UDR: UDT + RSYNC

Open Source Fast File Transfer

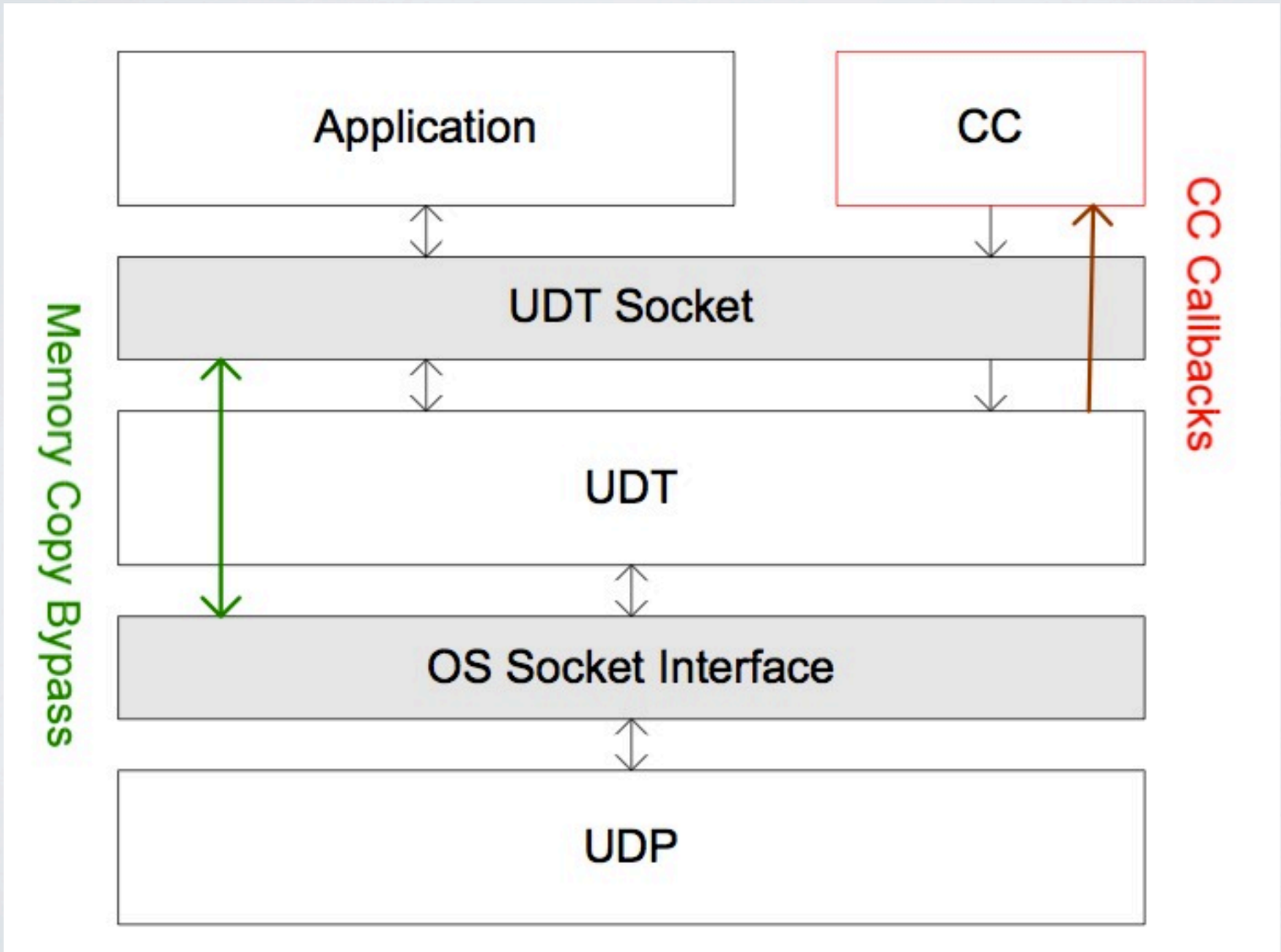
Allison Heath
University of Chicago

Motivation for High Performance Protocols

- High-speed networks (10Gb/s, 40Gb/s, 100Gb/s, ...)
- Large, distributed datasets
- TCP does not scale well as network bandwidth-delay product (BDP) increases

UDT: UDP-based Data Transfer

- Reliable UDP based application level protocol



A Brief History of UDT

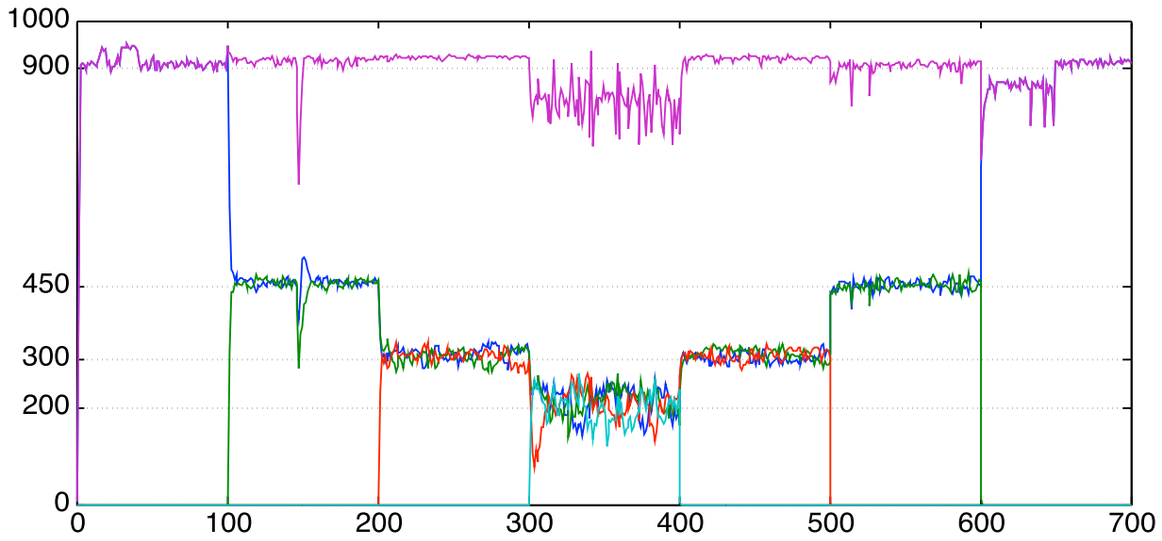
- Started in 2001 (SABUL)
  - Bulk data transfer: control information over TCP, data over UDP
  - Naïve congestion control
- UDT v2
  - Both control and data over UDP
  - Fast, fair, and friendly congestion control
- UDT v3
  - Configurable congestion control
  - Partial reliability & messaging
- UDT v4
  - Multiple UDT sockets on single UDP port
  - Better efficiency and congestion control

Current UDT Feature Summary

- Protocol design to support reliable, efficient packet processing
- Efficient native congestion control algorithm
- Configurable congestion control
- Optimized implementation as a user level C++ library
- API, as similar as possible to BSD sockets
- Supports Linux, BSD, OS X, and Windows
- Open source BSD license
- Available at udt.sourceforge.net

UDT Results

- 2006, 2008, 2009 Supercomputing Bandwidth Challenge Winner
- Disk to disk 9 Gbps over 10 Gbps long distance links
- Efficient, fair and stable



Flow 1	902	466	313	215	301	452	885
Flow 2		446	308	216	310	452	
Flow 3			302	202	307		
Flow 4				197			
Efficiency	902	912	923	830	918	904	885
Fairness	1	0.999	0.999	0.998	0.999	1	1
Stability	0.11	0.11	0.08	0.16	0.04	0.02	0.04

Moving Forward with UDT

- Open source and BSD license: udt.sourceforge.net
- Why has UDT not been more widely adopted?
- Some answers:
  - Lack of accessibility
  - Lack of knowledge for tuning/configuration required to achieve best performance
- Goal: Create a suite of easy to use data transfer applications that utilize UDT

UDT: Buffers and CPUs

Configuration Change	Observed Transfer Rate	Time to Transfer 1 TB (minutes)
UDT and Linux Defaults	1.6 Gbps	85
Setting buffers sizes to 64 MB	3.3 Gbps	41
Improved CPU on sending side with processor affinity	3.7 Gbps	36
Improved CPU on receiving side with processor affinity on	4.6 Gbps	29
Improved CPU on both sides with processort affinity on both sides	6.3 Gbps	21
Removing CPU clock scaling	6.7 Gbps	20

UDStar

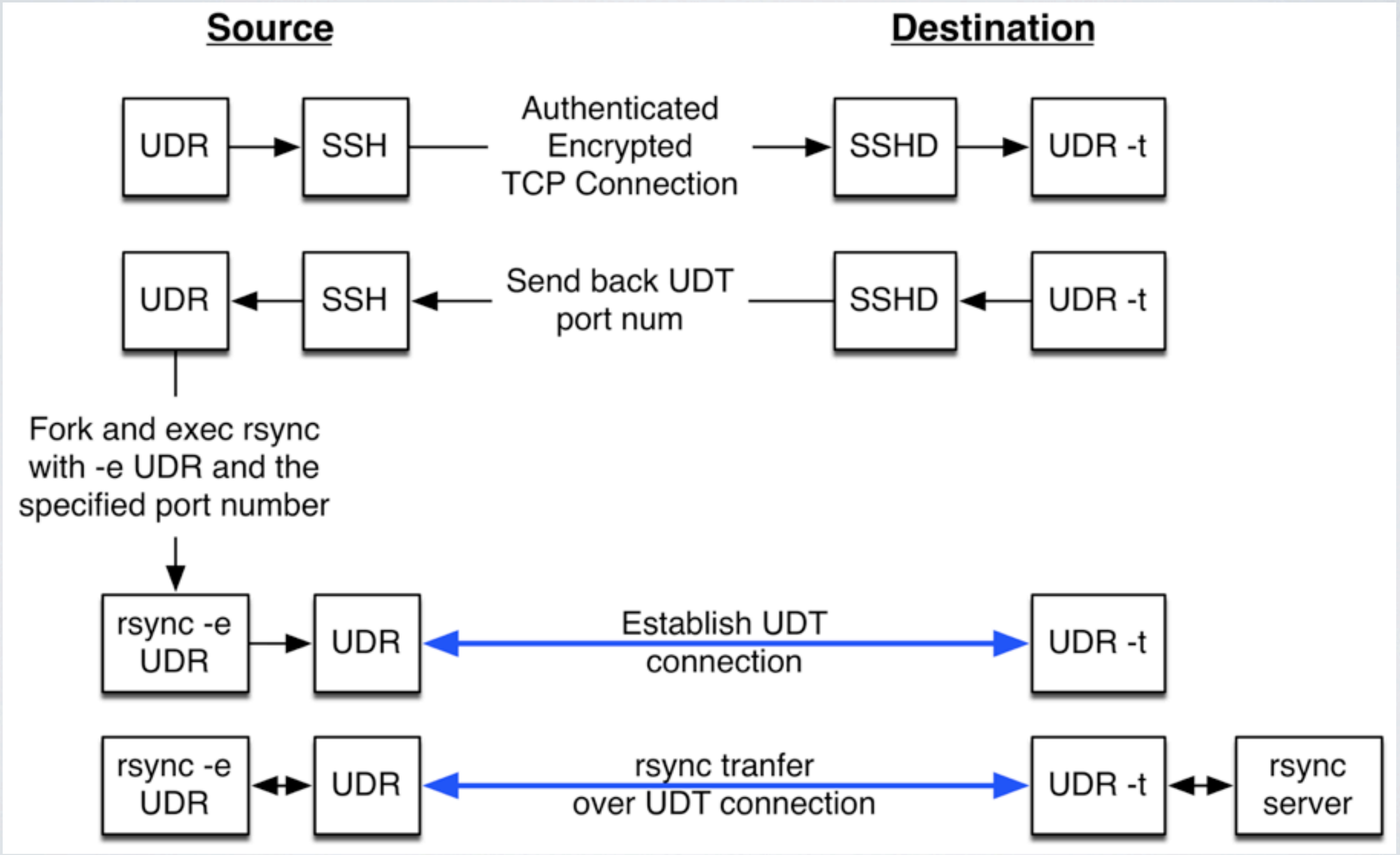
- There are a number of familiar data transfer tools that are in use: rsync, scp, ftp, (your favorite here)

- Goal: make them UDT-enabled with as little effort on the user's part as possible

UDR

- First component of UDStar
- UDT + rsync
- Lightweight wrapper around rsync that allows it to use UDT
- Made possible because rsync communicates via pipes to the remote shell
  - Utilize the rsync -e option

UDR Implementation



UDR: Current User Perspective

- Download from GitHub: github.com/LabAdvComp/UDR
  - Apache 2.0 Licensed
- Compile, only dependency is OpenSSL
- Once installed on both sides:
  - udr [udr options] [standard rsync command]
- Includes encryption based on OpenSSL (off by default)
  - 128, 192, and 256 AES
  - 3DES
  - Blowfish

UDR Server

- Modeled after rsync server
- Lightweight python server to handle UDR requests and run the appropriate process on the server
- Enabled users without accounts to download data
- Most configurations options are obtained from a provided rsyncd.conf

Evaluating Disk to Disk Transfer

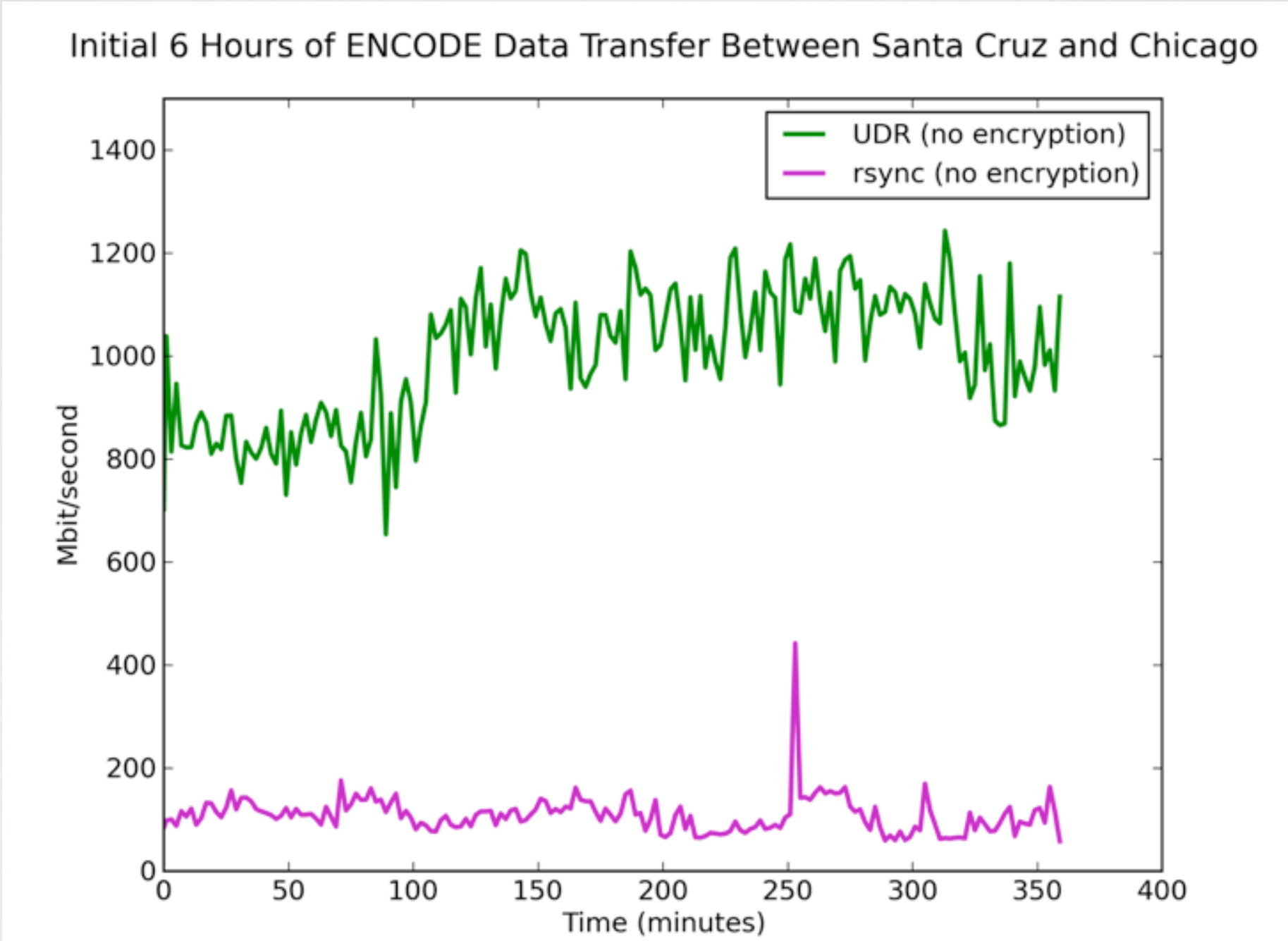
- Transfers limited by a number of factors:
  - Read speed on source data volume
  - Write speed on destination data volume
  - Lowest bandwidth on connections linking the volumns
  - Round Trip Time (RTT)
- Normalize performance to be independent of particular disks used:
  - transfer speed / min(source data volume read speed, destination data volume read speed)
  - Long Distance to Local Ratio (LDLR)

UDR Results: 100 GB Transfer

- Tested between UChicago Kenwood Facility and FIU
- 10G connection, RTT is 52ms
- 100 GB transfer

	Avg Mbps	LDLR
UDT Memory to Memory	6699	N/A
UDT Send/Recv File	2185	0.98
UDR 0.9 (no encryption)	916	0.41
UDR 0.9 (aes-128)	391	0.18
rsync (no encryption)	423	0.19
rsync (aes-128)	249	0.11

UDR Results: ENCODE



UDR Results: UCSC Genome Browser

Source	Destination	UDR (Mbps)	rsync (mbps)
Santa Cruz, CA	Milwaukee, WI	500	160
Santa Cruz, CA	Detroit, MI	600	150
Santa Cruz, CA	Bielefeld, Germany	600	6
Santa Cruz, CA	Aarhus, Denmark	350	6
Santa Cruz, CA	Brisbane, Austrialia	500	3

OSDC Public Data via UDR

- Data currently available via rsync and UDR
- www.opensciencedatacloud.org/publicdata

Download/synchronize Sloan Digital Sky Survey Data Release 8:

- Using rsync: `rsync -avzu guest@128.135.107.145:/glusterfs/osdc_public_data/sdss_bestdr8/ /path/to/local_copy`
- Using UDR: `udr rsync -avzu guest@128.135.107.145:/glusterfs/osdc_public_data/sdss_bestdr8/ /path/to/local_copy`

Next Steps for UDR

- Rsync-like server functionality for UDR
- Testing and improvements
- Packaging for easier installation
- Stable 1.0 release in 2013
  - Current: 0.9.3

Beyond UDR: UDStar

- Initial UDR component developed and released
- Next: UDT-enabled SCP
  - FTP? SFTP?
- Interested in hearing from potential users
- Other language wrappers for UDT to encourage development, some interest in python, java, others?

THANK YOU

github.com/LabAdvComp/UDR
udt.sourceforge.net
www.opensciencedatacloud.org

OSDC

LAC

THE UNIVERSITY OF CHICAGO