Mining,
University of Illinois at Chicago, Illinois, USA

[b]Open Data Partners, Oak Park, Illinois, USA

[c] University of Amsterdam, The Netherlands

Abstract

The TCP transport protocol is currently inefficient for high speed data transfers over long distance networks with high bandwidth delay products. The challenge is to develop a protocol which is fast over networks with high bandwidth delay products, fair to other high volume data streams, and friendly to TCP-based flows. We describe here a UDP based application level transport protocol named UDT (UDP based Data Transfer) with these properties and which is designed to support distributed data intensive computing applications. UDT can utilize high bandwidth efficiently over wide area networks with high bandwidth delay products. Unlike TCP, UDT is fair to flows independently of their round trip times. In addition, UDT is friendly to concurrent TCP flows, which means it can be deployed not only on experimental research networks but also on production networks. To ensure these properties, UDT employs a novel congestion control approach that combines rate based and window based control mechanisms. In this paper, we describe the congestion control algorithms used by UDT and provide some experimental results demonstrating that UDT is fast, fair and friendly.

Keywords: UDT, Bandwidth Delay Product (BDP), Grid Networking, Transport Protocol, High Speed WANs

1 Introduction

Data-intensive distributed and Grid applications often involve transporting, integrating, analyzing, or mining one or more very high volume data flows or, what we call in this paper, teraflows. Developing teraflow applications was impractical until recently when network infrastructures supporting 1 Gbit/s and 10 Gbit/s links began emerging.

As networks with these bandwidths begin to connect computing and data resources distributed around the world, the limitations of current network protocols are becoming apparent. Currently deployed network transport protocols face several difficulties in effectively utilizing high bandwidth, especially over networks with high propagation times. These difficulties grow commensurately with the bandwidth delay product (BDP), which is the product of the bandwidth and the round trip time (RTT) of the path. In particular, it is an open problem to design network protocols for mixed traffic of teraflows and commodity traffic which are fast, fair and friendly in the following senses:

1. *Fast*: the protocol should be able to transfer data at very high speed. The throughput obtained by the protocol should only be limited by the physical characteristics of the network and some reasonable overhead of the lower level protocols. For example, a single flow of the protocol with no competing traffic should be able to use all of the available bandwidth B, even for $B = 1$ Gbit/s and $B = 10$ Gbit/s links.

2. *Fair*: the protocol has the ability to share bandwidth resources with other flows using the same protocol in the sense that m high speed flows on a high bandwidth delay product link with no competing commodity traffic will each use about B/m of the bandwidth.

3. *Friendly*: the protocol can coexist with commodity TCP flows. The protocol should not use up all the available bandwidth and prevent concurrent TCP flows from getting a fair allocation of bandwidth. More specifically, TCP flows should have approximately the same throughput in two situations: 1) only $m + n$ TCP flows exist, 2) m TCP flows and n high speed flows exist.

TCP is a protocol that becomes less than optimal as network bandwidth and delay increase, although it is still dominant on today's Internet. During its congestion avoidance phase, TCP is designed to increase its sending rate by one segment per RTT when there is no congestion event as indicated by three duplicate acknowledgements; each time there is a congestion event, the sending rate is decreased by half. This approach, called additive increase multiplicative decrease (AIMD), is very inefficient for networks with high BDPs. The recovery time for a single lost packet can be very high. It is also unfair to competing flows in the sense that flows with shorter RTTs can obtain more bandwidth. In this paper, we concentrate on the congestion avoidance behavior of TCP. TCP slow start, fast retransmit, and fast recovery are out of the scope of this paper. Detailed description of the shortcomings of TCP in high BDP networks can be found in [6] and [29].

In this paper, we show that a new protocol called UDT (for UDP-based Data Transfer) can be used to support teraflows for distributed and grid computing applications. We also show experimentally that UDT is fast, fair and friendly in the senses given above, even over networks with high BDPs. In addition, since UDT is implemented at the application level, it can be deployed today without any changes to the current network infrastructure. Of course, UDT can also be deployed more efficiently by modifying the operating system kernel.

UDT uses UDP packets to transfer data and retransmit the lost packets to guarantee reliability. Its congestion control algorithm combines rate based and window based approaches to tune the inter-packet time and the congestion window, respectively.

It is not a goal of UDT to replace TCP in the Internet. Instead, protocols like UDT can be used to supplement TCP in networks with large BDPs, where some applications require one or several teraflows and these must co-exist with commodity flows.

The rest of this paper is organized as follows. The control mechanism of UDT is described in section 2. In section 3 we introduce experimental results. Section 4 looks at related work. Section 5 contains concluding remarks.

2 Description of UDT Protocol

We begin this section with an overview of the UDT protocol. Next, we describe the reliability and congestion control mechanisms used in UDT.

2.1 Overview

UDT is derived from the Simple Available Bandwidth Utilization Library (SABUL) [11] [12] [4]. Its new congestion control algorithm satisfies fairness while achieving similar throughput to SABUL. SABUL uses a Multiplicative Increase Multiplicative Decrease (MIMD) control algorithm to tune the sending rate based on the current sending rate. UDT uses the AIMD control algorithm, but the parameters are updated dynamically by a bandwidth estimation technique. In addition, UDT removes the TCP control channel of SABUL by sending control information in the same UDP data channel.

UDT transfers data between a pair of UDP ports. All UDT data packets have the same size, which is set to the Maximum Transmission Unit (MTU) of the path. Each data packet is assigned an increasing sequence number. This sequence number is used by UDT's reliability mechanism. The first bit of the UDT header is used to distinguish a data packet from a control packet. The data transfer mechanism of UDT is presented in Figure 1 and described in more detail in the sections below. Here is a brief overview.

A UDT application consists of a sender and a receiver. The sender maintains a protocol buffer of packets to be sent and a loss list that contains the sequence numbers of lost packets. An application buffer can be linked directly to the UDT protocol buffer using overlapped input/output, without the necessity of a memory copy. The sender either retransmits a packet from the loss list if it is not empty, or sends a new packet from the protocol buffer.

The receiver maintains a protocol buffer that stores the arriving packets and updates a loss list that records the sequence numbers of lost packets. In the overlapped input/output mode, the incoming data is written directly into the application buffer.

There is only one UDP connection (a pair of UDP ports) between any two connected UDT entities. Application data is sent from the sender of one UDT application to the receiver of the other UDT application, whereas control information is exchanged directly between the two receivers. In Figure 1 the arrows represent the directions of the data and control flows.

2.2 Reliability Control

UDT uses two kinds of acknowledgments to guarantee reliability. Selective positive acknowledgments (which we denote as ACK) are sent back at constant intervals as long as new packets continue to arrive in sequence. So, an ACK usually can acknowledge the successful receiving of a block of packets. An ACK provides the sender with the largest sequence number of packets that have been received in order. A negative acknowledgement (NAK) is generated each time a packet loss is detected by the receiver. Information about lost packets may be resent if the receiver has not received the retransmission within a certain amount of time, which is increased each time

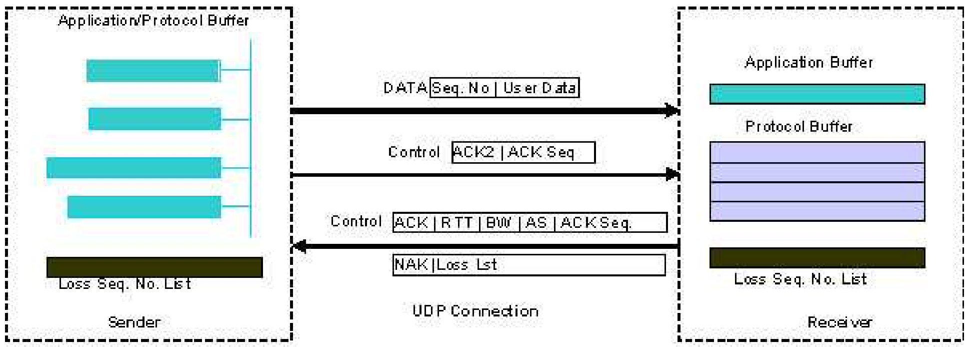

Figure 1: Data and control paths in UDT.

the same packet loss information is sent. The sender retransmits the lost packets once it has received the NAK packet.

By using selective acknowledgements, UDT has only a very small portion of control packets. More importantly, this also significantly reduces the computation overhead of the end systems, compared to TCP for example, which acknowledges at most two packets at a time.

2.3 Congestion Control

The congestion control of UDT has two independent parts: flow control and rate control.

The flow window is a threshold to limit the number of unacknowledged packets. It is dynamically set to $AS * (RTT + SYN)$. Here, SYN is a constant interval for rate control. In practice, SYN is set to 10 ms. AS is the current packet arrival speed.

The receiver records the packet arrival intervals. Once an ACK is to be sent, it calculates the median of the packet arrival intervals (M) since the last ACK, calculates the packet arrival speed (AS = 1/M), and attaches it to the current ACK packet. We use the median instead of the mean value because the packet sending may not be continuous and the packet arrival interval may be unnecessarily large due to the interruption in packet sending. The UDT receiver then computes the flow window size according to the estimated RTT and the packet arrival speed, and sends back (to the sender) the minimal value between the flow window size and the available receiver buffer size. Once the UDT sender receives this ACK, it updates its flow window size to this value.

To calculate RTT, the UDT sender will send an ACK2 packet immediately after it receives an ACK packet. The UDT receiver can then estimate the RTT value according to the timestamps when the ACK leaves and when the ACK2 arrives. UDT uses ACK sub-sequencing (i.e., each ACK is assigned a unique ACK sequence number) to match the {ACK,ACK2} pair accurately. We used ACK2 because both sides of UDT need to know the RTT value.

Flow control is used to reduce the number of packets sent during congestion, helping to clear the congestion. At the same time, UDT also uses rate control to limit the packet sending speed. Rate control updates the inter-packet time according to a modified AIMD principle.

Once a NAK is received, the inter packet time will be increased by 1/8, which is equivalent to decreasing the sending rate by 1/9 (=1−1/(1+1/8)) This is the MD part of the AIMD algorithm. The increase parameter is proportional to the end-to-end link capacity, estimated by a packet pair using UDT's own data traffic.

The rate increase is triggered every 10 ms if no NAK was received during the last 10 ms. This constant rate control interval is to eliminate the fairness bias caused when different flows have different RTTs. First, an increase parameter of the number of additional packets to be sent during the next 10 ms is calculated. Based on this information, the inter-packet time is calculated. This is the AI part of the AIMD.

If the estimated end-to-end capacity is B (in number of packets per second), the current sending rate is C (in number of packets per second), and the fixed UDT packet size is $\mathcal{S}$ (including all packet headers, in bytes), then the number of packets to be increased is:

$$\mathcal{N}_{\text{inc}} = \beta \max\left(10^{\left(\lceil \log_{10}(\mathcal{S}\,(B-C)*8)\rceil - 9\right)}, \frac{1}{1500}\right)$$

where $\beta = 1500/\mathcal{S}$. Note that in practice, the value of C can be equal to or greater than the value of B; in this case, the value of $\mathcal{N}_{\text{inc}}$ is $1/\mathcal{S}$.

The optimal UDT packet size is MTU. In this paper, we consider 1500 bytes as the typical value of MTU. When the packet size is 1500 bytes, we hope to increase the number of packets to send per SYN according to predefined values as shown in Table 2.3. But if the packet size is not 1500 bytes, we need to correct the increment by $1500/\mathcal{S}$, which is the functionality of β.

To estimate the end-to-end bandwidth, UDT sends a packet pair back to back every 16 data packets by omitting the inter-packet waiting time

Available Bandwidth ($B - C$)	$\mathcal{N}_{inc}$
100 - 1000 Mbit/s	1 packet (= 1500 bytes)
10 - 100 Mbit/s	0.1 packets (= 150 bytes)
1 - 10 Mbit/s	0.01 packets (= 15 bytes)

Table 1: Available bandwidth vs. rate increment

between every 16th packet and its successor. The receiver then calculates the estimated bandwidth and sends it back through an ACK packet.

With this approach, flows with higher sending rates do not get larger increase parameters and all flows sharing a common bottleneck converge to an appropriate fairness equilibrium. This fairness is independent of RTT due to the constant rate control interval.

3 Experimental Studies

In this section we describe the testbeds and the experimental studies we performed.

3.1 Testbeds

The tests for this paper used two testbeds. The first is the DataTAG Testbed [5]. For the tests described here we used six servers at StarLight in Chicago and six servers at CERN in Geneva connected by a 10 Gbit/s transatlantic optical path. All servers have 2.4 GHz Intel dual Xeon processors and optical Gigabit Ethernet cards.

The second testbed is called the Tera Wide Data Mining (TWDM) Grid Testbed [25]. The TWDM Grid testbed is an application testbed operated by the National Center for Data Mining at the University of Illinois at Chicago and built on top of three high speed networks: NetherLight [19], constructed and operated by SURFnet [24], OMNInet [20], and Abilene [1]. The three networks are interconnected at StarLight [23] in Chicago.

Several of the experiments below were conducted during the SC 03 Conference [22]. During SC 03 in Phoenix, AZ, our research booth was connected through the conference SCInet network to StarLight via four 1 Gbit/s links. The testbed used at SC 03 is described in Figure 2.

NetherLight is located in Amsterdam, The Netherlands. It has three 10 Gbit/s connections to North America. Two of them are used in the testbed.

One of the 10 Gbit/s connections is connected to Abilene in New York, and from there all regular traffic is routed to StarLight through the Abilene 10 Gbit/s backbone. The other 10 Gbit/s connection was used to set up an optical path directly between NetherLight and StarLight. Both StarLight and OMNInet are located in Chicago. Our lab, the National Center for Data Mining (NCDM) at UIC, is connected to StarLight via a 4 Gbit/s path over OMNInet.

For the tests described below, clusters of workstations in Amsterdam, StarLight, NCDM/UIC, and Phoenix were used. The Amsterdam cluster consisted of ten Dell PowerEdge 2650 with 350 GB (GBytes) of disk space each. This cluster is attached to NetherLight. The StarLight cluster consisted of seven similarly configured nodes. The NCDM/UIC cluster consisted of five similar servers. During SC 03, our research booth contained an 8 node cluster, each node having 100 GB to 250 GB of disk space and dual Xeon processors.

The nodes in the cluster ran either Redhat Linux 7.3 (kernel 2.4.18-4smp) or Debian Linux. Each node had either copper or fiber Gigabit Ethernet Network Interface Cards (NICs) installed. Either hardware RAID or software RAID was configured on all of the servers used. In total, the testbed had about 30 servers with 6 TB (Terabytes) of disk space.

At the same time, two Itanium systems were used, one attached to StarLight and the other attached to NetherLight. Both of them had an Intel Pro/10GbE LR network interface card installed. A dedicated 10 Gbit/s optical path was used to connect these two nodes. Redhat Linux 7.3 was installed on these machines.

Compared to the DataTAG Testbed, the TWDM Grid Testbed had a longer RTT (190 ms vs. 116 ms) and a higher incidence of dropped packets.

3.2 Fairness and Friendliness Experiments on the TWDM Grid Testbed

During SC 03, we did two fundamental experimental studies. One was designed to demonstrate the fairness and friendliness of UDT. The other was designed to demonstrate UDT's efficient use of available bandwidth over networks with high BDPs, i.e., to demonstrate that UDT is fast.

In this section, we describe experiments using the TWDM Grid Testbed to measure the fairness and friendliness of UDT. For these experiments, the traffic goes through a 10 Gbit/s connection to Abilene in New York. The traffic then passes through Abilene's 10 Gbit/s backbone and arrives at the show floor of SC 03. The RTT between these two sites is about 190 ms.

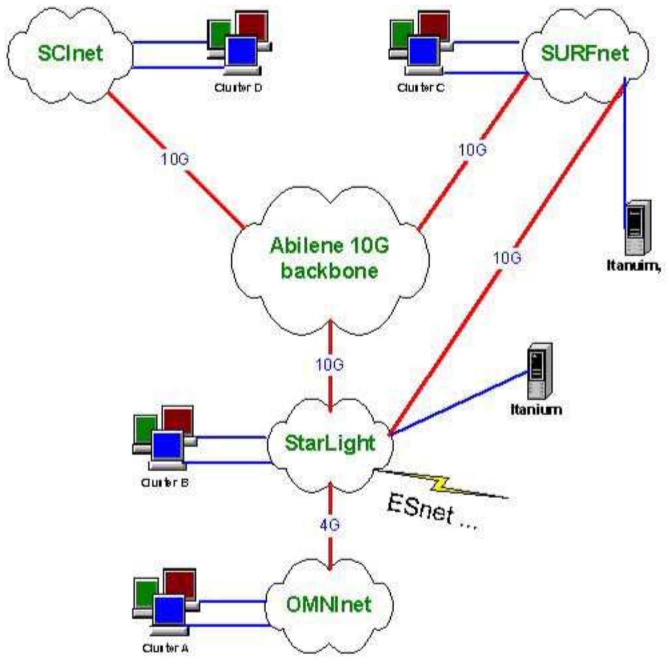

Figure 2: Network architecture of the TWDM Grid Testbed

For this test, we connected four nodes in the NetherLight cluster with four nodes in our research booth in Phoenix. Each node was limited to 1 Gbit/s by its 1 Gigabit NIC.

One node in Amsterdam sent 1 teraflow to a corresponding node in Phoenix and 50 TCP flows. The second node in Amsterdam sent two teraflows to the corresponding node in Phoenix and 50 TCP flows. The third node in Amsterdam sent three teraflows to the corresponding node in Phoenix and 50 TCP flows, while the fourth node in Amsterdam sent four teraflows to the corresponding node in Phoenix and 50 TCP flows. So in total, there were 10 teraflows and 200 TCP flows between the two four-node clusters.

Our goal was to demonstrate that UDT was fast, fair and friendly, as defined in Section 1. This experiment is summarized in Tables 2 and 3. In these tables, the throughput of each flow is first computed by averaging the throughput of the flow during the tests, with a sampling time of 1 second. Next, the mean μ, standard deviation σ, and ratio σ/μ are computed across the 50 TCP flows and then, separately, across the 1-4 UDT flows. For the TCP flows, we also report the minimum and maximum throughput.

UDT flows (Mbit/s)						50 TCP flows (Mbit/s)			
UDT_1	UDT_2	UDT_3	UDT_4	μ	σ/μ	μ	Min.	Max.	σ/μ
667				667	0	4.23	3.40	8.75	18%
311	320			315.5	2.0%	4.19	3.31	7.96	15%
209	218	239		222	6.9%	4.05	3.17	7.85	22%
172	156	180	169	169.25	5.9%	3.81	2.92	6.44	14%

Table 2: Throughput statistics for fairness and friendliness tests on the link between NetherLight and SC 03. The mean μ and the standard deviation σ is measured across flows. This test was run for 30 minutes.

UDT			TCP			Overall
# Flows	Average	Aggregate	# Flows	Average	Aggregate	Throughput
1	667	667	50	4.24	212	878
2	315.5	631	50	4.20	210	841
3	222	666	50	4.06	203	869
4	169.3	677	50	3.82	191	868

Table 3: Summary of average and aggregated throughput of TCP and UDT for the fairness and friendliness tests on the link between NetherLight and SC 03. (unit: Mbit/s)

The following conclusions can be drawn from these two tables.

1. First, UDT is fair in the sense that σ/μ is relatively low, indicating that the available bandwith left for the teraflows is shared relatively equally between the teraflows. For example, in the case of 4 UDT flows, σ/μ is only 5.0%. In fact, in this experiment, the ratio σ/μ is slightly lower for the teraflows than for the commodity TCP flows.

2. Second, UDT is friendly in the sense that the aggregate bandwidth used by 50 TCP flows is relatively constant, varying between 191 and 212 Mbit/s, as the number of UDT teraflows varies. Similarly, the average TCP throughput for the 50 TCP flows is relatively constant, varying between 3.91 and 4.24, as the number of UDT teraflows varies.

3. Third, we note that UDT makes relatively efficient use of the available bandwidth in the sense that the aggregate throughput varies between 841 and 878 Mbit/s.

3.3 Fairness and Friendliness Experiments on the DataTAG Testbed

In this section, we summarize fairness and friendliness experiments conducted on the DataTAG Testbed.

We performed a number of experiments using the link between CERN and StarLight. The results are presented in Tables 4 and 5. The average round trip time between CERN and StarLight was 116 ms, while the round trip time between NetherLight and SC 03 was 190 ms.

The conclusions of these tests are the following.

- The throughputs were computed as described above: first, by averaging over samples obtained at 1 second intervals; second by computing the average and standard deviations across the UDT flows, and separately, across the 50 TCP flows.

- Due to the shorter RTT, the mean TCP throughput is higher on the DataTAG testbed than on the SC 03 TWDM Grid Testbed. For example, for four UDT teraflows, the average TCP throughput on the DataTAG testbed is 6.04 Mbit/s, whereas on the SC 03 TWDM testbed it is only 3.81 Mbit/s.

- In contrast, the UDT throughput is essentially the same on the two testbeds, demonstrating that UDT's throughput is independent of RTT. For example, for four UDT teraflows, the average throughput is 169 Mbit/s on the SC 03 TWDM testbed and 160 Mbit/s on the DataTAG testbed.

- Overall, the ratios σ/μ are smaller on the DataTAG testbed than on the TWDM testbed. This is probably due to the fact that more packets were dropped on the latter testbed.

- The ratio σ/μ for the UDT flows is relatively small, demonstrating that UDT is fair in the sense defined above. In fact, these ratios are smaller for the DataTAG tests than for the SC 03 TWDM tests. For the DataTAG tests, the ratio σ/μ is modestly smaller for TCP flows compared to the UDT flows. This was not the case for the SC 03 TWDM tests.

- As before, the aggregate TCP bandwidth is relatively constant, as the number of UDT flows varies from 1 to 4. In fact the aggregate bandwidth only varies beteen 302 Mbit/s and 317 Mbit/s. This demonstrates that UDT is friendly in the sense defined above.

- As before, the overall throughput is relatively high, indicating that UDT can efficiently use the available bandwidth. Indeed, the bandwidth utilization is modestly higher on the DataTAG testbed compared to the SC 03 TWDM testbed (945 Mbit/s compared to 878 Mbit/s for a single UDT flow, for example).

UDT flows (Mbit/s)						50 TCP flows (Mbit/s)			
UDT_1	UDT_2	UDT_3	UDT_4	μ	σ/μ	μ	Min.	Max.	σ/μ
628				628	0	6.34	6.22	6.32	0.3%
318	311			314.5	1.6%	6.18	5.64	6.29	0.6%
218	206	202		208.7	4.0%	6.10	6.96	6.23	0.8%
162	157	159	160	159.5	1.3%	6.04	5.94	6.14	0.7%

Table 4: Throughput statistics for fairness and friendliness tests using the DataTAG testbed. The mean μ and standard deviation σ were computed over the UDT flows, and, separately, over the 50 TCP flows. This test was run for 1000 seconds, i.e., 16.7 minutes.

UDT			TCP			Overall
# Flows	Average	Aggregate	# Flows	Average	Aggregate	Throughput
1	628	628	50	6.34	317	945
2	314.5	629	50	6.18	309	938
3	208.7	626	50	6.10	305	931
4	159.5	638	50	6.04	302	940

Table 5: Summary of average and aggregated throughput of TCP and UDT flows for the fairness and friendliness tests using the DataTAG testbed. (unit: Mbit/s)

3.4 Bandwidth Utilization Experiments using the TWDM Grid Testbed

For the next experiment, we transported Sloan Digital Sky Survey data [9] across the Atlantic from Amsterdam to Chicago. For this experiment, we sent data from NetherLight to StarLight via two different paths simultaneously: one is the 10 Gbit/s routed Abilene link and the other is the 10 Gbit/s dedicated optical path between NetherLight and StarLight. Up to 10 UDT flows ran over the routed link. Since every server has a bottleneck of 1 Gbit/s throughput, we expected to get 10 Gbit/s on this path. At the same time, we ran several flows over the 10 Gbit/s optical path. We hoped to get up to 10 Gbit/s throughput on this path too.

With this setup, the maximum bandwidth is 20 Gbit/s. To test the bandwidth available in practice, we used Iperf [26] in UDP mode and obtained a bandwidth of about 12.5 Gbit/s. The difference between the 12.5 Gbit/s measured by Iperf and the theoretical 20 Gbit/s upper limit can be explained by limitations in the network infrastructure and is discussed in more detail below. It is important to recall that Iperf uses a simple transport mechanism and does not guarantee packet delivery.

The 10 Gbit/s NIC we used appears to place limitations on the maximum overall throughput possible. The tests used an Intel 10 Gbit/s NIC installed on a PCI-X (64 bit, 133 MHz) bus. The bus speed of the PCI-X is about 8.5 Gbit/s. With the communication overhead, the card can only provide about 5 to 6 Gbit/s. The new generation PCI-X 2.0 may alleviate this problem in the future.

When two UDT teraflows are sent over the 10 Gbit/s optical path, each gets about 2.5 Gbit/s throughput, so the aggregated throughput is about 5.0 Gbit/s. A single UDT teraflow running on the 10 Gbit/s optical path gets about 4.5 Gbit/s throughput.

On the routed 10 Gbit/s path, we ran 8 UDT flows and got about 6.6 Gbit/s throughput. The main bottleneck resides in the switch. The backplane of the switch (CISCO 6059) has a limitation of 8 Gbit/s. The overhead of the communication on the bus and the overhead of the UDT/UDP/IP stack also makes the throughput a little bit lower than expected.

Table 6 includes statistics on the throughput of these UDT flows, including mean, standard deviation and maximum. The results in the table are based on a sampling rate of 1 second. From the table, we can see that the standard deviation of the two flows on Itanium systems is very low, less than 10. This also indicates that the performance between the Itanium systems is very stable. We observed that competition exists on the routed link,

	Mean (Mbit/s)	σ (Mbit/s)	Maximum (Mbit/s)
Flows on 10 Gbit/s optical paths			
Flow 1	2485	7.38	2726
Flow 2	2485	2.40	2492
Routed Flows			
Flow 3	797	137	953
Flow 4	449	80	713
Flow 5	761	145	949
Flow 6	759	142	920
Flow 7	794	130	953
Flow 8	505	189	925
Flow 9	766	144	926
Flow 10	582	196	938
Aggregated Throughput for 10 flows			
Overall	10386	565	11647

Table 6: Throughput statistics on the 10 Gbit/s optical paths and routed links between NetherLight and StarLight using the TWDM Grid Testbed. The mean μ and the standard deviation σ are computed for each flow over time. This test ran for 20 minutes.

which explains why the standard deviations of the 8 routed flows are much higher. We noticed that, on the 10 Gbit/s optical path, the bottleneck is on the end systems. However, the bottleneck on the routed links resides in the switches.

The results are also summarized in Figure 3. The figure shows the throughput of UDT flows in a demonstration that ran for 20 minutes, with a sampling rate of 10 seconds. In the figure, the upper line represents the overall throughput during the demonstration. The individual throughputs are represented by the lower lines. Two streams on the Itanium system have very smooth values around 2.5 Gbit/s and they share the bandwidth equally. In the graph, these two lines are overlapped. The throughputs of these 8 UDT flows are presented on the bottom of the figure. Eight UDT flows compete for the 6.8 Gbit/s available bandwidth. This is why the throughputs of these 8 UDT flows fluctuate greatly.

In summary, we got up to 11.6 Gbit/s throughput transporting astronomical data across the Atlantic from Amsterdam to Chicago. Compared

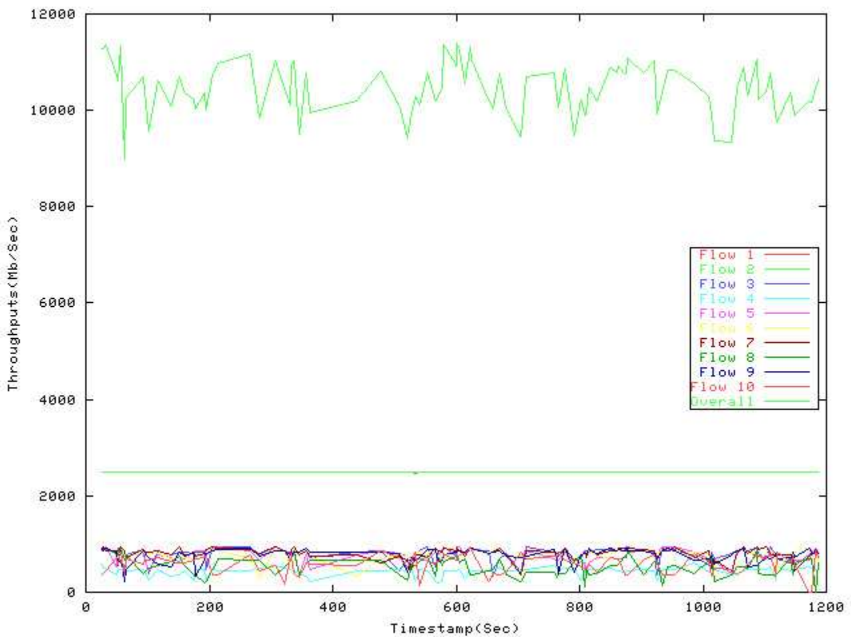

Figure 3: This graph details the throughput of individual UDT teraflows, as well as the aggregated throughput (green line) between Amsterdam (NetherLight) and Chicago (StarLight).

to the 12.5 Gbit/s bandwidth obtained by Iperf, this represents an effective utilization of about 94% of the available bandwidth.

3.5 Bandwidth Utilization Experiments using the DataTAG Testbed

In another experiment, we transferred the same astronomical data through the DataTAG testbed between Geneva (CERN) and Chicago (StarLight). The results are summarized in Table 7. From these results, we can see that UDT can get up to 5.67 Gbit/s. Considering that the maximum throughput between 6 pairs of machines is 6 Gbit/s, UDT achieves 94.5% utilization.

The results are also shown in Figure 4. These tests were also run during SC 03. The figure shows results for more than 30 minutes. The top line represents aggregated throughput. The lower six lines represent individual UDT flows.

	Mean μ (Mbit/s)	σ (Mbit/s)	Maximum (Mbit/s)
Flow 1	881	160	958
Flow 2	785	209	958
Flow 3	930	122	958
Flow 4	810	183	965
Flow 5	803	174	972
Flow 6	768	186	891
Overall	4977	175	5666

Table 7: Throughput statistics on DataTAG testbed between CERN and StarLight. The mean μ and standard deviation σ are computed over time for each flow. This test run lasted 32 minutes.

4 Related Work

Currently, there are four approaches being investigated for high performance data transport: employing parallel TCP connections, modifying standard TCP, creating new protocols based upon UDP, and developing entirely new network transport layer protocols [15].

Using parallel TCP connections to achieve higher performance is intuitive [10] and also widely available because the current version of GridFTP employs this approach [2]. However it poses several problems. For example, careful tuning is required to determine both the window size and the number of flows required. In addition, we still do not fully understand how to dynamically adapt the number of concurrent flows as the network changes,

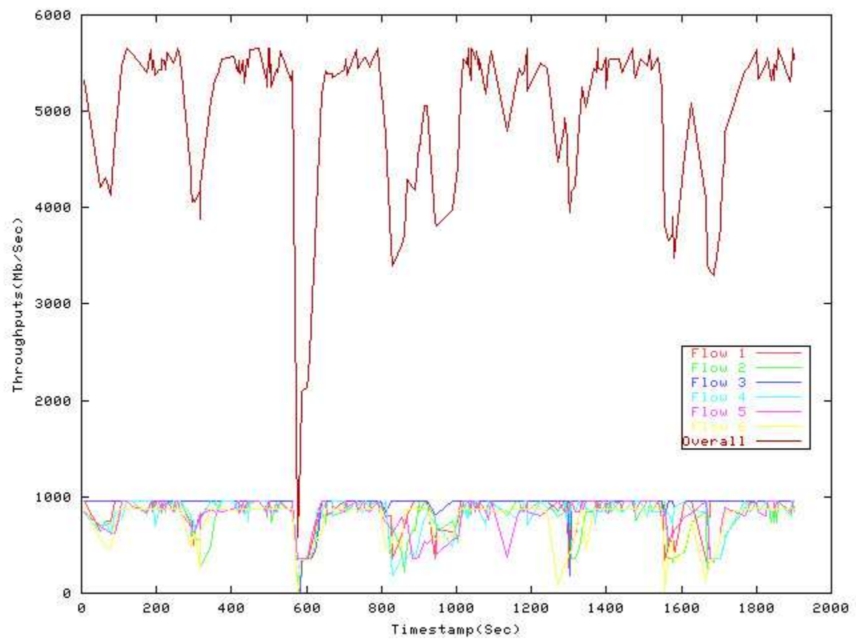

Figure 4: Throughput of UDT teraflows

and how to overcome the RTT bias problem of current implementations of TCP [13]. Another problem is that large oscillations can arise when multiple TCP flows coexist, causing imbalance among the parallel connections.

A second approach is to modify TCP. Many authors proposed to modify the additive increase multiplicative decrease (AIMD) congestion control algorithm in standard TCP. There is a great deal of work in this area, resulting in a series of TCP variants.

Scalable TCP [17] and HighSpeed TCP [7] use more aggressive congestion control algorithms to replace the standard AIMD algorithm. As the congestion window becomes larger, they both increase faster. HighSpeed TCP also decreases slower, whereas Scalable TCP has a constant decrease factor of 1/8.

When the window size is below a threshold window size, Scalable TCP and HighSpeed TCP handle congestion using the standard TCP mechanism. In this way, they are friendly to standard TCP flows for traditional low BDP networks, whereas they are more aggressive over high BDP networks Beyond a threshold, Scalable TCP adopts a multiplicative increase, whereas HighSpeed TCP calculates the increase parameter according to the current congestion window size.

The MIMD algorithm used in Scalable TCP is not fair according to [21]. The control algorithm used in HighSpeed TCP, although less aggressive than that of Scalable TCP, can also lead to unfairness since flows with higher congestion windows have a greater increase parameter [7].

FAST TCP [28] extends TCP Vegas [3] by defining a new function to calculate the congestion window using the packet delay information, rather than increasing or decreasing by predefined values. Using packet delay as an indication of congestion can predict the congestion earlier so that packet loss may be avoided, or at least reduced. However, packet delay may not actually indicate congestion [18], and it can be severely affected by the traffic on the reverse route.

A third approach is to employ entirely new network transport protocols, such as open loop control. XCP [16] generalizes Explicit Congestion Notification (ECN) by using precise congestion signaling, where the routers expressly tell the sender the state of congestion and how to react to it. In XCP, routers monitor the input traffic rates of each of their output queues and tell each of the flows sharing that link to increase or decrease their congestion windows, by annotating the congestion headers of data packets; the latter are then returned to the sender in an acknowledgment packet from the receiver. XCP has a MIMD efficiency controller and an AIMD fairness controller.

Improvements to TCP (Approach 2) require changes to the network stack in operating systems. New network protocols (Approach 3) require, in addition, changes to the network infrastructure itself, such as new code in routers. For this reason, significant time and investments are required for standardizing and deploying new protocols such as the ones described above.

In contrast, the fourth approach, in which new algorithms are deployed at the application layer, can have an impact in the short term. It is important to note that new flow control and congestion control algorithms that are developed for application layer deployment can always be implemented in the network stack in the long term, after a standard approach has emerged.

There have been several independent implementations of application layer protocols (Approach 4), including SABUL [11], Reliable Blast UDP (RBUDP) [14], Tsunami [27] and Fast Object Based Data Transfer System (FOBS) [8]. The basic idea is to use a UDP based data channel and a separate control channel to provide reliability and/or congestion control.

RBUDP, Tsunami, and FOBS use simple rate based UDP with reliability control. These three protocols employ a simple rate control mechanism based upon packet loss. They use a constant sending rate, which is determined by the application. This enables an efficient use of the bandwidth but requires that the appropriate constant sending rate be determined externally. Fairness and friendliness issues are not addressed since these three protocols are designed to be used in private or QoS enabled networks.

SABUL is a more complex protocol that also uses UDP but has a fully functional congestion control algorithm to address the efficiency, fairness, and friendliness issues. UDT extends SABUL by introducing bandwidth estimation and flow control into SABUL's congestion control algorithm. In addition, UDT employs a UDP-based control channel while SABUL uses a TCP-based control channel. Both SABUL and UDT employ a UDP-based data channel.

5 Conclusion

In this paper, we have described a new application level protocol called UDT. UDT is built over UDP. UDT employs a new congestion control algorithm that is designed to achieve intra-protocol fairness in the presence of multiple high volume flows. UDT is designed to be fast, fair and friendly as defined in Section 1.

We showed in our experimental studies that UDT can effectively utilize the high bandwidth of networks with high BDPs in situations where currently deployed versions of TCP are not effective. We also demonstrated that UDT is fair to other UDT teraflows. Finally, we proved that UDT is still friendly to concurrent TCP commodity flows.

As UDT is built over UDP and is implemented at the application layer, it can be deployed at lower cost because it does not require any modifications to the network infrastructure or changes to any operating systems.

In the future, we will continue our study of UDT's rate control and flow control mechanisms. We will also model UDT and continue to study it using simulations. In particular, we plan to see if simpler versions, which are easier to analyze and model, can provide broadly similar performance. We will also examine alternate methods for estimating available bandwidth. We are currently developing a kernel-level version of UDT in order to assess the performance improvement this provides.

Acknowledgements

This work was supported by NSF grants ANI-9977868, ANI-0129609 and ANI-0225642.

References

[1] Abilene network. http://abilene.internet2.edu.

[2] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke. Data management and transfer in high performance computational grid environments. *Parallel Computing Journal*, 28:749–771, 2002.

[3] Lawrence S. Brakmo and Larry L. Peterson. TCP vegas: End to end congestion avoidance on a global internet. *IEEE Journal on Selected Areas in Communications*, 13(8):1465–1480, 1995.

[4] A. Chien, T. Faber, A. Falk, J. Bannister, R. Grossman, and J. Leigh. Transport protocols for high performance: Whither TCP? *Communications of the ACM*, 46(11):42–49, 2003.

[5] DataTAG, http://datatag.web.cern.ch/datatag/.

[6] Wu chun Feng and Peerapol Tinnakornsrisuphap. The failure of TCP in high-performance computational grids. In *Proceedings of Supercomputing*, 2000.

[7] Sally Floyd. Highspeed TCP for large congestion windows. *IETF*, RFC 3649, December, 2003.

[8] FOBS. http://omega.cs.iit.edu/ ondrej/research/fobs.

[9] Jim Gray and Alexander S. Szalay. The world-wide telescope. *Science*, 293:2037–2040, 2001.

[10] R. L. Grossman, S. Bailey, A. Ramu, B. Malhi, H. Sivakumar, and A. Turinsky. Papyrus: A system for data mining over local and wide area clusters and super-clusters. In *Proceedings of Supercomputing*. IEEE, 1999.

[11] R. L. Grossman, M. Mazzucco, H. Sivakumar, Y. Pan, and Q. Zhang. SABUL - simple available bandwidth utilization library for high-speed wide area networks. *Journal of Supercomputing*, to appear.

[12] Robert L. Grossman, Yunhong Gu, Dave Hanley, Xinwei Hong, Dave Lillethun, Jorge Levera, Joe Mambretti, Marco Mazzucco, and Jeremy Weinberger. Experimental studes using photonic data services at igrid 2002. *Future Generation Computer Systems*, 19(6):945–955, 2003.

[13] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *Proceedings of the 16th IEEE-CS/ACM International Parallel and Distributed Processing Symposium (IPDPS)*, 2002.

[14] E. He, J. Leigh, O. Yu, and T. DeFanti. Reliable blast UDP: Predictable high performance bulk data transfer. In *IEEE Cluster Computing*, 2002.

[15] M. Goutelle, Y. Gu, E. He, et al. A Survey of Transport Protocols other than Standard TCP. Global Grid Forum, Data Transport Research Group, work in progress, April 2004. https://forge.gridforum.org/forum/forum.php?forum_id=410

[16] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the ACM Sigcomm*, 2002.

[17] Tom Kelly. Scalable TCP: improving performance in highspeed wide area networks. *ACM SIGCOMM Computer Communication Review*, 33:83–91, 2003.

[18] J. Martin, A. Nilsson, and I. Rhee. Delay-based congestion avoidance for TCP. *IEEE/ACM Transactions on networking*, Vol. 11, No.3, pages 356–369, 2003.

[19] Netherlight: Proving ground for optical networking, constructed and operated by SURFnet. http://www.netherlight.net.

[20] Optical Metro Network Initiative. http://www.icair.org/omninet.

[21] R.N.Shorten, D.J.Leith, J.Foy, and R.Kilduff. Analysis and design of congestion control in synchronized communication networks. In *Proceedings of the 12th Yale Workshop on Adaptive and Learning Systems*, May 2003.

[22] SC 03 Conference. http://www.supercomp.org.

[23] StarLight. http://www.startap.net/starlight.

[24] SURfnet: High-quality internet for education and research. http://www.surfnet.nl/en/.

[25] TeraFlow Testbed. http://www.ncdm.uic.edu.

[26] Ajay Tirumala, Feng Qin, Jon Dugan, Jim Ferguson, and Kevin Gibbs. IPERF version 1.7.0. http://dast.nlanr.net/Projects/Iperf.

[27] Tsunami. http://www.anml.iu.edu/anmlresearch.html.

[28] D. X. Wei C. Jin and S. H. Low. Fast TCP: motivation, architecture,algorithms, performance. In *Proceedings of IEEE Infocom*, pages 81–94, 2004.

[29] Y. Zhang, E. Yan, and S. Dao. A measurement of TCP over long-delay network. In *Proceedings of the 6th International Conference on Telecommunication Systems, Modeling and Analysis*, pages 498–504, 1998.