UDT: UDP based Data Transfer
Protocol, Results, and Implementation Experiences

Yunhong Gu & Robert Grossman
Laboratory for Advanced Computing / Univ. of Illinois at Chicago

Bill Allcock & Raj Kettimuthu
Globus Alliance / Argonne National Laboratory
Outline

- UDT Protocol
- UDT Congestion Control
- Implementation/Simulation Results
- Implementation Experiences at ANL
Design Goals and Assumptions

- Fast, Fair, Friendly
  - High utilization of the abundant bandwidth with either single or multiplexed connections
  - Intra-protocol fairness, RTT independence
  - TCP compatibility

- Low concurrency, high bandwidth, bulk data
  - A small number of sources share abundant bandwidth
  - Most of the packets can be packed in maximum segment size (MSS)
What’s UDT?

- UDT: UDP based Data Transfer
  - Reliable, application level, duplex, transport protocol, over UDP with reliability, congestion, and flow control
  - Implementation: Open source C++ library

- Two orthogonal parts
  - The UDT protocol framework that can be implemented above UDP, with any suitable congestion control algorithms
  - The UDT congestion control algorithm, which can be implemented in any transport protocols such as TCP
UDT Protocol

Packet Scheduling

Sender

Receiver

DATA

ACK

ACK2

NAK

Sender

Receiver

Redirect timers:

- ACK Timer
- NAK Timer
- Retransmission Timer
- Rate Control Timer

Packet scheduling diagram.
UDT Protocol

- Packet based sequencing
- ACK sub-sequencing
- Explicit loss information feedback (NAK)
- Four timers: rate control, ACK, NAK and retransmission timer
  - Rate control and ACK are triggered periodically
  - NAK timer is used to resend loss information if retransmission is not received in an increasing time interval
Congestion Control

- Rate based congestion control (Rate Control)
  - RC tunes the packet sending period.
  - RC is triggered periodically.
  - RC period is constant of 0.01 seconds.

- Window based flow control (Flow Control)
  - FC limits the number of unacknowledged packets.
  - FC is triggered on each received ACK.

- Slow start is controlled by FC
  - Similar to TCP, but only occurs at the session beginning.
Rate Control

- AIMD: Increase parameter is related to link capacity and current sending rate; Decrease factor is 1/9, but not decrease for all loss events.

- Link capacity is probed by packet pair, which is sampled UDT data packets.
  - Every 16th data packet and it successor packet are sent back to back to form a packet pair.
    - The receiver uses a median filter on the interval between the arrival times of each packet pair to estimate link capacity.
Rate Control

- Number of packets to be increased in next rate control period (RCTP) time is:
  \[ inc = \max(10^{\log_{10}((B-C)\times MSS\times 8)} \times \beta / MSS, 1 / MSS) \]
  where \( B \) is estimated link capacity, \( C \) is current sending rate. Both are in packets per second. MSS is the packet size in bytes. \( \beta = 1.5 \times 10^{-6} \).

- Decrease sending rate by 1/9 when a NAK is received, but only if:
  1. largest lost sequence number in NAK is greater than the largest sequence number when last decrease occurred; or
  2. The number of NAKs since last decrease has exceeded a threshold, which increases exponentially and is reset when condition 1 is satisfied.
## Rate Control

B = 10Gbps, MSS = 1500 bytes

<table>
<thead>
<tr>
<th>C (Mbps)</th>
<th>B - C (Mbps)</th>
<th>Increase Param. (Pkts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 9000)</td>
<td>(1000, 10000]</td>
<td>10</td>
</tr>
<tr>
<td>[9000, 9900)</td>
<td>(100, 1000]</td>
<td>1</td>
</tr>
<tr>
<td>[9900, 9990)</td>
<td>(10, 100]</td>
<td>0.1</td>
</tr>
<tr>
<td>[9990, 9999)</td>
<td>(1, 10]</td>
<td>0.01</td>
</tr>
<tr>
<td>[9999, 9999.9)</td>
<td>(0.1, 1]</td>
<td>0.001</td>
</tr>
<tr>
<td>9999.9+</td>
<td>&lt;0.1</td>
<td>0.00067</td>
</tr>
</tbody>
</table>
Flow Control

- \[ W = W \times 0.875 + AS \times (RTT + ATP) \times 0.125 \]
- ATP is the ACK timer period, which is a constant of 0.01 seconds.
- AS is the packets arrival speed at receiver side.
  - The receiver records the packet arrival intervals. AS is calculated from the average of latest 16 intervals after a median filter.
  - It is carried back within ACK.
Implementation: Performance

![Graph showing performance over time with different lines indicating traffic to StarLight, 40us RTT, traffic to Canarie, 16ms RTT, and traffic to SARA, 110ms RTT.](image-url)
Implementation: Intra-protocol Fairness

[Graph showing throughput in Mbps over time for different RTT values: to StarLight, 40us RTT; to Canarie, 16ms RTT; to SARA, 110ms RTT.]
Implementation: TCP Friendliness
Simulation: TCP Friendliness
Simulation: RTT Independence
Simulation: Convergence/Stability

![Graph showing simulation results](image-url)
For More Information

- LAC: www.lac.uic.edu
- Internet Draft: draft-gg-udt-xx.txt
- UDT: sourceforge.net/projects/dataspace
Implementation Experiences of UDT Driver for Globus XIO

Bill Allcock & Raj Kettimuthu
Globus Alliance
Argonne National Laboratory
Globus XIO

- Extensible input/output library for the Globus toolkit®.
- Simple intuitive open/close/read/write API.
- Provides a driver development interface.
- Framework takes care of non-protocol ancillary requirements such as error handling etc.
- As Globus XIO has a built-in UDP driver, the framework assists greatly in developing reliable layers on top of UDP.
- More details can be found at http://www-unix.globus.org/developer/xio
Improvements Made to UDT

- To make UDT closely resemble TCP, developed server interface to handle multiple connection requests
- Server listens on a known port for receiving connection requests
- Upon receiving a request, a new socket created and the port information communicated to the client
Improvements Made to UDT (cont.)

- Client establishes a new connection to this port for data transfer
- Introduced some changes to the handshake mechanism
- Requirements that we had
  - Receiver not expected to know the transfer size.
  - Sender does not communicate the transfer size to the receiver.
Improvements Made to UDT (cont.)

- Completion of transfer intimated by closing UDT
- Had to introduce a close state machine into the protocol
- Included new control messages for close handling
Performance

- Initial results
  - Average throughput of 97 MBps on a GigE LAN
  - Average throughput of 33 MBps over the wide area link from ANL to LBL (bottleneck is OC12 link)
  - Throughput over the wide area link is low compared to the throughput achieved by the UIC implementation
Performance (cont.)

- Exploring the cause for the difference in performance
- Known differences
  - Used non threaded flavor of Globus
  - Smaller protocol buffer
  - Driver operates on vectors as opposed to buffers