UDT as an Alternative Transport Protocol for GridFTP

John Bresnahan,[1,2,3] Michael Link,[1,2] Rajkumar Kettimuthu,[1,2] and Ian Foster[1,2,3]
[1]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439
[2]Computation Institute, University of Chicago, Chicago, IL 60637
[3]Department of Computer Science, University of Chicago, Chicago, IL 60637

Abstract

GridFTP has emerged as a de facto standard for secure, reliable, high-performance data transfer across resources on the Grid. By default, GridFTP uses TCP as its transport-level communication protocol. It is well known that TCP Reno cannot provide satisfactory performance on high-speed, long-delay networks. In this paper, we describe how we enabled GridFTP to use UDT as an alternative transport-level communication protocol. We compare the performance of GridFTP over UDT with GridFTP over TCP on various test beds. We also study the impact of UDT on bulk TCP flows.

1. Introduction

GridFTP [1] has been commonly used as a data transfer protocol in the Grid. The GridFTP protocol extends the standard FTP protocol and provides a superset of the features offered by the various Grid storage systems currently in use. Key features of GridFTP include the following:

Security: The Globus GridFTP [2] server/client utilizes the GSI protocol, which not only enables a secure Public Key Infrastructure (PKI) interface but also adds the capability of delegated authority via X.509 certificates. Delegated authority is critical for large collaboration efforts and enables single sign-on in virtual organizations, thereby eliminating the need for the user to enter passwords onto what can be hundreds of different sites. Kerberos is also supported.

Parallelism: On wide-area links, using multiple TCP streams in parallel between a single source and destination can improve aggregate bandwidth relative to that achieved by a single stream. GridFTP supports such parallelism via FTP command extensions and data channel extensions.

Striping: Additionally, GridFTP supports striped data movement, in which data distributed across, or generated by, a set of computers or storage systems at one end of a network is transferred to another remote set of storage systems or computers.

Third-Party Control: GridFTP also allows secure third-party clients to initiate transfers between remote sites, thereby facilitating the management of large datasets for distributed communities.

Partial File Transfer: Some applications can benefit from transferring portions of files rather than complete files. GridFTP supports requests for arbitrary file regions.

Reliability: GridFTP provides support for reliable and restartable data transfers.

Negotiation of TCP buffer/window sizes: GridFTP employs FTP command and data channel extensions to support both automatic and manual negotiation of TCP buffer sizes for large files as well as large sets of small files.

The Globus implementation of GridFTP provides a software suite optimized for the gamut of data access issues—from bulk file transfer to the details of getting data out of complex storage systems in sites. Although GridFTP supports multiple TCP streams to overcome the limitations of TCP congestion control algorithm for long, fat networks [3-5], it is still not possible to utilize the available bandwidth optimally in some situations. UDP based Data Transfer protocol (UDT) [6] is a popular application level data transport protocol that addresses the limitations of TCP in fast, long-distance networks. In this paper, we describe the following:

- Development of a Globus XIO driver for UDT

- Enhancements done to Globus GridFTP to use UDT as an alternate transport protocol
- Experimental study comparing the performance of GridFTP with TCP and UDT as the transport level protocols
- Experimental study of the impact of GridFTP with UDT on GridFTP with TCP flows.

The rest of the paper is organized as follows. In Section 2, we provide an introduction of UDT. In Section 3, we describe the development of Globus XIO driver for UDT. In Section 4, we highlight the enhancements done to Globus GridFTP. In Section 5, we present the experimental studies; and in Section 6, we summarize our results.

2. UDT

TCP's congestion control algorithm is window based and every time a congestion event is detected, the window size is reduced to half. After the detection of first congestion event, the window size is increased at most one segment each round-trip time regardless of how many acknowledgments are received in that round-trip time. This congestion control algorithm has significant limitations on fast, long-distance networks. Researchers have come up with numerous solutions to address the limitations of the above-mentioned TCP's AIMD-based congestion control mechanism [7]. These solutions include improvements to TCP [8-14], new transport protocols such as XCP [15], XTP [16] and reliable layers on top of UDP [6, 17-23].

UDT [6] is an application-level data transport protocol that uses UDP to transfer bulk data, while implementing its own reliability and congestion control mechanisms. UDT achieves good performance on high-bandwidth, high-delay networks in which TCP has significant limitations. UDT uses UDP packets to transfer data and retransmit the lost packets to guarantee reliability. UDT's congestion control algorithm combines rate based and window based approaches to tune the inter-packet time and the window size, respectively. The congestion control parameters (window size and inter-packet time) are updated dynamically by a bandwidth estimation technique. UDT is popular among the application level solutions to address TCP limitations and it has been shown to be friendly to short-lived TCP flows [6].

3. Globus XIO Driver for UDT

GridFTP uses the Globus Extensible Input Output (XIO) [24] interface to invoke network I/O operations. The Globus XIO framework presents a single, standard open/close, read/write interface to many different protocol implementations, including TCP, UDP, HTTP, and file—and now UDT. The protocol implementations are called drivers. Once created, a driver can be dynamically loaded and stacked by any Globus XIO application. Many drivers have been created by using the native Globus XIO assistance APIs, including TCP, UDP, HTTP, File, Mode E [1], Telnet, Queuing, Ordering, GSI, and Multicast Transport [25]. For other evolving protocols, if an implementation already exists, requiring the developers to implement their protocols by using native Globus XIO APIs would necessitate considerable effort.

Recognizing the benefit of providing wrapper code to hook these libraries into the Globus XIO driver interface, we therefore introduce the *wrapblock* feature to Globus XIO. This is a simple extension to the original Globus XIO driver interface that allows for much easier creation of drivers. The stock Globus XIO driver interface is written in an asynchronous model. While this is the most scalable and efficient model, it is also the most difficult to code against. Further, many existing protocol implementations do not have asynchronous APIs, and transforming them into an

asynchronous model can be time consuming. The wrapblock functionality uses thread pooling and event callback techniques to transform the asynchronous interface to a blocking interface. This makes the task of creating a driver from an existing library easier.

```
static
globus_result_t
globus_l_xio_udt_write(
        void * driver_specific_handle,
        const globus_xio_iovec_t * iovec,
        int iovec_count,
        globus_size_t * nbytes)
{
   globus_result_t result;
   xio_l_udt_handle_t * handle;
   GlobusXIOName(globus_l_xio_udt_write);
   handle = (xio_l_udt_handle_t *)
                driver_specific_handle;
   *nbytes = (globus_size_t) UDT::send(
        handle->sock, (char*)iovec[0].iov_base,
        iovec[0].iov_len, 0);
   if(*nbytes < 0)
   {
      result = GlobusXIOUdtError("UDT::send failed");
      goto error;
   }
   return GLOBUS_SUCCESS;
   error:
      return result;
}
```

Fig. 1. Sample wrapblock write interface implementation for UDT.

To illustrate the ease in creation, we present in Figure 1 the code required to implement write functionality for UDT. The implementation requires a simple pass-through call to the UDT library. The number of bytes written is passed back to Globus XIO by reference in the nbytes parameter. The data structure xio_l_udt_handle_t is created in the open interface call and is passed to all other interface calls as a generic void * memory pointer. This allows the developer to maintain connection state across operations. Similar code is written to handle reading data. In the open and close interface function, the initialization and cleanup of resources are done as would be expected. The code inside the driver looks much like a simple program using the third-party API. There is little Globus XIO–specific code beyond the interface function signatures. There are driver-specific hooks that allow the user to directly interact with the driver in order to provide it with optimization parameters. This interaction is handled via cntl functions that look much like the standard UNIX ioctl().

4. GridFTP Enhancements

A new command has been added to the GridFTP protocol to add drivers to the data channel stack.

SITE SETDCSTACK {<driver name>[:<driver options>],}+

The second parameter to the site command is a comma-separated list of driver names optionally followed by a : and a set of driver specific url-encoded options. From left to right the driver names form a stack from bottom to top. For security reasons the GridFTP server does not allow clients to load arbitrary xio drivers into the server. The GridFTP server admin must white list the driver individually, using the ‘-dc-whitelist‘ option to the server.

5. Experimental Studies

We compare the performance of Iperf, scp, bbcp, GridFTP over TCP (both single and multiple streams), GridFTP over UDT, and raw UDT on four different networks—a wide-area network between Argonne National Laboratory (ANL) and the University of Auckland, New Zealand, with a round-trip time of 204 ms; a wide-area network between ANL and Los Angeles, with a round-trip time of 60 ms; a wide-area network between the Ohio State University and JA site in Japan, which is a part of the Japan Gigibit Network II project, with a round-trip time of 193 ms; and a wide-area network between the JA site and Oak Ridge National Laboratory, with a round-trip time of 194 ms. To the best of our knowledge, all the pairs of the sites used in the experiments have 1 Gbit/s (maximum possible bandwidth)

connectivity. The operating system on all the nodes used in the experiments is Linux. TCP Reno is used in all the experiments.

Table 1: Throughput (in Mbit/s) achieved when transferring 1 GB of data over two wide-area networks, using various mechanisms

Testbed / Transport Mechanism	ANL - NZ	ANL - ISI	BMI - Japan	Japan – ORNL
Iperf – 1 stream	19.7	74.5	59	110
Iperf – 8 streams	40.3	117.0	115.3	592.4
scp – 1 stream	2.96	9.6	3.13	3.14
bbcp mem – 1 stream	-	35.3	5.84	118.2
bbcp mem – 8 streams	-	59.2	11.7	467.25
bbcp disk – 1 stream	-	35.15	5.85	112.6
bbcp disk – 8 streams	-	54.8	11.7	451.3
GridFTP mem TCP – 1 stream	16.4	63.8	79.6	123.3
GridFTP mem TCP – 8 streams	40.2	112.6	222	586.5
GridFTP disk TCP – 1 stream	16.3	59.6	73.6	113.5
GridFTP disk TCP – 8 streams	37.4	102.4	201.6	574.6
GridFTP mem UDT	189.7	426.6	238	382.5
GridFTP disk UDT	187.9	418.3	220.5	380.6
UDT mem	202.3	432.5	246.3	397.2
UDT disk	174.2	398.0	211.6	374.5

5.1 Throughput

In these experiments, 1 GB of data was transferred between the end points. Table 1 shows the throughput achieved in megabit per second. We noted that the performance of GridFTP over TCP is comparable to the performance of iperf and is significantly better than scp and bbcp. GridFTP over UDT outperforms the best possible throughput obtained with TCP by a factor of 4 on two testbeds (ANL-NZ and ANL-ISI).

Table 2: Throughput (in Mbit/s) for nonconcurrent and concurrent GridFTP over TCP and GridFTP over UDT flows on Japan-ORNL testbed

Nonconcurrent flows	GridFTP-TCP	132.1
	GridFTP-UDT	416.8
2 Concurrent TCP flows	GridFTP-TCP	120.1
	GridFTP-TCP	118.6
2 Concurrent UDT flows	GridFTP-UDT	403.2
	GridFTP-UDT	404.6
2 Concurrent flows (1 TCP and 1 UDT)	GridFTP-TCP	122.6
	GridFTP-UDT	404.3
3 Concurrent flows (2 TCP and 1 UDT)	GridFTP-TCP	122.6
	GridFTP-TCP	123.2
	GridFTP-UDT	405.7

GridFTP over UDT outperforms GridFTP over TCP (single stream) by a factor of 3 on the other two testbeds (BMI-Japan and Japan-ORNL). GridFTP over TCP (with 8 streams) performs as well as GridFTP over UDT on the BMI-Japan testbed and interestingly it outperforms GridFTP over UDT on the Japan-ORNL testbed. Two concurrent UDT flows achieve about 400 Mbit/s each (Table 2) but a single UDT flow alone gives only about 400 Mbit/s on the Japan-ORNL link. We suspect that, for some reason, the probing mechanism employed by UDT to determine the available bandwidth detects the available bandwidth to be around only 400 Mbit/s on the Japan-ORNL link. We could not get bbcp numbers on the ANL-NZ testbed because of firewall problems. The BMI-Japan testbed and the Japan-ORNL testbed had autotuning of TCP buffer size enabled; hence, the TCP buffer size was not explicitly set to the bandwidth-delay product (in fact, setting the TCP buffer size to bandwidth-delay product on these nodes reduced the throughput by multiple folds). On the other two test beds, the TCP buffer size was

set to bandwidth-delay product for the GridFTP over TCP transfers. We note that the performance of GridFTP over UDT is close (about 95% or more) to that of raw UDT in case of memory-to-memory transfers. It is interesting to note that GridFTP over UDT outperforms raw UDT in case of disk-to-disk transfers. We believe that this is because the UDT test application may not be optimized for disk I/O.

5.2 Impact of UDT on TCP flows

We ran GridFTP over UDT and GridFTP over TCP transfers concurrently to measure the impact of UDT on bulk TCP flows. For these tests, we did only memory-to-memory transfers (/dev/zero to /dev/null). We did not transfer a fixed amount of data for these tests, rather we ran the transfers until they attain steady throughput. This is the reason for the higher throughput for nonconcurrent UDT flows in Table 2 and Table 3 compared to those of in Table 1 for the BMI-Japan and Japan-ORNL testbeds.

Table 3: Throughput (in Mbit/s) for nonconcurrent and concurrent GridFTP over TCP and GridFTP over UDT flows on BMI-Japan testbed

Nonconcurrent flows	GridFTP-TCP	80.1
	GridFTP-UDT	240.2
2 Concurrent TCP flows	GridFTP-TCP	78.6
	GridFTP-TCP	79.2
2 Concurrent UDT flows	GridFTP-UDT	120.5
	GridFTP-UDT	126.4
2 Concurrent flows (1 TCP and 1 UDT)	GridFTP-TCP	42.4
	GridFTP-UDT	187.6
3 Concurrent flows (2 TCP and 1 UDT)	GridFTP-TCP	26.5
	GridFTP-TCP	34.6
	GridFTP-UDT	164.4

From Table 2, we see that UDT does not affect the throughput of the concurrent bulk TCP flows on Japan-ORNL testbed, whereas on the BMI-Japan testbed, UDT does have some impact on the bulk TCP flows. But we note from Table 3 that UDT does reduce its rate to accommodate the TCP flows.

We also ran some tests to compare the system resources utilized for GridFTP-TCP and GridFTP-UDT transfers. For these tests, we used the TeraGrid [26] network between ANL and ORNL, as the performance of TCP and UDT is comparable on this testbed. Both TCP and UDT achieved a throughput around 700 Mbit/s on this testbed. The CPU utilization for TCP transfers was in the range of 30–50%, whereas for UDT transfers it was around 80%. The memory consumption was around 0.2% for TCP and 1% for UDT.

6. Summary

We described how we enhanced the Globus GridFTP server and the framework to use UDT as an alternative transport level protocol for TCP. We discussed the wrapblock feature in Globus XIO that aided in the creation of the UDT driver. We presented experimental studies comparing the performance of GridFTP over UDT with GridFTP over TCP. We showed that UDT outperformed single-stream TCP on all the four testbeds and multistream TCP on three testbeds. We also reported on the impact of UDT on bulk TCP flows and the system resources utilized by UDT and TCP transfers. Overall, although UDT does use more system resources than TCP, it achieves significantly higher throughput compared to TCP and reduces its rate to accommodate the competing TCP flows.

Acknowledgments

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357, and in part by National Science Foundation's CDIGS.

References

[1] W. Allcock, "GridFTP: Protocol extensions to

FTP for the Grid," Global Grid ForumGFD-R-P.020, 2003.
[2] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," SC'05, ACM Press, 2005.
[3] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," ACM/SIGCOM, 2002.
[4] S. Floyd, "High-speed TCP for large congestion windows," RFC 3649, Experimental, December 2003.
[5] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in Proceedings of the IEEE Infocom, March 2004.
[6] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," Comput. Networks 51, 7 (May. 2007), 1777–1799. DOI=http://dx.doi.org/10.1016/j.comnet.2006.11.009
[7] Allman, M., Paxson, V. and Stevens, W. TCP Congestion Control. IETF, RFC-2581, 1999.
[8] Floyd, S. HighSpeed TCP for Large Congestion Windows. IETF, RFC 3649, 2003.
[9] Jin, C., Wei, D.X. and Low, S.H., FAST TCP: motivation, architecture, algorithms, performance. IEEE Infocom, 2004.
[10] Kelly, T., Scalable TCP: Improving Performance in High-Speed Wide Area Networks. First International Workshop on Protocols for Fast Long Distance Networks, 2003.
[11] Ha, S., Rhee, I., and Xu, L. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. SIGOPS Oper. Syst. Rev. 42, 5 (Jul. 2008), 64-74. DOI= http://doi.acm.org/10.1145/1400097.1400105
[12] R. Shorten, and D. Leith, "H-TCP: TCP for High-Speed and Long-Distance Networks," Second International Workshop on Protocols for Fast Long-Distance Networks, February 16-17, 2004, Argonne, Illinois USA
[13] T. Hatano, M. Fukuhara, H. Shigeno, and K. Okada, "TCP-friendly SQRT TCP for High Speed Networks," in Proceedings of APSITT 2003, pp455-460, Nov 2003.
[14] L. Xu, K. Harfoush, and I. Rhee, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," INFOCOM 2004, March 2004
[15] Katabi, D., Handley, M. and Rohrs, C., Congestion Control for High Bandwidth-Delay Product Networks. Sigcomm, 2002.
[16] Strayer, W.T., Lewis, M.J. and Cline Jr., R.E. XTP as Transport Protocol for Distributed Parallel Processing. USENIX Symposium on High-speed Networking. 1994.
[17] Tsunami Network Protocol Implementation, 2004. www.indiana.edu/~anml/anmlresearch.html.
[18] Chien, A., Faber, T., Falk, A., Bannister, J., Grossman, R. and Leigh, J. Transport Protocols for High Performance: Whither TCP? Communications of the ACM, 46 (11). 42-49. 2003.
[19] Clark, D., Lambert, M. and Zhang, L. NETBLT: A Bulk Data Transfer Protocol. IETF, RFC 998, 1987.
[20] Gu, Y. and Grossman, R.L., UDT: An Application Level Transport Protocol for Grid Computing. Second International Workshop on Protocols for Fast Long-Distance Networks, 2003.
[21] He, E., Leigh, J., Yu, O. and DeFanti, T.A., Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. IEEE Cluster Computing, 2002.
[22] Sivakumar, H., Grossman, R.L., Mazzucco, M., Pan, Y. and Zhang, Q. Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks. Journal of Supercomputing. 2004.
[23] Qishi Wu, Nageswara S. V. Rao: Protocol for high-speed data transport over dedicated channels. Third International Workshop on Protocols for Long-Distance Networks (PFLDnet 2005), Lyon, France, Feb. 2005
[24] W. Allcock, J. Bresnahan, R. Kettimuthu, and J. Link, "The Globus eXtensibleInput/Output System (XIO): A protocol independent I/O system for the Grid," in Proceedings of the 19th IEEE international Parallel and Distributed Processing Symposium (Ipdps'05) - Workshop 4, Vol. 5 (April 4-8, 2005). IPDPS. IEEE Computer Society, Washington, DC, 179.1. DOI= http://dx.doi.org/10.1109/IPDPS.2005.429
[25] K. Jeacle and J. Crowcroft, "A multicast transport driver for Globus XIO," in WETICE '05: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise. Washington, DC: IEEE Computer Society, 2005, pp. 284–289.
[26] TeraGrid. http://www.teragrid.org.