IEEE International Parallel and Distributed Processing Symposium
(IPDPS 2004), Santa Fe, New Mexico, 26–30 April 2004. LA-UR 04-0494.

Re-Architecting Flow-Control Adaptation for Grid Environments*

Adam Engelhart, Mark K. Gardner, and Wu-chun Feng
{adame, mkg, feng}@lanl.gov

Research & Development in Advanced Network Technology (RADIANT)
Computer & Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract

The performance of TCP in wide-area networks (WANs) is becoming increasingly important with the deployment of computational and data grids. In WAN environments, TCP does not provide good performance for data-intensive applications without the tuning of flow-control buffer sizes. Manual adjustment of buffer sizes is tedious even for network experts. For scientists, tuning is often an impediment to getting work done. Thus, buffer tuning should be automated.

Existing techniques for automatic buffer tuning only measure the bandwidth-delay product (BDP) during connection establishment. This ignores the (large) fluctuation of the BDP over the lifetime of the connection. In contrast, the dynamic right-sizing algorithm dynamically changes buffer sizes in response to changing network conditions.

In this paper, we describe a new user-space implementation of dynamic right-sizing in FTP (drsFTP) that supports third-party data transfers, a mainstay of scientific computing. In addition to comparing the performance of the new implementation with the old in a WAN-emulated environment, we give performance results over a live WAN. The new implementation produces transfer rates of up to five times higher than untuned FTP.

1. Introduction

For over twenty years, TCP has provided reliable service to Internet users. The algorithms used have been tuned, extended, and tested over decades to adapt to varying conditions in the global Internet and serve as the basis for Internet-based applications such as FTP, the World Wide Web, and distributed computing infrastructures, such as grids [2, 5, 6, 10, 11].

Unfortunately, stock TCP is not well-suited for widely-distributed grid environments running over networks with high bandwidth-delay products (BDPs). The problem lies in the parameters of the TCP flow-control algorithm—the default buffer sizes are static, and are only appropriate for local-area network environments. Since only one full buffer's worth of data can be in transit at any time, small buffers hobble data transfers by leaving network resources relatively unused.

This problem is well-known and has been solved by adjusting the buffer sizes manually [4, 25]. However, this solution requires specialized experience to do well, including proficiency with various network measurement tools, such as *iperf* [27], *netspec* [28], *nettest* [14], *nettimer* [15–17], *pchar* [19], or *pipechar* [13], and a knowledge of how to set the operating system's networking parameters.

This gives rise to the so-called "wizard gap" [20], the gap between the performance seen when the system is tuned by a networking expert versus the performance that a non-networking expert gets. However, users whose expertise lies in other areas (e.g., visualization or bioinformatics) should be able to take advantage of modern WAN technology without becoming networking experts as well.

There are several techniques for adjusting buffer sizes automatically, including AutoNcFTP [18] and Enable [26]. While these tools eliminate the need for a network expert, they only set the buffer size once—at the start of the connection. This is fine if the BDP is relatively static, but it is not. The BDP of a WAN can vary widely over the course of a connection, as demonstrated in Section 2. In order to achieve the best performance possible, TCP buffer sizes must be adjusted throughout the lifetime of the connection

*This work was supported by the U.S. Department of Energy through Los Alamos National Laboratory contract W-7405-ENG-36 with funding provided by the Office of Science Base Program of DOE. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE or Los Alamos National Laboratory.

for proper flow-control adaptation.

Two approaches to dynamically tuning buffer sizes are auto-tuning [23] and dynamic right-sizing (DRS) [7,9]. The former is a sender-based approach to flow control, while the latter is a receiver-based approach. While the sender-based approach yields excellent performance, there is a risk of overflowing the receiver's buffers. As a receiver-based approach, DRS ensures no buffers will overflow while still providing excellent performance.

We are working to ensure that DRS is widely adopted by operating system vendors where it will be available to all applications without modification. Until then, we make a patch available for the Linux kernel. However, end users are often unable to patch and recompile the kernel due to lack of knowledge or lack of administrative permissions on the machines involved. As an alternative solution, we developed an earlier user-space implementation for bulk-data transfer [12].

In this paper, we describe a new implementation of dynamic right-sizing in FTP (drsFTP) which uses a different mechanism for computing round-trip times. Unlike the previous version [12], the new mechanism also allows third-party bulk-data transfer connections to use the DRS algorithm. In addition to comparing the performance of the new and old implementations in an emulated environment, we present the performance of the new implementation in a live WAN environment, for third-party as well as client-server transfers.

2. Background

TCP can apply two independent "brakes" to network traffic. The first, flow control, ensures that the sender does not overrun the receiver's available buffer space. The second, congestion control, ensures that the sender does not overrun the capacity of the network.

To do this, TCP maintains a flow-control window, $fwnd$,[1] and a congestion-control window, $cwnd$. The sender infers $cwnd$ from the network's loss behavior and receives $fwnd$ from advertisements sent by the receiver. It then uses the smaller of the two ($min(fwnd, cwnd)$) as the effective window ($ewnd$) and adjusts its rate to send at most one $ewnd$ worth of data per round-trip time (RTT).

Traditionally, $fwnd$ has been a relatively small static value, as TCP was first implemented when BDPs were small and receivers were short on memory for buffer space. By default, most TCP implementations set $fwnd$ to the largest size available without scaling, approximately 64 kB [3]. However, BDPs commonly range from a few bytes for short-haul modem connections (56 kbps $\times$ 5 ms $=$ 36 bytes) to several megabytes for high-speed WANs (10 Gbps $\times$ 200 ms $=$ 25 MB). Default buffer sizes waste resources to the point of profligacy—the modem connection above wastes over 99% of the buffer space (36 bytes / 64 kB $=$ 0.05% buffer utilization), while the WAN connection could over to 99% of the network's capacity (64 kB / 25 MB $=$ 0.26% network utilization).

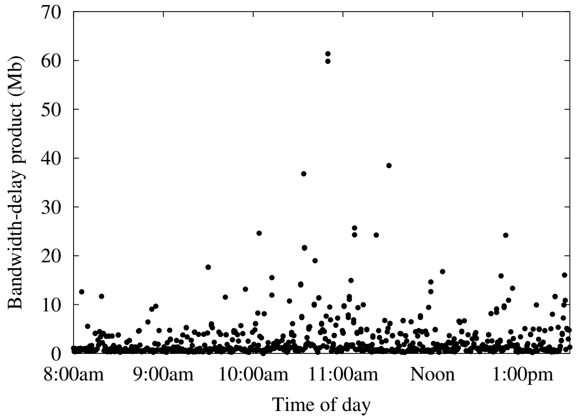

Figure 1. Bandwidth-delay product at 20-second intervals.

Furthermore, both bandwidth and delay can change within the lifetime of a single connection, particularly in a WAN, due to congestion, queueing, and routing changes. This effect is clearly illustrated in Figure 1, which plots the BDP of a link between Los Alamos and New York at 20-second intervals. The bandwidth of the link ranges from 26 kbps to 28.5 Mbps, with an average of 17.2 Mbps. The RTT ranges from 119 ms to 475 ms, with an average of 157 ms. The BDP can therefore vary by as much as 61 Mb or 7.6 MB.

Since the BDP fluctuates so widely, even on an intra-transfer timescale, any solution that does not re-adjust $fwnd$ throughout the connection potentially wastes memory or bandwidth. For the purposes of efficiency and high throughput, dynamic flow-control adaptation is essential, and DRS provides this adaptation.

3. Algorithm

DRS attempts to maximize the transfer rate of TCP connections by ensuring that the flow-control window is always larger than the congestion-control window (within the limits of available buffer space on the hosts), thus making the network the limiting factor in the transfer. It does this by setting the send and receive buffer sizes to the BDP. [2]

In kernel space, the DRS algorithm has access to the TCP connection state and can derive the BDP by calculating the

[1] This variable is traditionally called $awnd$, for "advertised window," because the receiver advertises it to the sender in each packet.

[2] In order not to throttle the connection during slow start, when the flow-control window is doubling every round-trip time, drsFTP sets the buffer sizes to twice the BDP.

bandwidth from the rate at which the sequence number advances and by using TCP's estimate of the round-trip time. However, in user space, this information is not available; DRS-enabled applications must rely on their own coarser-grained measurements. As will be seen in Section 5, these measurements still result in significantly improved network throughput.

3.1. Determining bandwidth

As long as an FTP data connection is open, the sender will attempt to send data as quickly as possible but within the constraints of the congestion- and flow-control windows and the receiver will accept that data as quickly as possible. The end hosts can therefore estimate the bandwidth simply by dividing the amount of data received (or sent) by the time required to receive (or send) it.

When DRS is implemented in the kernel, it must calculate the bandwidth on the recieving host, because the new bandwidth-delay product must be communicated to the sender in the advertised window. However, drsFTP is under no such constraint, because buffer sizing information is exchanged on a bidirectional DRS channel (Figure 2).

Because applications can write data in larger bursts than the network can transmit due to buffering, the bandwidth estimate computed by the sender may exceed the theoretical capacity of the network over some intervals. The bandwidth estimate on the receiver, on the other hand, is much more realistic. Therefore, we calculate the bandwidth on the receiver.

Selecting the interval over which to compute the bandwidth presents a problem. If the interval is too short, the overhead increases and the bandwidth estimate can fluctuate wildly. If the interval is too long, DRS loses its ability to respond quickly to rapidly changing network conditions. Currently, drsFTP computes the bandwidth every time the round-trip time is calculated.

3.2. Determining round-trip time

Unlike bandwidth, round-trip time can not be measured from user space without injecting additional data into the network—user-space programs have no access to the TCP state variables where the information is kept and therefore must resort to their own devices.

In drsFTP, messages are sent on a DRS channel (as shown in Figure 2), which is separate from the standard FTP control channel. The addition of the separate DRS channel not only ensures that the stream on the control channel is compatible with existing FTP implementations, but also allows third-party transfers in which the control channel and data channel involve different pairs of machines, as shown in Figure 3.

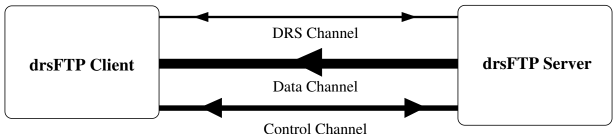

Figure 2. A drsFTP transfer.

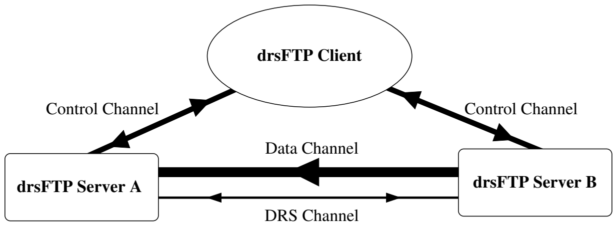

Figure 3. A third-party drsFTP transfer.

3.3. Setting the flow-control window

The techniques described above allow drsFTP to estimate the bandwidth-delay product. Since user-space applications cannot directly set the flow-control window, drsFTP uses the `setsockopt` call to set the receive buffer to twice the measured BDP. (The BDP is doubled because it is possible that the sender is in slow start, and therefore doubling its window size with every RTT. Since it is difficult to determine when slow start ends in user space, we always double the buffer size.)

For maximum performance, the sender should keep its buffer size synchronized with the receiver's. This requires the receiver to inform the sender whenever it changes its buffer. Since we are already sending RTT probes over the DRS control channel, we combine the buffer size communication and RTT computation mechanisms into a single type of message exchange.

The format of the message resembles the SBUF command from the draft GridFTP standard [1]. While the old drsFTP implementation [12] uses the GridFTP SBUF command format, the new implementation uses an extended format for probing the RTT and setting the window size, and restricts its use to the DRS control channel. We will return to the extended SBUF format in Section 4.

3.4. On window scaling

For DRS to work properly, the TCP window scaling factor must be set high enough to enable sufficiently large buffers. Since the scaling factor is set when the connection is established, drsFTP must increase the buffer sizes before the connection is open, and then return the buffer sizes to the initial value once the connection is established. Currently, drsFTP uses a scaling factor that allows windows of up to 16 MB, if allowed by the operating system.

Most operating systems' limits on buffer sizes must be set to larger sizes in order to allow 16 MB windows. Instructions for doing this can be found in [4].

3.5. TCP-friendliness

DRS is TCP-friendly in the sense that on a fully utilized network, the bandwidth of N flows (whether or not they are DRS-enabled) will eventually converge to 1/Nth the total bandwidth of the network. To see this, consider what happens in the presence of network congestion. DRS-enabled flows, which have exactly the same congestion-control mechanism as the stock flows, will be constrained by their diminishing congestion windows in exactly the same way as non-DRS flows. Thus, the available bandwidth will be fairly shared, and DRS is TCP-friendly.

On the other hand, when the congestion dissipates, the stock flows will once again be artificially limited by their flow-control windows, while the DRS flows will not. As a result, the DRS flows will consume the leftover bandwidth not used by the stock flows. This is not unfriendliness on the part of DRS; it is purely an artifact of the flow-control throttling of the stock flows.

4. Reimplementation of drsFTP

Previous versions of drsFTP interpret RFC959 [21] as allowing multiple outstanding commands[3] and hence the SBUF dialogue took place on the control channel during the data transfer. The majority, if not all, of the extant FTP implementations do not expect a new command before the previous one completes. Multiple outstanding commands prevent complete compatibility with existing clients and servers.

While motivated by a desire to improve compatibility, the real reason for the redesign is to allow automatic buffer sizing during third-party transfers, which was not possible under the previous approach.

In the new implementation, the dialogue over the FTP command channel maintains a single outstanding command at a time, as expected by extant clients and servers. All DRS traffic that occurs during a transfer is sent over a separate channel, the DRS channel. (See Figure 2.)

While the command channel is client/server-oriented, the DRS channel is peer-to-peer. As with the previous implementation, SBUF messages are used to synchronize the receiver and sender buffer sizes. The SBUF message is also used to compute RTT. However, the syntax of the SBUF message has also changed in the new implementation:

```
sbuf ::= SBUF <SP> <timestamp>
  <SP> <buffer-size>
timestamp ::= <number>
buffer-size ::= <number>
```

This command is always sent over the DRS channel, which has the same endpoints as the data transfer channel.

When a sender receives an SBUF command, it sets its buffer size to `buffer-size`, subject to the limitations of its available buffer space, and echoes the command exactly as received. When a receiver receives the echoed SBUF, it determines the RTT using the `timestamp`. Using the bandwidth estimate it computes, the receiver computes a new BDP. The process is then repeated.

Previously, the receiver in drsFTP would wait for the SBUF to return before setting the buffer size, so the receiver's buffer space stays synchronized with the sender when the sender had limited buffer space available. The new implementation sets the receiver buffer size before sending the SBUF message, favoring network performance over memory frugality.

The `timestamp` field is new. It is generated when the receiver sends the SBUF message. The current implementation sends the time returned by `gettimeofday`, in seconds, with as much precision as is available.

The sender's response to the SBUF message is to simply echo the SBUF message it received. (The DRS channel is peer-to-peer; there is no notion of independent "commands" and "replies" as on the standard FTP control channel.) Upon receipt of an SBUF message, the receiver computes the RTT. The estimated bandwidth and the RTT are then used to compute the new BDP.

We note using a separate DRS channel eliminates a limitation of the previous version of drsFTP [12]—the inability of the technique to support DRS-enabled third-party data transfers. Since the DRS channel connects the same two machines as does the data channel (Figure 3), SBUF messages traverse the same route as data packets, and hence the BDP is correctly computed during a third-party transfer.

The previous implementation of drsFTP only allows one outstanding SBUF message. This becomes a problem when the RTT increases to the point where a new SBUF message is sent before the previous echo is received. Since SBUF messages in the new implementation contains all the information necessary to determine the RTT, the server does not need to maintain any state regarding outstanding SBUF messages.

[3] RFC 959 states, "The communication between the user and server is *intended* to be an alternating dialogue ... The user *should* wait for ...," emphasis added.

5. Experiments

In this section we quantify the improvement in bulk-data transfer performance as a result of the re-architectured drsFTP. We measure the performance of four different cases—FTP with stock buffer sizes, FTP with statically-tuned buffer sizes set to the BDP, FTP with over-provisioned buffer sizes, and drsFTP. We present results for both emulated and live WAN environments and for both client-server and third-party transfers.

5.1. Experimental setup

For the emulated WAN environment, we use three identical machines with 100-Mbps Fast Ethernet cards. Each machine has two 500-MHz Pentium III processors and 1 GB of memory. (While a Fast Ethernet connection might seem underpowered in a cluster environment, it is still representative of many current grid nodes. Furthermore, it demonstrates the benefits of DRS even in smaller-BDP networks—benefits which are magnified in larger-BDP networks.) One machine, running TICKET [30], acts as the WAN emulator; it forwards packets at line rate between the two machines with a user-settable delay. For these tests, the delay is set to 100 ms. The peak bandwidth between the machines, as measured by *iperf* [27], is 95 Mbps. The bandwidth-delay product is therefore approximately 1.2 MB.

In the live WAN tests, we use an FTP server in Los Alamos and an FTP client at the Oregon Graduate Institute. The server has two 933-MHz Pentium III processors, 1 GB of RAM, and a 3Com 3C985 Gigabit Ethernet card. The client has one 1.8-GHz Pentium 4 processor, 512 MB of RAM, and an ADMtek Comet Fast Ethernet card.

Over a two-hour period of time, the average bandwidth and RTT measured every second by *iperf* [27] and *ping* ranges from 0-93 Mbps and 72-257 ms, respectively. The bandwidth-delay product ranges from 0.175-268 kB. The connection behaves very much like that shown in Figure 1 with respect to the variance in the BDP.

In addition the stock and drsFTP cases, we report results for an over-provisioned buffer size of 16 MB and an "optimally"-tuned buffer size. (The "optimally"- tuned buffer size is set equal to the highest measured bandwidth-delay product, 268 kB, measured above.[4]) The OS parameters are set such that DRS connections can use up to 16 MB of buffer space.

[4] We were very generous in selecting a large BDP sample (one that corresponds to one of the few high points in Figure 1). Had we sampled the BDP at a low point, the statically-tuned performance would be much worse than that shown in Figure 5.

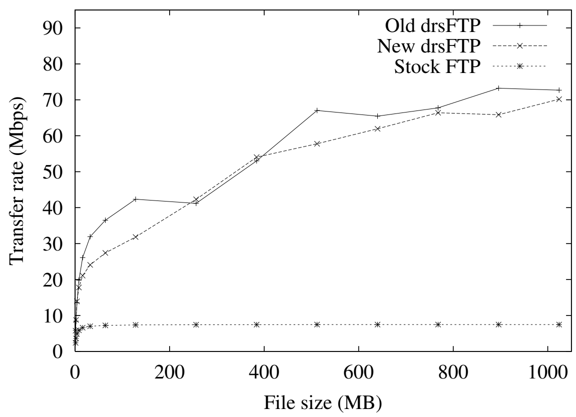

Figure 4. Comparison of drsFTP versions and stock FTP using an emulated WAN.

5.2. Experimental method

We transfer a set of files ranging from 1 MB to 1024 MB in size using each of the four FTP variants. (The exact file sizes range from 1 to 128 MB by powers of two and from 128 MB to 1024 MB in increments of 128 MB.) We perform at least 30 transfers at each file size. For the live WAN tests, we also vary the times of day at which the transfers take place.

5.3. Comparison of old and new drsFTP

We compare the performance of the old and new drsFTP implementations over the WAN emulator. The results are shown in Figure 4. The new drsFTP compares quite well with the old drsFTP. There is no significant difference between the two. Stock FTP with default (64 kB) buffers is also presented. Compared to stock FTP, DRS can speed up transfers by as much as nine times when file sizes are sufficiently large.

5.4. Live WAN results

We now examine the behavior of drsFTP in a live WAN environment. Figure 5 shows the performance of all four FTP variants over the live WAN. drsFTP attains a maximum bandwidth of approximately 55 Mbps, over 5.5 times better than the 10 Mbps of stock FTP.

Perhaps paradoxically, drsFTP performs better than the "optimally"-tuned connection at large file sizes (55 Mbps vs. 33 Mbps). We found that while the drsFTP bandwidth estimates are approximately what we would have expected, the RTT estimates often are significantly higher than those mentioned in Section 5.1. We believe that this is due to *ping*'s use of ICMP to determine RTT versus our use of SBUF messages.

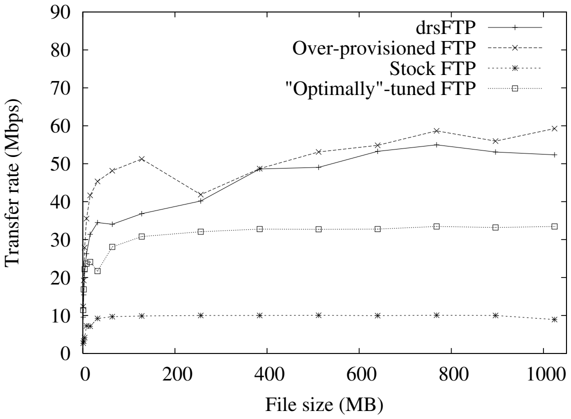

Figure 5. Comparison of the FTP variants over a live WAN.

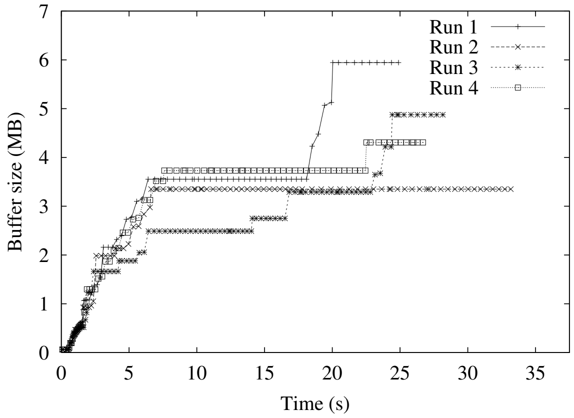

Figure 6. drsFTP buffer sizes for 128-MB transfers in the daytime.

ICMP packets are processed by the kernel, whereas SBUF messages require application processing, and hence RTTs computed using SBUF messages will be longer. The mean RTT for ICMP, from Section 5.1, is 76.7 ms, while the mean RTT as measured by drsFTP during a 128 MB transfer is 180.6 ms. The difference is due to protocol overhead, the need to go between kernel space and user space, and scheduling latency. Hence, for static buffer tuning, a larger buffer size than would be predicted by the usual manual-tuning method is likely needed to achieve the best performance. For this reason, it is possible for dynamic techniques, such as DRS, to outperform static tuning that is performed at connection set-up.[5]

Figure 6 shows the DRS buffer size growing over time for four 128 MB transfers. Note the wide range in final buffer sizes—from 3.3 MB to nearly 6 MB—due to the varying BDP of the network. We acquire the measurements from which we generate this graph during the daytime, making the effect more pronounced. However, the effect is still clearly visible in data gathered during the night and early morning. Transfers with larger average buffer sizes finish earlier.

Figure 7 shows the bandwidth estimates used by drsFTP to calculate the buffer sizes in one 128-MB transfer. The transfer rate rises very quickly during the first two seconds of the transfer, suggesting that the connection is in slow start. At this point, the connection experiences congestion, and the rate begins to increase linearly due to the additive-increase phase of congestion control.

Figure 8 shows the RTT estimates from the run in Figure 7. The estimates are scattered widely over a range from

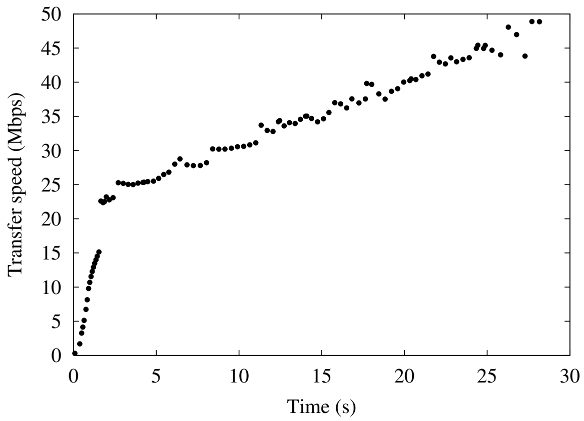

Figure 7. Delivered bandwidth during a 128-MB WAN file transfer.

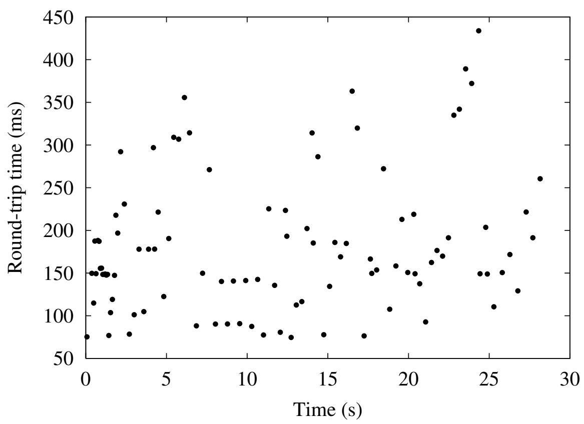

Figure 8. Round-trip time as measured by drsFTP during a 128-MB WAN file transfer.

[5]This is in addition to the fact that DRS's dynamic nature not only makes it immune to getting a bad sample at the start of the connection (corresponding to one of the low points in Figure 1), but also lets it take advantage of any extra bandwidth that should become available during the transfer.

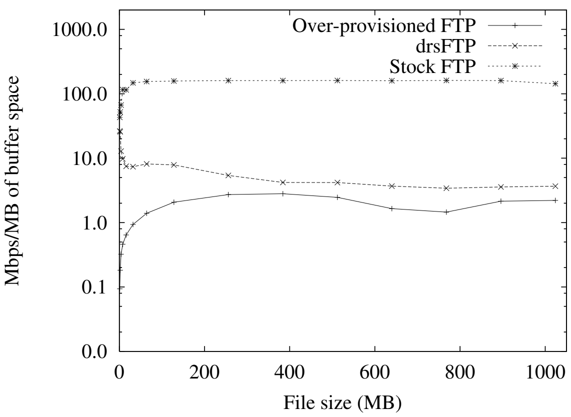

Figure 9. Bandwidth attained over the WAN per MB of buffer space for the FTP variants.

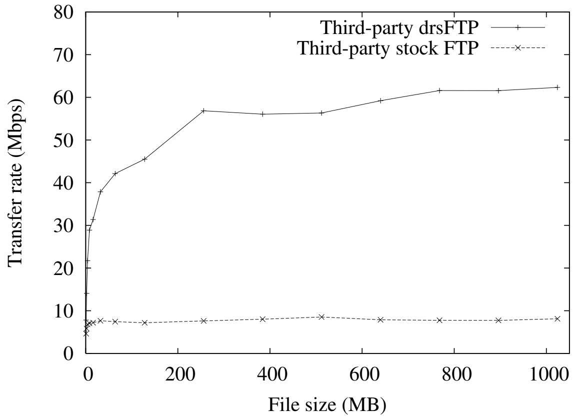

Figure 10. Bandwidth attained over the WAN by third-party drsFTP transfers.

75 ms to 434 ms, showing the extreme volatility of WAN and host conditions.

5.5. Memory efficiency

We now turn our attention to the efficiency of the variants with respect to memory. One relevant metric here is bandwidth achieved per megabyte of buffer space, measured in Mbps/MB.

When grids become part of production environments, there will be more competition for memory, and grid users will have to be careful not to reserve too much of the system's memory for buffer space. Therefore, achieving the best bandwidth for the quantity of buffer space allocated becomes a priority.

Figure 9 shows that drsFTP is significantly more efficient in its use of buffer space than FTP with over-provisioned buffers. DRS ranges from 14.8 Mbps/MB for 1-MB transfers to 4 Mbps/MB at 1024-MB transfers, while the over-provisioned variant approaches 3.6 Mbps/MB for a 1024-MB transfer.

We caution, however, that this metric does not supersede the more traditional metric of bandwidth. As a case in point, while stock FTP is much more frugal in its use of buffer space than drsFTP, attaining over 100 Mbps/MB, it only attains one-sixth the transfer rate of the DRS case. Clearly, users must take both buffer usage and transfer rate into account when making comparisons.

5.6. Third-party transfers

In this section, we present the performance results for third-party transfers, comparing drsFTP to stock FTP.

The FTP specification, RFC 959 [21], allows direct server-to-server data transfers. The process is diagrammed in Figure 3. In brief, the user issues the PASV command to one FTP server, causing it to listen on an arbitrary port and return the IP address and port on which it is listening. The user then issues a PORT command to the other FTP server, instructing it to connect to the given port on the first server. Finally, the user issues the STOR and RETR commands to the appropriate servers to start the transfer. Section 5.2 of RFC 959 gives more information on this.

The results are presented in Figure 10. For the series of runs shown, third-party drsFTP attains over six times the bandwidth of stock FTP.

6. Conclusions

In this paper, we present a new implementation of drsFTP, a user-space dynamic right-sizing technique for increasing the performance of bulk-data transfers over connections with high bandwidth-delay products. drsFTP requires no kernel modifications or expert assistance to use, apart from setting the maximum buffer size sufficiently large to allow DRS room to increase the flow-control window. (This would also have to be done to statically set buffer sizes by hand.)

We have shown a bandwidth increase of a factor of five to nine over stock FTP and an increase in buffer-usage efficiency of a factor of up to four over FTP with over-provisioned buffers. Furthermore, once drsFTP is installed and set-up, it requires no further attention from the user. drsFTP therefore provides significant gains in transfer rate, memory efficiency, and convenience over existing user-space buffer-tuning techniques.

The interval over which drsFTP computes the bandwidth attained by the connection was determined arbitrarily. Currently, it is statically set at compile time. While the existing algorithm gives good performance, we believe that additional research is required. It is possible that changing this interval in response to network conditions will result in more reliable bandwidth data and therefore better efficiency with respect to memory. In a similar vein, we have several ideas for validating the accuracy and reliability of round-trip time estimates. Care is needed to retain responsiveness to changing WAN conditions, however.

Like the previous version of drsFTP, the new implementation will be released as open-source software.

Acknowledgements

We gratefully acknowledge the kind support of Wu-chang Feng of the OGI School of Science & Engineering at the Oregon Health & Science University for providing a remote machine for live WAN testing.

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] ANL, CalTech, LBL, SLAC, JF, U. Wisconsin, BNL, FNL, and SDSC. The Particle Physics Data Grid. `http://www.cacr.caltech.edu/ppdg/`.

[3] D. Borman, R. Braden, and V. Jacobson. TCP Extensions for High Performance (RFC 1323), May 1992.

[4] Pittsburgh Supercomputing Center. Enabling High-Performance Data Transfers on Hosts. `http://www.psc.edu/networking/perf_tune.html`.

[5] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *International Journal of Supercomputer Applications*, 23(3):187–200, July 2001.

[6] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi. Access Grid: Immersive Group-to-Group Collaborative Visualization. In *Proceedings of the 4th International Immersive Projection Workshop*, 2000.

[7] M. Fisk and W. Feng. Dynamic Adjustment of TCP Window Sizes. Technical Report Los Alamos Unclassified Report (LAUR) 00-3221, Los Alamos National Laboratory, July 2000.

[8] M. Fisk and W. Feng. Dynamic Right-Sizing in TCP. In *Proceedings of the Los Alamos Computer Science Institute Symposium*, Oct 2001. LA-UR 01-5460.

[9] M. Fisk and W. Feng. Automatic Flow-Control Adaptation for Enhancing Network Performance in Computational Grids. In *Journal of Grid Computing*, Vol. 1, No. 1, 2003, pp. 63–74.

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

[11] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 2001.

[12] M. K. Gardner, W. Feng, and M. Fisk Dynamic Right-Sizing in FTP (drsFTP): Enhancing Grid Performance in User-Space In *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, July 2002.

[13] G. Jin, G. Yang, B. Crowley, and D. Agrawal. Network Characterization Service. In *Proceedings of the IEEE Symposium on High-Performance Distributed Computing*, August 2001.

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

[15] K. Lai and M. Baker. Measuring Bandwidth. In *Proceedings of IEEE INFOCOMM 1999*, March 1999.

[16] K. Lai and M. Baker. Measuring Link Bandwidths Using a Deterministic Model of Packet Delay. In *Proceedings of ACM SIGCOMM 2000*, August 2000.

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

[18] J. Liu and J. Ferguson. Automatic TCP Socket Buffer Tuning. In *Proceedings of SC 2000: High-Performance Networking and Computing Conference (Research Gem)*, November 2000. `http://dast.nlanr.net/Projects/Autobuf`.

[19] B. Mah. pchar: A Tool for Measuring Internet Path Characteristics. `http://www.employees.org/~bmah/Software/pchar`.

[20] M. Mathis. Pushing Up Performance for Everyone. `http://www.ncne.nlanr.net/news/workshop/19999/991205/Talks/mathis_991205_Pushing_Up_Performance/`.

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

[22] SC2001 Bandwidth Challenge Proposal: Bandwidth to the World. `http://www-iepm.slac.stanford.edu/monitoring/bulk/sc2001/proposal.html`.

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

[24] S. Thulasidasan, W. Feng, and M. K. Gardner Optimizing GridFTP through Dynamic Right-Sizing In *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.

[25] B. Tierney. TCP Tuning Guide for Distributed Applications on Wide-Area Networks. In *USENIX & SAGE Login*, February 2001. `http://www-didc.lbl.gov/tcp-wan.html`.

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

[27] A. Tirumala and J. Ferguson. IPERF. `http://dast.nlanr.net/Projects/Iperf/index.html`.

[28] Information & Telecommunication Technology Center, University of Kansas. NetSpec: A Tool for Network Experimentation and Measurement. `http://www.ittc.ukans.edu/netspec/`.

[29] E. Weigle and W. Feng. A Comparison of TCP Automatic-Tuning Techniques for Distributed Computing. In *Proceedings of the Eleventh IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, July 2002.

[30] E. Weigle and W. Feng. TICKETing High-Speed Traffic with Commodity Hardware and Software. In *Proceedings of the Third Annual Passive and Active Measurement Workshop (PAM2002)*, March 2002.