Phoebus: A system for high throughput data movement

Ezra Kissel\textsuperscript{a,}\textsuperscript{*}, Martin Swany\textsuperscript{a}, Aaron Brown\textsuperscript{b}

\textsuperscript{a} Department of Computer & Information Sciences, University of Delaware, 18 Amstel Avenue, Rm. 101, Newark, DE 19716, United States
\textsuperscript{b} Internet2, 1000 Oakbrook Drive, Suite 300, Ann Arbor, MI 48104, United States

\begin{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. We have developed interfaces to Phoebus to allow its use in various real applications and data movement services. This paper extends our earlier work with tests of Phoebus-enabled applications on both real-world networks as well as over configurable network testbeds that allow us to modify latency and loss rates. We demonstrate that Phoebus improves the performance of bulk data transfer in a variety of network configurations and conditions.
\end{abstract}

\section{Introduction and motivation}

Despite continuing advances in the link speeds of networks, data movement remains a key problem in parallel and distributed computing. Applications in both science and industry are becoming increasingly data intensive. The viability of many distributed computing paradigms depends on the ability to have data transfer speeds scale up as computing power increases. This paper describes and investigates the efficacy of a network middleware system for improving data transfer performance for data-intensive applications and systems.

Despite the protocol advances and heroic network performance achieved through the years, most users of advanced distributed computing environments struggle with adequate network performance. There are many sources of information about network tuning available online, and certainly there are many published techniques to improve network performance, but the so-called "wizard gap" remains — achievable network performance fails to scale up with network speed. In some sense, the fact that a high performance distributed system user needs to understand host tuning and things like protocol windows points to a failure of the network ecosystem. For example, scientists, computer or otherwise, do not need to understand Coulomb's law to plug our systems into power outlets or the Nyquist–Shannon sampling theorem to make a telephone call.

Phoebus provides a path to solving this critical problem. Phoebus is an infrastructure for automatically 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 reserve network resources, to use segment-specific transport protocols between gateways, as well as to apply other performance-improving techniques on behalf of the user. The Phoebus system addresses network performance issues data intensive applications by assisting these applications with resources "in the network".

A new protocol or middleware will only see widespread adoption if users can easily integrate it into their existing systems and applications. We have previously implemented a driver which allows GridFTP\cite{14} to automatically use Phoebus when available. More recently, we have collaborated with the REDDnet\cite{34} group to provide support for Phoebus within their data logistics software suite. This integration allows the Phoebus architecture to be used in a configurable and transparent manner to assist with large-scale data movement between storage depots within the REDDnet network.

In our earlier work\cite{22}, we demonstrated the achievable performance gains of Phoebus in a testbed environment using GridFTP over a variety of simulated network conditions. We showed how Phoebus can effectively use alternate transport protocols to improve file transfer throughput. The performance improvements are the most dramatic when considering less than optimal network conditions and when bottleneck links are present, e.g. while simulating dynamic networks with reservable bandwidth. Unfortunately, our Phoebus experiments at the time
were confined to a testbed environment only and limited to 1 Gb/s network speeds, leaving scalability beyond 1 Gb/s and demonstrable performance gains on real networks as future work.

This paper expands upon our previous results and provides a new set of simulated and real-world experiments while contrasting against our earlier results. In addition, we provide an overview of the existing Phoebus architecture and describe additional features that allow for the efficient and transparent use of an increasingly heterogeneous network landscape. The detailed contributions of this paper are:

1. Describing the Phoebus system: We describe the Phoebus middleware architecture, which includes the notion of session-layer framing and the intelligent segmentation of the end-to-end network path. A session layer protocol provides a number of benefits in terms of network adaptability and configurability. We describe 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.

2. Extending applications and libraries to provide Phoebus support: We describe how Phoebus may be used with existing applications, focusing on the integration of GridFTP and REDDnet tools which allow them to take advantage of the Phoebus architecture. Our performance testing compares data transfer throughput when utilizing Phoebus against existing approaches.

3. Presenting testbed results of Phoebus performance in a variety of network conditions: We employ the widely-used GridFTP data transfer tool to demonstrate how Phoebus can improve performance over a variety of network paths as well as how single stream Phoebus transfers are able to outperform parallel streams in many cases. We show that Phoebus is able to scale up to higher network capacities by examining performance on both 1 Gb/s and 10 Gb/s networks.

4. Examining 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.

5. Demonstrating real-world performance gains: We show how Phoebus-enabled applications achieve better throughput via a number of real-world experiments.

The remainder of this paper is organized as follows: Section 2 will briefly discuss some background and Section 3 will provide an overview of the Phoebus system. Details of Phoebus application integration are outlined in Section 4, Section 5 will discuss the effects of transport layer protocols and give an overview of how Phoebus can translate between them. We will present real-world results from a number of use-cases in Section 6. Finally, we will present experimental results based on our testbed in Section 7 and conclude in Section 8.

2. Background

For decades, the end-to-end argument [36] has provided the conceptual basis for transport protocols. The common interpretation of this argument states 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 of 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 many heuristics have been proposed to improve the ability of transport protocols to discern congestion related loss and react accordingly.

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. Reliable backbone networks with low loss and high latency, combined with shared access networks with some loss and low latency, can interact to cause performance problems. A typical connection may cross multiple domains and these interactions affect the throughput of the connection. The current Internet model must simply ignore the differences in networks and trust the end nodes to address them as best they can.

Due to these factors, the ubiquitous Transmission Control Protocol (TCP) is known to have performance issues. There is a tremendous body of research [19,24,27,10,31], too vast to properly cite here, devoted to understanding and improving TCP’s performance. The performance effects are particularly significant in long-distance, high-bandwidth networks. While there have been countless proposals to change TCP, none have been a panacea.

3. Phoebus architecture

Phoebus [8] is a system that implements a new protocol and associated forwarding infrastructure for improving throughput in today’s networks. The current Internet model hides 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) located at strategic locations in the network take responsibility for shepherding users’ data to the next PG in the path, or to the destination host, as depicted in Fig. 1.

The Phoebus network “inlay” allows us 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. Phoebus bundles a variety of tuning and adaptation into a networked data movement service. As mentioned, 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. In that spirit, the Phoebus system offers a way to offload network tuning to network experts and the network itself.

3.1. Related work

There are many approaches that attempt to address the problem with TCP mentioned in Section 2. Higher-speed variants of TCP [26,43,11] aside, the idea of “splitting” TCP has been explored in e.g. I-TCP [4], which spoofs TCP connections to deal with loss over wireless connections. Performance enhancing proxies (PEPs) are described in RFC 3135 [7] and are commonly used to handle the issues that occur when using TCP over wireless or satellite links. However, the focus of PEPs has been to address

---

1. Phoebus is a descendant of the Logistical Session Layer (LSL) [38], which used a similar protocol.
performance problems on specific, less common links and they do not propose a general solution, nor are they targeted at bulk data movement.

Phoebus is similar in spirit to work in application level routing (or overlay networks) and non-default route selection [37, 2, 33]. That work has addressed a number of issues including route asymmetry and optimal, or parallel, route selection. Phoebus differs in that it is a vehicle for bulk data transfer in research and education networks rather than a workaround for ineffective Internet routing policies, or a way to "route around congestion".

The high-performance computing and networking community has circumvented the TCP problem in two major ways. The first is the use of parallel TCP streams [17, 44]. The second is with user-space protocols such as UDT [15] that take advantage of the User Datagram Protocol (UDP) for data transfer and circumvent TCP altogether. These approaches place the effort required to improve performance into the user and application domain.

3.2. Dynamic networks

More and more research and education network providers are deploying network reservation technology that allows network bandwidth to be dynamically allocated. We define Dynamic Networks (DNs) to include all networks in which dynamic signaling can be used to allocate or reserve bandwidth, or to negotiate some other Quality of Service (QoS) metric. These DNs are useful in support of high-performance and Grid computing applications that demand network capacity. These networks complete the on-demand computing landscape by doing for networks what the grid paradigm does for computing and storage. Phoebus is a key technology for enabling broad access to these DNs in that it provides a seamless mechanism to bridge the gap between the traditional shared packet environment and on-demand network segments.

Dynamic networks, in particular "circuit" or "bandwidth-on-demand" networks, are being deployed by major research and education networks like Internet2, the US Department of Energy's ESnet, and GÉANT3 in Europe. These networks are developing a compatible signaling interface, based on the OSCARS [16] system, that allows the 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 DN without provisioning a circuit to every end host. This also allows simplicity at the edge of the network, with the PG acting as a negotiation and translation agent. DNs also present a situation in which the inherent heterogeneity of the network is very apparent — some portions of the network are dedicated and far less subject to loss.

The Phoebus platform is currently being deployed on the Internet2 Network and will enable users to automatic access its dynamic network. Phoebus 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.

4. Application use and integration

The introduction of a new protocol and network system necessitates the means to enable their use in existing applications. To that end, we have developed two methods that allow applications to make use of Phoebus that require no changes in existing code: a Phoebus wrapper library and transparent redirection. The wrapper library overrides the standard operating system socket calls so that any application linked against it can transparently use the Phoebus infrastructure. The application can be explicitly linked against the library at compile time or it may enable the wrapper by setting the LD_PRELOAD environmental variable before executing the application. Using transparent redirection involves taking advantage of the firewall capabilities in Linux and transparently redirecting TCP connections destined for specific hosts or services through the Phoebus architecture. Additional details on these implementations may be found in our earlier work [8].

We recognize that there are several cases where transparent operation proves cumbersome to effectively use existing applications. In most cases it is ideal for applications to be modified to ensure that they can make use of the new features offered by the protocol and have finer-grained control as to when and how it is used.

The Globus Toolkit’s [13] GridFTP [14] is an example of where “smarter” use of Phoebus requires more integration. GridFTP’s status as a key application in the Grid computing environment suggested a closer tie-in to allow users to easily use the Phoebus infrastructure in their data transfers. We have since focused on providing Phoebus support to additional data-intensive projects such as REDDnet using the Phoebus client library, discussed in further detail below.

4.1. A Globus XIO driver for Phoebus

Globus XIO is an extensible input/output library within the Globus Toolkit [13]. 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 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. Any 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.

4.2. REDDnet and Phoebus

The REDDnet project is designed to provide a large distributed storage facility for data intensive collaboration. REDDnet is a 700TB deployment of Logistical Networking [5] infrastructure designed to be shared by several application domains using the concept of “working storage” to help manage the movement of large amounts of storage over wide area networks. With REDDnet supporting Tier-1 data movement from projects such as the CMS collaboration at CERN (LHC), the adoption of performance-enhancing middleware systems like Phoebus promises to show great potential.

The REDDnet data dispatcher has been modified to take advantage of Phoebus for its data movement operations. Data movement is performed via third party transfer between REDDnet depots at the behest of the data dispatcher and decisions it makes about the approximate best use of available resources. By extending the underlying protocol used by the dispatcher, Phoebus routes may be statically or programmatically configured. The routes may also be parsed from a memory mapped routing table file that in turn encodes an extended data copy operation with the Phoebus path to use, providing the basis for data movements between depots.

In addition to the dispatcher, many lower-level and user-space tools have been extended and linked against the Phoebus client library providing native support. These tools provide a set of command line and web-based file download clients, as well as performance benchmarks, that can now easily take advantage of Phoebus with a simple enabling option.

5. 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.

As mentioned in Section 2, an end-to-end transport protocol 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), 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 will 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 [28,45,18]. 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 better, 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 in this paper as it has shown good performance in 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 UDT, facilitating more direct comparison.

5.1. 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 [25,32]. 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 [39,20,40,43]. 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 transport-layer techniques to be utilized sooner.

5.2. 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 [15,18,41]. Non-TCP protocols implemented in the operating system kernel do exist [23,30], 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 on 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 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 5.3, 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.

5.3. 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 daemon via a callback function.

The other form of abstraction is the object representing an open connection. This object provides a consistent interface no matter 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 [42] 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.

5.4. UDT adaptation

We created an implementation of the abstraction layer for UDT, as described above. As mentioned, UDT is a protocol from the University of Illinois designed for transfers over wide-area, high-speed networks [15]. 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 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 protocol data units (S-PDUs). This introduces a header for each PDU, 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 PDUs. If it is a shutdown message, the header contains a field describing which direction, reading or writing, is being shutdown by the PDU’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 S-PDU, 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 S-PDU consisting of a header and the encapsulated data. When the PG needs to shutdown one side of a connection, it creates a new shutdown PDU and sends it with no payload.

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

Recall Fig. 1, which illustrates an end-to-end connection in the Phoebus model. The transport protocol between the PGs need not be the same as that on the edges, so it is easy to see how we can use UDT protocol adaptation between 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.

5.5. Phoebus compared with other approaches

As mentioned previously, improving the performance of bulk data transfers has primarily been accomplished in three ways: (1) striping data over parallel TCP streams, (2) taking advantage of user-space protocols such as UDT and (3) employing TCP protocol variants suited for high-speed and long distance transfers. Applications must be designed to take advantage of the first and second approaches, while the third approach requires that host protocol stacks be explicitly configured and tuned to achieve benefits. The use of TCP variants and protocol tuning has been extensively studied and is not reproduced here in our analysis of Phoebus performance. Instead, our results focus on how Phoebus improves the performance of existing applications over a number of network environments. We will note, however, that Phoebus is able to take advantage of TCP variants when necessary by adapting and optimizing protocols at PGs in the network thereby removing this burden from end users.

A number of applications (notably GridFTP) take advantage of parallelism by striping data over multiple, concurrent TCP connections with each sharing some portion of the available bandwidth. The benefits come from smaller window sizes for N parallel streams (leading to faster recovery times) and the reduced severity of a loss event by a factor of 1/N. This approach can substantially improve application throughput but often at the expense of competing flows, raising questions about fairness. In addition, this solution will typically not scale well beyond 10 Gb/s where parallel streams incur a costly overhead for the application. Simply adding more streams provides diminishing returns and can lead to a reduction in overall performance [1]. One of the goals of Phoebus is to reduce the dependence on parallel streams while still supporting parallelism when useful. The following results and performance analysis compares the benefits for applications that utilize both approaches while allowing each to compete fairly.

We have described how user-space transport protocols have been employed to increase performance for transfers over long-haul networks. This technique allows applications to either directly implement or programmatically select the protocol at run-time and avoid the well-known pitfalls of using TCP over long, fat networks. This approach also has its disadvantages: a more aggressive protocol may not be suitable for use over shared network links and there is the issue of high overhead for the host system, often limiting the effectiveness of such protocols well below 10 Gb/s. The user-space protocol adaptation done in Phoebus allows the use of such protocols on network segments where appropriate (e.g. WAN links) while allowing applications to transparently take advantage of the performance gains. We focus on UDT in this paper to compare it directly with the XIO implementation available with GridFTP.

Additionally, work has been done in performance enhancing overlays implemented directly within the Globus GridFTP architecture [35,21]. These overlays utilize similar techniques to Phoebus (e.g. splitting TCP connections) but are tied to the GridFTP server implementation and do not propose a general architecture for enhanced data movement. Our analysis of TCP–TCP adaptation performed by Phoebus in a realistic environment demonstrates comparable or better performance achieved by these overlay approaches while providing additional flexibility and optimization through the session layer.

6. Real-world testing

This section presents two different real-world experiments: The first demonstrates performance gains using Phoebus with a common network performance benchmarking tool between locations on the East and West coasts of the US. The second examines how Phoebus improves transatlantic data transfers between REDDnet storage sites at CERN, Switzerland and locations in the US. Both experiments utilize segments of Internet2’s IP backbone network and focus on TCP adaptation between PGs.

6.1. Internet2 experiments

We tested the effectiveness of Phoebus using the prototype Phoebus infrastructure available on Internet2. This provided us with the opportunity to observe how the Phoebus concepts played out in the real world as well as to see how well it would work for the users of Internet2’s Phoebus service.

6.1.1. Test setup

Internet2’s Phoebus infrastructure consists of gateway software running on machines located at the various Internet2 Points-of-Presence (POP). For the purposes of this test, we used the machines located in Los Angeles, Chicago and New York. These machines are Dual Processor Pentium 4 Xeons with 4G of RAM and a single 10 Gb/s S2io NIC connecting it to the outside world. The backbone network that they are all connected to is a 10 Gb/s network and is the same network used by regular Internet2 traffic.

The systems we used included a machine from SDSC whose link to Internet2 passed through the Los Angeles POP and a machine from Columbia whose link to Internet2 passed through the POP in either New York City or Chicago, depending on its destination. If the machine in Columbia were connecting to a cross-country host, its path would take it through Chicago. If the destination was to the east coast, the path would take it through New York. Each of these machines had a 1 Gb/s connection to the outside world.

For this experiment, we performed 74 tests. Each test consisted of running iperf2 between each node in the test for a duration of 120 s. This allowed us to see the performance of each link to give us an idea of how they affected the overall transfer rates. The

Phoebus portion of each test consisted of two separate runs. One used Chicago as the edge POP and the other used New York City as the edge POP. The run using Chicago as the POP was necessary to be able to make valid comparisons between a direct connection and Phoebus. Including New York City in the comparison allows us to see how well the system can perform using the best route available, an option not actually available using direct connections. In all of these tests, the data flowed from SDSC on the west coast to Columbia on the east coast.

### 6.1.2. Throughput results

Table 1 gives the observed bandwidth for tests performed along the entire end-to-end path as well as each segment along the path. Performing a direct connection from SDSC to Columbia, we were able to obtain 243 Mb/s. However, by enabling iperf to use Phoebus through the transparent wrapper library, keeping all other variables the same, we were able to obtain a transfer rate of 382 Mb/s, an improvement of 57%.

The results support the notion that a Phoebus connection is hindered by the slowest link in the path [38]. This limitation means that the end-to-end connection performs somewhat slower than the speed of the slowest link. In the connection between SDSC and Columbia using Chicago as the egress point, the slowest link is the link between the Chicago PG and Columbia which runs at around 419 Mb/s. In this case, Phoebus performs close to the available link capacity at 383 Mb/s. The path using New York City as the egress point provides a completely different view. In that path, we found the slowest link to be the connection between SDSC and the Los Angeles PG. The overall Phoebus connection using this path also performs within a reasonable margin below the slowest link at 715 Mb/s.

To determine the effects of various transfer sizes on overall bandwidth, we ran a series of iperf tests with transfer sizes between 32M and 4G bytes. Fig. 2 shows the performance difference between the direct transfers and those utilizing the Phoebus infrastructure. These tests were also run between SDSC and Columbia using the default egress point at the Chicago POP and exhibits similar performance along that path as shown in Table 1. For small transfers, using Phoebus provides an increase of around 300% over direct transfers. As the amount of data transferred increases, the initial TCP connection startup is amortized over the length of the transfer and we see a Phoebus performance improvement between 50% and 60% up to 4G bytes.

Using the Phoebus infrastructure as configured, a typical user would have been able to achieve an approximately 50%–60% increase in throughput assuming the default path was available during a sustained transfer. Transferring smaller amounts provides an even larger benefit. Achieving these results required no special knowledge of the network as the PGs are configured to dynamically determine the appropriate egress POP. Using Phoebus with a global directory service to provide information on alternate available routes, or by specifying the route directly, the faster path through New York City could have been selected. This modification would have obtained a 293% improvement in throughput overall.

### 6.2. REDDnet experiments

REDDnet deploys “storage depots” at a number of sites located within the US as well as at CERN to support its distributed data storage infrastructure. At any particular site, there are a number of individual hosts configured as depots that then advertise their storage space as part of the REDDnet network. Each of these systems is 1G-connected and is typically configured with a dual-core AMD Athlon X2 processor, 4 GB of RAM and running a recent 2.6 Linux kernel.

We utilized a benchmark called ibp_perf that has been integrated with the Phoebus client library for the purposes of these tests. The ibp_perf tool provides a command line interface to simulate data copying between depots using the same protocol and configurable options as the REDDnet data dispatcher, thus ensuring an accurate measurement of expected performance. A typical copy operation between two depots involves a number of simultaneous data transfers per node in order to take advantage of the existing benefits in using parallel TCP streams. Our figures show runs of ibp_perf while using 16 parallel streams per node for a total of 64 parallel TCP connection in each direction. We also varied the total transfer size per node between 16 MB and 4 GB for each test and our throughput results are aggregated over the total number of nodes used between sites.

Fig. 3 compares transfers of increasing size between 4 depots at CERN and Vanderbilt using direct connections with those utilizing a specified Phoebus path. Since there are 4 1G-connected depots on each end of the transfer, the total available bandwidth is limited to 4 Gb/s. The ibp_perf benchmark supports testing both “read” and “write” performance, which is reflected in our results. This indicates the direction of the transfer with CERN as the testing site. A “read” denotes that the data flows from Vanderbilt to CERN, and the opposite for a “write”. The Phoebus path taken is CERN-Chicago-Atlanta-Vanderbilt, utilizing the PGs installed at Chicago and Atlanta Internet2 POPs.

Using Phoebus provides a significant increase in performance for both the “read” and “write” cases. With a 154 ms RTT between CERN and Vanderbilt, the connections take a considerable amount of time to achieve good performance. At between 2 and 4 GB of data transferred, the Phoebus case provides over 100% improvement in “write” performance and close to 50% improvement for “reads”. There is some noticeable asymmetry in the path, although the Phoebus case helps close the gap that increases between “reads” and “writes” in the direct transfers.

Fig. 4 shows a similar graph as above but here we tested from CERN to the University of Michigan REDDnet site. The
Phoebus path was also different in this test. We were given access to a temporary PG in Amsterdam to use for testing purposes during our experiments and our Phoebus path for this test was CERN-Amsterdam-Chicago-UMich. Unfortunately, the link between Amsterdam and CERN was limited to 3 Gb/s due to administrative reasons and that fact is reflected in our results. Again, the Phoebus case is able to outperform the direct TCP connections by a similar margin in the "write" test and exceeds the throughput of the "read" tests. At 2–4 GB transferred, the Phoebus case performs only slightly better than the direct case, which we speculate is due to an asymmetry in the link between Amsterdam and CERN.

These results show the efficacy of Phoebus in improving the performance of a data intensive application such as REDDnet where managing many parallel streams of data and overcoming high network latencies can severely impact the achievable throughput using traditional means. We are working with the REDDnet group to more closely integrate Phoebus into their developmental software suite and expect to achieve further performance gains as additional network testing becomes possible.

7. Testbed results

Our goal in this section is to test a real data transfer tool, GridFTP, using the Phoebus infrastructure in a variety of network conditions. Despite the availability of a prototype Phoebus infrastructure in Internet2 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 difficult as we can attest to in gathering our real-world results above. 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.

The Linux kernel has a module available called netem [29] 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. Both of our testbeds consisted of seven machines connected as depicted in Fig. 5. 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 suggests 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 1G-connected network testbed consisted of Intel Pentium 4 Xeon 2.8 GHz HT CPUs, 4 GB RAM and 1 Gb/second Ethernet links. While it may seem that 1 Gb/s Ethernet is rather pedestrian in this age of 10Gb/s Ethernet, we again note that most of the day-to-day high-performance distributed computing for scientific applications today is well below 1 Gb/s. In addition, the prototype Phoebus service on Internet2 uses 1 Gb/s Ethernet for the most part, and providing a reliable data transfer service at this rate is quite significant.

With the acquisition of new, more capable nodes and Myricom NICs, we were able to construct a 10 Gb/s Ethernet testbed with an identical layout and configuration as our 1 Gb/s testbed with which to evaluate the scalability of Phoebus. The edge and netem nodes consisted of single and dual quad-core AMD Opteron CPUs, and 4–8 GB of RAM while the PGs were quad-core AMD Phenom II processors with 8 GB of RAM. These system specifications helped ensure that we would not run into many hardware limitations while testing applications at 10 Gb/s speeds, although 10 Gb/s performance is certainly attainable with less capable hardware.

7.1. Achieving scalability with Phoebus

Software routing and forwarding performance depends greatly on a number of hardware factors including CPU clock cycles and number of cores, memory type and speed, as well as the system bus architecture. In particular, the effects of memory latency and
shared bus architectures on software routing implementations are well documented in [9,36]. While scaling hardware over time to keep pace with new technologies is an inescapable necessity, a number of intelligent software enhancements may also be employed to improve performance.

To that end, modifications to the forwarding backend used by the initial Phoebus implementation were required in order for Phoebus to effectively scale up to 10 Gb/s speeds across a number of platforms. Our original backend employed a traditional send/recv loop with a configurable buffer that required memory copies between kernel and user address space. Depending on the deployed hardware, reasonable performance at 10 Gb/s speeds was frequently unobtainable especially on lower-end systems.

To overcome this limitation, we took advantage of the splice system call that has reached maturity in recent Linux kernel releases. The splice call uses a pipe to move data between two file descriptors. On Unix-like systems, a pipe is effectively implemented as an in-kernel memory buffer and the splicing of data through this buffer avoids expensive userspace memory copies when moving data between network sockets. With a newly implemented and configurable forwarding backend based on splice, we have seen a number of significant performance gains both in terms of total throughput and lowered CPU utilization. Table 2 demonstrates single TCP stream performance observed between the Phoebus send/recv and splice forwarding backends when tested on 10 Gb/s networks deploying otherwise capable hardware. Due to the scarcity of 10 Gb/s capable edge hosts, the Internet2 tests were performed between backbone systems solely to demonstrate the improvements in the Phoebus forwarding backend.

<table>
<thead>
<tr>
<th></th>
<th>Direct</th>
<th>Phoebus-sendrecv</th>
<th>Phoebus-splice</th>
</tr>
</thead>
<tbody>
<tr>
<td>10G Testbed (Gb/s)</td>
<td>9.9</td>
<td>8.6</td>
<td>9.9</td>
</tr>
<tr>
<td>10G Internett2 (Gb/s)</td>
<td>9.2</td>
<td>7.4</td>
<td>9.1</td>
</tr>
</tbody>
</table>

7.2. Experimental configuration

For our experiments, we tested LAN packet loss rates of 0.001, 0.01, and 0.1, with WAN latencies of 25 ms, 50 ms, 100 ms and 150 ms. We also induced 4 ms of latency for each LAN segment to simulate latencies over edge networks. On the 10 Gb/s testbed, we increased the netem queue size appropriately to accommodate the higher interface rate. 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 25 ms to 75 ms within the continental United States and exceed 100 ms on transcontinental links. Our set of experimental cases represent a set of conditions that can be expected on current real-world networks.

Each system in our 1 Gb/s testbed was running a vanilla 2.6.26 Linux kernel with web100 patches while the 10 Gb/s testbed ran a more recent 2.6.32 kernel in order to provide the most recent driver support. 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.

Table 2 Phoebus forwarding backend comparison.

Over each of these 1 Gb/s 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.

We performed identical measurements over our 10 Gb/s testbed with the exception of the UDT tests to demonstrate that Phoebus is able to scale at these speeds. At 10 Gb/s speeds, UDT becomes an impractical solution in terms of the amount of overhead incurred due to userspace copying. Testing the 10 Gb/s direct UDT performance with the GridFTP UDT driver delivered no more than 2.6 Gb/s while fully taxing the GridFTP server severely limiting its effectiveness for most applications. We are investigating additional low-level protocol adaptations for Phoebus that take advantage of the programmable nature of 10 Gb/s network interface cards.

7.3. Throughput results at 1 Gb/s

Our 1 Gb/s test results demonstrated the efficacy of Phoebus in realistic wide-area configurations. To compare with our more recent 10 Gb/s results, we reproduce the 100 ms latency graphs here. Figs. 6 and 7 show the performance of direct, single and parallel stream tests along the comparable Phoebus tests. In these 100 ms latency cases, Phoebus-UDT clearly outperforms the other configurations, including that of direct UDT, demonstrating the effectiveness of protocol translation. In low loss environments, parallel Phoebus-TCP performs worse than parallel TCP for short duration transfers due to additional connect latencies incurred when using Phoebus (see Section 7.7). This initial performance penalty becomes negligible as transfer times increase.

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.

7.4. Throughput results at 10 Gb/s

At 10 Gb/s speeds, we see comparable relative performance with the 1 Gb/s results and a noticeable improvement in
Fig. 6. 1 Gb/s performance at 100 ms WAN Latency, 30.001 LAN loss.

Fig. 7. 1 Gb/s Performance at 100 ms WAN Latency, 30.01 LAN loss.

Phoebus-TCP performance. Figs. 8 and 9 show the consistently good performance of Phoebus-TCP with 8 parallel streams and single-stream Phoebus-TCP providing a dramatic improvement over single-stream direct TCP when loss is a factor. In both cases, Phoebus-TCP is competitive or better than the direct case with 8 parallel streams.

Fig. 10 demonstrates how many parallel GridFTP streams can limit achievable throughput with both single-stream cases outperforming the parallel streams cases at 25 ms latency and no loss. With the more capable hardware and reduced overhead provided through the splice backend implementation, Phoebus can show performance gains compared to direct transfers across high-speed, low latency links.

7.5. CPU utilization

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

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. Table 3 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.

### 7.6. 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 DN service allows for allocations in 51 Mb/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 500 Mb/sec WAN link, with 40 ms of latency. Again, the link is configured to have no loss, although the presence of a bottleneck clearly induces loss.

Figs. 11–13 show 600 s experiments with 9 *iperf* 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. Fig. 11 shows GridFTP using 8 parallel TCP streams from end-to-end. In this case, the GridFTP streams clearly take more of the available bandwidth, but falls well short of saturating the 500 Mb/sec bottleneck. Fig. 12 shows GridFTP using Phoebus with one TCP stream. Fig. 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.

### 7.7. Connect latencies

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 instability is visible in Fig. 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.

While the bandwidth using Phoebus shows an improvement in many cases, the results implicitly include the extra cost of the connect time. Intuitively, if the time to connect is excessive, the data transfer might finish earlier if the extra bandwidth
Tables4providesconnectlatenciesforbothasinglePGandtwo
and10Gb/timeeachconnecttook.Thebenchmarkwasrunonbothour1Gb
repeatedlyconnecttotheendhostandmeasuretheamountof
connecttimes,wecreatedasimpleconnectbenchmarkthatwould
ConnectandTransferLatenciesat1Gb/s.

table 4
betweenPhoebusGatewaysduringconnectionestablishmentin
to the additional session protocol negotiation that takes place
aslongastheinter-gatewayWANlatency.Thisisprimarilydue
usingTCPovermorethanonePGtakeapproximately3times
PGsasconfiguredinour10Gb
beexchangedatconnecttimeandhavingmoreoverheaddueto
dependentonRTTthanTCP,likelyrequiringmoreinformationto
suchsituationsisdiminished.
Inallthesecases,therelativetimeofthePhoebusconnectionsis
notablyhigherthanthedirectconnectcase.However,thearchitecture
timetellsamuchdifferentstory.Intheworstcasescenariofor
2PGsoverapathwithhighlatency,thePhoebusconnectiontakes
3timesaslongasthe directTCPconnection.This translates to
approximately210msdifferencebetweenthetwo.IfthePhoebus
connectionisabletoperformeven25%fasterthandirect
connectionthen,afteraround1s,thePhoebusconnectionwill
have surpassed the direct transfer.

7.8. Transferlatencies

Anotherusefulmetrictotestisthe processing overhead that
theuseofPhoebusaddstoanend-to-endconnection.Ifan
applicationwereheavilydependentonrequest–responsestyle
communication,theaddedlatencyofPhoebuscouldcauseaperformance slowdown.Toquantifythis,wecreatedabenchmark
to test the RTT for end-to-end connections. This application
connectstotheendhost,eitherdirectlyorviaPhoebus,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
singlebyteinsteadofamortizedoveralargernumberofbytes.Inall
of these tests, we did not add any loss to the links since the chance
ofalossoccurringwithsuchashortpacketexchangemakesthe
ofthesetestswedidnotaddanymizontothe linksinsincethechance
ofa loss occurring with such a short packet exchange makes the
chancesofitsubstantiallyaffectingtheoutcome remote.

As shown in Tables 4 and 5, in all cases, the direct connection
performs the fastest, as would be expected. When in very close
proximity, the Phoebus connections using TCP vary depending on
the latency of the inter-gateway path. With no induced latency,
the Phoebus connect time is approximately 4 to 6 times that of
the direct connection. Considering the good performance of TCP
on low latency, low loss links, the suitability of using Phoebus in
such situations is diminished.

Connections using UDT at 1 Gb/s are even slower, at between
2 and 4 times slower at any latency. This implies that UDT is more
dependent on RTT than TCP, likely requiring more information to
beexchangedatconnecttimeandhavingmoreoverheaddueto
running in userspace.

Fig. 5 provides connect latencies for both a single PG and two
PGs as configured in our 10 Gb/s testbed. Connects via Phoebus
using TCP over more than one PG take approximately 3 times
as long as the inter-gateway WAN latency. This is primarily due
to the additional session protocol negotiation that takes place
between Phoebus Gateways during connection establishment in
addition to the overhead incurred from establishing the underlying
transport layer connections. This connect latency when using
Phoebus accounts for the lower performance numbers during short
transfers, as can be seen in some of the testbed results over the
10–20 s intervals.

Inallthesecases,therelativetimeofthePhoebusconnectionsis
notably higher than the direct connect case. However, the absolute
timetellsamuchdifferentstory.Intheworstcasescenariofor
2PGsoverapathwithhighlatency,thephoebusconnectiontakes
3 times as long as the direct TCP connection. This translates to
approximately210msdifferencebetweenthetwo.IfthePhoebus
connectionisabletoperformeven25%fasterthandirect
connectionthen,afteraround1s,thePhoebusconnectionwill
have surpassed the direct transfer.

Table 4
Connect and Transfer Latencies at 1 Gb/s.

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

Fig. 14. GridFTP transfers over 500 Mb/s bottleneck with 5.01% loss at edges.

Table 5
Connect and transfer latencies at 10 Gb/s.

<table>
<thead>
<tr>
<th>Type</th>
<th>Latency</th>
<th>Connect Mean (ms)</th>
<th>Connect σ</th>
<th>Transfer Mean (ms)</th>
<th>Transfer σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>None</td>
<td>0.24</td>
<td>0.00</td>
<td>0.25</td>
<td>0.01</td>
</tr>
<tr>
<td>1PG Phoebus-TCP</td>
<td>None</td>
<td>0.84</td>
<td>0.01</td>
<td>0.28</td>
<td>0.01</td>
</tr>
<tr>
<td>2PG Phoebus-TCP</td>
<td>None</td>
<td>1.42</td>
<td>0.02</td>
<td>0.32</td>
<td>0.01</td>
</tr>
<tr>
<td>Direct</td>
<td>Moderate</td>
<td>58.36</td>
<td>0.00</td>
<td>58.37</td>
<td>0.02</td>
</tr>
<tr>
<td>1PG Phoebus-TCP</td>
<td>Moderate</td>
<td>67.09</td>
<td>0.01</td>
<td>58.42</td>
<td>0.01</td>
</tr>
<tr>
<td>2PG Phoebus-TCP</td>
<td>Moderate</td>
<td>167.8</td>
<td>0.02</td>
<td>58.50</td>
<td>0.01</td>
</tr>
<tr>
<td>Direct</td>
<td>High</td>
<td>108.36</td>
<td>0.00</td>
<td>108.36</td>
<td>0.02</td>
</tr>
<tr>
<td>1PG Phoebus-TCP</td>
<td>High</td>
<td>117.09</td>
<td>0.02</td>
<td>108.42</td>
<td>0.01</td>
</tr>
<tr>
<td>2PG Phoebus-TCP</td>
<td>High</td>
<td>317.66</td>
<td>0.03</td>
<td>108.49</td>
<td>0.01</td>
</tr>
</tbody>
</table>
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 dynamic networks.

We have shown promising results with protocol adaptation between TCP and more configurable protocols, using UDT as an exemplar. 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 ever-increasing network capacities. Using PGs built with commodity hardware, we have observed and presented results of Phoebus forwarding at over 9 Gb/s for a single stream, and close to 10 Gb/s for multiple streams, with our current code. There are 10 Gb/s Ethernet Phoebus nodes under experimentation in Internet2’s network, with a plan proposed by the community working group to extend this and complete a permanent Phoebus deployment in Internet2.3

For performance scalability, we are experimenting with network processor based hardware, programmable network interface cards, and integration with emerging programmable extensions to high-end routers. We are confident that the Phoebus model will continue to scale to higher speeds. We are also investigating scaling to 100G via Phoebus clusters, with colleagues from Department of Energy. A coordinated, Phoebus-based approach is seen as one promising technique to utilize 100G networks.

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 are being explored in the rethinking of traditional network models.12 However, in a world of extreme performance demands, the Phoebus architecture addresses a very real need.

Acknowledgment

Phoebus was supported by the Department of Energy Office of Science under FG02-04ER25642.

References


3 There are currently five Phoebus nodes deployed in Internet2 as of July 2010.


Ezra Kissel received the B.Sc. degree in Computer Science from the University of Delaware in 2003. After working in private industry for 2 years, he returned to the University of Delaware earning the M.Sc. degree in 2007 and is currently a Ph.D. candidate in the CIS department. His research interests include high-performance networking, network protocol design, grid computing, and network security.

Martin Swany is an Associate Professor in the Department of Computer and Information Sciences at the University of Delaware. He received his B.A. and M.S. from the University of Tennessee in 1992 and 1998, respectively. He completed his Ph.D. at the University of California, Santa Barbara in 2003 and joined the faculty of the University of Delaware that year. Since 2005, Swany has been the Internet2 Faculty Fellow involving work in network metrics and performance-enhancing middleware. His research interests include high-performance parallel and distributed computing and networking.

Aaron Brown graduated with a Bachelor’s degree in Computer Science from Clark University in 2003. In 2006, he received a Master’s degree from the University of Delaware where his research interests included high-performance networking and measurement. He is currently employed as a network software engineer at Internet2, working on the perfSONAR project as well as Internet2’s dynamic circuit networking initiative.