UDR: UDT Enabled Rsync for Large Data Transfers

Allison Heath, Robert Grossman

Problem Overview

UDT is a proven technology for transporting large datasets over high performance networks and has powered applications that have won the Supercomputing Bandwidth Challenge in 2006, 2008 and 2009. However, the scientific community has been slow to adopt UDT, likely because of a lack of easy to use tools. The goal of this project is to enable familiar tools with UDT. As a first step towards this goal we have created UDR, a lightweight wrapper that enables rsync to use UDT for data transfers.

UDT

UDT is a UDP-based, application level protocol. It is designed to support transferring large datasets over high speed wide area networks, where TCP has been known to be extremely ineffective.

UDT's features include:

- UDP-based, application level protocol
- Protocol design to support efficient packet processing
- Configurable congestion control
- Efficient native congestion control algorithm
- Optimized implementation as a user level C++ library
- Supports Linux, BSD, UNIX, and Windows
- API very similar to BSD Socket
- Open source BSD license
- Available at udt.sourceforge.net

UDT's results include:

- Send data from disk-to-disk at greater than 9Gb/s over a 10Gb/s wide area
- Powered Sector to enable over 100Gb/s data exchange among 4 data centers across the US
- Forms the underlying technology for products from 8 different companies.
- Used to move files by thousands of users each day who subscribe to the Hi Messenger service offered by Baidu

UDR Design and Implementation

UDR design goals:

- As similar to use as possible to rsync
- Minimize dependencies
- Simple installation
- Security support

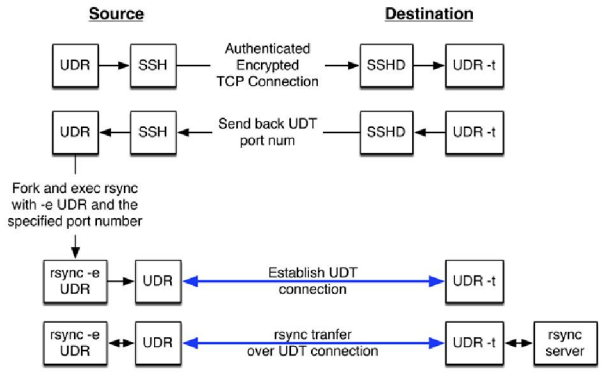
Current UDR installation and usage:

- Download from GitHub (github.com/LabAdvComp/UDR)
- Requires one dependency: OpenSSL
- Compile with make, specifying OS and architecture
- Creates a single binary 'udr'
- Prefix the current rsync command used to transfer data with 'udr', e.g.:

```
udr rsync -avz /home/user/tmp/ hostname.com:/home/user/tmp
```

- Any rsync options can be used

UDR does not change rsync, it works by creating a UDT connection and then places the connection between the rsync client and server. Below is a high level diagram of how UDR functions:



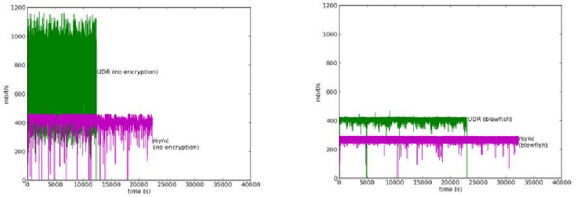
Results

We monitored transfer speeds of a 108 GB and a 1.1 TB dataset, between OSDC nodes located in Chicago and LVOC with a round trip time of 104ms. The table below contains the results of these transfers:

	108 GB Dataset		1.1 TB Dataset	
	mbit/s	LLR	mbit/s	LLR
UDR (no encryption)	752	0.66	738	0.64
rsync (no encryption)	401	0.35	405	0.36
UDR (blowfish)	394	0.35	396	0.35
rsync (blowfish)	280	0.25	281	0.25
rsync (3des)	284	0.25	285	0.25

- The current implementation of UDR only uses the blowfish cipher for encryption so rsync using blowfish as well as the default 3des encryption are included for comparison.
- LLR is defined as the ratio between the transfer speed and the minimum of the source disk read speed and the target disk write speed. In our experiments the local source disk read speed was 3072 mbit/s and local target disk write speed was 1136 mbit/s, so the denominator for the LLR is 1136 mbit/s.

1.1 TB transfer speed over time:



Future Work

- More complete security, investigating DTLS
- Rsync server capabilities (right now there is a development branch on GitHub)
- Enabling other transfer tools in a similar manner
- Integration into other OSDC capabilities
- Always open to feedback and suggestions

Questions or Comments: Allison Heath (aheath@uchicago.edu)

OSDC OPEN SCIENCE DATA CLOUD

IGSB Institute for Genomics & Systems Biology

OCC OPEN CLOUD CONSORTIUM

LAC LABORATORY FOR ADVANCED COMPUTING www.labcomputing.org

THE UNIVERSITY OF CHICAGO