Enhancement of GridFTP through Hardware Offloading

Mohammad Javad Rashti, Gerald Sabin
RNET Technologies, Inc.
Dayton, OH 45459, USA
{mrashti, gsabin}@rnet-tech.com

Rajkumar Kettimuthu
Mathematics and Computer Science
Argonne National Laboratory
Argonne, IL 60439, USA
kettimut@mcs.anl.gov

***Abstract* — Extreme-scale scientific collaborations require high-performance wide-area end-to-end data transports to enable high data volumes. GridFTP is the de facto protocol for Grid-wide transfer of data. Existing predominant network transport solutions such as TCP have serious limitations that consume significant CPU power and prevent GridFTP from achieving high throughput on long-haul networks. These limitations have caused underutilization of existing high-bandwidth links in scientific and collaborative grids. To address this situation, we have enhanced Globus GridFTP, the most widely used GridFTP implementation, by developing transport offload engines such as UDT and iWARP on a programmable 10GbE NIC. Our results show significant reduction in server utilization and full line-rate sustained bandwidth in high-latency networks, as measured for up to 100 ms of network latency. In our work, we also offload OpenSSL security on SmartNIC to reduce host utilization for secure file transfers. The offload engine can provide line-rate data channel encryption/decryption on top of UDT offload without consuming additional host CPU resources. Lower CPU utilization leads to increased server capacity, which allows data transfer nodes to support higher network and data-processing rates. Alternatively, smaller or fewer DTNs are needed for a particular data rate requirement.**

Keywords— Scientific Collaboration; GridFTP; UDT; Network Offload; RDMA

I. Introduction

Extreme-scale scientific computations, experiments, and collaborations require unprecedented wide-area end-to-end capabilities in the form of high-throughput data transports to enable the transfer of high data volumes between organizations or facilities. Such requirements arise in a number of science areas including climate, high energy physics, astrophysics, combustion, Nano-science, and genomics. The modeling of complex systems, such as climate or supernovae, at higher and higher fidelity generates proportionately larger volumes of data that must be visualized, examined, and analyzed by widely dispersed scientist teams looking for insight and discovery. Unfortunately, the amount of data being created by many computational codes faces significant transfer limitations, despite the available 10 Gbps and 100 Gbps capacities of grid network backbone connections. Supernovae simulation, genomics, and combustion modeling are some of the areas affected by such shortcomings.

The current data transfer limitations are no longer an artifact of the limited capacity in the network backbone as originally surmised. Indeed, the 10Gbps and 100Gbps backbones of either the Energy Science Network (ESNet) [3] or Internet2 [15] can currently offer such capacity to connect pairs of sites for extended periods. These networks enable scientific applications to transfer extremely large data files, primarily by using high-performance data transfer protocols such as GridFTP [2]. However, the fact that science users rarely see or use this bandwidth is symptomatic of much deeper challenges, which will only get worse with the next generation of multi-petascale and future exascale projects. Effectively utilizing the available bandwidth requires efficient underlying transport protocols and end-to-end implementations that can sustain a high bandwidth in high-latency transfers.

Historically, wide-area data transport has been handled mostly by the Transmission Control Protocol (TCP), which has been the basis for the File Transfer Protocol (FTP), GridFTP [2], bbcp [4], and HyperText Transfer Protocol (HTTP). The unprecedented demands that extreme-scale applications place on data transport, and the geo-diversity of today's scientific collaborations have pushed TCP beyond its useful envelope. The fundamental problem with TCP ultimately reduces to its treatment of bandwidth as a shared resource. A number of efforts have been made to develop high-performance versions of TCP [5][36][37], but with only limited success.

Research shows that the UDT transport protocol [8], a reliable user-level protocol on top of UDP, can achieve a much better throughput than a TCP connection can over a long-haul network [24]. Nevertheless, the user-level processing incurred by UDT consumes many more resources than does a TCP connection (for the same data rate), effectively preventing the widespread realization of UDT benefits. In addition, host resource requirements are expected to limit the maximum achievable bandwidth of GridFTP, since on many systems the processing time required to process the UDT messages will be even larger than the data transfer time. Clearly needed, therefore, is a means to achieve near "wire" transfer speeds, while using fewer host resources, allowing for more host capacity.

Another challenge is data transfer security. Many applications (e.g., medical, pharmaceutical, aerospace, and

This material is based upon work supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research SBIR program under Awards Number #DE-SC0002182 and #DE-FG02-08ER86360.

security applications) require secure GridFTP communications. GridFTP offers options to secure data transfers. However, the resources required for performing operations such as encryption and decryption greatly reduce the achievable bandwidth. Therefore, users either must see greatly reduced throughput or risk sending unencrypted data [35].

Besides the actual protocol processing, data transfer nodes (DTNs) [16] at the edge of the networks are responsible for several other data-processing tasks. Such tasks include file integrity check by computing the checksum after the file is written to disk (as done by the Globus transfer service [39]), data compression, and data reduction. Therefore, it is crucial to free some of the DTN's processing power for these additional data-processing tasks without affecting the observed throughput.

To address these challenges, we present a set of asynchronous network interface card (NIC) offload engines that improve the GridFTP long-range throughput and host resource utilization. Using a SmartNIC 10GbE card [25], we examine the benefits of offloading UDT to allow for line-rate bandwidth on long-range networks. We also use the UDT engine as the underlying transport protocol for offloading the Remote Direct Memory Access (RDMA) and Secure Sockets Layer (SSL) protocols to further reduce resource consumption. Our experiments show that by using NIC offload, near-line-rate throughput can be achieved through a single data stream on high-performance long-haul networks such as ESNet, while offering a multifold reduction in host CPU utilization.

The rest of this paper is organized as follows. Section II provides a background on the protocols and tools used in this work. Section III expands on the motivations of our work. Section IV discusses the related state of the art. In Section V, we discuss our detailed design and implementation of NIC offload engines. In Section VI, we present experimental results, followed by analysis and conclusion in Section VII.

II. Background

In this section, we briefly discuss GridFTP, UDT, and RDMA, along with the SmartNIC user programmable card used in this project.

A. GridFTP

GridFTP is a high-performance, secure and reliable data transfer protocol optimized for high-bandwidth wide-area networks [2]. It is based on the Internet FTP protocol, with extensions for high-performance operation and security. GridFTP is the preeminent standard for science projects requiring secure, robust, high-speed bulk data transport.

The GridFTP protocol is a backward-compatible extension of the legacy RFC 959 FTP protocol [18]. It maintains the same command/response semantics introduced by RFC 959. It also maintains the two-channel protocol semantics. The separation of the control and data channels in GridFTP enables third-party transfers, that is, the transfer of data between two end hosts, mediated by a third host. This functionality made it possible to develop hosted clients such as Globus Transfer [39] for GridFTP servers.

The de facto implementation of GridFTP is the Globus distribution [1][26]. This implementation is used by thousands of users with millions of data transfers per day. The Globus implementation of GridFTP provides a software suite optimized for a wide range of data access issues, from bulk file transfer to the details of getting data out of complex storage systems within sites.

Key features of GridFTP and its predominant implementation (Globus) include the following:

1) Parallel TCP streams: GridFTP uses parallel streams to overcome the inherent limitation in AIMD-based TCP congestion control algorithm. Typically, it provides orders of magnitude higher performance compared with that of standard FTP.

2) Cluster-to-cluster data movement: GridFTP can do coordinated data transfer by using multiple nodes and streams at the source and destination. This approach can increase performance by another order of magnitude.

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

4) Multiple security options: The Globus GridFTP framework supports various security options, including Grid Security Infrastructure (GSI) [9], anonymous access, username- and password-based security similar to that of regular FTP servers, SSH-based security, and Kerberos.

5) Modularity: The XIO-based [10] Globus GridFTP framework makes it easy to plug in other transport protocols. The Data Storage Interface (DSI) [11] allows for easier integration with various storage systems.

6) Third-party control: GridFTP also allows secure third-party clients to initiate transfers between remote sites.

7) Partial file transfer: Scientists often want to download only portions of a large file, instead of the entire file. GridFTP supports this capability by specifying the byte position in the file to begin the transfer.

8) Negotiation of TCP buffer/window sizes: GridFTP employs FTP command and data channel extensions to support both automatic and manual negotiation of TCP to get optimal performance.

B. Alternative Transports in GridFTP – UDT and RDMA

UDT is a reliable messaging protocol developed by the University of Illinois [8]. It is built on the top of UDP with reliability control and congestion control. With its congestion control algorithm, UDT can efficiently utilize the high-speed wide-area networks with a high bandwidth-delay product.

RDMA is one of the most significant breakthroughs in efficient network data transfer. RDMA allows for direct transfer of host buffers residing on distinct memory spaces without intermediate copies, host CPU involvement, or operating system overheads. The Internet Wide-Area RDMA Protocol (iWARP) specification (proposed by the RDMA Consortium [38] to the IETF in 2002) defines a stack of RDMA layers on top of standard TCP/IP over Ethernet. The stack decouples the processing of Upper Layer Protocol (ULP) data from the operating system and reduces the host CPU utilization by avoiding copies during data transfer. In order to achieve these goals, iWARP must be fully offloaded to an RDMA-capable NIC (RNIC) on top of an offloaded transport protocol. While standard iWARP specifies only TCP and SCTP as the underlying protocols, it can be ported to other transports, such as UDP [22][23] (with modifications) or UDT.

C. *SmartNIC – The Network Offload Platform*

The 10 Gbps Ethernet SmartNIC [25] that we have used as our offload platform in this project consists of a Cavium OCTEON Plus CN5750 network processor [27], with 12 cnMIPS64 cores running at 750 MHz. An on-board 2 GB DRAM allows ample packet buffering space to enable deep message inspection and large windows. The SmartNIC has a PCIe x8 host connection and an SFP+ connector for the 10GbE networking connection.

The OCTEON network processors contain hardware processing units to accelerate common networking related functions, deep packet inspection, compression, data deduplication, RAID, security, DMA, and packet scheduling. The C-based programmability of the OCTEON enables the SmartNIC to be easily programmed with software both during development and by the end user.

III. Motivations

Several shortcomings prevent applications from achieving high throughput on long-range grid connections. Besides TCP throughput issues, network processing overhead can limit the overall throughput a DTN node can provide, leading to deployment of more high-end servers to support the required throughput. While data transfer may be the main job of a DTN, other data-processing tasks are also expected to be performed on the data, which require extensive CPU power on the DTN. Unfortunately, when the amount of host-based network protocol processing is high, little CPU power will be left to perform other data operations. These shortcomings serve as the main motivations for our work presented in this paper.

The TCP protocol is not designed for high-throughput, high-latency links. Therefore, a TCP-based protocol stack cannot achieve good utilization in such circumstances. Because of its RTT bias problem [12][13], which can lead to a drop or fluctuation in the observed throughput, it cannot efficiently handle data transfers over computational grids. Processing the entire TCP-based networking stack can also consume a large percentage of host resources. For example, a TCP message on a 10Gbps Ethernet link can easily consume most of the CPU cycles using a 2.2 GHz Opteron CPU [5].

InfiniBand (IB) [6] was introduced as an alternative to Ethernet and TCP/IP, revolving around RDMA technology, to reduce host overheads and increase maximum achievable throughput. However, IB has received limited adoption among networks interconnecting scientific grids. It was originally designed for short-range data transfers typically between high-performance storage and computing systems. Devices such as Longbow [7] extend the reach of IB over wide-area connections, but with limited market. Efforts such as porting IB transport over an Ethernet link layer (e.g., RoCE [28]) have also received limited use, partially because of the lack of support for wide-area networks[1]. Therefore, traditional TCP/IP protocols are still the predominant choice for scientific data transfers.

Using UDT as the transport protocol, GridFTP has seen a much higher percentage of the available bandwidth. Nevertheless, while UDT has proven to be a better alternative for sustaining higher throughput over high-latency networks [24], its CPU usage can be prohibitive, causing limited practical use in high-speed data transfer applications. With such limitations, a hardware offloading of UDT to the NIC seems to be inevitable, in order to achieve both high network throughput and higher server capacity.

Data security is another bottleneck. GridFTP is a command/response protocol. GridFTP, like any FTP, uses two channels: a control channel and a data channel. The control channel is used for sending commands and responses and is encrypted for security reasons. The data channel is used for transferring the actual data of interest. By default, the data channel is authenticated at connection time, but no integrity checking or encryption is performed because of performance overheads. Enabling encryption or integrity protection can greatly reduce the throughput, sometimes by 90%. Despite having accelerated encryption instructions in modern Intel processors (i.e., AES-NI and vector instructions), the processing requirements for data channel security are still high (as we discuss later). These issues have led many application users not to enable encryption or integrity checking on GridFTP data channel. Here is where NIC offload can help, significantly improving the observed throughput.

IV. Related Work

Several attempts have been made to improve the performance of GridFTP file transfers over scientific grids. We discuss some of these in this section.

Bresnahan et al. compared the performance and system resources utilized for GridFTP-TCP and GridFTP-UDT transfers [24]. The experiments, which were conducted on the TeraGrid [14] network between Argonne and Oak Ridge national laboratories, showed that while UDT can achieve higher throughput than TCP can on a long-haul network, the CPU utilization for TCP transfers was lower, in the range of 30–50%, compared with that of UDT transfers, which was around 80%.

Kissel and Swany [21] presented an RDMA XIO driver for GridFTP, which uses revolving sender- and receiver-side queues for buffering RDMA messages over an emulated long-haul network based on RoCE. They showed how RDMA can sustain line-rate throughput over their emulated network. In an earlier attempt, Subramoni et al. [34] extended GridFTP to support RDMA over InfiniBand. The experiments were done using Longbow WAN routers [7], and close to line-rate throughput was demonstrated for short- to medium-range network latencies.

[1] RoCEv2, under development, can support WANs using IPv6 routing.

Gunter et al. [33] investigated the benefits of using parallel networks, one IP-based and one circuit-based network between two endpoints, to increase the overall throughput of GridFTP.

Vardoyan et al. [35] investigated file transfer bottlenecks in GridFTP especially when security processing is involved. They demonstrated a threaded mechanism with which multiple transfer threads can utilize system resources effectively for protocol and security processing.

While these projects have significantly contributed to the file transfer performance of GridFTP using various transport protocol techniques, our work in this paper utilizes programmable NICs for offloading CPU-intensive protocol processing to achieve higher performance and lower utilization. Besides improving the observed file transfer throughput over long-haul networks, this method increases the available processing power for the GridFTP server host, allowing for higher data handling and transfer capacity.

V. NIC Offload Engines

In this section, we discuss the design and implementation of our NIC offload engines.

A. *UDT (Reliable UDP)*

We have ported the UDT v4 implementation [29] to the SmartNIC, with substantial modifications to make it work efficiently on OCTEON processors. The NIC-resident firmware leverages the OCTEON's unique Schedule/Synchronization/Order (SSO) unit, hardware timers (TIM), locking mechanisms, atomic operations, and network accelerators for parallelization and workload partitioning [41]. Here we highlight some features of the implementation.

Network Processing of Data Flow: Fig. 1 depicts the arrangement of the OCTEON cores for network stack processing, including UDT, SSL, and RDMA. Blue (dark) boxes represent OCTEON integer cores, which are logically divided into a send and a receive side. Note that 10 cores participate in different data-processing tasks, including send and receive, in a data-flow model. They process work requests submitted by the OCTEON Packet Order and Work (POW) unit [41] upon network or host arrival. The remaining 2 cores are dedicated for asynchronous sending of UDT messages and UDT timing control. The asynchronous arrangement of protocol processing steps increases the utilization of the OCTEON cores, resulting in higher capacity for packet processing, as rarely is a core left idling for synchronization.

At the send side, data are pushed by the host through PCIe interface and OCTEON's Packet Input Processing (PIP) unit [41]. Messages are then sent, using DMA, to the card and put on the UDT send queue. A dedicated asynchronous send core transfers the UDT data (through the lower network layers) to the link.

At the receive side, the UDT receive queue is integrated with the existing control loop that terminates incoming network data. Upon arrival and after initial checking and time management, any UDT packet is placed on the receive queue for delivery to the upper layer protocols.

Each UDT socket has a callback function, which is invoked by using asynchronous timers after initial processing of the incoming data in the asynchronous receive. The callback function is assigned based on the upper layer protocol that receives the UDT data. The UDT protocol data (belonging to host-based UDT sockets) are pushed up to the host receive rings by this callback (using DMA, through OCTEON's Packet Output – PKO – engine [41]). Configurable interrupts generated by the NIC inform the host of the arrival of the UDT packets.

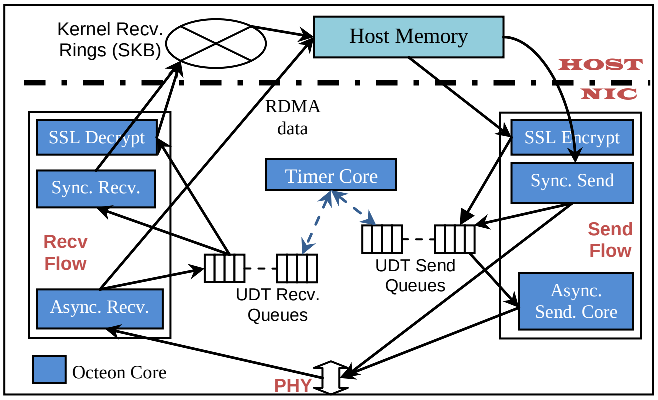


Fig. 1. Transport offload packet processing dataflow on SmartNIC

Zero-Copy Processing: All protocol processing on the SmartNIC is performed without data copies (zero copy). Packet data is not being copied to NIC memory buffers in any of the processing stages. The same packet buffers that are allocated by the hardware at ingress are re-chained and forwarded to the egress path. This approach saves a significant amount of processing power and latency, while allowing for line-rate throughput. At the host side, however, a travel through the kernel and a single copy is inevitable for UDT sockets, to transfer the data out of the kernel's socket buffers into the user space and vice versa (for non-RDMA cases).

B. *iWARP and RDMA*

A version of iWARP that runs over UDT has been designed for SmartNIC. RDMA Protocol (RDMAP) and Direct Data Placement (DDP) layers are offloaded to the card, while the verbs layer, including queue pair processing, is partially managed at the host.

DDP Layer: DDP enables a ULP to send data to a data sink without requiring an intermediate buffering. A TCP-based DDP is typically implemented on top of the Marker PDU Aligned (MPA) framing protocol. One can, however, implement DDP on top of any protocol that follows the reliability requirements set forth in RFC 5041 [19]. Using datagram-based UDT releases the requirement of using the MPA layer, thus removing both latency and bandwidth overheads associated with using markers. DDP uses special callbacks for asynchronous transfer of data to the host using OCTEON's DMA engines, to help increase the overall capacity of the NIC.

RDMAP Layer: The RDMAP firmware is layered immediately on top of offloaded DDP. RDMAP relies on the DDP layer for transmission and placement of data into tagged buffers and the passing of untagged buffers. This implementation of RDMAP is based on the specification described by RFC 4296 [20]. RDMAP is generally a thin layer that uses DDP header fields for different RDMA operations. Our current implementation of RDMAP supports RDMA send, receive, and write operations; and we are planning to add RDMA read operation in the future.

iWARP Driver Implementation: The host driver implementation is compatible with the Open Fabrics Alliance's OFED [30]. The SoftiWARP software stack [31] is used as a base for our host- side iWARP support, in which the software processing of RDMAP and DDP are bypassed to the card. We have modified the base OFED SoftiWARP software to support the SmartNIC offload engines.

RDMA Connection Manager (CM) is an essential piece to transparently support IB and iWARP networks using the same OFED API. SoftIWARP implements a version of RDMA CM that utilizes TCP/IP. We have modified this implementation to perform over the offloaded UDT protocol. When establishing an RDMA connection, a pre-connected UDT socket that has been offloaded to the card is registered to the RDMAP and DDP layers. At this stage the socket is considered to be "in RDMA mode" and all its data will be processed by the NIC iWARP layers.

C. Security Offload and OpenSSL

SSL is one of the essential standards utilized by GridFTP for security processing. GSI is built on SSL/TLS for encryption and mutual authentication. GridFTP-Lite [40] also uses SSL for its security purposes. OpenSSL is an open-source SSL/TLS toolkit utilized by GridFTP.

Cavium has provided the core functionality for card-side SSL, in an API that offers base functions for context management, encryption/decryption, digital signature, and so forth, utilizing OCTEON's per-core security coprocessors. We have created an API that utilizes this functionality for OpenSSL security processing in an asynchronous data-flow fashion on the NIC.

The base functions are inefficient, since they essentially use only one OCTEON core and perform several data copies, causing the overall throughput to be very low. The low performance is partially due to the library's internal mechanism for handling SSL sequence numbers that is non-scalable and serializes the process, not allowing multiple blocks to be processed in parallel. We modified the library to handle SSL sequence numbers outside of the core functions, using OCTEON's powerful atomic operations. With this optimization, since we can concurrently utilize up to 10 cores on the 12-core OCTEON processor for a single SSL stream, we are able to fully parallelize the cryptography process on independent blocks, increasing the observed throughput by eight to nine times.

We have further improved the SSL offload performance by removing data copies and enabling concurrency in SSL API processing on the card. We have eliminated memory copies that happen on the card, both in our library and in Cavium's base crypto library. Note that the inherent data touch in the security processing is inevitable, where the data is read from the source packet buffer, and the encrypted/decrypted data is written into the destination packet buffer). We also use cache prefetching and aligned buffers to improve the memory access time for the one-time data touch. Prefetching and buffer alignment further improved the SSL throughput by ~20%.

To reduce firmware complexity, we keep the buffer size equal to a packet buffer size (currently near 9 KB). This helps avoid extra copies for moving the entire data in an intermediate contiguous buffer for cryptography, as the card side packet buffers belonging to a ULP data unit are not necessarily contiguous in memory.

D. GridFTP over NIC Offload Engines

To enable the Globus GridFTP to use the offloaded transport protocols, we use XIO (eXtendible Input/Output System) drivers. XIO in GridFTP [10] provides a modular framework in which different communication protocols can be utilized by using a common standard interface supporting operations such as open/close/read/write. The XIO interface connects to the communication protocol using a driver structure, in which multiple drivers for supported protocols can be used by the GridFTP server/client.

We have used two XIO drivers for UDT and RDMA. The UDT XIO driver in GridFTP works on top of UDT offload engine on the card. For RDMA, we have used the XIO-RDMA driver developed as part of Phoebus project [21], as the base for our iWARP (RDMA) XIO driver on top of the SmartNIC iWARP stack. This driver uses multiple threads at server and client sides for performing RDMA operations. RDMA Send/Recv operations are used for control purposes. We have modified this driver to support both Send/Recv and RDMA Write for data transfer.

A socket-based channel is used in the original RDMA XIO driver for handshaking and control. Since RDMA Write operations are one-sided, the server and client need a separate mechanism to inform the data sink (receiver) of the completion of an RDMA Write operation. The control messages need to complete only *after* the data transfer is completed on the RDMA write channel. The original driver uses a separate non-RDMA socket for control-channel completion notifications. We have moved the control messages over to the RDMA channel to allow for guaranteed completion sequence. In fact, two mechanisms are used for completion checking:

- Destination buffer checking/polling (using a small header and footer around the data in XIO driver's internal SLAB buffers). These checks are intermittent, in order to avoid high CPU utilization.
- Two-sided (Send/Recv) RDMA operations to inform the data sink of a completed RDMA Write (recommended by the standard).

At completion, the data sink sends a TCP or RDMA Send/Recv based receipt to the data source, indicating the full reception of data, so that source SLAB resources can be freed. Our performance results are best when using the first completion method; however, this method is nonstandard and may not be used on systems where memory placement of network data can be out of order.

For increased performance, we also modified the XIO driver to send larger data chunks of up to a certain size to the card. The best RDMA performance on OCTEON is gained when the data is pushed from the host (using the scatter/gather-based DMA capabilities of OCTEON's IPD/PIP unit) instead of asking the OCTEON cores to initiate DMA operations. Such data-pushing operations however are limited to no more than 14 user buffers; and since we are dealing with 4 KB user pages as the largest buffer size in the RDMA scenario, our DMA size is limited to less than 56 KB. Note that this does not negatively affect performance or functionality and is completely opaque to the GridFTP user.

VI. Experimental Results

To evaluate the offload engines and their benefits for throughput and host CPU utilization, we begin with some microbenchmark results for several offload engines, including UDT, RDMA, and SSL offload. We then present GridFTP results over an emulated long-delay network. To further verify our findings, we also present results for GridFTP over UDT on a long-haul 10GbE link reservation on ESNet.

A. *Experimental Platform*

The tests were performed on two platforms:

1) RNET_P is a cluster, each node with two quad-core Intel Xeon E5620 processors running at 2.4 GHz and 12 GB of RAM. The nodes run CentOS 5.3, Linux kernel 2.6.18-128. We use synthetic delay insertion on a middle node to emulate a long-haul network with various latency numbers.

2) UC_P is a two-node configuration at the Argonne/University of Chicago Computation Institute. Each node has a quad-core Intel Xeon E5504 processor running at 2 GHz, with 4 MB of cache. The nodes have 4 GB of RAM and run RedHat Enterprise Linux 6 with kernel version 2.6.32-358. We use an OSCARS [17] 10GbE virtual circuit reservation over ESNet (~96 ms latency), when using this platform.

On both platforms, each node has an RNET 10GbE card (SmartNIC) installed in an x8 PCIe slot. We turn off CPU frequency scaling and processor c-state switching to allow for maximum host performance.

B. *Microbenchmark Results over the LAN*

In this section, we present results from benchmarking the UDT, RDMA and SSL offload engines using simple throughput microbenchmarks.

1) UDT Offload Engine: In the UDT microbenchmark, a one-way stream of messages is transferred from one node to the other one, and a final small acknowledgment is sent back. The card is set to coalesce interrupts in order to reduce the burden on the host, an approach that has a positive effect on throughput. Jumbo frames are used, and the user-level API chunks the data in 16 KB pieces. The UDT offload single-stream test can reach up to 9.8 Gbps.

We use the *mpstat* tool in Linux to report CPU utilization. Based on the measurements, offloaded UDT requires nearly 50% of a core power at the sender side and nearly 80% of a CPU core power at the receiver side to reach line-rate throughput. The CPU utilization for UDT shows that at the sender side, CPU is consumed mostly in system calls, which are due to the UDT driver handling send requests. At the receiver side, almost all the consumed CPU cycles are in soft IRQ (interrupt) processing for the arrived data. We have moved a significant portion of the host-side processing to the interrupt bottom half (soft IRQ), where the socket buffers containing arrived data are posted into work queues for user-space processing. Eliminating multiple sender-side and receiver-side copies at the host has helped reduce the overall CPU utilization. Most of the CPU utilizations observed are due to kernel-space socket buffer processing and polling activities to check for the availability of data; no UDT protocol processing occurs at the host side.

2) RDMA Engine: To measure RDMA throughput, we use RDMA Write (memory semantics) in a one-way stream of data, terminated by a final acknowledgment message from the receiver. The observed one-way streaming throughput is over 9.8 Gbps; and the CPU utilization is low, due to using offloaded processing and direct memory transfer into user-space buffers. The sender-side CPU utilization is ~11% of a CPU core, spent mostly in system calls to post data to the card. The receiver-side CPU utilization is negligible (less than 2% of a CPU core), since no CPU involvement is required in order to place the data in user space. The small CPU power consumed is to periodically check the arrival of data.

3) SSL Offload Engine: For benchmarking the SSL offload, we have used an emulated HTTPS server/client, where a one-way stream of HTTPS data is sent and finalized by a final response from the receiver to end the session. Using this benchmark, we observe 6.2 Gbps throughput in a single flow of OpenSSL-based data transfer (Fig. 2).

The main reason for not reaching line rate in full OpenSSL processing on the card is OCTEON's relatively low performance of MD5 hash processing. The MD5 engine performance is about 1.3 Gbps per core. If we skip the card-side MD5 calculations (and leave the final integrity check of the file data to the host), the card's capacity will increase, making the total single-stream observed throughput to reach 9.9 Gbps using offloaded encryption. With the next generation 4-port SmartNIC, which is based on a 32-core OCTEON II processor at 1.2 GHz, up to 27 Gbps of full OpenSSL throughput is expected.

Undoubtedly, the main benefit for SSL offload is reduced host CPU utilization. The offloaded SSL implementation consumes no more CPU cycles than does a UDT offload (essentially getting free security processing). This is due to the fact that SSL data essentially pass through the same data path that UDT data do, with little extra processing. We discuss OpenSSL CPU utilization in the next section, where we also present long-delay network results.

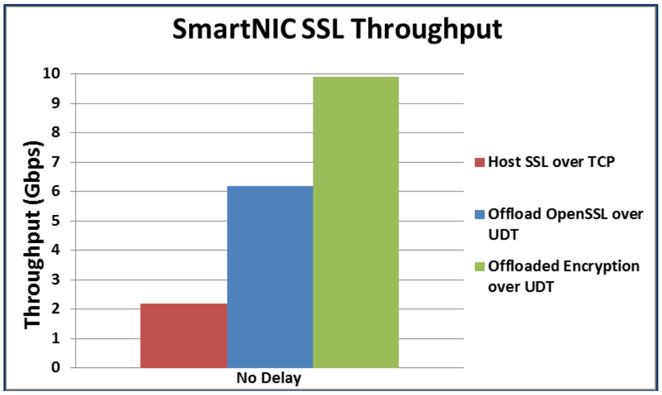


Fig. 2. SSL offload vs. host throughput

C. *GridFTP with Offload Support*

Here we present the results for benchmarking Globus GridFTP over SmartNIC offload engines in a local-area network setting. We run our benchmark with and without an injected network delay. We focus on GridFTP throughput and CPU utilization when using UDT and RDMA. OpenSSL integration with GridFTP is under way, for which in this section we present further microbenchmark results.

The TCP stack is tuned based on the TCP performance tuning recommendations by ESNet [32]. We also use 4 MB GridFTP block sizes, jumbo frames, and H-TCP congestion control (that shows a better throughput over long delays). For host-based UDT, 8 KB maximum segment sizes are used, similar to those for the offloaded UDT (this is different from upper layer buffer sizes of 4 MB used for the GridFTP software).

No-Delay Network Results: We first measured GridFTP memory-to-memory file transfer throughput over the local-area network with a ~500 µs RTT. The leftmost side of Fig. 3 shows the observed throughput for the local-area network with no injected delay. Except for host-based UDT, other transports are able to saturate the link. While host-based UDT takes close to a minute to reach its maximum throughput, offload UDT takes only about 4–5 seconds.

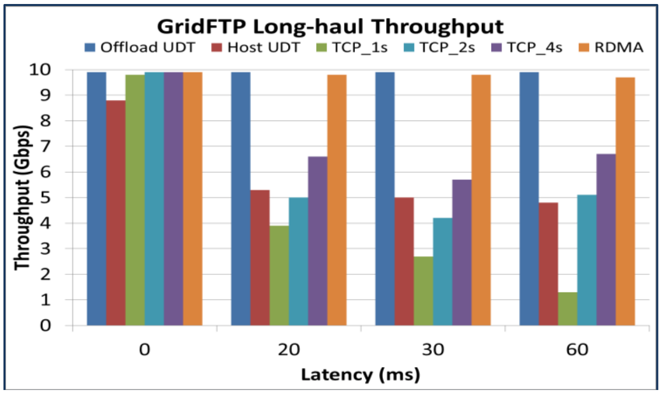


Fig. 3. GridFTP throughput over emulated long-delay network

Fig. 4 shows CPU utilization at sender (data source) and receiver (data sink) of a GridFTP file transfer session, when using any of the above transports. We measure the total host CPU utilization and calculate the average amount of one CPU core power that is utilized for 1 Gbps of throughput. As expected, RDMA transport shows the lowest CPU utilization for GridFTP, because of little CPU involvement in data transfer (especially at the receiver side). After RDMA, the UDT offload shows the next lowest CPU utilization at both sides. Clearly, host-side UDT has the highest CPU utilization. Regardless of how we bind the processes to CPU cores, one core is completely saturated in host UDT case; hence lower than line-rate bandwidth is saturated for host-side UDT.

Table 1 shows CPU usage of different processing operations for various transports (shown as percentage of their total CPU utilization). The numbers are presented in the form of sender-side/receiver-side values. The majority of offloaded UDT time is spent in system calls; negligible time is spent in user space. In contrast, the host UDT spends a large portion of its sender time in user space. At the receiver side, a significant part of offload UDT processing is spent in IRQ processing, which includes checking for data arrival and handling packet buffers. We plan to improve this portion by using kernel-assisted wait queues and reduced kernel-side polling. For TCP, most of the processing at both sides occurs in system calls and soft IRQ processing.

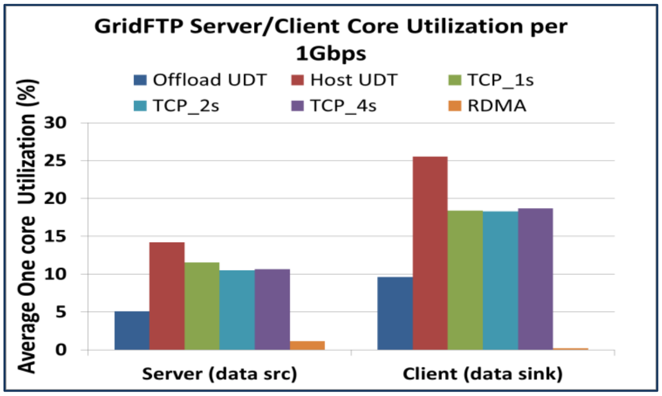


Fig. 4. GridFTP CPU utilization with different transports over RNET_P

TABLE 1 - CPU UTILIZATION FOR TCP AND UDT

GridFTP Transport	IRQ %	Syscall %	User space%
TCP (4 stream)	45 / 60	51 / 35	4 / 5
Host UDT	21 / 43	31 / 16	48 / 41
Offload UDT	16 / 71	81 / 26	3 / 3

Long-Delay Network Setup: One of the main expected benefits of using UDT as the underlying transport is its ability to sustain high throughput over long-haul networks, that is, wide-area networks with high latencies. To examine the GridFTP performance with various network latencies, we used an emulated long-delay network, where the latency is induced by a middle node, as depicted in Fig. 5. The middle node has a two-port OCTEON Plus 10GbE card and uses OCTEON's timers to schedule each arrived message at its ingress port for forwarding at the egress port at a requested later time, inducing an artificial delay. This method of injecting artificial delay also introduces a small amount of jitter, where less than 10% of the packets experience a delay variance higher than 20%. In addition, the setup also introduces a mild (less than 10%) packet reordering. These characteristics are reasonable for emulating a long-haul network, where mild jitter and reordering are present. Little to no packet drops are present in this experiment. We are planning to examine the effects of various packet drop scenarios using artificial packet drops in our future experiments.

In order to sustain 10 Gbps, the middle node needs to have buffering space for the bandwidth-delay product of the network. For example, for 10 ms latency, we need to have $10ms * 10Gbps = 100Mb$ of buffering. This includes buffering at the UDT protocol level, mainly the space for the arrived packet buffers, before being forwarded. Due to the elimination of card side copies for UDT processing, the only data-dependent buffering required is the amount of packet buffers held in buffers during the delay. For RDMA to be able to sustain this capability, beside UDT's internal buffering, we also need to have enough buffering at the XIO driver's SLAB interface for a message's acknowledgment to arrive from receiver to sender, before we discard the message at the sender side. For this purpose, we use a SLAB buffer with 2,048 partitions of up to 64 KB each.

Fig. 3 presents the GridFTP file transfer throughput over several transports using various injected delays. The delays are shown as round-trip latencies in milliseconds and the performance is the maximum reached (over a long window to ensure stability). Offloaded UDT and RDMA on top of it are the only transports that can offer a sustained near-line-rate throughput for GridFTP. On the other hand, both host-based TCP and UDT suffer significantly from long delays, and even multistream TCP is not able to compensate, despite improving TCP throughput. The other observation is that TCP-based transfers take minutes to reach their maximum throughput, while UDT offload takes less than five seconds.

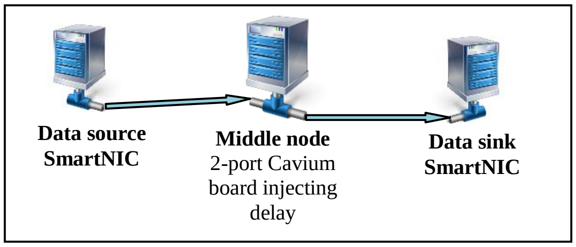
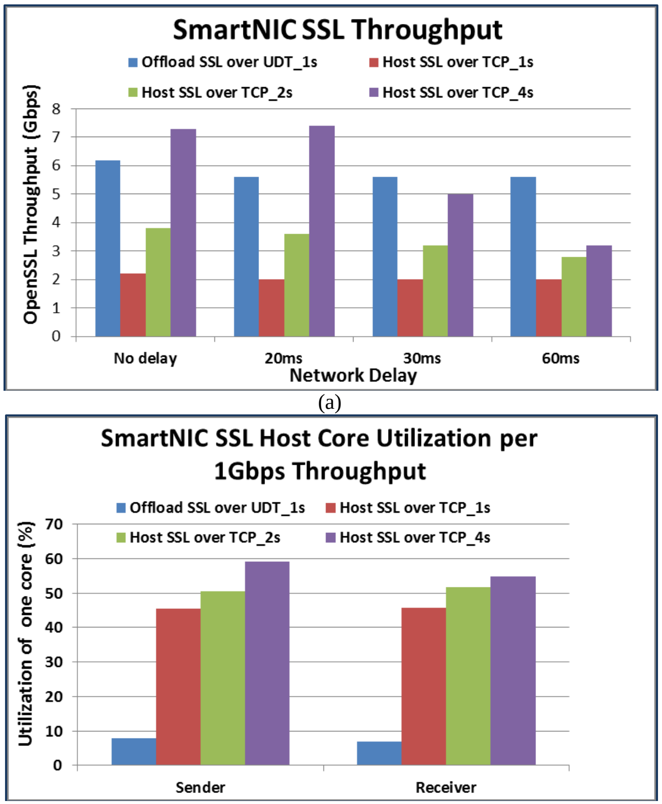


Fig. 5. Injecting long delays to emulate a long-haul network

OpenSSL Results over Long-Delay Network: We run the OpenSSL microbenchmark over the emulated long-delay network, with the aim of comparing throughput and CPU utilization for the single-stream offloaded OpenSSL with host-based OpenSSL over TCP when using multiple TCP streams. For the multistream TCP test, we use multiple independent process pairs and utilize MPI calls to synchronize among them. We run the test for multiple consecutive windows (transferring near 8 Gb of data in each window), until the throughput converges. For TCP we use 1 MB buffer sizes.

In Fig. 6(a) we present the throughput under various network delays. As one can see, multistream TCP-based SSL can do better than single-stream offloaded SSL over UDT in a no-delay case. The reason is that the processing power of the current-generation SmartNIC limits the throughput to about 6.2 Gbps of OpenSSL + UDT, as also shown in Fig. 2. As the network delay increases, however, while there is about 10% drop in SSL offload throughput (sustained regardless of the amount of network latency), the multistream TCP throughput drops more dramatically, falling by up to 50% to about 3 Gbps.

Fig. 6(b), which depicts normalized CPU utilization, clearly shows that using offloaded SSL can significantly benefit host CPU usage, decreasing it by 80% or higher. Using two SmartNIC boards, we can expect near twice the observed offload performance (over 11 Gbps) over a long-delay network while utilizing the power of less than 2 host cores (less than 20% of the host power). Conversely, for a host-based TCP solution to achieve 10 Gbps, the power of 5 to 6 host cores (~70% of the host power in our RNET_P platform) is required.



(a)

(b)

Fig. 6. SSL streaming performance over long-delay network: (a) throughput, (b) host core utilization per 1 Gbps throughput

D. GridFTP Results over ESNet Long-Haul Connection

To verify the benefits of using offloaded UDT-based file-transfer over real networks, we examined our implementation on a long-haul scientific grid connection over ESNet. ESNet is a high-performance grid network run by the U.S. Department of Energy, connecting hundreds of research institutions, facilitating their scientific collaboration.

As shown in Fig. 7, in this setup the UC_P platform nodes at the University of Chicago are connected through a loop-back long-range network that travels through ESNet to a switch residing at NERSC in Berkeley, California. The total round-trip latency is ~96 ms.

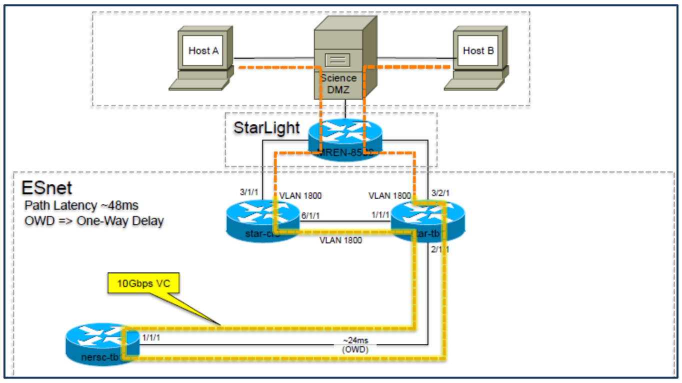
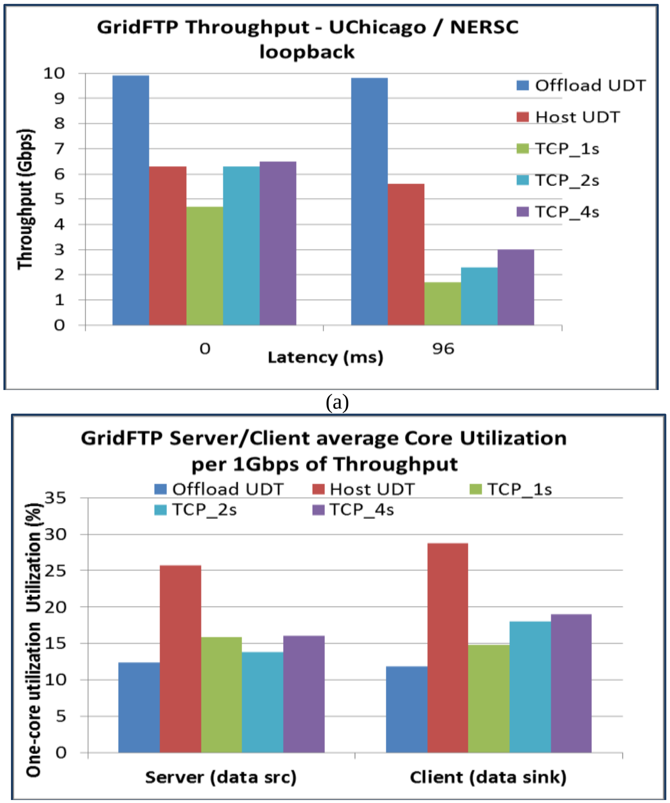


Fig. 7. Long-haul loopback network set-up at UChicago through ESNet



(a)

(b)

Fig. 8. GridFTP test results on UC_P: (a) maximum throughput on back-to-back connection and ESNet long-haul network, (b) CPU utilization

Fig. 8(a) shows the maximum observed throughput of GridFTP file transfer using offloaded UDT, compared with that of host-based TCP and UDT. The results are presented for both a back-to-back connection and the long-haul network. TCP performance tuning similar to those on the RNET_P platform were performed. We can see that only the offloaded UDT can nearly saturate the link under high latency. Our investigations show that the link introduces jitter and some mild reordering of packets, which contribute to TCP's poor performance. The amount of packet drops over this link is negligible.

Increasing UDT's internal bandwidth estimation window as well as increasing the next packet expected time improved its sustained throughput and helped UDT avoid fluctuations in the observed throughput. We note that such parameters should be adjusted based on network conditions (such as RTT and jitter), which could be a potential burden on the user.

Despite the tuning efforts, host-based UDT is unable to saturate the link in either case (no-delay and long delay). However, it still shows significantly less throughput drop compared to that of TCP, when going from no-delay (back-to-back connection) to the long-haul network. The reason both host-based UDT and TCP are not able to saturate the link even when the nodes are connected back-to-back is that the UC_P nodes are less capable than the RNET_P nodes and the interrupt processing fully saturates a CPU core. This situation results despite using interrupt coalescing, larger packet push-up from the card (4 KB instead of 1536 B), presence of the IRQ-balance tool (to spread IRQ processing), and jumbo frames.

Fig. 8(b) shows the CPU utilization of UDT and TCP on UC_P, in terms of the utilization of one CPU core per 1 Gbps of data transfer throughput. The measurements show twofold to threefold improvement in the CPU utilization when using the offloaded vs. host-based UDT.

To have a better insight into the transient behavior of these transport protocols on the long-haul connection, we also examined the observed instant file transfer throughput (as reported by GridFTP) over a 100-second window (Fig. 9). In this window of time, offloaded UDT is able to transfer over 113 GB of data, whereas the amount for host-based UDT and 4-stream TCP is only 59 GB and 24 GB, respectively. We also observe that GridFTP over offloaded UDT is able to sustain the maximum throughput over the long run. On the other hand, significant throughput variation is observed for TCP-based transfers.

The initial warm-up time for offloaded UDT, as observed in the beginning of the graph, is longer than expected, particularly longer than what we observe on the RNET_P platform, as reported in Section VI.C. The cause is most probably related to the UDT parameter settings, which we are currently investigating.

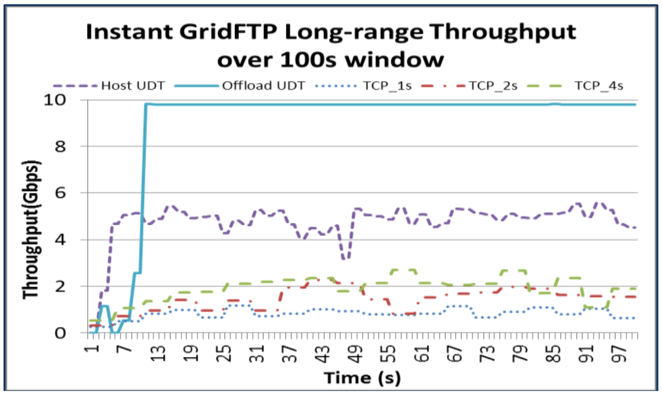


Fig. 9. Instant throughput of GridFTP long-range file transfer over a 100-second window of data transfer

VII. Conclusions and Future Work

In this paper we demonstrated how long-haul secure and regular file transfers can benefit from using an alternative transport protocol such as UDT that is offloaded to the network interface card. The benchmarks show how the UDT protocol offload can reduce the processing burden on DTNs, allowing for more communication capacity or data-processing capability. GridFTP over offloaded UDT can achieve over 60% reduction in host CPU usage, while near-line-rate throughput can be sustained over high-latency high-performance networks. Moreover, CPU usage can reach near-zero when utilizing RDMA over UDT. The results also show that high-throughput secure data transfer over long delays can be achieved by using SSL and UDT protocol offloading, while freeing the host processors for other purposes.

In addition to the above, many applications can now opt for secure data transfers, something significantly CPU-intensive when using a host-based approach on a DTN. The freed host resources can be utilized for various use-cases, including higher network capacity (using more network ports), data compression, and, more importantly, end-to-end file integrity check, which is a resource-intensive task. Moreover, the observed reduction in DTN processing requirements (up to 6 times for the same throughput in the case of SSL) can lead to significantly smaller or fewer DTNs.

In our next experiments we will examine the effect of packet loss on long haul data transfers using UDT and TCP. We also plan to complete the integration of offload engines, particularly SSL, with a production GridFTP release. Moreover, we plan to examine the potential usage of OpenSSL over RDMA that could further reduce the observed host utilization. We also plan to run our experiments over other scientific grids such as Internet2 or SCinet. To further examine the benefits of network offloading for increasing the capacity of a data transfer node, we plan to utilize multiple SmartNIC (and next-generation SmartNIC II) cards, and simultaneously perform other operations on the DTN, such as file integrity check or compression.

Acknowledgment

We thank the Argonne/University of Chicago Computation Institute, as well as Starlight Networks and ESNet, for providing the resources for long-haul network tests.

References

[1] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, “The Globus striped GridFTP framework and server,” in Proceedings of the SC05, page 54. IEEE Computer Society, 2005.

[2] W. Allcock, T.Perelmutov, “GridFTP v2 protocol description,” Global Grid Forum, May 2005.

[3] Energy Science Network (ESNet), Available: http://www.es.net.

[4] A. Hanushevsky, “Peer-To-Peer secure fast copy – bbcp,” SLAC, Available: http://www.slac.stanford.edu/~abh/bbcp/

[5] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda, “Performance characterization of a 10-Gigabit Ethernet TOE,” in Hot Interconnect (HOTI), August 2005.

[6] InfiniBand Trade Association, Available: http://infinibandta.org/.

[7] Longbow Infiniband Range Extenders. Obsidian Strategies, Available: http://www.obsidianresearch.com/products/longbow/index.html.

[8] Y. Gu and R. L. Grossman, “UDT: UDP-based data transfer for high-speed wide area networks,” Comput. Netw., 51(7):1777–1799, 2007.

[9] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, “A security architecture for computational grids,” in CCS ’98: Proceedings of the 5th ACM conference on Computer and communications security, pages 83–92, New York, NY, USA, 1998. ACM.

[10] W. Allcock, J. Bresnahan, R. Kettimuthu, and J. Link, “The Globus eXtensible Input/Output System (XIO): A protocol-independent I/O system for the grid,” in High- Performance Grid Computing and High-Level Parallel Programming Models in conjunction with IPDPS, 2005.

[11] R. Kettimuthu, J. Link, J. Bresnahan, and W. Allcock, “Globus data storage interface (DSI) – enabling easy access to grid datasets,” in First DIALOGUE Workshop: Applications-Driven Issues in Data Grids, 2005.

[12] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, “Modeling TCP throughput: a simple model and its empirical validation,” in ACM SIGCOMM, 1998.

[13] P. Prakash, A. Dixit, Y. Hu, and R. Kompella, “The TCP outcast problem: exposing unfairness in data center networks,” in Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2012.

[14] TeraGrid. Available: https://www.xsede.org/tg-archives.

[15] Internet2. Available: http://www.internet2.edu/.

[16] “Science DMZ: data transfer node,” Energy Science Network (ESNet), Available: http://fasterdata.es.net/science-dmz/DTN/

[17] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston, “Intra and interdomain circuit provisioning using the OSCARS reservation system,” in 3rd International Conference on Broadband Communications, Networks, and Systems, 2006.

[18] J. Postel and J. Reynolds, “File transfer protocol (FTP),” IETF, RFC 959, 1985.

[19] H. Shah, J. Pinkerton, R. Recio, P. Culley, “DDP - direct data placement over reliable transports,” IETF Network Working Group, RFC 5041, October 2007.

[20] S. Bailey, T. Talpey, “The architecture of direct data placement (DDP) and remote direct memory access (RDMA) on Internet protocols,” IETF Network Working Group, RFC 4296, December 2005.

[21] E. Kissel, M. Swany, “Evaluating high performance data transfer with RDMA-based protocols in wide-area networks,” IEEE HPCC-ICESS 2012, Liverpool, UK, June 25-27, 2012. IEEE Computer Society.

[22] R. Grant, M. Rashti, A. Afsahi, P. Balaji, “RDMA capable iWARP over datagrams,” in Proceedings of International Parallel & Distributed Processing Symposium (IPDPS), IEEE Computer Society, 2011.

[23] M. J. Rashti, R. E. Grant, P. Balaji, and A. Afsahi, “iWARP redefined: scalable connectionless communication over high-speed Ethernet,” 17th International Conference on High Performance Computing (HiPC 2010), Goa, India, December 19-22, 2010.

[24] J. Bresnahan, M. Link, R. Kettimuthu and I. Foster, “UDT as an alternative transport protocol for GridFTP,” in Proceedings of the 7th PFLDNeT Workshop, Tokyo, Japan, May 2009.

[25] RNET’s User Programmable 10Gigabit Ethernet SmartNIC. Available: http://www.rnet-tech.com/us/products/2-specialities/2-smartnic.

[26] Globus GridFTP. Available: http://toolkit.globus.org/toolkit/docs/latest-stable/gridftp/.

[27] Cavium CN57XX Octeon Plus Network Processor. Available: http://www.cavium.com/OCTEON-Plus_CN57XX.html.

[28] Infiniband Trade Association, “RDMA over converged Ethernet,” Press release, 2010. Available:

http://www.infinibandta.org/content/pages.php?pg=press_room_item&rec_id=663

[29] Y. Gu, “UDT: breaking the data transfer bottleneck,” Available: http://udt.sourceforge.net/.

[30] OpenFabrics Alliance., “OpenFabrics enterprise distribution (OFED),” Available: http://www.opernfabrics.org.

[31] B. Metzler, F. Neezer, P. Frey, “A software iWARP driver for OpenFabrics,” OpenFabrics Spring SONOMA Workshop, 2009.

[32] Tips for tuning Linux & TCP for high performance long-haul networks. ESNet, Available: http://fasterdata.es.net/host-tuning/linux/.

[33] D. Gunter, R. Kettimuthu, E. Kissel, M. Swany and J. Zurawski, "Exploiting network parallelism for improving data transfer performance," IEEE/ACM Annual SuperComputing Conference (SC12) Companion Volume, Nov. 2012.

[34] H. Subramoni, P. Lai, R. Kettimuthu, D.K. Panda, "High performance data transfer in grid environment using GridFTP over InfiniBand," Proceedings of the 10th IEEE/ACM CCGrid Symposium, May 2010.

[35] G. Vardoyan, R. Kettimuthu, S. Tuecke and M. Link, "Characterizing throughput bottlenecks for secure GridFTP transfers," International Conference on Computing, Networking and Communications, Internet Services and Applications Symposium, Jan. 2013.

[36] S Floyd, "HighSpeed TCP for large congestion windows," IETF RFC 3649, 2003.

[37] D. Leith, R. Shorten, "H-TCP: TCP congestion control for high bandwidth-delay product paths," IETF Draft, draft-leith-tcp-htcp-06.txt, 2008.

[38] RDMA Consortium, "Architectural specifications for RDMA over TCP/IP," Available: http://www.rdmaconsortium.org/

[39] Globus Transfer Service, Available: https://www.globus.org/file-transfer

[40] Globus GridFTP-Lite, Available: http://toolkit.globus.org/toolkit/data/gridftp/quickstart.html.

[41] "OCTEON Programmer's Guide, The Fundamentals", Cavium Networks, 2010.