Evaluating High Performance Data Transfer with RDMA-based Protocols in Wide-Area Networks

Ezra Kissel*, Martin Swany†
* School of Computer and Information Sciences, University of Delaware, Newark, DE 19716
† School of Informatics and Computing, Indiana University, Bloomington, IN 47405

***Abstract*—**
The use of zero-copy RDMA is a promising area of development in support of high-performance data movement over wide-area networks. In particular, the emerging RDMA over Converged Ethernet (RoCE) standard enables the InfiniBand transport for use over existing and widely deployed network infrastructure. In this paper, we evaluate the use of RDMA over Ethernet in two deployment scenarios: 1) a *gateway* approach that adapts standard application connections to an RDMA-based protocol for transmission over wide-area network paths, and 2) the integration of our RDMA implementation into GridFTP, a popular data transfer tool for distributed computing. We evaluate both approaches over a number of wide-area network conditions emulated using a commercial network emulation device, and we analyze the overhead of our RDMA implementations from a systems perspective. Our results show a significant increase in network utilization and performance when using RDMA over high-latency paths with a reduced CPU and memory I/O footprint on our gateways and end host applications.

I. Introduction

High performance data movement over wide-area networks (WANs) is a key issue in distributed computing. Current network technologies and protocols are increasingly ineffective in the face of ever-growing data movement demands. This affects key domains from data intensive science to electronic commerce, and compute grids to commercial computing clouds. Data movement time is critical to performance in all of these environments.

There are a plethora of solutions, software and hardware, with different interconnects, APIs and protocols. No solution is a panacea as all have issues with overhead, infrastructure support, or ease of adoption. The emerging RDMA[1] over Converged Ethernet (RoCE) [5] standard lets users take advantage of low-latency, efficient, high-performance protocols that are commonly used in the data center, like InfiniBand (IB). RoCE is basically the InfiniBand protocols made to work over Ethernet infrastructure.

The notion of "converged Ethernet", also known as enhanced or data center Ethernet, is that of including various extensions to the IEEE 802.1 standards to provide prioritization, flow control and congestion notification at the link layer. Since the IB protocols operate in networks that are virtually loss-free, the motivation for this is clear. The protocol, however, does not directly require any of these extensions and thus it is possible to use RoCE in WAN environments. However, we do require certain guarantees about path characteristics. The path should be virtually loss-free and should have deterministic and enforced bandwidth guarantees. Virtual circuits, and more recently, building dedicated paths with Software Defined Networks (SDNs) give us that capability.

Shared network infrastructure presents other challenges for enabling RDMA-based transfers. It may also not be feasible to have all applications be RDMA-capable, in particular due to the asynchronous nature of the API. We show that extensions to our Phoebus Gateway [27] system allows legacy applications to take advantage of RDMA over the WAN using protocol adaptation at the edge of the site. This adaptation can be done either transparently or explicitly when signaled by the application.

Applications that support RDMA directly are also viable candidates for long distance RoCE transfers when virtual circuits can be extended to end hosts. As the dominant data transfer tool in grid environments, we target GridFTP [18] as our candidate application. GridFTP is built on the Globus XIO framework [23], which provides extensibility by enabling modular "drivers" that can take advantage of different underlying protocols. We develop and evaluate an RDMA-capable driver for GridFTP that requires no modifications to the existing XIO framework.

Evaluating the performance of RDMA protocols over the WAN remains a challenge in today's networks. Current production infrastructure is often not capable of supporting RDMA transfers and many research testbeds have practical restrictions that limit the set of tunable path parameters. Contention for available resources and the repeatability of experiments are other factors that must be considered. The evaluation in this paper makes use of a dedicated network emulation device, Spirent XGEM [1], to emulate a wide range of WAN conditions. This gives us a high degree of confidence that our designs will operate in comparable conditions on real-world networks and provides a solid basis with which to evaluate the effectiveness of RoCE in WAN settings.

The following summarizes the contributions made in this paper: 1) We describe a session-based *gateway* approach that allows legacy applications to transparently use RDMA over WANs, 2) We develop a novel approach for application data transfer called Bulk Asynchronous Get, 3) We develop an RDMA-enabled GridFTP driver that supports edge hosts that have capable NICs, 4) We evaluate RoCE performance over various WAN conditions using a dedicated network simulation device, and 5) We perform a system-level analysis of RDMA as used by our implementations.

[1]Remote Direct Memory Access.

II. Motivation

Achieving reliable, high-speed data transfer performance in the wide area remains a "holy grail" for many in the research and education (R&E) and e-Science communities, and it is increasingly important for the commercial sector as well. While available link and backbone capacity have rapidly increased, the achievable throughput for typical end-to-end applications has failed to increase commensurately. In many cases, application throughput may be significantly less than what is theoretically achievable unless a considerable amount of effort is spent on host, application, and network "tuning" by users and network administrators alike. The growing WAN acceleration industry underscores this need.

We recognize that TCP is ubiquitous and is used as the default transport protocol in many existing transfer applications. Practical deployment and adoption concerns have limited the ability of new protocol and architectural designs to effect significant changes in response to specific application demands. While frameworks like Globus allow users to experiment with new approaches implemented as loadable drivers, most legacy applications are left to deal with the limitations of prevailing protocol support and their deficiencies in providing adequate performance over high-latency, high-speed networks.

As networks continue to get faster, another key performance consideration is not only the efficiency of the underlying transport protocols but also the level of operating system (OS) involvement in supporting data movement between end hosts. With 10Gb/s network interfaces becoming more common in high-performance computing environments, applications struggle to achieve expected throughput without significantly taxing CPU resources, or at least some significant fraction of available cores. Although there are a number of optimizations available, from kernel socket splicing to TCP offload engines, this situation is not sustainable as data sets grow larger and the demand on networks increases. Of course, this has been the case previously, with each order of magnitude increase in network speed. The end of processor frequency scaling means that we cannot simply wait for processors to get faster and solve the I/O performance issue. New approaches are needed.

Architectures that enable high-performance data movement, including our own work, frequently depend on the ability of the network to provide dedicated, stable, and often highly configurable network resources. In support of scientific computing and R&E communities, one approach to managing a specific class of backbone network resources has resulted in Software Defined Networks, or SDNs. These SDNs allow network resources to be provisioned "on the fly" by users, services and advanced applications.

Demand-driven allocation of these ephemeral, dedicated links and paths enables optimization of network utilization and is an ideal tool for demanding network applications. These "circuit networks" currently support high performance and Grid computing applications that must reliably and quickly move large quantities of data, and there are currently a number of existing SDN control plane technologies in general use [17], [31]. In addition, a considerable amount of effort is now being focused on the problem of local, or end-site network configuration. In many cases this involves extending a circuit provisioned via existing SDN systems into the local-area network of a particular institution where it can provide a dedicated, end-to-end virtual path for the requesting application or user. Efforts such as PWE3 [8] are exploring similar edge-to-edge capabilities over label-switched paths. Our approach takes advantage of these efforts to configure the network in support of RDMA transfers, both across the end-sites and the WAN.

III. Background

This paper draws upon our previous work in supporting high performance networking through a session layer protocol implementation, a performance enhancing gateway, and a buffer-and-burst model for bulk data movement. We now provide an overview of these components that have been extended and used in our evaluation of RDMA over wide-area networks.

A. eXtensible Session Protocol (XSP)

The eXtensible Session Protocol (XSP) [25] was designed as a flexible protocol architecture for managing the interactions of applications and network-based services, and among the devices that provide those services. Residing in layer-5 of the OSI network model [13], XSP can provide both control and data encapsulation of the underlying transport layer protocol data units (TPDUs) into Session layer PDUs (SPDUs). Through an abstraction known as the *Protocol Channel* service handler, XSP allows us to implement and use interchangeable transports among applications and services that speak our session protocol implementation.

Additionally, XSP provides mechanisms to manage the configuration of dynamic networks services with the ability to perform both in-band and out-of-band signaling of metadata between session entities. Although outside the scope of this paper, these features allow us to configure on-demand circuits on behalf of applications and bind the state of the network, and associated transport connections, to a particular session. A session, in our model, is generically "a period of a particular activity." In this work, we are interested in using XSP to manage periods of activity devoted to bulk data movement and its use in our network-based services to optimize for high-performance transfers.

B. Phoebus

Our previous work on Phoebus [27] has been a direct response to many of the performance issues described above, particularly when bulk data movement is concerned. Phoebus is a middleware system that applies our XSP session layer, along with associated forwarding infrastructure, for improving throughput in today's networks. Using XSP, Phoebus is able to explicitly mitigate the heterogeneity in network environments by breaking an end-to-end connection into a series of connections, each spanning a different network segment. In this model, Phoebus Gateways (PGs) located at strategic locations in the network take responsibility for forwarding users' data to the next PG in the path, and eventually to the destination host. A Phoebus network "inlay" of intelligent gateways allows data transfers to be adapted at application run time, based on available network resources and conditions.

In order to adapt between protocols along different network segments, Phoebus uses the *Protocol Channel* handler available in XSP to implement a number of transfer backends. These backends can then be used interchangeably via the shared XSP API while the underlying XSP framework handles any differences in protocol semantics. The existing Phoebus implementation has developed and experimented with a number of protocol backends, including TCP, UDP, and Myrinet Express (MX), as well as user space protocol implementations such as UDT [19].

C. Session Layer Burst Switching (SLaBS)

Building upon our experiences with Phoebus, we developed a modular extension to the Phoebus architecture called Session Layer Burst Switching, or SLaBS [24]. SLaBS uses the XSP session layer to enable an intelligent store-and-forward data movement service that takes advantage of large buffers at PG adaptation points to form data bursts and optimizes their transmission over dedicated network resources. Appropriately enough, we call these bursts "slabs" and the process of forming slabs "slabbing". In essence, slabs are SPDUs that are formed by coalescing smaller SPDUs from the edge (e.g. user application flows) that arrive at PGs. Incoming SPDUs are multiplexed, or reframed, into larger SPDUs (slabs) which are more suitable for efficient transmission over wide-area networks using the protocol adaptations available with Phoebus. The XSP session layer provides the mechanism for exchanging slab information over a SLaBS control session and the multiplexing/demultiplexing of SPDUs between PGs running the SLaBS implementation. Additional details are available in our previous work [24], [27].

Our SLaBS model draws inspiration from data center environments where switched-fabric interconnects defined by the InfiniBand Architecture (IBA) and related RDMA protocols have played a significant role in enabling massive parallelization with improvements to throughput while also reducing latency and overhead. In particular, RDMA operates on the principle of transferring data directly from the user-defined memory regions of one system to another, across a network, while bypassing the operating system and eliminating the need to copy data between user and kernel memory space. These direct memory operations are supported by enabling network adapters to register, or "pin", memory preventing that memory from being paged out, and giving it a consistent virtual to physical mapping. RDMA can then directly access these explicitly allocated regions without interrupting the host operating system. In this paper, we investigate this direct memory transfer approach as it applies to efficiently moving slab buffers over long-distance networks.

Finally, our evaluation depends on the availability of RoCE network adapters (rNICs) that support RDMA over existing Ethernet infrastructure. This is an area of active development with a proposed standard [5] and implementations working within currently available hardware.

IV. Implementation

This section describes our RDMA implementations in the Phoebus Gateway and as a modular driver for GridFTP.

A. Bulk Asynchronous Get

Up to now, our forwarding routines within Phoebus and SLaBS have used a synchronous, *"send when available"* approach. We now describe an extension to SLaBS that takes advantage of an asynchronous, *"get when ready"* model for the transfer of buffered data (i.e. slab SPDUs) supported by XSP signaling.

The benefits of an asynchronous communication pattern include the ability to decouple the data movement over the network from the involvement of the operating system (OS) itself. In this manner, sending a resource, whether it be a set of files or data already within the page cache, involves letting the OS simply stage the buffered data in memory on behalf of the requesting application while allowing the remote host to asynchronously "get" the prepared memory regions via some transport mechanism. We call this particular type of transfer scenario Bulk Asynchronous Get, or BAG.

The BAG approach entails having a producer make some resources available for a remote consumer to access, during some particular window of activity, and notifying the consumer when the data is ready. The BAG operation is asynchronous in that the producer is only notified of a transfer completion if requested or required by the implementation, and typically via an out-of-band message. Our notion of an XSP session allows us to define such a period of activity as the duration of an active session over which control signaling can be performed. This period involves the exchange of metadata about active memory regions and supports the notification of both sender and consumer of completion status for a given transfer event.

What our BAG model requires is a transport mechanism that removes the sender (i.e. the host operating system) from the task of moving application buffers through the network. Along with an out-of-band signaling mechanism provided by XSP, this asynchronous communication model frees up a system to perform other tasks. The semantics used in RDMA protocols provides exactly this mechanism.

We have developed a conceptual model of a BAG service that uses an RDMA transport and have implemented the necessary components within the XSP session layer to support our evaluation. This has involved two main tasks: 1) creating an RDMA *Protocol Channel* for use by the Phoebus forwarding routines, and 2) extending the related messaging support in XSP that allows Phoebus to exchange the necessary metadata, perform the remote *Get* operations, and signal transfer completion events. The RDMA protocol handler developed for Phoebus uses the *rdmacm* and *ibverbs* libraries, available as part of the OpenFabrics Enterprise Distribution (OFED) [3], to establish an RDMA transport context and initiate the supported RDMA operations. Phoebus can use this new protocol implementation interchangeably along with the other transfer backends described previously.

Using the reliable connection (RC) InfiniBand transport, the *RDMA READ* operation defined in the IBA specification allows for consumer-initiated transfer of remote memory regions. We use this mechanism to provide the *"get when ready"* functionality described in our BAG model. To support these new memory semantics, we extended the XSP SLaBS control

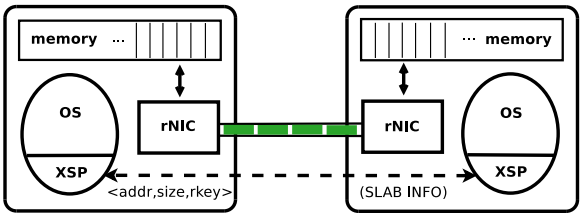


Fig. 1: System-level view of RDMA transfers with XSP-driven Bulk Asynchronous Get (XSP-BAG)

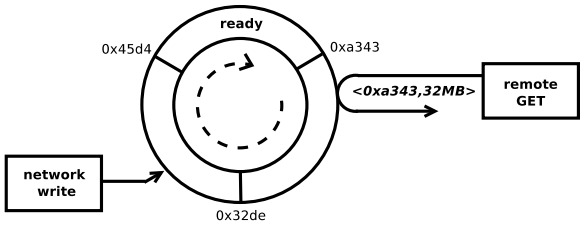


Fig. 2: Partitioned buffer design for SLaBS

PDUs to encode and exchange the necessary local and remote addresses, keys, and size of each memory region to transfer along with the original slab metadata. We call these control PDUs "SLAB" messages.

Figure 1 shows a system-level view of XSP providing the necessary signaling to maintain a BAG transfer. After the XSP session establishes the RDMA context, registers local and remote buffers, and exchanges pointers, the rNICs proceed to transfer data within the designated memory regions without further involvement from the OS. What is missing from this picture is the service that "stages" requested data in memory to be transferred. We now describe our SLaBS data movement system as one such in-the-network service.

B. RDMA for Phoebus and SLaBS

Our first implementation of the BAG approach extends SLaBS with the ability to use RDMA in order to more efficiently transfer slabs across dedicated network paths. Here, the memory regions to *Get* are the slab SPDUs being buffered at the SLaBS gateway. With the RDMA protocol handler available in Phoebus supported by XSP-BAG signaling in place, the remaining challenge involved extending the threaded buffer model within SLaBS to support efficient BAG transfers over high-latency WAN paths.

As network latency increases, it is well understood that pipelining the transmission of network buffers is required in order to continually keep data "in-flight" within the network. TCP solves this with a sliding window protocol clocked to the round-trip time (RTT) of the path, the effects of which become exaggerated over so-called "long fat networks" leading to considerable performance issues. In contrast, SLaBS maintains an open loop model over the dedicated core network paths which allows us to determine ahead of time what resources are necessary and to pace slab transfers based on buffer and throughput capabilities at various points in the network. This amounts to ensuring that the buffering implementation has at least one bandwidth-delay product[2] (BDP) sized buffer in transit at any given time. Thus, we use this value as a lower bound in the determination of slab formation size; in other words, the amount of buffered SPDUs that will generate the next XSP SLAB message and subsequent slab transmission to the destination PG.

Our SLaBS buffer implementation, illustrated in Figure 2, uses an adjustable size ring buffer with configurable memory region partitions within the overall buffer. To simplify the amount of metadata exchange required with XSP, we employ a simple "triple buffering" scheme where three logical memory regions are exchanged between SLaBS gateways in a round-robin fashion to keep the network saturated. While incoming SPDUs are written to one region ("network write" from the edge connections), the second region contains "ready" slabs and their associated metadata has already been sent to the remote side within an XSP SLAB message. The destination PG posts the *Get* operations as they are received over the XSP control session while slabs in a a third region are actively being retrieved via RDMA. In order to handle the multiplexing of multiple connections into slabs at an ingress PG, each of these regions in the ring buffer is further partitioned into smaller regions as more connections are added. In this case, each formed slab contains multiple memory regions that are exchanged via SLAB messages and subsequent *Get* operations between PGs.

Figure 3 illustrates a typical SLaBS protocol exchange between a source and destination PG. In this example, PG A is slabbing 4 incoming connections into memory regions *bufA* through *bufD*. Once enough data has buffered to form a full slab, PG A sends an XSP SLAB message, identified by SLAB sequence number 0, which contains the metadata necessary for PG B to *RDMA READ* the indicated memory regions and demultiplex the slab to the corresponding egress transport connections. Upon receiving the SLAB message, PG B immediately posts the *Get* call to the *ibverbs* library, which invokes the RDMA operations to begin transferring each buffer. As the RDMA transfer continues, PG A has formed two more slabs and has sent the associated SLAB1 and SLAB2 messages to PG B over the XSP control session. PG B posts additional *Get* calls while the SLAB0 regions are still being completed, queuing the next batch of RDMA operations. When there is continuous offered load at PG A, this pipelining is necessary to ensure that the network remains fully utilized. If PG A runs out of available buffers, it must wait for an acknowledgement from PG B that a previous slab buffer has been successfully received, e.g. ACK SLAB0, before it can begin to reuse the "in-flight" memory regions to buffer more data. In this case, the 4th slab sent, SLAB3, uses the same memory regions on PG A as our initial SLAB0.

We note that the ability to send BAG information ahead of the slab is an important feature in the SLaBS model. Connections to the ingress PG might stall, eliminating them from further slab bursts, and not all connections will be multiplexed within every slab. SLaBS may also resize or reallocate memory regions depending on the number of connections and available

[2]The bandwidth delay product of a network path is typically calculated as $BDP = RTT_{seconds} * Rate_{Bps}$

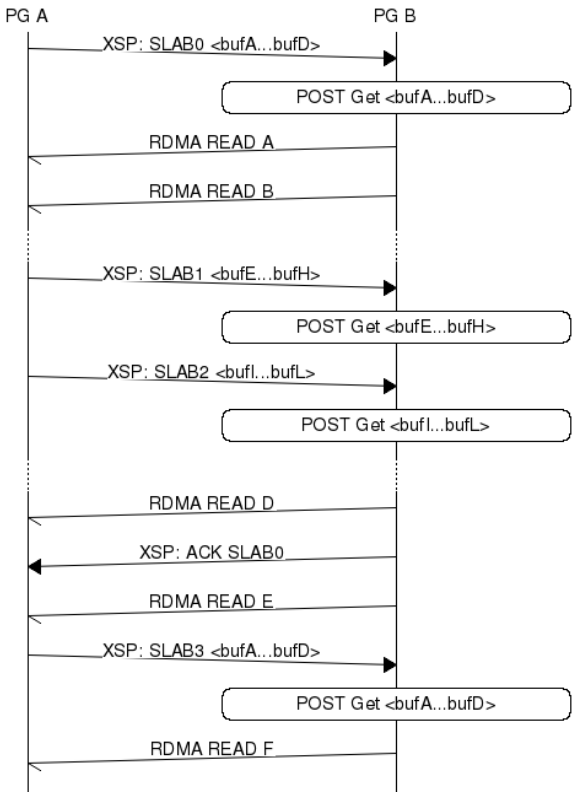


Fig. 3: SLaBS RDMA communication between two PGs

memory on the system, requiring new memory registrations for subsequent RDMA operations. Simply exchanging metadata once per connection is not sufficient and we rely on our active XSP session channel between PGs to handle this dynamic exchange.

C. An RDMA driver for GridFTP

The well-known file transfer application GridFTP [18] within the Globus Toolkit [16] uses an extensible input/output library called XIO [23]. By utilizing the concept of a driver stack, various modular drivers may be implemented independently and loaded dynamically at run time. This modularization facilitates re-usability for applications developed with the toolkit. Within the XIO framework, we have developed an RDMA transport driver, XIO-RDMA, that gives GridFTP the ability to work over both native IB and RoCE fabrics.

XIO provides an interface to standard *open*, *close*, *read*, and *write* calls as well as *accept* and *server init* hooks via pass-through and callback methods along the driver stacks. We use our SLaBS buffer model developed for BAG in Section IV-B to effectively form slabs within the XIO library by intercepting application data passed via *read* and *write* calls. Fully formed slabs are transferred over the network using an RDMA context established when the GridFTP servers initialize a new transfer. Since we are no longer multiplexing multiple connections, our session control channel for XIO-RDMA has reduced complexity compared to BAG, and we can perform a single metadata exchange to communicate memory region pointers at initialization. The client and server XIO-RDMA instances keep track of SLAB sequence numbers and completion events in order to pipeline RDMA operations over the network.

While other efforts such as RXIO [34] have optimized the XIO framework for RDMA-style communications, our work was primarily concerned with adding RDMA support within the existing XIO code in order to evaluate RoCE over WAN environments using a real-world application. As a result, while our XIO-RDMA driver incurs additional overhead within the XIO framework, we are able to provide WAN performance that exceeds previously published results. We envision that our SLaBS buffer optimizations for high-latency paths can be applied to extensions like RXIO to provide the best of both worlds.

V. Performance Evaluation

This section presents our experimental results as we evaluated the performance of our RDMA implementations. Our goal was to determine application speedup when using RoCE, either adapted at PGs with SLaBS or directly using XIO-RDMA, compared to end-to-end TCP. We also investigated performance issues over bottleneck conditions. We note that our evaluation makes use of the *netem* [2] Linux module to emulate "edge" latencies in certain cases. We have previously validated the accuracy of this approach in [26].

Results were collected from a testbed environment consisting of 4 nodes separated by an XGEM device used to emulate real-world network conditions, shown in Figure 4. The testbed formed a linear network topology with client and server nodes at the edges (i.e. LAN segments) and PGs on each end of a WAN segment emulated by the XGEM. Each of the 4 nodes contained dual-port 10Gb/s Mellanox ConnectX-2 VPI adapters set to Ethernet link-layer mode. The PG systems were additionally outfitted with two Mellanox ConnectX-2 dual-port Ethernet NICs. All have native RoCE support. Each of the 4 systems were compute nodes with 8-core Intel Xeon CPUs and 189GB of RAM. Our system profiling and PG data channel results were collected in a separate *netem*-only testbed on two dedicated PG system with AMD Phenom II X4 processors and 8GB of RAM. Each system ran a 2.6.32 kernel with standard Linux host tuning applied, including network interface and driver settings optimized for 10Gb/s speeds. CUBIC [20] was the default congestion control algorithm used on our systems. RDMA drivers and libraries were provided by the most recent OFED, version 1.5.4, installed on each system.

All of our transfer tests involved memory-to-memory copies to avoid disk I/O bottlenecks. Unless otherwise noted, network traffic was generated using GridFTP with servers running on each system. A client host on the management network initiated third-party transfers between source and destination servers.

The XGEM was used to emulate a number of network conditions that included variations over latency, loss, jitter, and bandwidth control. For the experiments involving PGs, *netem* was used to emulate LAN latencies on the end hosts; however, adding loss directly on source or destination hosts is known to cause poor interactions with the host's TCP stack [2]. To accommodate this constraint, we used the XGEM to verify end host to PG TCP transfer performance with the indicated amount of loss on the emulated LAN segments.

Fig. 4: 10G testbed with XGEM network emulator

A. Phoebus with RDMA

Our first experiment was designed to test the efficiency of Phoebus data channels in transferring slabs between two PGs. The *iperf* [21] network benchmark was used to initiate TCP connections to the ingress PG where they were buffered and multiplexed using SLaBS and then burst to the egress PG using the configured data channel, namely TCP, UDP, and RDMA. Simply buffering connections at the ingress PG allows the TCP connection to transfer at speeds near line rate.

Figure 5 shows that at 10Gb/s, the SLaBS PGs are not able to successfully form slab SPDUs and simultaneously burst buffered slabs over either TCP or UDP data channels near the available network capacity. TCP data channel performance reaches 8.1Gb/s while UDP approaches 8.5Gb/s. This limitation is primarily due to the available memory bandwidth on the PG hosts, and increasing the number of additional incoming streams does not significantly affect the buffering or backend performance. With the RDMA data channel enabled, the resulting reduction in system overhead allows the slab bursting performance to approach the rate of simply writing SPDUs into the slab buffer from the edge connections. In fact, the RDMA data channel transmits at the maximum bandwidth achievable by the rNIC using RoCE, approximately 9.7Gb/s. We note that this number is lower than the 9.9Gb/s "goodput" achievable with TCP over a 10Gb/s path due to a Maximum Transmission Unit (MTU) of 2048 bytes as defined in the current RoCE specification. This increases the protocol header and control overhead compared to the MTU of 9000 bytes configured for the non-RoCE tests.

Our second set of experiments evaluated GridFTP performance over a variety of network conditions, comparing the end-to-end TCP case with RDMA over the WAN via Phoebus. Each data point in the Figure 6 charts is the average throughput achieved by the *globus-url-copy* GridFTP client over multiple transfers with a duration of 60 seconds. GridFTP includes support for striping application data over parallel streams and we compare single stream TCP performance with 8 parallel streams in both the direct and Phoebus cases. To emulate typical end-site to regional network path conditions, a total of 4ms (2ms on each edge) of latency was added along the end host to PG links in addition to small amounts of loss that reflects contention over shared access networks. We also added 0.2ms of jitter to our WAN latency tests with a variation delta of 0.05ms to emulate minor packet delay variation commonly observed over backbone paths, e.g. due to buffering in routers.

As can be seen in Figure 6a, both single and parallel stream TCP performance is on par with the Phoebus case at low latencies and when there is no loss along the path. GridFTP performance is approximately 9.3Gb/s for both direct TCP and Phoebus cases. As WAN latency increases, the average single stream TCP performance begins to drop as its congestion window takes increasingly longer to grow, and at

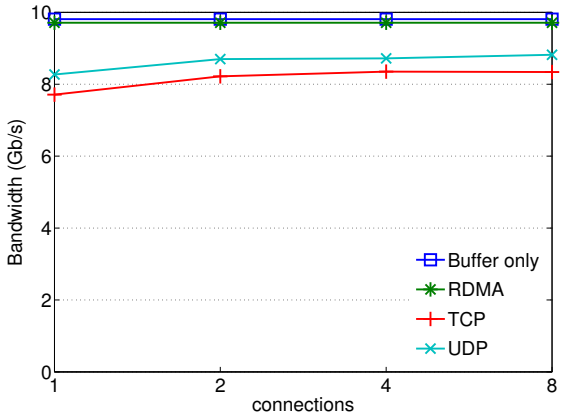


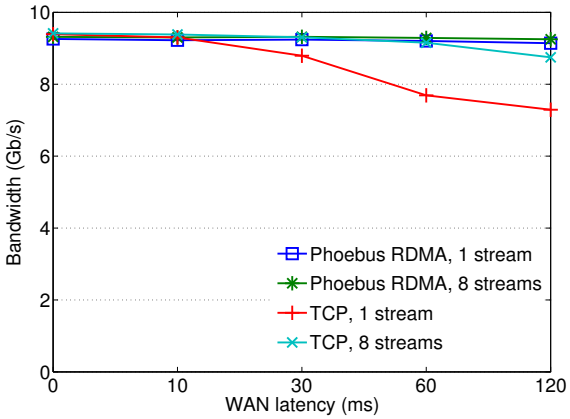
Fig. 5: Phoebus SLaBS data channel performance

120ms, parallel TCP performance also declines. In contrast, the Phoebus RDMA case remains consistent across all WAN latencies. At 120ms, single stream Phoebus provides a modest 5% improvement over direct TCP parallel streams and 20% over a single stream.

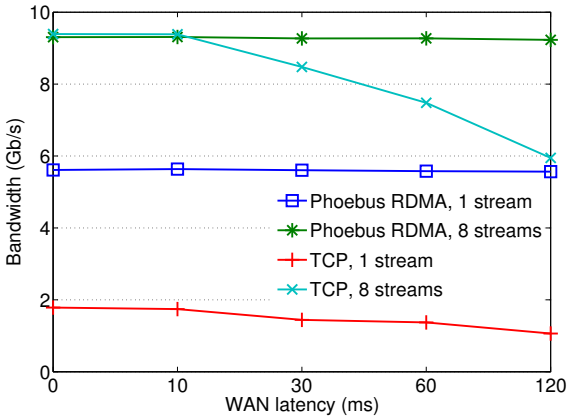
The situation changes dramatically when loss is introduced. Figure 6b and Figure 6c show the results of the above experiment run with 0.001% and 0.01% loss added, respectively. Using parallel TCP streams mitigates the effects of loss and increasing latency, as expected, but single stream TCP performance suffers considerably. In the 0.001% loss case, single stream Phoebus is able to consistently outperform the direct single stream TCP case by over 300%, and approaches parallel stream performance at 120ms WAN latency. With 0.01% loss, single stream Phoebus begins to improve on parallel TCP at approximately 50ms. In both loss cases, parallel streams over Phoebus perform nearly the same as when there is no loss, and we see over a 6-fold improvement in the highest latency and loss case.

We note that Phoebus RDMA performance remains very consistent across all WAN latency values. This is a direct benefit of using RDMA over the WAN segment where maximum bandwidth can be achieved almost instantaneously compared with sliding window protocols such as TCP. The limitation for Phoebus in the loss cases is the ability for TCP edge connections to achieve good performance to and from the PGs that segment the end-to-end path. Assuming a 2ms LAN segment to the ingress PG as used in our experiments, parallel TCP has little trouble achieving close to 10Gb/s throughput even in the higher loss case. As the data is buffered at the PG, our results show how the Phoebus RDMA data channel can maintain slab bursts over the high latency WAN segment at consistently high rates.

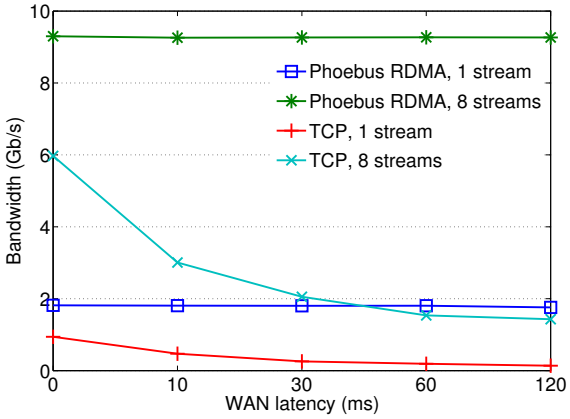
Lastly, we illustrate the importance of choosing a correctly sized SLaBS buffer to adequately maintain full utilization of the WAN path as latency increases. The buffer sizes indicated in Figure 7 represent the total allocated ring buffer within the SLaBS gateway. For an appropriately sized buffer for the given WAN latency, 1/3 of the total buffer partitions will be in-flight via a remote *Get* at any given time. As the number of buffers to *Get* approaches the BDP of the WAN path, performance of the RDMA data channel begins to decrease as the ingress



(a) No added loss



(b) 0.001% loss



(c) 0.01% loss

Fig. 6: Performance of Phoebus RDMA over increasing WAN latency, 4 ms LAN latency, and varying loss rates

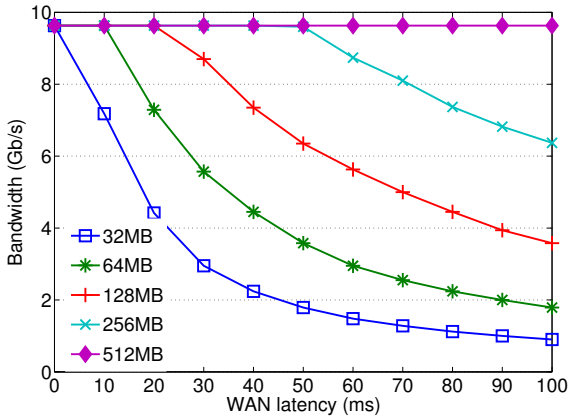


Fig. 7: The effect of latency on SLaBS buffer size

PG has to wait for the next slab record to pipeline, effectively "wasting" the available network capacity and increasing the signaling to data transfer ratio. Using a 512MB buffer, SlaBS with BAG is able to achieve full WAN network utilization well beyond 100ms, covering a large number of typical WAN path latencies while requiring a memory footprint achievable in most network service platforms. Buffer resources on the PG could also be better managed by sizing SLaBS buffers based on WAN latency, which could be learned from the network topology or observed with measurements.

B. XIO-RDMA Performance

In our XIO-RDMA experiments, the PGs served as our end hosts while we used the XGEM to emulate a direct, dedicated WAN path with varying latency between the two systems. Our testing compared the built-in XIO TCP driver with our XIO-RDMA implementation. Other user space GridFTP drivers such as UDT perform poorly in 10Gb/s environments due to the increased overhead of context switching and data copying as we have demonstrated in our previous work [27].

As shown in Figure 8, both XIO-RDMA and properly tuned TCP are able to quickly saturate a loss free, dedicated path with low to moderate amounts of latency. As latency increases beyond 60ms, both single and parallel TCP performance begins to drop while XIO-RDMA remains consistent. We also include results for untuned TCP (16M versus 256M buffers) to illustrate the importance of setting large TCP buffers in order to saturate 10Gb/s paths. An added benefit of XIO-RDMA is that the chore of system buffer tuning across platforms is eliminated. The driver is configured with adequate SLaBS buffers by default, and if desired, may be more directly controlled by the user via driver options passed by the GridFTP client.

Our primary use case for XIO-RDMA is in networks where on-demand, dedicated paths may be provisioned to the end-site. This allows GridFTP to take advantage of RoCE over long-distance paths, moving bulk data with improved network performance and increased host efficiency. To the best of our knowledge, our solution is the first to take advantage of RoCE for use over existing Ethernet WAN paths while outperforming other GridFTP RDMA approaches targeted for use in connecting distributed grid environments over IB [33].

C. Bottleneck Considerations

Existing circuit networks are subject to bandwidth availability and users may request or be provided with a path operating at some fraction of the native interface speed on the end host. This presents additional challenges as the resulting artificial bottlenecks can create performance issues for TCP, and extra considerations must be observed for maintaining good performance for RDMA transfers. Circuit bandwidth constraints are

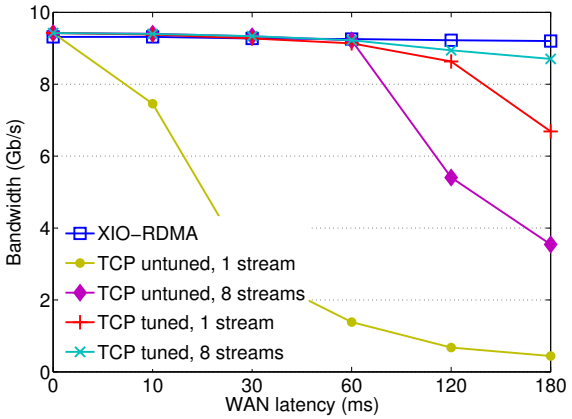


Fig. 8: Comparing GridFTP performance over increasing WAN latencies with and without XIO-RDMA driver

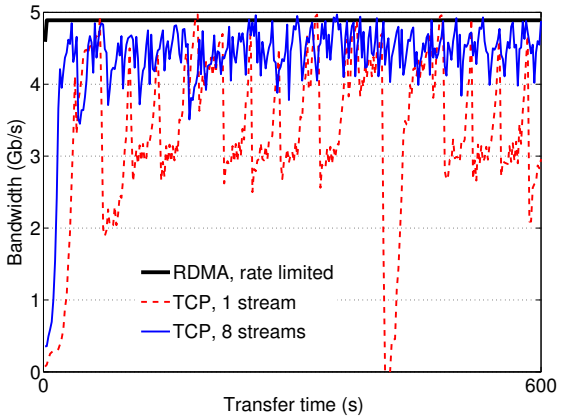


Fig. 9: Comparing TCP and rate-limited RDMA performance over 120ms WAN path with 5Gb/s policing

typically enforced with a combination of policing and shaping policies on switches along the provisioned path. Hard policing will drop frames if bandwidth thresholds are exceeded, or if the configured burst limit for the policy is set too low to handle the burstiness of incoming traffic.

The IB transport used in RoCE sends each application message (i.e. RDMA buffer) as a burst over the network at line rate. If the RDMA message size is greater than the burst limit configured by the circuit policing, RoCE frames will simply be dropped and the transfer will stall. We have observed this behavior directly when configuring the XGEM to police traffic between our RoCE-capable hosts. To overcome this, we must ensure that the maximum message size remains below the burst limit, and in addition, the rate of RDMA transfers must be paced to match the policed bandwidth.

We used an IB benchmarking tool to test the above hypothesis. Importantly, the benchmark allows for adjustments to the RDMA message sizes, provides adjustable rate limiting, and has the ability to specify the transmit queue depth used by the *ibverbs* library. Figure 9 shows that we were able to saturate a 5Gb/s, 120ms bottleneck path as long as the following conditions were met: 1) RDMA message sizes were kept below the policing burst limit configured on the XGEM (1MB in this case), 2) transmission of RDMA messages was rate limited, or paced, to the policer bandwidth, and 3) the transmit queue depth was sufficiently large enough to match the BDP of the path. The steady performance of the rate limited RDMA transfer contrasts sharply with TCP behavior. As the transfer rate ramps up past the bottleneck rate, TCP reacts to loss induced by the bandwidth policing and severely backs off, and even parallel TCP streams is unable to consistently saturate the path. The effects of TCP's congestion recovery mechanism are amplified by the high latency of the path.

Our SLaBS model is one that benefits from large RDMA message sizes to reduce overhead in both buffer management and the signaling of metadata between endpoints that use our RDMA implementation. For bulk data movement applications in particular, larger units of data exchange are desirable and SLaBS buffers are typically sent with RDMA message sizes ranging from 32MB to 256MB. Unfortunately, attempting to send these larger buffers exceeds the burst limits and overruns queues found in many network devices enabling circuit networks. Based on these findings, we are investigating modifications to our SLaBS buffering implementation to support adaptive rate limiting and the ability to efficiently manage smaller buffer partitions when required by the underlying network configuration.

VI. System Analysis

Reducing communication overhead remains a key issue for supporting increasingly faster networks. System resources not consumed by network operations can be allocated for other CPU and memory-intensive tasks, ideally allowing for efficient overlap of computation and communication. For example, software router platforms such as our Phoebus Gateways can take advantage of reduced communication overhead to provide additional services such as encryption or compression of the data stream.

In addition to the *zero-copy* techniques supported by RDMA protocols, we take advantage of the Linux kernel "splicing" support in our implementations. The *splice* and *vmsplice* system calls use a kernel memory pipe to "splice" or connect file descriptors, which may refer to network sockets, and memory pages while avoiding costly copying to and from user space buffers. These techniques allow us to further reduce system resource utilization when sending and receiving TCP streams in our PGs.

In order to better quantify the performance benefits of RDMA and splicing, we profiled our PG systems using the Linux *perf* tool [14] while running a series of benchmarks that allowed us to compare standard TCP, splicing, and RDMA approaches to network communication. *perf* provides access to software and hardware counters including spent CPU cycles, page faults, data and instruction cache loads, stores, and misses, context switches and number of CPU migrations. Our primary goal in this work is to reduce memory bus contention and CPU utilization, and thus our results focus on data cache profile statistics and cycles spent in each case.

We developed a network benchmark called *xfer_test* that implements a configurable SLaBS ring buffer and allows client/server transfer of buffer partitions over TCP, using both standard socket calls as well as splicing, in addition to RDMA

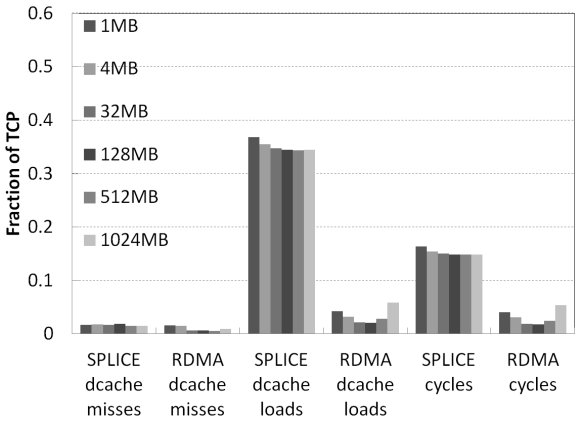


Fig. 10: System profiling results comparing buffer send overheads using splice, and RDMA. Normalized to TCP results.

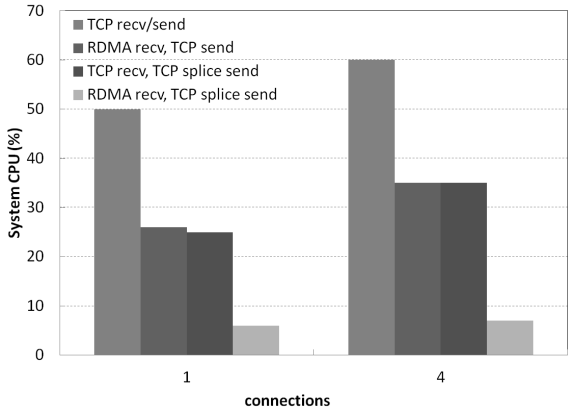


Fig. 11: Phoebus-SLaBS CPU overhead

support. Each test involved a 60 second transfer for each of TCP, splicing, and RDMA, which was repeated for SLaBS buffers of increasing size from 1MB to 1024MB. The *perf* tool was used to collect system profile information for each run and the results averaged over 20 total runs.

Figure 10 shows the profiling results for *xfer_test* runs across increasing buffer sizes for both the splice and RDMA cases on an otherwise unstressed system. The values shown are the ratio of data cache misses and stores, plus CPU cycles spent, normalized to the equivalent standard TCP transfer results. Clearly, both splicing and RDMA provide a significant reduction in terms of accesses to the system memory bus and total CPU utilization. RDMA improves upon the splicing case even further, with over 95% fewer data cache loads and CPU cycles spent compared to the equivalent TCP transfer across buffer sizes.

We also performed the same tests on a memory-stressed system using the *stress* system workload generator [4]. The relative performance was comparable to the non-stressed system with slight improvements for both splicing and RDMA compared to standard TCP socket operations. However, there was a significant increase in cache loads for the splicing case. Although splicing is a considerable improvement, and we recommend its use in TCP applications wherever appropriate, RDMA is clearly the network communication approach incurring the least amount of overhead.

Finally, we look at the benefits to overall system CPU utilization in an operating PG when using SLaBS with and without RDMA and splicing. Figure 11 compares TCP socket *write* calls, with and without RDMA, versus kernel socket splice optimizations, with and without RDMA, in the egress SLaBS gateway. Using only the splice calls instead of a standard TCP send, we can achieve a nearly 40-50% reduction in total CPU usage, even as the number of incoming connections increases. This overhead reduction significantly improves the PGs ability to scale and apply additional optimizations. With the RDMA data channel and splice, the picture changes even more dramatically and we see only 6-7% total CPU utilization when buffering and forwarding at 10Gb/s rates.

Our profiling results clearly show the benefits of using splicing and RDMA in order to improve the performance of data movement services that use SLaBS, or any network communication pattern that relies heavily on memory-intensive operations. Our performance evaluation in Section V of both Phoebus RDMA and XIO-RDMA empirically verifies the effective use of these optimizations in practical scenarios.

VII. Related Work

Being a recently proposed standard, there has been relatively little previous research in analyzing RoCE performance over existing Ethernet infrastructure, and we believe ours is the first to evaluate the application of RoCE over wide-area networks in particular. Other RDMA and *zero-copy* protocols not involving IB have been proposed to run over Ethernet. These include technologies such as Intel's Direct Ethernet Transport (DET) [6] and approaches that use iWARP-enabled NICs [12], [30]. Compared to RoCE and IB, DET does not provide full OS-bypass functionality with limited hardware support, while iWARP remains bound to the limitations of TCP/IP.

On the other hand, there have been active efforts involved with extending IB fabrics over WANs [11], [29] and comparisons of IB to existing 10G Ethernet in high-latency transfer scenarios [32]. These evaluations rely on IB extension devices which limit WAN performance to approximately 8Gb/s, whereas our approach shows that RoCE can easily saturate existing 10G networks. Other related work has investigated RDMA-capable storage protocols over WANs [9], [35] and explored system-level benefits of RDMA interfaces over 10G networks [7].

Our development of an XIO-RDMA driver for GridFTP shares common goals with a similar approach in ADTS [33]. As an RDMA implementation for GridFTP, the focus of ADTS is on disk-based transfers and their performance analysis was limited to I/O bound data movement over lower latency paths. RXIO [34] adapts the XIO framework for efficient RDMA communication but focuses on non-contiguous I/O operations within data center environments.

Considerable efforts have been made in modifying TCP's AIMD-based congestion control algorithm, resulting in numerous variations for improving performance over WANs. These include High-Speed TCP (HSTCP) [15] and Hamilton-TCP [28], among many others too numerous to mention here.

Others have investigated user space implementations such as UDT [19], which provides reliability over UDP but suffer from increased overhead, limiting their practical deployment in high-speed networking applications. Newer transport protocols such as SCTP [10] seek to improve performance through multistreaming, while others allow features such as explicit congestion feedback [22]. In all of these cases, improvements have been incremental while failing to address new modes of thinking in the transport of bulk data over long distances.

VIII. Conclusion

Advances in RDMA over Ethernet technologies now make it possible to efficiently support high-performance data movement over existing 10G wide-area infrastructure. At the same time, bringing these benefits to typical applications and end-users has remained a barrier for entry in many data center and grid environments. In this work, we have presented our Bulk Asynchronous Get approach as implemented in our Phoebus Gateway system, enabling the transparent use of RDMA protocols over the WAN for existing bulk data transfer applications without modifications. Our performance analysis has demonstrated considerable gains for GridFTP transfers using RDMA over a variety of emulated WAN conditions compared to end-to-end TCP. Depending on the severity of network impairments, our approach can provide between 20% to over 600% improvements in achievable throughput over high latency paths. Our work also considers native RDMA support within data transfer applications, and we have demonstrated an XIO-RDMA driver that allows GridFTP to maintain consistently high performance over dedicated high-latency WAN paths.

In order to be remain viable, our system must also scale to 40G and 100G network speeds. Our analysis numbers indicate that reductions in OS overheads through a combination of RDMA and system optimizations will allow our techniques to operate at such speeds on standard hardware platforms.

References

[1] Gem: Network impairment emulator. http://www.spirent.com/Solutions-Directory/Impairments_GEM.

[2] Net:netem. http://www.linux-foundation.org/en/Net:Netem.

[3] Ofed: Open fabrics enterprise distribution. https://openfabrics.org/ofed-for-linux-ofed-for-windows/ofed-overview.html.

[4] Stress workload generator. http://freecode.com/projects/stress.

[5] Infiniband. architecture specification release 1.2.1 annex a16: Roce. Infiniband Trade Association, 2010.

[6] Intel direct ethernet transport (det). http://software.intel.com/en-us/articles/intel-direct-ethernet-transport, 2010.

[7] P. Balaji. Sockets vs rdma interface over 10-gigabit networks: An in-depth analysis of the memory traffic bottleneck. In *In RAIT workshop .04*, page 2004, 2004.

[8] S. Bryant, G. Swallow, L. Martini, and D. McPherson. Pseudowire Emulation Edge-to-Edge (PWE3) Control Word for Use over an MPLS PSN. RFC 4385 (Proposed Standard), February 2006. Updated by RFC 5586.

[9] B. Callaghan, T. Lingutla-Raj, A. Chiu, P. Staubach, and O. Asad. Nfs over rdma. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*, NICELI '03, pages 196–208, New York, NY, USA, 2003. ACM.

[10] A. L. Caro, Jr., J. R. Iyengar, P. D. Amer, S. Ladha, G. J. Heinz, II, and K. C. Shah. Sctp: A proposed standard for robust internet data transport. *Computer*, 36:56–63, November 2003.

[11] S. Carter, M. Minich, and N. S. V. Rao. Experimental evaluation of infiniband transport over local- and wide-area networks. In *Proceedings of the 2007 spring simulation multiconference - Volume 2*, SpringSim '07, pages 419–426, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[12] D. Dalessandro and P. Wyckoff. A performance analysis of the ammasso rdma enabled ethernet adapter and its iwarp api. In *In Proceedings of the IEEE Cluster 2005 Conference, RAIT Workshop*, 2005.

[13] J. D. Day and H. Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

[14] A. C. de Melo. The new linux 'perf' tools. http://vger.kernel.org/~acme/perf/lk2010-perf-paper.pdf, 2010.

[15] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996.

[17] B. Gibbard, D. Katramatos, and D. Yu. Terapaths: A qos-enabled collaborative data sharing infrastructure for peta-scale computing research. In *Proceedings of the 3rd Intenational Conference on Broadband Communications (IEEE)*, 2006.

[18] GridFTP. http://www.globus.org/datagrid/gridftp.html.

[19] Y. Gu and R. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Computer Networks (Elsevier) Volume 51, Issue 7*, 2007.

[20] S. Ha, I. Rhee, and L. Xu. Cubic: A new tcp-friendly high-speed tcp variant. *ACM SIGOPS Operating Systems Review, Volume 42, Issue 5*, pages 64–74, 2008.

[21] IPerf. http://dast.nlanr.net/Projects/Iperf/.

[22] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 89–102, New York, NY, USA, 2002. ACM.

[23] R. Kettimuthu, W. Liu, J. M. Link, and J. Bresnahan. A gridftp transport driver for globus xio. In *PDPTA*, pages 843–849, 2008.

[24] E. Kissel and M. Swany. Session layer burst switching for high performance data movement. In *Proceedings of PFLDNet*, 2010.

[25] E. Kissel and M. Swany. The extensible session protocol: A protocol for future internet architectures. Technical Report UDEL-2012/001, Dept. of CIS, University of Delaware, 2012.

[26] E. Kissel and M. Swany. Validating linux network emulation. Technical Report UDEL-2012/004, Dept. of CIS, University of Delaware, 2012.

[27] E. Kissel, M. Swany, and A. Brown. Phoebus: A system for high throughput data movement. *J. Parallel Distrib. Comput.*, 71:266–279, February 2011.

[28] D. Leith and R. Shorten. H-TCP: TCP congestion control for high bandwidth-delay product paths. Internet Engineering Task Force, INTERNET-DRAFT, draft-leith-tcp-htcp-00.txt, 2005.

[29] S. Narravula, H. Subramoni, P. Lai, R. Noronha, and D. K. Panda. Performance of hpc middleware over infiniband wan. In *Proceedings of the 2008 37th International Conference on Parallel Processing*, ICPP '08, pages 304–311, Washington, DC, USA, 2008. IEEE Computer Society.

[30] M. Oberg, H. M. Tufo, T. Voran, and M. Woitaszek. Evaluation of rdma over ethernet technology for building cost effective linux clusters. In *7th LCI International Conference on Linux Clusters: The HPC Revolution*, 2006.

[31] ESnet On-demand Secure Circuits and Advance Reservation System (OSCARS). http://www.es.net/oscars/.

[32] N. S. V. Rao, W. Yu, W. R. Wing, S. W. Poole, and J. S. Vetter. Wide-area performance profiling of 10gige and infiniband technologies. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 14:1–14:12, Piscataway, NJ, USA, 2008. IEEE Press.

[33] H. Subramoni, P. Lai, R. Kettimuthu, and D. K. Panda. High performance data transfer in grid environment using gridftp over infiniband. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 557–564, Washington, DC, USA, 2010. IEEE Computer Society.

[34] Y. Tian, W. Yu, and J. S. Vetter. Rxio: Design and implementation of high performance rdma-capable gridftp. *Computers & Electrical Engineering*, 38(3):772–784, 2012.

[35] W. Yu, N. Rao, P. Wyckoff, and J. Vetter. Performance of rdma-capable storage protocols on wide-area network. In *3rd Petascale Data Storage Workshop PSDW 2008*, 2008.