Abstract for Second International Workshop on Protocols for Fast Long-Distance Networks

UDT: An Application Level Transport Protocol for Grid Computing

Yunhong Gu and Robert L. Grossman
Laboratory for Advanced Computing / National Center for Data Mining
University of Illinois at Chicago
700 SEO, M/C 249, 851 S Morgan St, Chicago, IL 60607, USA

1. Introduction
As the network bandwidth and delay increase, TCP becomes inefficient [1, 5, 8, 9], due to the problems of slow recovery from loss and RTT bias inherit in its AIMD congestion control algorithm, as well as the bursting data flow caused by its window control. Data intensive applications over high BDP networks such as the computational grids need new transport protocols to support them.

To meet this timely requirement, we have developed an application level protocol built above UDP, which is named UDP based Data Transfer protocol (UDT). UDT has its own congestion control mechanism to reach the efficiency, fairness and stability objectives, whereas its application level nature enables it to be deployed with the lowest cost, without any changes in the network infrastructure and operating systems.

Related works include TCP enhancements such as FAST TCP, HighSpeed TCP and Scalable TCP, UDP based bulk data transport protocols like tsunami and RBUDP, and open-looped approach of XCP [10].

2. The UDT Protocol

UDT adds reliability and congestion controls onto UDP. It uses packet based sequencing, i.e., the sequence number is increased by 1 for each packet, which is in MTU size including all headers. Selective positive acknowledgement (ACK) is sent every constant interval, whereas negative acknowledgement (NAK) is generated as soon as packet loss is detected. The congestion control combines rate based and window based control mechanisms to ensure efficiency and fairness, including TCP friendliness and delay independence.

Rate control tunes the inter-packet time every constant interval, which is called SYN. The value of SYN is 0.01 seconds, which is an empirical value according to a tradeoff among efficiency, fairness and stability. Every SYN time, if the packet loss rate during the last SYN time is less than a threshold (maximum possible link BER), the number of packets to be sent in next SYN time is increased by:

\[ inc = \max(10^{\frac{\log_{10}(B-C) \times MTU/8}{8} \times \beta}, 1/MTU) \]

where \( B \) is the estimated bandwidth and \( C \) is the current sending rate, both in number of packets per second. \( \beta \) is a constant value of 0.0000015. MTU is the maximum transmission unit, or the UDT packet size, in bytes. The inter-packet time is then recalculated according to the total numbers of expected sent packet in next SYN time. The estimated bandwidth \( B \) is probed by sampled UDT data packet pairs [2, 3].

The inter-packet time is increased by 1/8 (or equivalently, the sending rate is decreased by 1/9) as soon as the sender receives an NAK packet whose largest lost sequence number is greater than the largest sent sequence number when last decrease occurs or the number of NAKs since last rate decrease has exceeded an increasing threshold. No data is sent out during next SYN time after a rate decrease.

UDT also uses a flow control window to limit the number of unacknowledged packets. Once it is time to feed back ACK, the UDT receiver calculates the packet arrival rate (AS) using a median filter on the recent packet arrival intervals it recorded and attaches AS within the ACK packet. At the sender side, when an ACK is received and the AS value is greater than 0, the flow window size \( W \) is updated as:

\[ W = W \times 0.875 + (RTT + SYN) \times AS \times 0.125 \]

Rate control is used to reach fast bandwidth discovery, fast loss recovery, and intra-protocol fairness, whereas flow control helps to reduce packet loss and oscillation and to avoid congestion collapse [7]. Meanwhile, both of them contribute to TCP friendliness.

During congestion, loss reports from the receiver may also be dropped or delayed, so the sender may keep sending new packets and worsen the congestion. Flow control prevents this from happening. By reducing packet loss, flow control also helps to improve efficiency and fairness, since low loss rate reduces the frequency of sending rate decreases and makes UDT less aggressive.
Rate control decides the throughput when there are only small number of concurrent sources. As the RTT and the number of concurrent flows increase, flow control begins to limit the throughput. In such situation, flow control helps to reduce the oscillations that would be higher due to the high rate increase per RTT.

The flow window size starts from 2 as slow start, and is updated to the number of acknowledged packets when the sender receives an ACK. Slow start ends when the sender receives an NAK or reaches the maximum window size, after which the congestion control algorithm described above starts to work. The inter-packet time is 0 during slow start phase and is set to the packet arrival interval when slow start ends.

3. Simulation and Implementation

We have developed a UDT NS-2 simulation module and an open source C++ library working on several platforms, which are both available online [4]. Simulations and experiments on real networks have been done to examine the efficiency, fairness, and stability features. Meanwhile, the UDT library has been used in several practical data intensive applications in grid computing environments.

The impact of bandwidth and RTT are little on UDT since the increase parameter is related to end-to-end link capacity and the rate control interval is independent to RTT. UDT can utilize available bandwidth efficiently even with rapid changes of background. This fast adaptation is especially useful for applications involving much disk IO, whose speed is oscillating due to file locating and disk scheduling. UDT can reach about 950Mbps on 1Gbps link over 110ms path from Chicago to Amsterdam. During IGrid 2002 3 parallel UDT connections on the same link reach 2.8Gbps [6].

For concurrent UDT flows sharing a single bottleneck, max-min fairness can be reach in short time because slower UDT flows has higher increase parameter according to the rate control algorithm and they have same decrease factor. This fairness is still observed even if the RTT varies for different flows. In multiple bottleneck topologies, the flow on the narrowest link is guaranteed to obtain at least half of its fair share.

When competing with bulk TCP (Reno) flows, TCP can obtain more bandwidth than UDT in moderate network BDP scenarios, whereas its bandwidth share decreases as the BDP increases. However, since TCP cannot utilize the bandwidth efficiently in high BDP links, it is reasonable that UDT acquires more bandwidth. On the other hand, UDT has little impact on web like short life TCP flows in all scenarios.

In summary, we expect UDT to be used in the environments where small number of sources share the high bandwidth. (We have tested 400 concurrent bulk flows in 1Gbps bandwidth / 1-second RTT link.) Traditional well performed TCP flows such as web traffic will still be safe in the existence of UDT. Meanwhile, we have not tested the situation of concurrent bulk or short life UDT flows in the large number similar to that of the TCP flows on the Internet, which is not an objective use scenario of UDT.

4. Reference