UDT: UDP based Data Transfer Protocol

http://udt.sourceforge.net

Breaking the Data Transfer Bottleneck

National Center for Data Mining
University of Illinois at Chicago

The Problems

- Bulk data transfer in data intensive applications
- Cooperation between large number of data flows
- High computation overhead, large data copy, bursting disk-network IO
- TCP is ineffective

The Challenges

- Easy to deploy: user space and end-to-end approach without router feedback
- High performance, fast data transfer
- Intra-protocol fairness without RTT bias
- TCP friendliness

The Solutions

- UDP-based, application level protocol
- Protocol design to support efficient packet processing
- Configurable congestion control
- Efficient native congestion control algorithm
- Optimized implementation

The Features

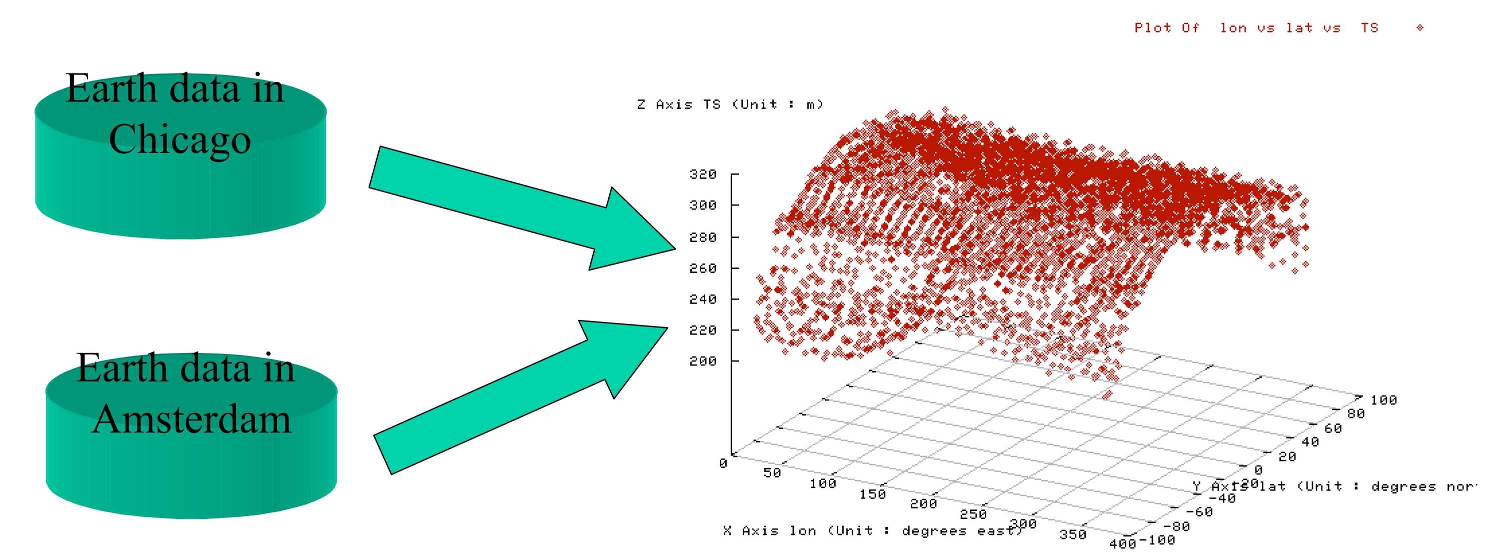
- Fast.
- Fair and friendly.
- Easy to use.
- Highly configurable.
- Firewall friendly.

The Results

- SC03: 10 UDT flows and 200 TCP flows from Amsterdam to Phoenix – fair and friendly
- SC06: transfer SDSS data disk-disk between Chicago and Tampa at 8Gb/s using Sector & UDT. BWC winner.
- SC08: Large area cloud computing with Sector/Sphere. UDT supports 120*120 flows in the system.

The Software

- Open source BSD license
- User level C++ library
- Support Linux, BSD, UNIX, and Windows
- API very similar to BSD Socket
- 20,000 downloads so far, used in numerous commercial and research products.



Background

Inexpensive storage and high bandwidth optical networks have facilitated the rapid increase of distributed data intensive applications, especially in the field of E-Science.

TCP's problem: poor bandwidth utilization in high BDP networks, RTT bias, and prone to queuing and reverse traffic.

Figure on the left: two earth observation data streams from Chicago and Amsterdam were joined and analyzed in real time during iGrid 2002.

Symbols and Abbreviations

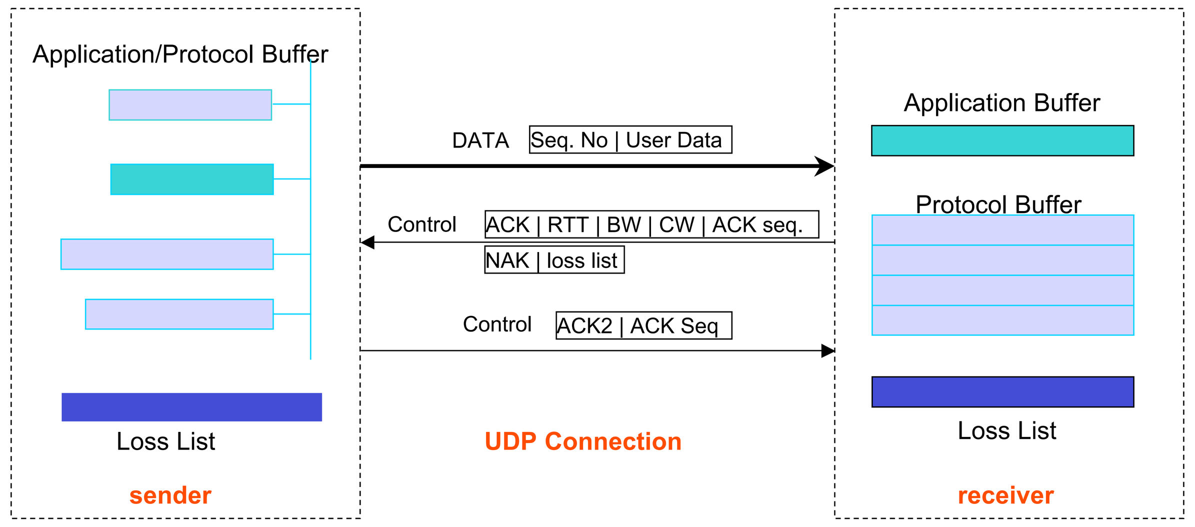
SYN: Synchronization Time. A UDT constant time that is 0.01 seconds.
RTT: Round Trip Time
MSS: Maximum Segment Size (in bytes)
R: Packet Sending Rate (in packets/s)
W: Congestion Window Size (in packets)
B: Estimated Bandwidth (in bits/s)
ACK: Acknowledgement
NAK: Negative Acknowledgement (loss report)
AS: Packet Arrival Speed (in packets/s)

UDT Architecture

UDT is an application level transport protocol over UDP. It is duplex. Each UDT entity has both a sender and a receiver. Two UDT entities communicate through a pair of UDP ports.

UDT uses packet-based sequencing and timer-based selective acknowledgement. ACK is sent every *SYN* and NAK is sent once a packet loss event is detected.

UDT uses a hybrid rate-window congestion control. Rate control is triggered every *SYN*, whereas window control is triggered every ACK.



Rate Control

Rate control tunes the packet sending rate. No more than one packet can be sent during each packet sending period.

Additive Increase: Every *SYN*, if there is no NAK, but there are ACKs received, the increment of next *SYN* is given by:

$$inc = \max(10^{\lceil \log_{10} B \rceil - 9}, 1/1500) \times 1500 / MSS$$

Multiplicative Decrease: For a random chosen NAK

$R = R * 8/9$

B (Mb/s)	Increment (packets)
$B \le 0.1$	0.00067
$0.1 < B \le 1$	0.001
$1 < B \le 10$	0.01
$10 < B \le 100$	0.1
$100 < B \le 1000$	1
$1000 < B \le 10000$	10
…	…

* MSS = 1500 bytes

Window Control

Window control limits the number of unacknowledged packets. It is done at the receiver side. Once an ACK is to be sent, update the window size to:

$W = W * a + AS * (RTT + SYN) * (1 - a) \quad 0 < a < 1$

The minimum value between *W* and the receiver's available buffer size (flow control) is sent to the sender in ACK.

Bandwidth Estimation

UDT uses receiver based packet pairs (RBPP) to estimate link capacity *L*. Suppose the current sending rate is *C*, then

if *C* is less than the last decreased sending rate

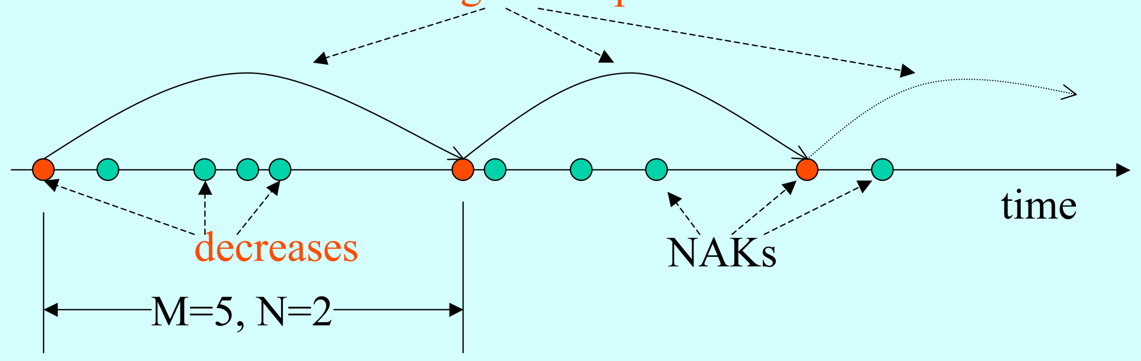
$B = L - C$

else

$B = \min \{L-C, L/9\}$

De-Synchronization

A randomization method is used to remove the negative impact of loss synchronization. It also avoids drastic changes in the sending rate.

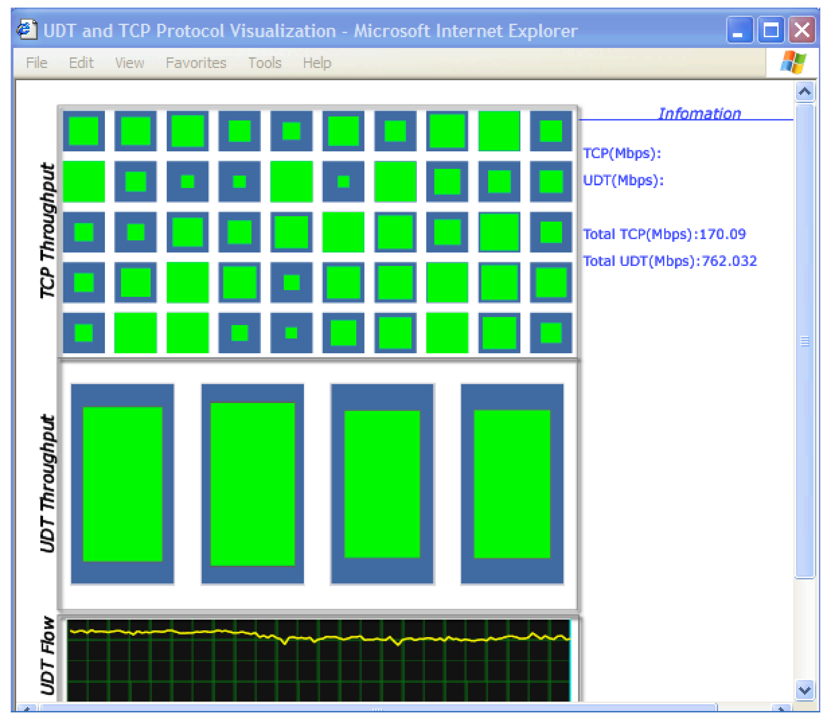


Implementation

UDP multiplexer allows multiple UDT connections to share single port.

Efficient processing on protocol data structures, including memory copy avoidance, self-clocking, and fast scheduling of UDT connections.

Supports multi-core processing, buffer auto-sizing, rendezvous connection setup, etc.



Demonstration

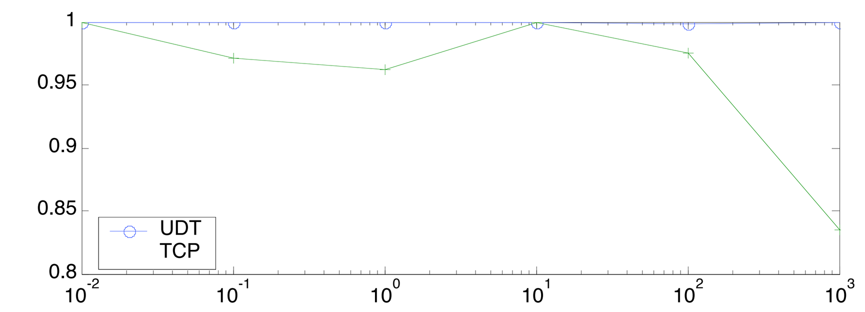
Figure on the left: 50 TCP flows and 4 UDT flows share a 1Gb/s, 180ms RTT link. The TCP window size is configured so that the maximum TCP throughput is 5Mb/s. The size of each green square represents the current transfer speed of that flow. The size of each blue square in the TCP group is 5Mb/s, and it is 250Mb/s in the UDT group.

Reference

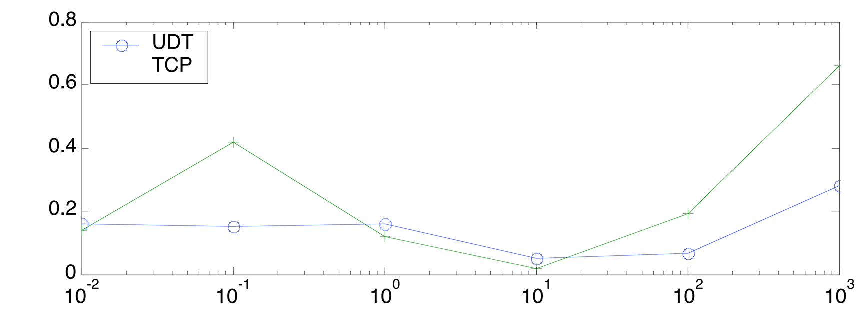
UDT web site: http://udt.sourceforge.net

Internet Draft: Yunhong Gu, Robert L. Grossman, UDT: A high performance data transfer protocol, draft-gg-udt-02.txt

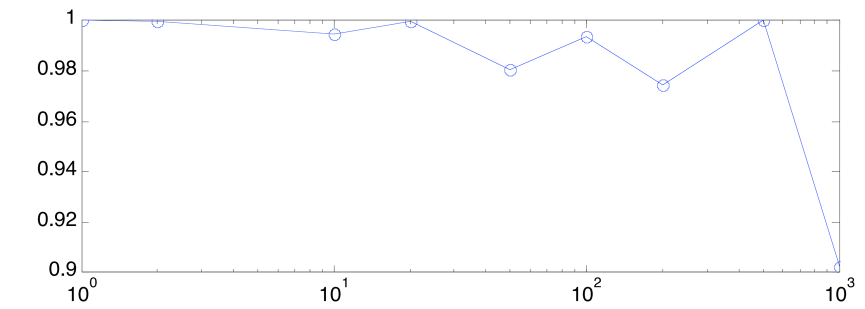
Paper: Yunhong Gu and Robert L. Grossman, UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Computer Networks (Elsevier). Volume 51, Issue 7. May 2007



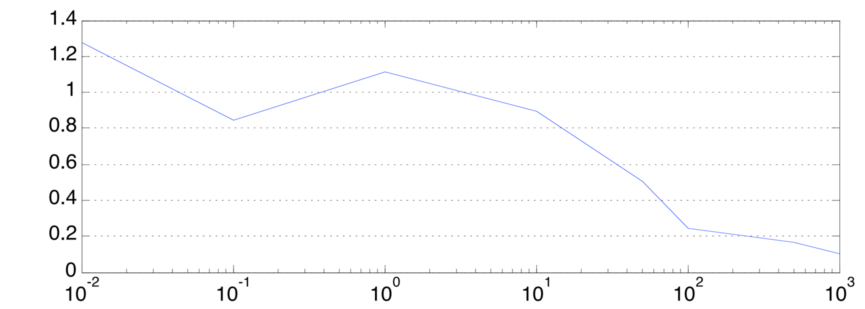
Jain's fairness index of UDT and TCP



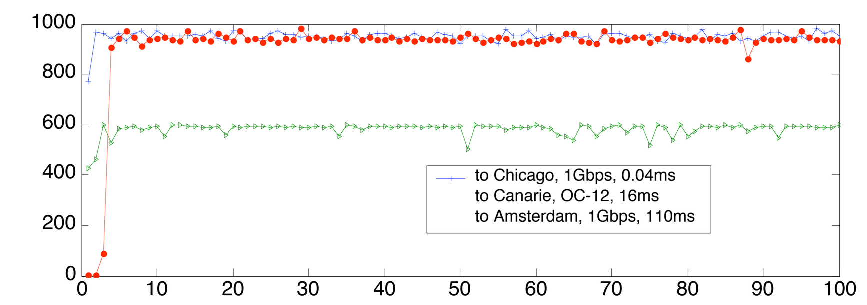
Stability Index of UDT and TCP: smaller value is more stable



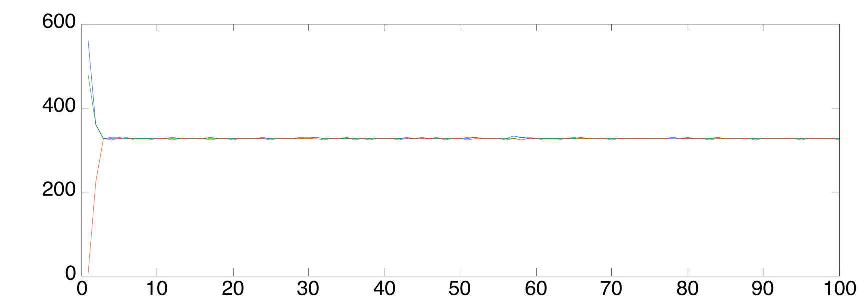
UDT's RTT fairness: values are the throughput ratio of two flows, one has a fixed *RTT* of 1ms, the other ranges its *RTT* from 1ms to 1000ms.



UDT's TCP friendliness index: values are mean throughput of 10 TCP flows with 5 other UDT flows vs. with 5 other TCP flows.



UDT's efficiency in real networks



UDT fairness in real networks: 3 UDT flows in real networks with different *RTT*s (0.04ms, 16ms, 110ms) and bottlenecks (OC12, 1Gb/s, 1Gb/s)