Improving GridFTP Performance Using The Phoebus Session Layer

Ezra Kissel and Martin Swany
Department of Computer & Information Sciences
University of Delaware, Newark, DE 19716
{kissel, swany}@cis.udel.edu

Aaron Brown
Internet2
1000 Oakbrook Drive, Ann Arbor MI 48104
aaron@internet2.edu

ABSTRACT

Phoebus is an infrastructure for improving end-to-end throughput in high-bandwidth, long-distance networks by using a "session layer" protocol and "gateways" in the network. Phoebus has the ability to dynamically allocate network resources and to use segment-specific transport protocols between gateways, as well as to apply other performance-improving techniques on behalf of the user.

One of the key data movement applications in high-performance and Grid computing is GridFTP from the Globus project. GridFTP features a modular library interface called XIO that allows it to use alternative transport mechanisms. To facilitate use of the Phoebus system, we have implemented a Globus XIO driver for Phoebus.

This paper presents tests of the Phoebus-enabled GridFTP over a network testbed that allows us to modify latency and loss rates. We discuss use of various transport connections, both end-to-end and hop-by-hop, and evaluate the performance of a variety of cases. We demonstrate that Phoebus can easily improve performance in a diverse set of scenarios and of cases, in many instance it outperforms the state of the art.

1. INTRODUCTION AND MOTIVATION

Despite continuing advances in the link speeds of networks, data movement remains a key problem in distributed computing. The viability of many distributed computing paradigms depends on the ability to have data transfer speeds scale up as computing power increases. This paper investigates the efficacy of a network middleware system for improving data transfer time.

A typical end-to-end path through the Internet may traverse a variety of network technologies, each of which can have a unique set of characteristics. Everything from shared wireless links to dedicated optical circuits can be utilized as data travels from source to destination. Network segments often have dramatically different latencies, jitter and loss rates, and the interactions between them can lead to less than desirable end-to-end performance using existing transport protocols.

SC09 November 14-20, 2009, Portland, Oregon, USA

One situation in which the inherent heterogeneity of the network is most apparent is in the emerging paradigm of dynamic networks, in which network resources can be requested and reserved. These networks complete the on-demand computing landscape by doing for networks what the grid paradigm does for computing and storage. This model effectively creates a "hybrid" network that involves both shared network segments and dedicated circuits in an end-to-end path. While these networks are emerging as a powerful tool, their effective use is still an open issue.

Phoebus is an infrastructure for improving end-to-end throughput in high-bandwidth, long-distance networks. Phoebus augments the current Internet model by utilizing a "session layer" protocol and "gateways" in the network. Phoebus has the ability to dynamically allocate network resources, to use segment-specific transport protocols between gateways, as well as to apply other performance-improving techniques on behalf of the user. Phoebus is particularly suited for hybrid network environments, in which links at the edges are shared and core network circuits are dedicated.

Our new protocol will only see widespread adoption if users can easily integrate it into their existing systems and applications. Our previous enabling approaches consisted of methods to transparently allow existing applications to use the Phoebus infrastructure. However, this transparent operation proved cumbersome to effectively use with GridFTP [10]. GridFTP's status as a key application in the Grid computing environment necessitated a closer tie-in to allow users to easily use the Phoebus infrastructure in their data transfers.

In this work, we have built upon the existing Phoebus architecture and have provided additional features that allow for efficient and transparent use of this increasingly heterogeneous network landscape. The specific contributions of this paper are:

- *A study of in-the-network protocol translation to better utilize network links with unique characteristics:* We demonstrate how adapting data movement by using different protocols along an articulated network path can increase network throughput. We describe how this adaptation is implemented within our network middleware system.
- *A Globus XIO driver to enable Phoebus transfers for GridFTP:* We describe how the driver enables transparent use of the Phoebus architecture for a critical Grid application. Our performance testing compares GridFTP data transfer throughput while utilizing the Phoebus driver with the existing approaches.
- *Testbed results of Phoebus performance in a variety of net-*

work conditions: We show how Phoebus can improve GridFTP performance over a variety of network paths as well as how single stream Phoebus transfers are able to outperform parallel streams in many cases.

- *An examination of the efficacy of Phoebus in using dynamic networks.* We demonstrate how Phoebus can optimize the use of hybrid networks with static and dynamic components by examining its efficacy in cases with dedicated, but limited bandwidth.

The remainder of this paper is organized as follows: Section 2 will briefly discuss some background and provide an overview of the Phoebus system. Details of the Phoebus XIO driver implementation are outlined in Section 3. Section 4 will discuss the effects of transport layer protocols and give an overview of how Phoebus can translate between them. Finally, we'll present experimental results based on our testbed in Section 5 and summarize and discuss future work in Section 6.

2. BACKGROUND

For decades, the end-to-end argument [18] has provided the conceptual basis for transport protocols. The common interpretation of this argument says that the core of the network should remain simple, and that all protocol functionality, beyond merely forwarding packets, should be handled by the end hosts. This absolutist interpretation of the end-to-end argument forces all control and optimizations to the edge. This control mechanism needs to infer the state the network and when a packet is lost, due to any number of factors, the protocols must assume why the loss occurred and react accordingly. This inference and reaction is necessary to ensure fairness among flows traversing the same links. Over the years a number of heuristics have been proposed to improve the ability of transport protocols to discern congestion related loss and react accordingly.

The ubiquitous Transmission Control Protocol (TCP) is known to have performance problems in long-distance, high-bandwidth networks [15]. While there have been countless proposals to change TCP, none have been a panacea. Many TCP variants exist, but none are in widespread use. The high-performance computing and networking community has circumvented the problems in two major ways. The first is the use of parallel TCP streams [12, 24]. The second is with user-space protocols such as UDT [11] that take advantage of the User Datagram Protocol (UDP) for data transfer.

2.1 Phoebus and Dynamic Networks

Phoebus [6] is a system that implements a new protocol and associated forwarding infrastructure for improving throughput in today's networks. Phoebus is the follow-on to the Logistical Session Layer (LSL) [19], which used a similar protocol. Details on the Phoebus architecture can be found in our earlier work, but we provide a brief description below for completeness.

The current Internet model binds all end-to-end communication to a "Transport" layer protocol such as the Internet Protocol (IP) suite's Transmission Control Protocol (TCP). The Phoebus model binds end-to-end communication to a "Session" protocol, which is a layer above the Transport layer. Thus, Phoebus is able to explicitly mitigate the heterogeneity in network environments by breaking the end-to-end connection into a series of connections, each spanning a different network segment. In our model, Phoebus Gateways (PGs)

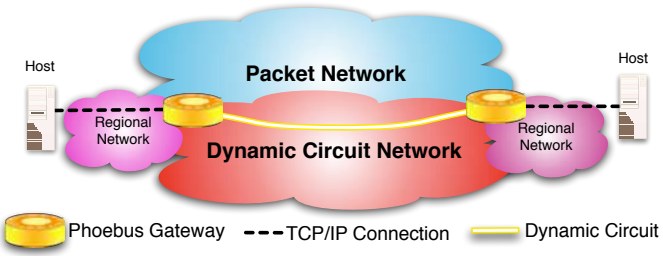

Figure 1: Phoebus Gateways at the border of regional networks and backbone and dynamic circuit networks like those deployed by Internet2.

located at strategic locations in the network take responsibility for shepherding users' data to the next host in the chain. This network "inlay" allows Phoebus to adapt the data transfer at application run time, based on available network resources and conditions. The Phoebus infrastructure creates an intelligent, articulated network. This network can take responsibility for ensuring good throughput for applications, while acting as an adaptation point and "network on-ramp" to different network architectures.

2.2 Dynamic Networks

Phoebus bundles a variety of tuning and adaptation into a networked data movement services. Despite the protocol advances and heroic network performance numbers reported through the years, most users of advanced distributed computing environments struggle with adequate network performance. There are many sources of information about network tuning [1, 3], and certainly there are many techniques to improve network performance, but the so-called "wizard gap" remains. In some sense, the fact that a high performance distributed system user needs to understand host tuning and congestion windows points to a failure of current network tools. We as computer scientists don't need to understand the standard model to plug our systems into wall sockets or the Nyquist-Shannon sampling theorem to make a telephone call. Without a sea-change in the way we think about networks, we will continue to be frustrated. In that spirit, the Phoebus system offers a way to offload network tuning to network experts and the network itself.

More and more research and education network providers are deploying network reservation technology that allows network bandwidth to be dynamically allocated. Dynamic Circuit Networks (DCNs), as these network are often called, support high-performance and Grid computing applications that demand network capacity. Phoebus is a key technology for enabling broad access to these DCNs in that it provides a seamless mechanism to bridge the gap between the traditional shared packet environment and on-demand circuits.

Dynamic circuit (or bandwidth-on-demand) networks are being deployed by major research networks like Internet2, the US Department of Energy's ESnet, and GÉANT2 in Europe. These networks are developing a compatible signaling interface that allows allocation of circuits across campus, regional, national and international networks. This dynamic network cloud allows users to set up dedicated, guaranteed network paths, on demand, for high-performance data transfers. However, in many cases, it is not feasible to bring these new circuit capabilities to every resource than can benefit from them, e.g. directly to each user's desktop. Phoebus provides a way to allow users to utilize a DCN without provisioning a circuit to every end host.

The Phoebus platform is currently being deployed on the Internet2 Network to enable users to automatic access its new DCN. Phoe-

bus will form the basis of a new data movement service which intends to transparently enable members of the research and education community to access the network in order to experience improved data transfer performance without any modification by the end users.

To handle this case, the Phoebus infrastructure includes an interface to the common DCN infrastructure as well as a client library that is a transparent replacement for the standard "sockets" API. When a new connection is initiated, the edge host contacts a PG. This PG can allocate a dynamic circuit across the backbone network, possibly crossing administrative domain boundaries, to the PG closest to the destination host. The far-end PG connects to the destination host. In this instance, the end-to-end flow will consist of two edge Transport-layer connections passing over shared Ethernet networks, and a third connection traversing the dedicated link.

This model applies to more than just flows crossing dynamic circuits. Many backbone networks, through proactive monitoring and improvements, have been able to reduce the loss along the backbone to near zero. This means that their users should see higher performance, but due to shared infrastructure on the edges, the users' performance is limited as compared to the available bandwidth. While not specifically allocating resources like in the DCN case, the model of shared edge networks and a near lossless backbone would equally apply to these kinds of networks.

3. A GLOBUS XIO DRIVER FOR PHOEBUS

Globus XIO is an extensible input/output library within the Globus Toolkit [9]. By utilizing the concept of a driver stack, various protocol drivers may be implemented independently and loaded dynamically at run time. This modularization facilitates reusability for applications developed with the toolkit. Using the XIO framework, we were able to create a Phoebus transport driver that allows Globus applications to natively take advantage of the Phoebus platform. In particular, we are interested in the performance of GridFTP transfers when utilizing the Phoebus driver.

Our driver is based on the built-in XIO TCP driver distributed in the Globus Toolkit. The driver was extended to support instantiating a Phoebus session when initiating outgoing connections. In most uses of the Phoebus system, the last PG in the series removes session headers and framing and then uses a TCP connection to communicate with the application's server, which is unaware of the use of Phoebus. By using the explicitly-loaded Phoebus XIO driver, the user is able to choose to use Phoebus, and both sides of the connection are aware of the session-layer semantics.

The XIO framework maintains a clear distinction between transport and translation drivers, providing a way to modify both the control and data channels during a transfer. The Phoebus driver is purely a transport driver. As such, when GridFTP is used with GSI authentication, for instance, and the Phoebus driver is requested, the control channel is unmodified and authentication proceeds as it would over the standard TCP driver even though the data path may now traverse a number of independent PGs.

An application may invoke the Phoebus driver by simply pushing it onto the driver stack. The well-known client, *globus-url-copy*, utilizes the `-dcstack` flag to specify the data channel stack to be used during the transfer, allowing the GridFTP server to load the Phoebus driver when requested. It is also conceivable that a GridFTP server administrator may configure the server to utilize

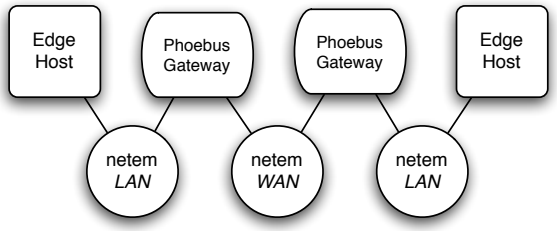

Figure 2: Testbed Configuration

the Phoebus transport driver by default and thereby make the use of Phoebus completely transparent to the end-user.

In order to specify which PG to use, the driver can make use of environmental variables that specify the full Phoebus path or simply the first hop along the path. To allow for a more programmatic approach, these values can also be set by including driver-specific options. We were also able to maintain virtually all of the existing TCP driver options and attributes while supporting additional Phoebus-specific functionality. Socket options specified with *setsockopt* calls are applied to standard TCP sockets as well as sockets associated with newly created Phoebus sessions.

Deployment of the XIO Phoebus driver is accomplished by simply installing it along with any other transport drivers during a Globus Toolkit installation. The driver source is configured, compiled, and installed along with other Globus components, similar to the included UDT driver, easing the configuration burden for large-scale Grid deployments. The driver code is expected to be released in a forthcoming Globus Toolkit version.

4. PHOEBUS SERVICES

A key tenet in the Phoebus model is that an end-to-end connection, articulated via a series of Transport protocol adapting Session gateways, can often outperform a single end-to-end transport protocol. A session-layer connection such as this can also outperform parallel connections in many cases, though Phoebus itself can also make use of parallel connections.

An end-to-end transport must behave conservatively as it may cross a wide variety of network conditions and technologies. A long-distance connection may pass over shared and dedicated links, long-distance loss-free networks and links with non-congestive loss. While protocols have been designed to handle each of these scenarios separately, no single transport connection can deal efficiently with all these network types. In addition, TCP is sensitive to the round trip time (RTT) [15], so simply reducing the RTT that a single TCP connection is faced with will improve TCP's ability to react and thus, its performance.

This issue is especially apparent for utilizing dedicated circuits. For a host to directly communicate over a dedicated channel, the host must be able to allocate an end-to-end circuit between itself and the destination. Unless the networks for both hosts allow for the creation of the end-to-end circuit, there is going to be some portion of the end-to-end connection that does not pass over the dedicated channel. This segment will likely occur over shared Ethernet meaning the connection will be sharing that segment with other TCP flows. While there have been a number of protocols written to maximize utilization of an allocated circuit, these protocols often will not retain TCP friendliness [16] [25] [13]. This leaves users with two options: use a suboptimal protocol to not impair other connections but waste some of the allocated bandwidth or use a bet-

ter, more aggressive protocol that potentially interferes with other connections.

In the Phoebus model, the end-to-end connection passing over the links mentioned above can be divided into a series of transport layer connections. Each of these transport layer connections can be adapted for the network environment the connection is utilizing. Indeed, each of these transport layer *instances* will dynamically adapt to the characteristics of the link.

To perform protocol adaptation and translation, PGs can be deployed inside the network whose function is to buffer the data and transmit it using the new protocol. These devices, having the most knowledge about the network between them, can choose the specific protocol or protocol settings that are best suited for the network path connecting them. The choice of protocol depends on a number of factors including the network conditions between the two devices, network resource type, network policy.

We have chosen to use UDT [11] in this paper as it has shown good performance in the WANs. It attempts to be responsive to cross-traffic, but is more aggressive in bandwidth utilization. In addition, we can tune the rate of transmission with relative ease. Finally, Globus GridFTP also features an XIO driver for it, facilitating more direct comparison.

4.1 Basic TCP Adaptation

The most basic protocol adaptation that can be performed is to change the settings used in the TCP connection. Since most users will not be transferring large amounts of data, operating systems vendors often leave the default TCP settings rather conservative to avoid wasting CPU and memory. There are numerous tuning guides available to teach users the specific set of options they should set on a TCP connection to achieve good throughput [1, 3]. The suggestions that they give fall broadly into two categories: increasing the send or receive buffers and changing the congestion control algorithm. Setting these options can have a significant impact on the performance of a TCP connection.

Buffer Sizes

Phoebus includes the ability to calibrate the size of the send and receive buffers for TCP connections. Phoebus can use this configurability to tailor the buffers to the exact distance between the devices. This ensures that longer distance connections will have enough buffer space to perform well while preventing shorter distance connections from wasting memory.

Congestion Control

There are a wide variety of congestion control algorithms available for TCP [14, 20, 21, 23]. These advanced algorithms see little use due to the difficulties and time involved in deploying a new protocol or implementation. Simply utilizing a different congestion control mechanism over part of the connection could allow new techniques to be utilized sooner.

4.2 Adaptation to Non-TCP Protocols

Adapting to a protocol other than TCP can improve performance while still allowing ease of deployment. A large number of network protocols have been written whose implementations reside entirely in user-space [4, 11, 13]. Non-TCP protocols implemented in the operating system kernel do exist [7, 17], but these suffer from the same defects as TCP when it comes to updating to new versions, leaving our focus on the user-space implementations.

User-space implementations of protocols generally take the form of a library providing the normal connect, send, recv and close primitives. These libraries generally use the User Datagram Protocol (UDP), which provides unreliable packet transmission, in a manner similar to how TCP uses IP. This leaves the protocol library responsible for providing correct and in-order reception of the data on the far side of the connection as well as any necessary congestion control.

Running in user-space makes these protocol libraries significantly easier to deploy. In most cases, the protocol library can be installed in the user's home area instead of needing to be globally installed, like a kernel module. Administrators will also be significantly more comfortable with a user-space library since the negative effects of an implementation bug will only be borne upon the user installing the software instead of by the operating system as a whole. It will also be more likely that a protocol written in user-space will be ported to the operating system of interest, since most of these protocols only depend on the socket API which has been effectively standardized across a wide range of operating systems.

Using the protocol libraries in user-space comes at a cost. The implementation of handling TCP variants along with a range of user-space protocols will be significantly more difficult since some form of abstraction layer, described in Section 4.3, will need to be implemented. Running a protocol in user-space also incurs penalties in switching between user-space and kernel-space that do not apply to kernel-space protocols.

4.3 Protocol Abstraction Layer

In order to utilize protocols that run in both user-space and kernel-space, we augmented the PG software with an abstraction layer. Similar to the ubiquitous sockets interface, this abstraction layer allows us to keep the PG's forwarding routines simple by hiding the differences between the protocols.

The abstraction layer can be broken down into two areas: the functions that are used to return new connections and the connection objects themselves.

When a PG connects to its next hop, it specifies the settings for the connection, including the host name, port and any protocol specific options like buffer sizes or the congestion control algorithm. The abstraction layer then allocates the connection using the specified settings, and returns an object representing the new connection. If a connection must traverse a bottleneck link, as in the case of a provisioned circuit with a reserved bandwidth, the abstraction layer may also rate-limit suitable protocols to improve overall performance. Since PGs may act as on-ramps to circuit networks, reservation information such as circuit bandwidth and duration are available to the abstraction layer.

When a PG waits for incoming connections from other PGs or end hosts, it can use functions in the abstraction layer to create listener objects which wait for incoming connection requests. The PG specifies protocol settings with the listener object which are then applied to the incoming requests for that particular listener. When a client connects to the listener, it applies the requested protocol settings and creates a new object representing the connection. This object is passed back to the program by way of a callback function.

The other form of abstraction is the object representing an open connection. This object provides a consistent interface no matter

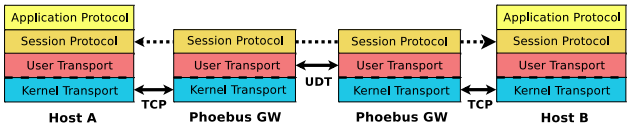

Figure 3: An end-to-end path with protocol adaptation performed at Phoebus Gateways in the network.

the underlying protocol. The objects contain functions for reading or writing the connection, shutting down the read or write side of the connection, functions for modifying protocol settings (when possible) and a function that retrieves statistical information about the connection.

The only function listed that does not have an analogous function in the standard socket API is the function to retrieve statistical information. Most statistical information (bytes sent, transfer rates, etc) could easily be tracked in a protocol independent fashion. However, some protocols may be obtaining this information already or may be able to more accurately track the information. For example, the web100 project [22] has produced a patch which instruments the TCP protocol in the Linux kernel. If the PG were to collect its own statistics, it would be redundant. By creating a higher level function to return statistics, each protocol is given the option of either using a protocol independent collection mechanism or reusing the statistics collection routines available to it.

4.4 UDT Adaptation

We created an implementation of UDT for the abstraction layer described above. UDT is a protocol from the University of Illinois designed for transfers over wide-area, high-speed networks [11]. The protocol provides reliability along with a modular congestion control algorithm. While the protocol provided a sockets-like API, there were some minor differences that needed to be addressed.

In TCP, the *shutdown* function can be used on a socket to close the reading or writing side. Once one side has been shutdown, any attempts by the remote host to read or write that side will fail. This set of semantics has produced a common approach to implementing socket based applications where the client sends a request, shuts down the write side of the socket and waits for the response. The server gets the request, handles it and then checks if another request has come in. Since the client has closed the write side, the server knows that the client is finished sending requests and closes down the socket. Under Phoebus, using TCP as the transport, the *shutdown* is received by the first PG and so the first PG shuts down the write side of its connection to the second PG who, in turn, shuts down the write side of its connection to the end host, effectively propagating the *shutdown*.

UDT does not currently implement *shutdown*, only *close*, whose semantics differ from those of *shutdown*. When the close function is used to terminate a socket, both the read and write sides of the socket are shutdown simultaneously. In the scenario described above, a problem occurs after the first host has received the *shutdown*; the host has no way of shutting down the write side of the connection. If the host uses close to terminate the connection, it will close the read side of the socket as well, preventing the response from propagating back. If the host chooses not to terminate the connection, the end server will continue waiting for the next request since it still perceives the client connection as being open and able to send more requests. This will cause the end-to-end connection to hang.

To handle this difference, the Phoebus UDT implementation was augmented to provide simple session-layer framing. This framing introduces a header for each session-layer frame, which optionally "contains" a payload. This is analogous to the operation of the lower layers of the stack.

The header has a type field used to differentiate between shutdown or data frames. If it is a shutdown message, the header contains a field describing which direction, reading or writing, is being shutdown by the frame's sender. The header also contains a 32-bit length field describing the length of its payload. In the case of a shutdown message, this will be zero. In the case of a data packet, this will correspond to the amount of data being transferred.

Phoebus uses this simple protocol to emulate the semantics of shutdown. When a PG needs to send data via a UDT connection, it creates a new data packet consisting of a header and the encapsulated data. When the PG needs to shutdown one side of a connection, it creates a new shutdown packet and sends it with no payload.

The receiver side initially reads in the header for each packet, and reacts accordingly. In the case of a shutdown packet, it sets flags to emulate the shutdown semantics on local UDT sockets. In the case of a data packet, it sets a flag, reading in the data as requested by the PG.

Figure 3 illustrates an end-to-end connection with UDT protocol adaptation performed across PGs. One of the advantages of this model is that the adaptation is transparent to the end hosts, which initiate standard TCP connections to the PGs and requires no special modifications to the application.

5. EXPERIMENTAL RESULTS

5.1 Testbed

Our goal is to test a real data transfer tool, GridFTP, using the Phoebus infrastructure in a variety of network conditions. Despite the availability of a test Phoebus infrastructure in Internet2 router points of presence (POPs) and test deployments in various other networks, getting access to a wide variety of end-to-end network paths is challenging. Even then, we have been at the mercy of prevailing network conditions, making experiment repeatability virtually impossible. This led us to build a testbed to emulate a range of network conditions in a controlled environment, in which we could make repeated experiments with the same configuration and effective conditions. It is important to note that Phoebus has demonstrated performance improvement in real networks, as presented in our previous work [6] and has demonstrated benefits for real applications (e.g. `http://www.internet2.edu/performance/phoebus/200804-phoebus-infosheet.pdf`).

The Linux kernel has a module available called *netem* [2] that makes emulating different network conditions possible. The module enables modification of how packets are handled by outgoing IP interfaces. It can buffer packets to create artificial latency as well as cause loss of packets. For our testing environment, we used the *netem* module to emulate various distances and loss rates. The setup consisted of seven machines connected as depicted in Figure 2. There were two end hosts that were used as the GridFTP source and destination servers. There were also two PGs, one at either side of the "backbone" network. The *netem* module sug-

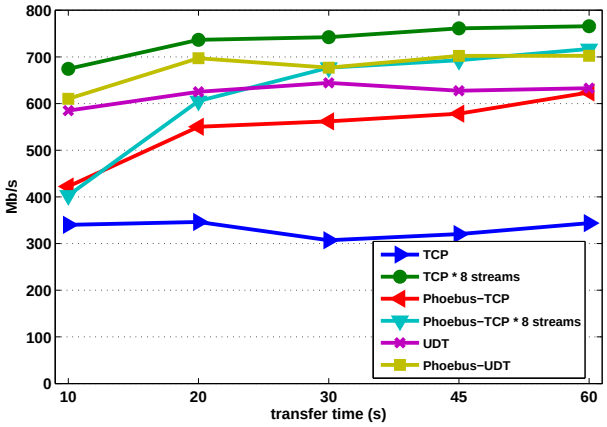

Figure 4: 25ms WAN Latency, %0.001 LAN loss

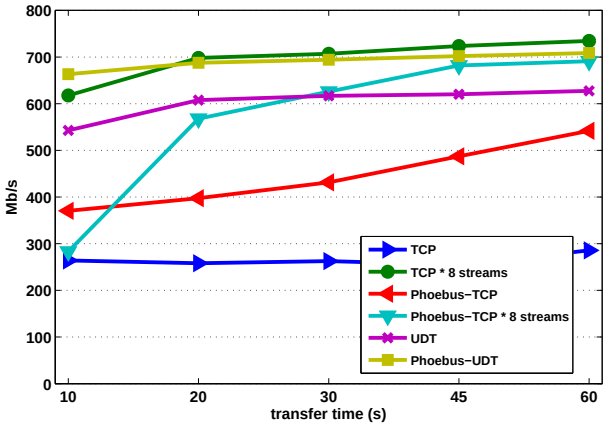

Figure 5: 50ms WAN Latency, %0.001 LAN loss

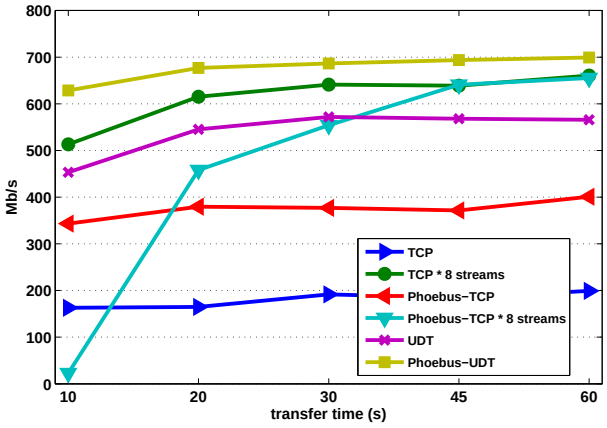

Figure 6: 100ms WAN Latency, %0.001 LAN loss

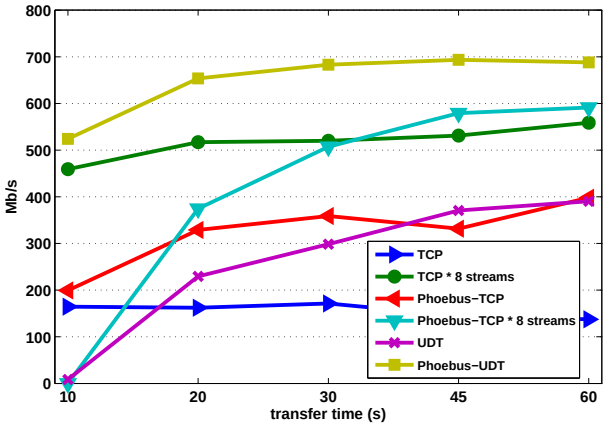

Figure 7: 150ms WAN Latency, %0.001 LAN loss

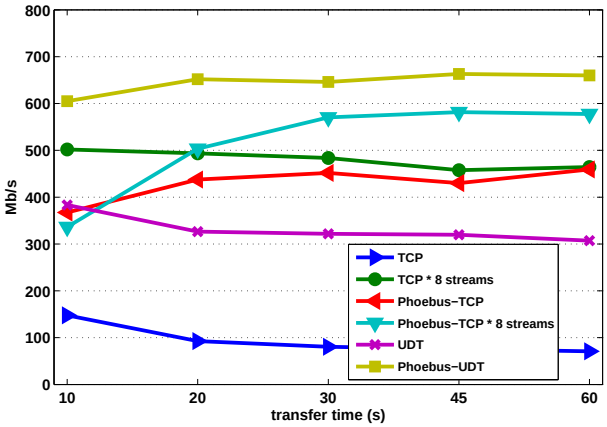

Figure 8: 25ms WAN Latency, %0.01 LAN loss

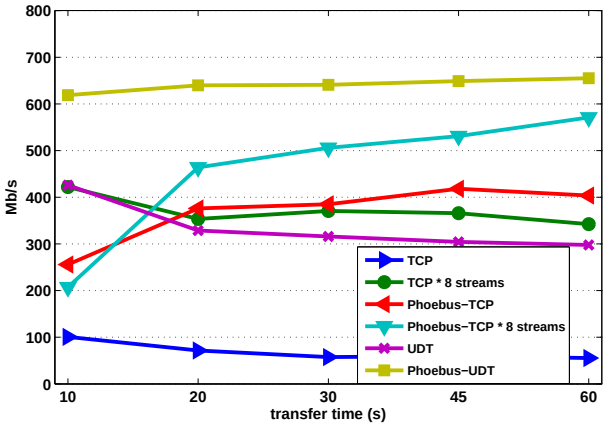

Figure 9: 50ms WAN Latency, %0.01 LAN loss

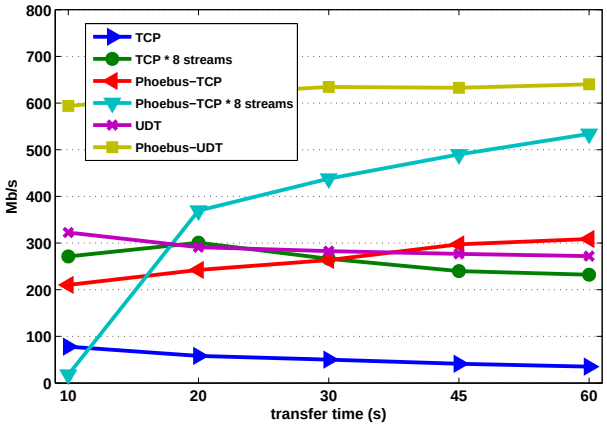

Figure 10: 100ms WAN Latency, %0.01 LAN loss

gests against using the module on the same host as the application sending or receiving data. This required us to add 3 dedicated hosts in the testbed to function as *netem* forwarding nodes. These nodes were configured to forward the data while transparently applying the latency and loss modifications. The *LAN* nodes emulate a shared local-area network with small amounts of loss, and the *WAN* nodes emulate the wide-area network, with varying amounts of latency introduced. This environment allowed us to test direct end-to-end connections as well as connections using Phoebus using the same paths, guaranteeing the same network conditions.

Our network testbed consists of Intel Pentium 4 Xeon 2.8GHz HT CPUs, 4GB RAM and 1Gb/second Ethernet links. While it may seem that 1G Ethernet is rather pedestrian in this age of 10G Ethernet, we again note that most of the day-to-day high-performance distributed computing for scientific applications today is well below 1Gb/second. In addition, the planned prototype Phoebus service on Internet2 will initially use 1G Ethernet and providing a reliable data transfer service at this rate is quite significant. Finally, we believe that the principles behind Phoebus are general and will apply equally to higher-speed links.

5.2 Experimental Configuration

For our experiments, we tested *LAN* packet loss rates of %0.001, %0.01, and %0.1, with *WAN* latencies of 25ms, 50ms, 100ms and 150ms. We also induced 4ms of latency for each LAN segment to simulate latencies over edge networks. Our chosen latencies are based on observed values between campus networks and nearby PGs as well as WAN paths on the Internet2 network and international R&E networks. Inter-gateway WAN latencies can range from 25ms to 75ms within the continental United States and exceed 100ms on transcontinental links. Our set of experimental cases represent a set of conditions that can be expected on current real-world networks.

Over each of these configurations, we measured direct GridFTP transfers over TCP and UDT, GridFTP transfers using Phoebus with both TCP and UDT over the *WAN* and 8 stream transfers using TCP over direct paths and using Phoebus. We also ran the same tests with the *netem* modules disabled, which results in three router hops with negligible latency and no loss.

Each system in our testbed was running a vanilla 2.6.26 Linux kernel with web100 patches. Standard TCP tuning was also applied to each system, so that the connections were not buffer limited. BIC was used as the TCP congestion control algorithm in these experiments. We also tested with CUBIC, which is the default congestion control algorithm in kernels since 2.6.19. However, we found that CUBIC performed best only in ideal network settings and that BIC was more resilient across the variety of network conditions over which we tested. We theorize that BIC responds better to loss in the network given that it is more aggressive as compared to CUBIC.

Each of the throughput experiment data points is the average of 20 identical runs. For the GridFTP benchmarks, we needed to ensure that the network was the bottleneck so that we could compare direct GridFTP connections with GridFTP connections over the Phoebus infrastructure. To remove the disk as a bottleneck, we employed memory-to-memory GridFTP using */dev/zero* and */dev/null* as the source and destination files respectively. To maintain consistency between the single stream and multiple stream tests, we forced all single stream GridFTP transfers to operate in extended block mode (MODE E) like the multiple stream tests. This forces the single stream tests to pay the overhead of including the per-block headers that are necessary for the multiple stream case.

One of the goals of Phoebus is to effectively replace (or at least reduce) the need to use parallel TCP instances commonly used in GridFTP transfers. To that end, we compare GridFTP with parallel streams against GridFTP using a single stream over the Phoebus infrastructure, as well as how GridFTP compares with parallel streams over Phoebus. In [5], the authors found that increasing the streams can decrease the GridFTP performance. Our intention was to compare Phoebus against direct connections while giving direct connections the best possible opportunity to compete.

5.3 Throughput Results

The test results demonstrate the efficacy of Phoebus in realistic wide-area configurations. For the shorter latency cases with low loss, like those in Figures 4 and 5, Phoebus-TCP shows unnecessary overhead, while Phoebus-UDT is on par with the best cases. As latency and loss increase along the path, the Phoebus cases begin to outperform all others. In the 100ms latency cases, Phoebus-UDT clearly outperforms the other configurations, including that of direct UDT. With *WAN* latencies of 150ms, Phoebus is the clear winner, with little performance degradation due to the significant latency. When loss increases to %0.01, with 25ms, 50ms and 100ms latency, Phoebus again suffers little performance loss, while other cases do.

A single end-to-end session using TCP at the edges and UDT over the WAN performs significantly better than any other configuration in environments challenged by latency and loss. This configuration is competitive with parallel TCP streams even under the better sets of network conditions, and has the added benefit of enabling applications other than GridFTP to take advantage of this performance without being forced to manage parallel connections. In addition, the impact of a single TCP stream at the shared edges of the network will be less than more aggressive approaches like UDT or parallel streams in the face of contention.

5.4 CPU Utilization

In addition to overall throughput, we also measured the CPU load of the client systems during the duration of the transfers. The CPU statistics for the GridFTP process running on the client were obtained from the */proc* file system and averaged over 1 second intervals. With the exception of the UDT tests, CPU utilization var-

ied within a few percentage points between the Phoebus and direct cases with comparable observed throughput. We found that the largest determining factor affecting CPU utilization in these cases was the overall throughput achieved. Thus when Phoebus was employed and the observed throughput increased, we observed additional CPU load as the client worked harder to process more packets.

Direct UDT	Phoebus w/ UDT	Direct UDT	Phoebus w/ UDT
no loss		0.001% loss	
62.1%	15%	38.4%	15.7%

Table 1: Comparison of CPU utilization using UDT from the edge host and via Phoebus.

Given that UDT is a user-space protocol implementation, we were not surprised to see much higher CPU loads during the UDT direct transfers. Although it provides consistently good performance in our tests, one of the tradeoffs includes the increased system requirements of the client. Table1 shows the disparity between running UDT natively through the GridFTP client and the Phoebus-UDT case where the UDT adaptation occurs along the WAN segment alone. Using Phoebus in this configuration provides a 25-45% decrease in client CPU utilization depending on the network conditions. CPU utilization drops for UDT with higher latency and loss as the throughput decreases, whereas the utilization for Phoebus-UDT remains nearly the same along with the observed throughput.

Obviously, the CPU overhead for UDT is simply moved from the end system to the PG, but this is a reasonable division of labor in some cases – compute nodes can focus on computation, while network-focused nodes can manage high-performance transfers. In the end, however, we are not relying on UDT to be the wide-area transport protocol for the PG. We are simply using it as a representative of other configurable protocols that a dedicated PG might use. On edge systems, however, it is a powerful and popular approach to improving throughput, at the expense of CPU overhead.

5.5 Performance Over Bottleneck Links

One of the promises of dynamic networks is the ability to allocate dedicated resources in application-specific amounts. For these networks to be viable, there must be mechanisms to insure that requests are reasonable in that resources are dedicated – if unused, they are wasted. Internet2's DCN service allows for allocations in 51Mb/sec increments. Phoebus can play a key role in adapting flows to "right-sized" circuit allocations. This is more difficult with end-to-end transport protocols connections.

To evaluate the performance implications of Phoebus in the face of a bottleneck link, we configured the testbed with a 500Mb/sec WAN link, with 40ms of latency. Again, the link is configured to have no loss, although the presence of a bottleneck clearly induces loss.

Figures 11, 12 and 13 show 600 second experiments with 9 Iperf[1] instances running from source to destination as background traffic. Throughput is reported at the TCP sink, via either Iperf or GridFTP. The background traffic's throughput over time is depicted with the colored lines. The black line shows the instantaneous throughput of the GridFTP transfer. Figure 11 shows GridFTP using 8 parallel TCP streams from end-to-end. In this case, the GridFTP streams clearly take more of the available bandwdith, but falls well short of

[1]Popular network measurement tool – see `http://iperf.sourceforge.net`

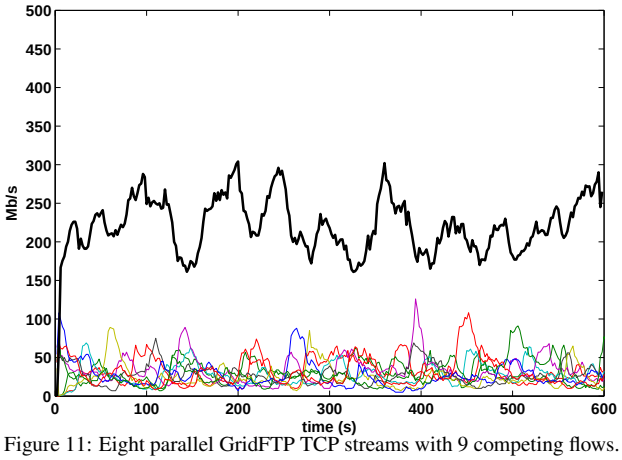

Figure 11: Eight parallel GridFTP TCP streams with 9 competing flows.

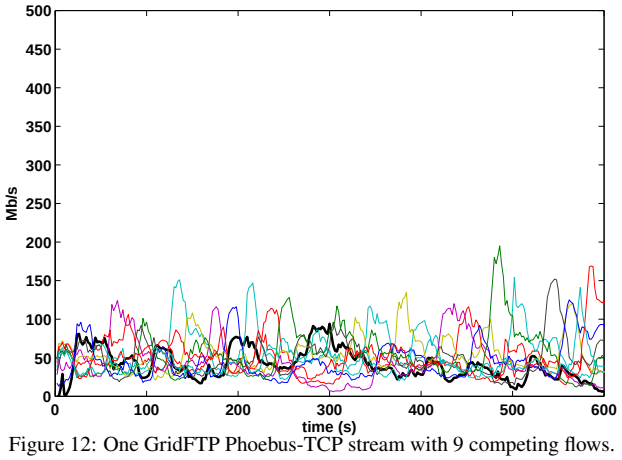

Figure 12: One GridFTP Phoebus-TCP stream with 9 competing flows.

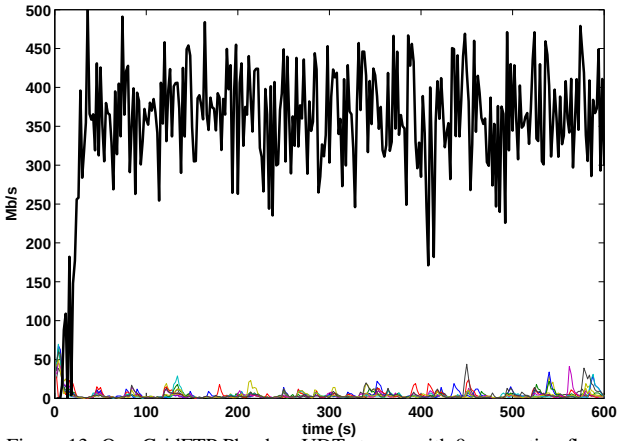

Figure 13: One GridFTP Phoebus-UDT stream with 9 competing flows.

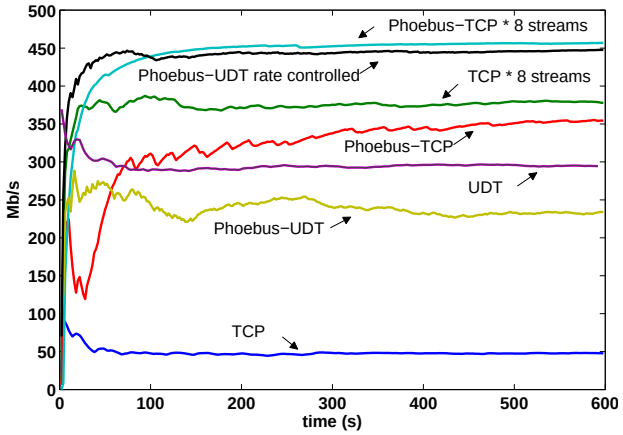

Figure 14: GridFTP transfers over 500Mb/s bottleneck with %.01 loss at edges.

saturating the 500Mbit/sec bottleneck. Figure 12 shows GridFTP using Phoebus with one TCP stream. Figure 13 shows GridFTP with TCP at the edges using UDT over the bottleneck link. This configuration is clearly able to make better use of the bottleneck link at the expense of the competing TCP flows.

In the circuit scenario, there will be no end-to-end competing cross traffic. There will be contention at the edges, but the circuit link will be dedicated. To capture this, we increased the loss on the LAN links to 0.01% to model cross-traffic and contention. We compared GridFTP transfers using one TCP and UDT connection and parallel transfers using 8 TCP streams, with Phoebus sessions using 1 and 8 TCP streams, and 1 UDT stream over the WAN link. Interestingly, Phoebus-UDT proved to be very unstable and performance suffered (indeed this instablilty is visible in Figure 14.) By sending traffic at the rate of the interface into a network that is less than interface speed, there are periods of loss. This phenomenon is common in hybrid networks. By rate-limiting UDT to the capacity of the dedicated circuit, we were able to come far closer to filling the circuit than 8 streams of TCP. Indeed this UDT configuration achieves performance on par with GridFTP making use of Phoebus and parallel streams.

5.6 Connect Latencies

While the bandwidth using Phoebus shows improvement in many cases, the results do not include the extra cost of the connect time. If the time to connect is excessive, the data transfer might finish earlier if the extra bandwidth obtained is more than offset by the connect time. To compare the connect times, we created a simple connect benchmark that would repeatedly connect to the end host and measure the amount of time each connect took.

For the tests, we ran 100 iterations. We tested the connect times for a direct connect, a connect using TCP as the inter-gateway protocol and a connect using UDT as the inter-gateway protocol. We also varied the distance of the network by changing the latency from 0ms (None), 50ms (Moderate), and 100ms (High). We did not add any loss as the chance of a loss occurring with such a short packet exchange is sufficiently remote as to be unlikely to have a major effect on the results.

As shown in Table 2, in all cases, the direct connection goes the fastest, as would be expected. When in very close proximity, the Phoebus connections using TCP are approximately 4 times slower, while the connections using UDT are an order of magnitude slower. Considering how quickly TCP can perform when the RTT is small, this shows that for very short distances Phoebus should not be used.

For the 50ms and 100ms distance tests, the Phoebus connect time is still slower, but not nearly as much so. At 50ms RTT, connects via Phoebus using TCP take between 12 and 25% longer than direct connections, and at 100ms, take between 7 and 10% longer. Connections using UDT are even slower than that, at between 2 and 4 times slower at either latency. This implies that UDT is more dependent on RTT than TCP, likely requiring more information to be sent back and forth at connect time.

In all these cases, the relative time of the Phoebus connections is notably higher than the direct connect case. The absolute time tells a much different story. In the worst case scenario for Phoebus over a path with some latency, the UDT connection took 4 times as long as the direct TCP connection. This translated to approximately 350ms time difference between the two. If the Phoebus connection is able to perform even 25% faster than the direct connection then, after 1.4 seconds, the Phoebus connection will have surpassed the direct transfer.

		Connect		Transfer	
Type	Latency	Mean (ms)	σ	Mean (ms)	σ
Direct	None	0.46	0.28	0.38	0.03
Phoebus-TCP	None	2.33	2.25	0.45	0.01
Phoebus-UDT	None	8.03	5.47	0.6	0.03
Direct	Moderate	64.02	0.31	64.04	0.4
Phoebus-TCP	Moderate	74.41	2.21	64.07	0.81
Phoebus-UDT	Moderate	143.33	26.17	64.04	0.41
Direct	High	114.38	3.27	114.6	3.57
Phoebus-TCP	High	125.27	3.49	112.16	0.94
Phoebus-UDT	High	245.32	48.45	112	0.03

Table 2: Connect and Transfer Latencies.

5.7 Transfer Latencies

Another useful metric to test is the processing overhead that the use of Phoebus adds to an end-to-end connection. If an application were heavily dependent on request-response style communication, the added latency of Phoebus could cause a performance slowdown. To quantify this, we created a benchmark to test the RTT for end-to-end connections. This application connects to the end host, either directly or via Phoebus, and repeatedly sends a single byte and waits for a single byte response. The sending of a single byte should provide the worst case scenario for Phoebus as any forwarding overhead will be charged to that single byte instead of amortized over a larger number of bytes. In all of these tests, we did not add any loss to the links since the chance of a loss occurring with such a short packet exchange makes the chances of it substantially affecting the outcome remote.

As shown in Table 2, for the case with no latency, the forwarding overhead is noticeable at between 15% and 33% slower for the TCP case and around 66% slower for the UDT case. When the hosts are further apart, however, this forwarding overheard becomes negligible. In all other cases, the differences between the direct versus Phoebus connections, no matter which protocol is chosen, are not significant.

6. SUMMARY AND FUTURE WORK

This paper presents a set of controlled experiments measuring the efficacy of the Phoebus system under a variety of network conditions. Specifically, we have shown how Phoebus with protocol adaptation can dramatically improve throughput over less than

ideal network conditions and provide dramatically improved single-stream application performance. This single stream performance is comparable with the best performance that parallel TCP can deliver when utilized over clean, low-latency paths. We developed and tested a Phoebus driver for use with GridFTP, which is perhaps the most widely-used system for high-performance file transfer in HPC and the Grid. The GridFTP results demonstrate that Phoebus can automatically improve network performance and serve as the basis for a GridFTP-based data transfer system that brings improved performance to end users with no tuning. We have also shown how Phoebus can more effectively utilize bottleneck links through path segmentation and protocol tuning, which is an increasingly prevalent scenario when Phoebus is used as a gateway to reserved capacity circuit networks.

Thus far we have shown promising results with protocol adaptation between TCP and more configurable protocols like UDT. Utilizing the abstraction layer in Phoebus and the session layer, we will include additional protocol functionality and investigate the performance of Phoebus with protocols designed for high-performance, dedicated and shared capacity links, in turn. We anticipate that Phoebus will provide further improvements when adapting and selecting protocols better suited for a specific network segment.

To be viable in modern networks, Phoebus must scale to 10Gb/s networks. Using a PG built with commodity hardware, we have observed local-area Phoebus forwarding over 9Gb/s with the current code. There are 10Gb Ethernet Phoebus nodes under experimentation in Internet2's network. We are also experimenting with network processor based hardware as well as programmable network interface cards. We are confident that the Phoebus model will scale to higher speeds.

Phoebus represents a significant change in the common models for using the network. In-the-network devices and protocol adaptation may never see use in the global Internet, although our approach bears similarity to ideas that explore rethinking the notion of traditional network models [8]. However, in a world of extreme performance demands, the Phoebus architecture addresses a very real need.

7. REFERENCES

[1] Enabling high performance data transfers. `http://www.psc.edu/networking/projects/tcptune/`.

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

[3] TCP tuning guide. `http://dsd.lbl.gov/TCP-tuning/linux.html`.

[4] Vfer: High performance data-transfer in user-space. `http://vfer.sourceforge.net/cgi-bin/index.cgi?page=home`.

[5] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link. The globus striped gridftp framework and server. In *Proceedings of the ACM/IEEE SC 2005 Conference*, page 54, 2005.

[6] A. Brown, E. Kissel, M. Swany, and G. Almes. Phoebus: A session protocol for dynamic and heterogeneous networks. UDCIS Technical Report 2008:334, `http://damsl.cis.udel.edu/projects/phoebus/phoebus_tech_report.pdf`.

[7] S. F. E. Kohler, M. Handley. Datagram congestion control protocol (DCCP). RFC 4340, March 2006.

[8] B. Ford and J. Iyengar. Breaking up the transport logjam. In *Proceedings of ACM HotNets*, 2008.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

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

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

[12] T. J. Hacker, B. D. Athey, and B. Noble. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 314, Washington, DC, USA, 2002. IEEE Computer Society.

[13] E. He, J. Leigh, O. Yu, and T. A. DeFanti. Reliable blast UDP:predictable high performance bulk data transfer. In *Proc. of theIEEE Cluster Computing 2002*, pages 317–324, 2002.

[14] K. Kaneko and J. Katto. TCP-fusion: A hybrid congestion control algorithm for high-speed networks. In *International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet)*, pages 31–36, 2007.

[15] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. Computer Communications Review, 27(3), July 1997.

[16] A. P. Mudambi, X. Zheng, and M. Veeraraghavan. A transport protocol for dedicated end-to-end circuits. In *Communications, 2006. ICC '06. IEEE International Conference on Communications*, pages 18–23, 2006.

[17] L. Ong and J. Yoakum. An introduction to the stream control transmission protocol (SCTP). RFC 3286, May 2002.

[18] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[19] M. Swany. Improving Throughput for Grid Applications with Network Logistics. In *Supercomputing 2004*, November 2004.

[20] K. Tan, J. Song, Q. Zhang, and M. Sridharan. Compound TCP: A scalable and TCP-friendly congestion control for high-speed networks. In *International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet)*, 2006.

[21] F. Vacirca, A. Baiocchi, and A. Castellani. Yeah-TCP: yet another highspeed TCP. In *International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet)*, pages 37–42, 2007.

[22] Web100. http://www.web100.org.

[23] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: Motivation, architecture, algorithms, performance. In *EEE/ACM Transactions on Networking (TON) Volume 14, Issue 6*, pages 1246–1259, 2006.

[24] Z. Zhang, G. Hasegawa, and M. Murata. Reasons not to parallelize TCP connections for long fat networks. In *Proceedings of SPECTS 2006*, 2006.

[25] X. Zheng, A. P. Mudambi, and M. Veeraraghavan. FRTP: Fixed rate transport protocol – a modified version of SABUL for end-to-end circuits. In *Proceedings of the 1st International Workshop on Provisioning and Transport for Hybrid Networks (PATHNETS)*, 2004.