IEEE Symposium on High-Performance Distributed Computing (HPDC-12/2003), Seattle WA, June 2003.

Optimizing GridFTP through Dynamic Right-Sizing *

Sunil Thulasidasan† Wu-chun Feng‡ Mark K. Gardner‡
{sunil,feng,mkg}@lanl.gov

†Basic & Applied Simulation Science (CCS-5)
Los Alamos National Laboratory
Los Alamos, NM 87544

‡Advanced Computing Laboratory (CCS-1)
Los Alamos National Laboratory
Los Alamos, NM 87544

Abstract

In this paper, we describe the integration of dynamic right-sizing — an automatic and scalable buffer management technique for enhancing TCP performance — into GridFTP, a subsystem of the Globus Toolkit for managing bulk data transfers across computational grids. Such grids are often characterized by networks with large bandwidth-delay products. Unfortunately, many of today's grid applications use only a small fraction of available bandwidth because the default buffer sizes in TCP are tuned for yesterday's WAN speeds. Buffer sizes can be manually tuned to allow TCP flow control to adapt to high-speed WAN environments, but this is a tedious process. Although recent work has shown how to automatically tune system buffers during connection set-up, these values may not be appropriate for the connection's lifetime due to varying network delay and throughput.

We show how using the technique of dynamic right-sizing (DRS) in GridFTP helps us optimize memory usage while maintaining high throughput over the lifetime of the connection. We also show how DRS enhances important GridFTP features such as striped and third-party data transfers in a scalable way. The technique is implemented entirely in user space so that end users do not have to modify the kernel.

Keywords— GridFTP, Globus, dynamic right-sizing, DRS, drsFTP, computational grid, bandwidth-delay product, auto-tuning.

1 Introduction

GridFTP [1] is a protocol extension to FTP [14] and its existing extensions [2, 8, 9] to manage large data transfers across computational grids [4]. GridFTP adds useful new features to the FTP specification that include secure transfers, third-party managed transfers, parallel and striped data transfers (where data is partitioned across multiple nodes) and partial file transfers among others. Presently, these have been implemented as part of the GridFTP module of the Globus Toolkitv2.0 [6]. The term "GridFTP" is used to refer to the protocol and the GridFTP family of tools that are distributed with the Globus Toolkit e.g., the GridFTP server, client tools and the client and control libraries.

GridFTP relies on TCP, the most widely used transport protocols for the Internet as well as computational grids. Such grids are often characterized by networks with large bandwidth-delay products (BDP). Currently, however, the default flow-control parameters in TCP are statically tuned to suit networks with small BDPs, and thus perform abysmally over today's grid networks with large BDPs.

Tuning of buffer sizes is necessary for maximizing throughput with TCP. For any given connection, the optimal TCP buffer size is equal to the BDP of the connection. Often, grid researchers manually tune the buffer sizes to keep the network pipe

*This work was supported by the U.S Department of Energy through LANL contract W-7405-ENG-36. This paper is a Los Alamos Unlimited Release LA-UR-03-2486.

full [13, 16] by using diagnostic tools to determine the round-trip time (RTT) and bandwidth of the connections. Such tools include `iperf` [18], `nettimer` [10] and `nettest` [11] among others. Because all these tools require a certain level of expertise to install and use, they are often not used by application end users.

Several services have been proposed to simplify this manual tuning process, e.g., AutoNcFTP [12] and Enable [17]. However, these tools only measure network characteristics at connection set-up time. As we shall see in section 2, the BDP of a connection can fluctuate enormously during the lifetime of the connection. Thus, this method may oftentimes end up either over-provisioning or under-allocating buffer space.

A more desirable approach would be to dynamically tune the buffers over the lifetime of the connection. There are two proposed implementations for this at the kernel level: auto-tuning [15] and dynamic right-sizing (DRS) [3]. The former implements sender-based, flow-control adaptation while the latter implements receiver-based, flow control adaptation. DRS in the kernel exhibits throughout speed-ups of up to eight over a typical WAN grid running stock TCP [3]. However, achieving such performance requires a DRS kernel patch for every pair of communicating hosts in the grid.[1]

DRS has also been implemented in user space [5], where it has been integrated into the Linux FTP client and the popular `wu-ftpd` [20] FTP server applications. While this is a coarser-grained implementation, it is more portable and remains transparent to the end user. The BDP of the network is continually monitored throughout the lifetime of the connection, and the buffers are dynamically tuned based on the estimated BDP. This has resulted in better adaptation and overall performance gain.

The GridFTP specification also has provisions for automatic buffer-size tuning [1], although this has been an unimplemented feature. In this paper, we shall describe the results of integrating DRS into the GridFTP control libraries. We show how GridFTP with DRS performs on par with statically tuned over-provisioned buffer sizes, but at a fraction of the memory cost. Further, the technique works with all of GridFTP's important data transfer features such as third-party data transfers and striped transfers. Considering the advantages, we believe that this is an important enhancement to the GridFTP framework.

[1]We note that DRS is fully compatible with stock TCP. Even if only one end host is DRS enabled, the connection will still work properly but without the performance benefits.

The rest of the paper is organized as follows. In section 2, we briefly discuss the GridFTP protocol and the motivation for integrating DRS into it. Section 3 briefly discusses the GridFTP structure and the design considerations for enabling DRS in GridFTP. In section 4, we discuss the implementation details of the DRS technique. Section 5 discusses the experimental results and the performance of GridFTP with and without DRS. Finally we conclude in section 6 with a brief discussion of the overall benefits of integrating DRS into GridFTP and avenues for further research.

2 Background

GridFTP is a set of extensions to the ubiquitous FTP protocol that facilitates reliable, secure and efficient bulk data transfers across computational grids. The original FTP protocol [14] and its existing extensions [2, 8, 9] are used as the base. The GridFTP specification [1] has additional features that include:

- Support for secure transfers (Grid Security Infrastructure).
- Parallel data transfers — where the data may be transferred in parallel streams between two nodes.
- Striped data transfers — where the data may be transferred to multiple nodes (stripes).
- Partial file transfers.
- Third-party control of data transfer.
- Automatic negotiation of TCP buffer sizes.

GridFTP is distributed as part of the Globus Toolkit and consists of a set of client and control libraries and associated Application Program Interfaces (APIs). The GridFTP server consists of a modified version of the `wu-ftpd` [20] server that supports most of the GridFTP protocol extensions. The client and server modules access a common set of libraries to manage the data and control connections of an FTP session. Section 3 has a more detailed discussion on the structure of GridFTP.

One of the unimplemented features in GridFTP has been the automatic negotiation of TCP buffer sizes. The key to maximizing transfer rate of TCP connections over high bandwidth-delay product (BDP) networks is to ensure that the connection throughput is not artificially limited by its flow-control window. To fully utilize available bandwidth

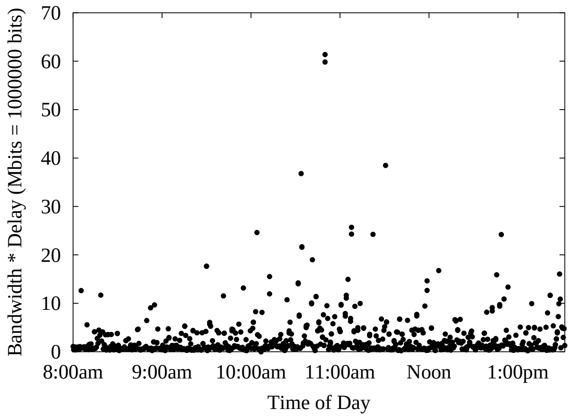

Figure 1. Variation of Bandwidth-Delay Product at 20-Second Intervals

and keep the network pipe full, the TCP buffers should be set at least as large as the BDP. Historically, a statically tuned flow-control window, $fwnd$, set to small default values sufficed for all communications since the BDP of networks was small. However, today's grid networks are characterized by large BDPs, and thus a small value for $fwnd$ leads to gross under-utilization of available bandwidth. Most operating systems today set the default $fwnd$ size to $\sim$ 64 KB — the maximum window size available without scaling. Yet BDPs range between a few bytes (56 Kbps $\times$ 5 ms = 36 bytes) and a few megabytes (622 Mbps $\times$ 100 ms = 7.8 MB). For the former case, the system wastes over 99% of its allocated memory(i.e., 36B/64KB = 0.05%). In the latter case, the system potentially wastes up to 99% of the network bandwidth (i.e., 64KB/7.8MB = 0.8%). Throughput can be increased many-fold by statically setting the default buffer sizes to the estimated BDP of the network. But even this may not be the best approach, since the BDP of a given connection can wildly fluctuate over its lifetime.

This fluctuation in BDP is illustrated in Figure 1 depicting the variation in the BDP between Los Alamos National Laboratory and a site in New York, sampled at 20-second intervals. The bottleneck bandwidth averages 17.2 Mbps with a low and a high of 26Kbps and 28.5 Mbps, respectively. The RTT delay also varies between 119ms and 475ms with an average delay of 157ms. As a result, the BDP of this connection varies by as much as 61Mb. Thus, ideally, $fwnd$ should vary dynamically with the BDP of the connection, thus providing the motivation for dynamic right-sizing (DRS).

The GridFTP specification has provisions for tuning buffer sizes, manually as well as automatically. The manual method allows clients to explicitly set the TCP buffer size for subsequent data connections. With the automatic method,[2] the buffer size is negotiated based on a RTT and bandwidth test at connection set-up time. We shall discuss these in more detail in section 4.2. The point to note, however, is that both these methods still represent a one-time tuning, carried out at connection set-up time, and can lead to over-allocation of memory or under-usage of bandwidth as the network conditions change over the connection's lifetime. Clearly, the grid community needs a solution that achieves optimal performance without wasting bandwidth or memory resources. We have already integrated the DRS technique into regular FTP [5], and the promising results combined with the increasing popularity of GridFTP and the Globus Toolkit among grid researchers provided us with enough motivation to apply this technique to GridFTP.

3 Applying DRS to the GridFTP Framework

In the following sections, we briefly discuss the GridFTP framework in the Globus Toolkit and the design considerations for implementing DRS in GridFTP.

3.1 The GridFTP Software Framework

The GridFTP implementation in the Globus Toolkit consists of a family of tools — the GridFTP server, client program, client library and the control libraries that manage the data and control connections of an FTP session.

Below, we briefly describe the main GridFTP modules that are of interest to us:

- globus-url-copy: This is the GridFTP client tool provided with the Globus Toolkit, which works with multiple protocols (http, https, ftp, gsiftp[3]). It takes a source URL and a destination URL as the arguments. The number of parallel streams can be specified as an optional argument.

- globus-gass-copy: The globus-url-copy is a simple wrapper around the globus-gass-copy program.[4]

[2] Unimplemented in the current GridFTP version as of the writing of this paper

[3] FTP with support for Grid Security Infrastructure.

[4] GASS is Globus Access to Secondary Storage.

This consists of an API for copying data from a source to a destination, specified by URLs. The protocols supported are http, https, ftp and gsiftp.

- globus-ftp-client: This is the FTP client library that supports standard FTP commands such as `get, put` etc. as well as the GridFTP extensions such as third-party-transfer.

- globus-ftp-control: This is the low level GridFTP driver, consisting of a set of libraries that handle the control and data channel management. The data channel management is symmetric for client *and* server sides, i.e, both client and server access a common set of functions for data transfer.

- GridFTP Server: This is a "globussified" version of the `wu-ftpd` server. GridFTP control and data channel management functions are invoked from within the main server module.

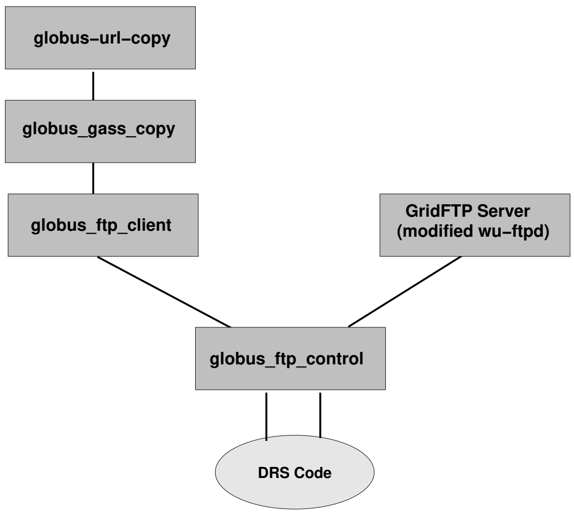

Figure 2. GridFTP Layout

To add DRS functionality to the GridFTP framework, most of the significant changes were made to the `globus-ftp-control` libraries. Figure 2 gives a general idea about the dependencies and interactions between the various modules described above. Note that the Globus Toolkit is a considerably complex piece of software. A detailed discussion of all the relevant modules is outside the scope of this paper, and the interested reader is referred to [6].

3.2 Data Transfer in GridFTP

The GridFTP protocol specifies two modes of data transfer:

- Stream Mode or `Mode S` is defined in the original FTP RFC [14]. Data is represented as a stream of bytes. There are no advanced features in this mode, except simple restart.

- Extended Block Mode or `Mode E`, which is an extension of the Block mode specified in [14]. `Mode E` supports advanced features such as 64-bit offset and length fields to the header, discontiguous, out-of-order transmission and also enables parallelism and striping.

DRS has been implemented in the extended block mode and is currently unavailable in stream mode. This approach allowed us the maximum room for maneuverability without significantly affecting the existing framework. Similar to the other advanced data transfer features in GridFTP (which are also unavailable in stream mode), specifying the DRS option forces the data transfer to take place in extended block mode. Figure 3 shows the structure of the extended-block header. Of particular interest to us is the eight-bit descriptor field. This field is used to indicate the type of data that follows — end-of-file, error-block, restart-marker etc. In section 4.2 we describe how we can exploit unused descriptor codes for implementing the DRS algorithm.

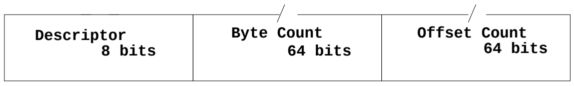

Figure 3. Extended Block Header

4 Implementing DRS in GridFTP

The DRS technique was originally implemented in the Linux protocol stack [3]. In order to maximize the transfer rate, DRS in kernel space tries to set the flow-control window of a TCP connection to be larger than the congestion-control window, thus ensuring that the connection is not artificially flow-control limited. It does this by inferring the instantaneous bandwidth and uses this value and the estimated round-trip time to calculate the BDP.

The DRS technique has been successfully extended to user space via the Linux FTP client and the wu-ftpd FTP server [5]. The main difficulty in extending DRS into a user-space implementation is that unlike the kernel, an application (e.g., an FTP client) does not have direct access to the state of the TCP stack. Thus, any DRS implementation in the application layer must rely on coarse-grained, user-space measurements rather than on fine-grained TCP connection state.

Bandwidth measurements are made by sampling the throughput at periodic intervals at the receiver end. To determine the round-trip delay, the FTP client sends a small packet periodically on the FTP control channel for the sender to echo back. The RTT is estimated by computing the difference between the sending and receiving times. The additional load on the network as the result of the RTT probe packets is generally small and depends on the sampling interval.

Note that in the above method, the RTT probes are sent on the control channel. Thus one of the implicit assumptions here is that the path delay on the control channel is the same as that of the data channel, i.e., essentially the control and data paths are assumed to follow the same route. This is not generally true in the case of third-party transfers, whereby the control commands may be sent on an entirely different path from the actual data-transfer path. Third-party transfers are an important feature of GridFTP; thus a different approach was needed to estimate RTT (described in section 4.2). Below we present the implementation details of DRS in GridFTP.

4.1 Determining Instantaneous Bandwidth

We know that the sender always has data to send throughout the lifetime of the FTP data connection. Thus, the sender will send as much data as possible, constrained by its idea of the congestion window or size of the receiver flow-control window (whichever is smaller). Further, data is received at the receiver end as quickly as current network conditions allow. Thus, the available bandwidth can be estimated by periodically computing the bytes received by the time taken to receive them.

The sampling interval needs to be chosen carefully. A sampling interval that is too short leads to increased overhead and reduces performance. Conversely, a long sampling interval leads to sluggish performance since DRS will not be able to set the buffer sizes quickly enough to adapt to changing network conditions.

Ideally, the sampling rate should be related to the round-trip time. However, for the current implementation we chose a fixed sampling time of 500ms, which is several times greater than cross-country RTTs. This value represents a fair compromise between overhead and adequate responsiveness. Dynamically varying the sampling time according to varying network delays is one of the modifications we propose to add in the future.

4.2 Determining RTT

The RTT of a data connection cannot usually be inferred from user space without injecting extra traffic into the network. This is because user-space code does not have access to the state variables in the TCP-stack, and hence cannot know when a packet was sent and its acknowledgement received. We shortly describe how we can circumvent this problem by using the advanced features of `Mode E`.

As mentioned earlier, in drsFTP, RTT was estimated by periodically sending small packets on the control channel, which the sender then echoes back. These packets served the dual purposed of carrying buffer-size information from receiver to sender[5] as well as being a vehicle for RTT estimation. This was based on the assumption that the control and data channels took the same path. As noted earlier, however, this technique will not work in GridFTP during third-party data transfers or for striped transfers (where multiple data connections are present).

The GridFTP specification has added two commands to the control channel protocol for setting TCP buffer sizes. Of particular interest to us is the `SBUF` command. This extension adds the capability for a client to explicitly set the TCP buffer size on the server for subsequent data connections.[6]

Syntax:
```
SBUF <SP> <buffer-size>
buffer-size ::= <number>
```

As mentioned in section 3.2, the DRS technique in GridFTP is implemented in the extended-block (`Mode E`) data transfer mode. Recall from section 3.2 that the `Mode E` header (ref. Figure 3) contains unused descriptor codes in the eight-bit descriptor field. For the purposes of estimation of RTT, we make use of the semantics of the `SBUF` command, but

[5] Sender's window should adjust in step with receiver's window in order to take full advantage of dynamically changing buffer sizes

[6] The second one is the `ABUF` command which enables clients to automatically set buffer-sizes based on a bandwidth and RTT test at connection set-up time. However, this is an unimplemented feature in the current version of GridFTP.

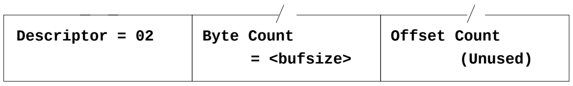

Figure 4. SBUF Message encapsulated in a Mode E Header.

encode it in a `Mode E` header and send it over the data channel. For this purpose, we define two new descriptor codes, making use of two unallocated bits in the descriptor field.

Descriptor Code	Meaning
02	Encoded SBUF message
01	Reply to SBUF message

Note that the `SBUF` messages are sent as *in-band* data, i.e., they are sent and received on the same data connection as the actual file transfer. This approach allows us to estimate RTT over the data channel itself.

Upon expiration of an alarm, the receiver sends a `Mode E` header (without any payload) to the sender. The descriptor field in the header is set to 02 (encoded `SBUF`) while the `<byte-count>` field is set to the value of the receiver's current buffer-size (Figure 4).

When the sender receives this packet on the data channel, it interprets this as an `SBUF` command using the descriptor code and tries to set its send-buffers to the value contained in the `<byte-count>` field. Also, an acknowledgement is sent by setting the descriptor code to 01 in the descriptor field of the next outgoing EB header.[7] When the client receives a header with the `SBUF` reply bit set, it calculates the RTT based on the reception time and the time the `SBUF` was originally sent. By using these `SBUF` messages as carriers for buffer-size information as well as a vehicle for RTT estimation, we eliminate the need for separate RTT probes.

Note that this method requires the data channel to be bidirectional during a data transfer, i.e., full-duplex communication must be supported over the data channel. This is a modification to the FTP specification as well as to the current GridFTP implementation. Bidirectional data transfers are not presently implemented in the Globus distribution of GridFTP. To enable DRS, we had to engineer bidirectionality into the library modules of GridFTP. Remember that the `SBUF` messages and its responses are sent over the existing TCP data connection and not over another connection going the opposite way. Thus this method will work even in the presence of firewalls (note that TCP itself is a bidirectional protocol over a single connection).

[7]Note that this is similar to the way TCP piggy-backs acknowledgements.

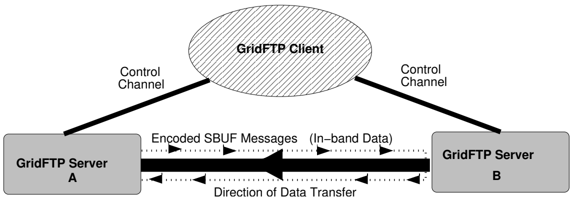

Figure 5. Data Transfer Dynamics During a Third-Party Transfer.

In drsFTP, the RTT estimates were made over the control channel. Thus for parallel transfers, we would need a control channel for each data connection (assuming they took the same route) — impeding the scalability of such an implementation. Moreover, third-party transfers could not be supported in this way. However, in GridFTP, by estimating all the required information i.e., bandwidth and RTT, over the data channel(s), we eliminate the need for any control channel information for DRS. Therefore, the DRS algorithm can be implemented between any two pair of communicating hosts involved in a data transfer. Thus, support for third-party transfers and parallel/striped transfers is implicit in this implementation.

Figure 5 illustrates a third-party transfer scenario. The `SBUF` messages are exchanged as in-band data on the data channel, parallel to the actual data transfer. In a two-party scenario, the controlling node itself would be at one of the ends of the data channel. We also note that the DRS state information associated with each data channel is only maintained at the end points of that particular data channel. In third-party and striped/parallel transfers, often a single node acts as the controlling entity that coordinates data transfers over multiple data channels. By implementing DRS solely over the data channel, we need not maintain any state information at the controlling node - making this a scalable approach.

4.3 Setting Receiver's Flow-Control Window

Using the estimates for bandwidth and RTT as described above, the bandwidth-delay product of the connection is calculated at the receiving end. User-space applications cannot directly set the flow-control window in the TCP stack. Instead, the window is set indirectly by setting the size of TCP receive buffer to at least twice the calculated value of the BDP. For this we employ the `setsockopt` system call that allows the user to manipulate TCP buffer sizes. The reason for doubling the receive buffer size is that in the worst case, the sender is in the slow-start phase and is doubling its window with every round-trip. It is difficult to estimate when the sender is out of slow-start, thus we increase the buffer size by a factor of two over the estimated BDP.

We note that the DRS technique is TCP-friendly in the sense that N flows, DRS-enabled or not, will each receive a long-term average of 1/N-th of the bandwidth of a fully utilized network. Since the congestion-control mechanism of TCP governs fairness and because GridFTP with DRS has the same congestion-control mechanism, it responds to congestion the same way as regular TCP. On an uncongested network, however, GridFTP with DRS will attempt to utilize the excess capacity that can exist when all the other connections are artificially limited by their flow-control windows. As the network becomes congested again, GridFTP with DRS throttles back and performs no better (or worse) than a regular TCP connection.

5 Experiments

We tested the performance of both two-party and three-party transfers using GridFTP with DRS against GridFTP with buffer sizes set statically to OS default values as well as to large values (greater than the BDP of the connection). The experiments were run through a WAN emulator that reproduces delay and loss based on a sample of such data between Los Alamos National Laboratory and a site in New York. Each version of GridFTP (drs, stock and static-optimal) is benchmarked across the WAN emulator.

5.1 Experimental Setup

Figure 6 shows our experimental setup for two-party transfers. It consists of three identical machines connected via 100-Mbps Ethernet. Each machine contains dual 500-MHz Pentium III processors with 1-GB RAM and a 100-Mbps on-board Intel Network Interface Card (NIC). One machine, containing another 100-Mbps NIC, acts as a WAN emulator. Each of its two NICs are connected to one of the other machines via hard-coded forwarding tables

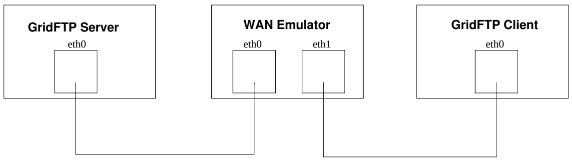

Figure 6. Experimental Setup for Two-Party Data Transfer

in the switch. The WAN emulator, implemented using TICKET [19] technology, forwards packets at line rate and has user-settable delay and drop probability. All traffic, both data and control, occurs through the WAN emulator. For testing parallel transfer, we start multiple data streams between the client and server machines. The number of parallel streams is specified as an option to the `globus-url-copy` program.

5.2 Experimental Method

For each version of GridFTP, we transfer a set of files ranging from 1 MB to 512 MB over the emulated WAN. For the GridFTP runs with OS-default buffers, we used buffers of 64 KB (the default under most modern operating systems). For both the OS-default and over-provisioned cases, we run GridFTP (without DRS) in regular stream mode (with one stream) as well as extended-block mode with one, two and four parallel streams. We then run GridFTP with DRS enabled for the same configuration, allowing the buffer sizes to vary in response to changing network conditions.

5.3 Results

We present a few results from our initial experiments for both two-party and three-party transfers. In the following results, the average round-trip time is 102.1 ms, and the DRS sampling interval is set to 500 ms. For the statically tuned, over-provisioned case, we use static buffer-sizes of 8 MB (which is greater than the bandwidth-delay product of 1.2 MB).

5.3.1 Two-Party Transfers

Figure 7 shows the performance of GridFTP (without DRS) with default buffer sizes (64K) in stream modes as well as `Mode E` with one, two and four parallel streams. The average transfer rate with four parallel streams (20 Mbps) is about four times faster than that with one stream (5 Mbps). Figure 8 shows

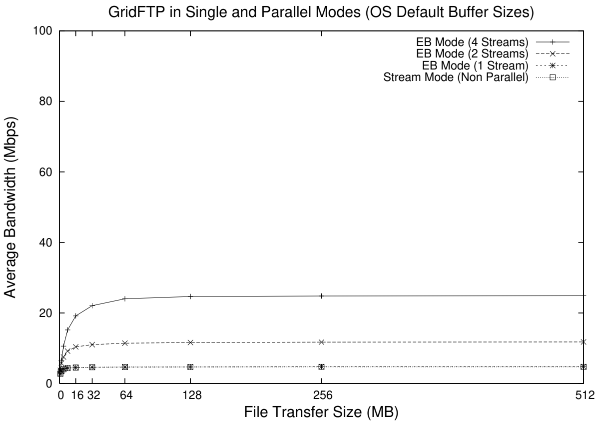

Figure 7. Performance of GridFTP with OS-default buffer size (64KB)

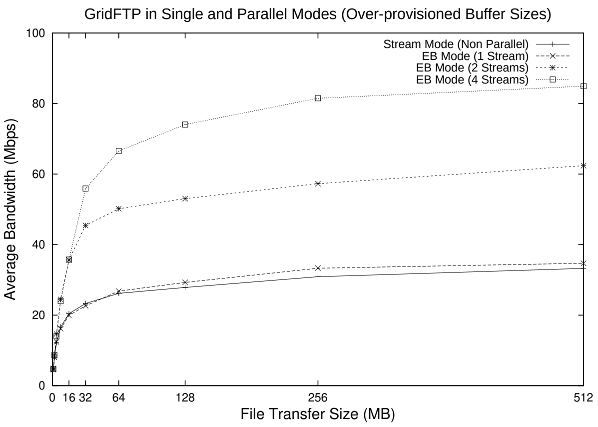

Figure 8. Performance of GridFTP with Statically Tuned Buffer Size (8MB)

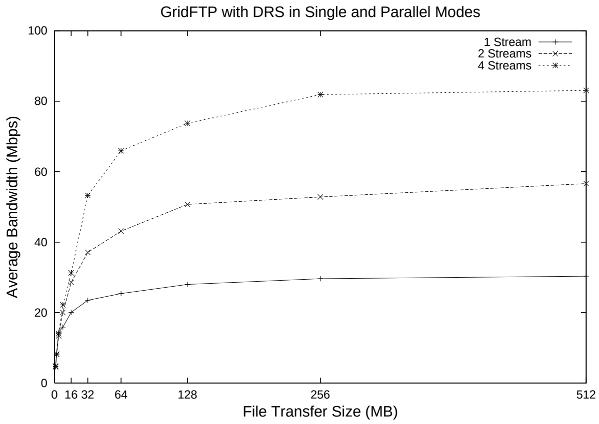

Figure 9. Performance of GridFTP with DRS - Dynamically Varying Buffer Sizes

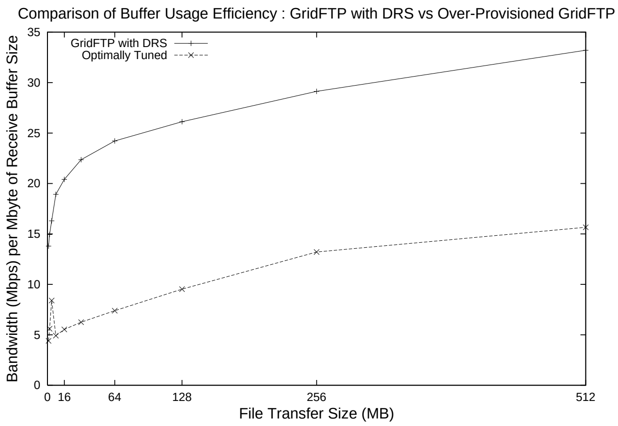

Figure 10. Buffer Usage Efficiency: Bandwidth Achieved per MB of Buffer

the results for the same file transfer sizes but with buffer size statically set to 8 MB. The maximum transfer rate goes up to 80 Mbps with four parallel streams, a four-fold improvement over the corresponding transfer rate with stock buffer sizes. Next we run the same set of tests but with DRS turned on. From Figure 9 we see that GridFTP with DRS performs as well as GridFTP with statically tuned over-provisioned buffers. Furthermore, the improvement over GridFTP with stock buffer sizes is four-fold. Buffer-sizes were observed to range from a paltry 700 KB to 3.5 MB. The reader may observe that buffer-sizes grow to values that are larger than the theoretical BDP of our setup (1.2 MB). There are two reasons for this — first, recall that we set the buffer sizes to twice the observed BDP to allow for the slow-start phase, where the sender may be doubling its window every round-trip time. Second, the coarse-grained, user-space timer tends to over-estimate the RTT up to a factor of 1.5. This is because the estimates are actually the end-to-end delay between the two application layers (including protocol stack overhead), and not the actual network path delay. These two factors combined result in a small amount of over-provisioning of buffers.

Figure 10 illustrates the efficiency of buffer usage by measuring the bandwidth achieved per MB of buffer allocated. We see that for a file size of 16 MB, DRS is able to achieve a throughput of almost 19 Mbps/MB as opposed to 5 Mbps/MB for the statically tuned, over-provisioned case. Thus the efficiency factor for DRS is greater by as much as a factor of four over the static configuration. As the transfer size (and thus transfer time) increases, efficiency picks up for over-provisioned GridFTP but it still remains well below that of DRS.

5.3.2 Third-Party Transfers

Figure 11 illustrates our experimental setup for third-party transfers. In this scenario, the data transfer takes place between two GridFTP servers, connected through the WAN emulator. The controlling node is the GridFTP client, which has low-latency, control-connections to either server. To initiate a third-party data transfer, the controlling node first establishes a control-channel connection with the receiving host (GridFTP server B in this case). The client then sends a PASV command on the control channel to the receiving host. This instructs the server to listen on a data port and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command is of the form (host,port) which contains the host and port address that this server will listen on. The client then sends a PORT command on the control channel to the sending host (GridFTP server A). This command is of the form

```
PORT host-address,host-port
```

The host address and port values are those that were returned by the receiving host in response to the PASV command. The PORT command instructs the sending server to use these values for the data connection. Figures 12 to 14 show the corresponding performance results in the third-party transfer scenario. As we can see, the performance is very similar to the two-party results discussed in the previous section. This is to be expected since DRS dynamics are implemented entirely over the data connection. The presence of a third-party controlling node will not affect the performance to any significant degree. Thus, implementing the DRS technique via in-band data messages over the data connection allows us to extend the performance benefits of DRS to third-party transfers - an important feature of GridFTP.

6 Conclusion

In this paper, we presented the results of integrating dynamic right-sizing into the GridFTP implemen-

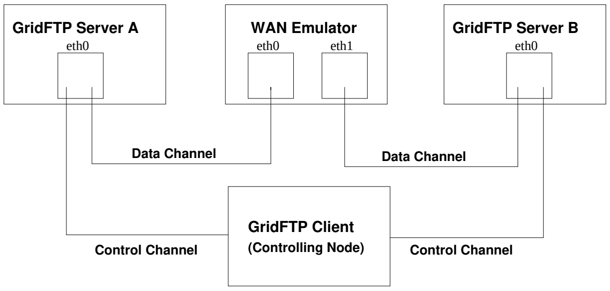

Figure 11. Experimental Setup for Third-Party Data Transfer

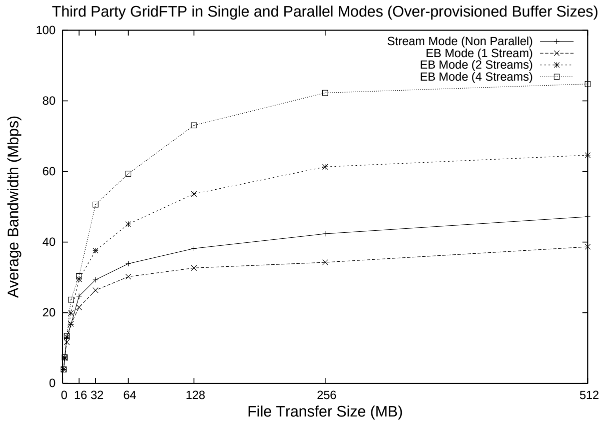

Figure 12. GridFTP Performance in Third-Party Transfer with Over-Provisioned Buffer size

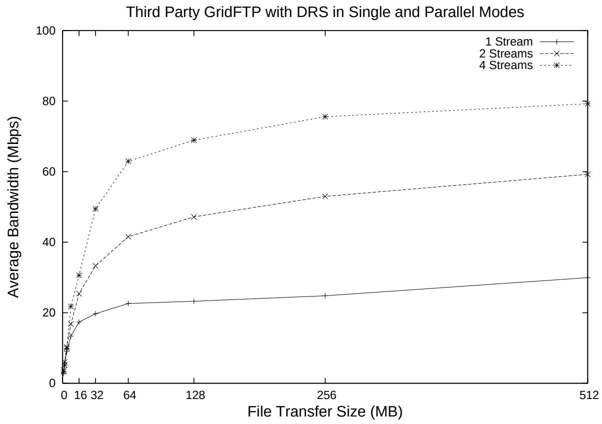

Figure 13. GridFTP Performance in Third-Party Transfer with DRS

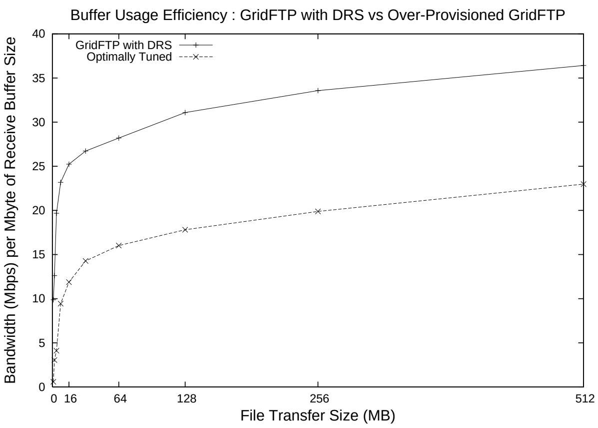

Figure 14. Buffer Usage Efficiency in Third-Party Transfer: Bandwidth Achieved per MB of Buffer

tation of the Globus Toolkit. We were motivated by the performance results obtained by integrating DRS into regular FTP. We showed how the implementation implicitly supports important GridFTP features like third-party, parallel and striped transfers in a scalable way. Future work will include performance results of DRS "in the wild" — GridFTP with DRS over a real-world WAN.

A requirement for our implementation was that the data connection needed to be bidirectional, in order to permit encoded `SBUF` messages to be sent for RTT estimation. However, one of the current limitations in the Globus Toolkit implementation of GridFTP is the lack of bidirectionality in the data channel, i.e., the data connection and data flow are required to be in the same direction.[8] To enable DRS, we engineered bidirectionality into the GridFTP implementation.

GridFTP uses parallel streams to achieve very high transfer rates. GridFTP with DRS and parallel streams is able to achieve the same throughput at a fraction of the memory cost - a highly beneficial aspect to the grid community. Furthermore, due to the static nature by which buffers are set for regular (non-DRS) GridFTP, i.e., once during connection setup, a connection can be unnecessarily throttled due to a bad BDP sample. GridFTP with DRS simply adapts automatically over the lifetime of the connection to simultaneously ensure high throughput and low memory usage. By integrating the efficiency of DRS into GridFTP, we believe we have combined the best of both approaches.

[8]This is not a restriction of the standard FTP protocol.

References

[1] W. Allcock et al. GridFTP: Protocol Extensions to FTP for the Grid. http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf, March 2001.

[2] R. Elz, P. Hethom. FTP Extensions IETF Draft, September 2000.

[3] M. Fisk, W. Feng. Dynamic Right-Sizing in TCP). In *Proceedings of the Los Alamos Computer Science Institute Symposium*, October 2001.

[4] I. Foster, C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1999

[5] M. Gardner, W. Feng, M. Fisk. Dynamic Right-Sizing in FTP (drsFTP): An Automated Technique for Enhancing Grid Performance. In *Proceedings of IEEE Symposium on High-Performance Distributed Computing*,July 2002.

[6] The Globus Project. http://www.globus.org

[7] Globus: Grid Security Infrastructure. http://www.globus.org/security/.

[8] P. Hethmon, R. Elz. Feature Negotiation Mechanism for the File Transfer Protocol. RFC 2389, August 1998.

[9] M. Horowitz, S. Lunt. FTP Security Extensions. RFC 2228, October 1997.

[10] K. Lai, M. Baker. Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, March 2001.

[11] Lawrence Berkeley National Laboratory Nettest: Secure Network Testing and Monitoring. http://www-itg.lbl.gov/nettest

[12] J. Liu, J. Ferguson. Automatic TCP Socket Bufer Tuning. In *Proceedings of SC 2000: High-Performance Networking and Computing Conference (Research Gem)*, November 2000.

[13] Pittsburgh Supercomputing Center. Enabling High-Performance Data Transfers on Hosts.

[14] J. Postel, J. Reynolds. File Transfer Protocol (FTP), October 1985

[15] J. Semke, J. Mahdavi, M. Mathis. Automatic TCP Buffer Tuning. *Computer Communications Review, ACM SIGCOMM*, 28(4), October 2001.

[16] B. Tierney. TCP Tuning Guide for Distributed Applications on Wide-Area Networks. In *USENIX and SAGE Login*, February 2001.

[17] B. Tierney, D. Gunter, J. Lee, M. Stoufer. Enabling Network-Aware Applications. In *Proceedings of the IEEE International Symposium on High-Performance Distributed Computing*, August 2001.

[18] A. Tirumala, J. Ferguson. IPERF. http://dast.nlanr.net/Projects/Iperf/index.html.

[19] E. Weigle, W. Feng. TICKETing High-Speed Traffic with Commodity Hardware and Software. In *Proceedings of the Third Annual Passive and Active Measurement Workshop*, March 2002.

[20] The *wu-ftpd* FTP Server. http://www.wu-ftpd.org