Network Tuning and Monitoring for Disaster Recovery Data Backup and Retrieval [∗∗]

[1] Prasad Calyam, [2] Phani Kumar Arava, [1] Chris Butler, [3] Jeff Jones
[1] Ohio Supercomputer Center, Columbus, Ohio 43212.
Email:{pcalyam, cbutler}@osc.edu
[2] The Ohio State University, Columbus, Ohio 43210.
Email:arava@cse.ohio-state.edu
[3] Wright State University, Dayton, Ohio 45435.
Email:jeff.jones@wright.edu

Abstract

Every institution is faced with the challenge of setting up a system that can enable backup of large amounts of critical administrative data. This data should be rapidly retrieved in the event of a disaster. For serving the above Disaster Recovery (DR) purpose, institutions have to invest significantly and address a number of software, storage and networking issues. Although, a great deal of attention is paid towards software and storage issues, networking issues are generally overlooked in DR planning. In this paper, we present a DR pilot study that considered different networking issues that need to be addressed and managed during DR planning. The study also involved performing a series of network measurements for large-scale data transfers on a DR pilot network testbed involving OSC's mass storage site and Ohio's academic institution DR-client sites. The measurement results demonstrate that adequate network tuning and on-going monitoring are critical for DR data backup and retrieval operations to be achieved in an efficient and timely manner.

1 Introduction

The need for Disaster Recovery (DR) planning primarily arises from every major institution's IT strategy and accounting regulations. They require off-site data backups that will allow data recovery in the event of a disaster. Although, the nature of data (e.g. data, images) differs between institutions, most institutions deal with backups that involve transferring increasingly large amounts of data to one or more off-site backup facilities. For this, institutions or DR-client sites invest in local storage, backup software, and in mechanisms for frequently moving a copy of their local data to tape-drives at off-site sites. With the increased access to high-speed networks, most copy mechanisms rely on IP networks (versus postal service) as a medium to transport the backup data between the institutions and off-site backup facilities.

Today's IP networks, which are designed to provide only "best-effort" service to network-based applications, frequently experience performance bottlenecks. The performance bottlenecks can be due to issues such as lack of adequate end-to-end bandwidth provisioning, intermediate network links congestion or device mis-configurations such as duplex mismatches [1] and outdated NIC drivers at DR-client sites. These bottlenecks in network paths severely impact data backup performance of all the backup jobs in the system i.e., they slow-down a few backup jobs involving problematic network paths, which in turn could hold-up speedy completion of several other backup jobs that traverse problem-free network paths. Also, having numerous slow backup jobs can cause long backup queues, which are highly undesirable in DR systems that have strict system resource constraints.

The performance bottlenecks also cause intermittent network disconnections in backup applications. As a result, most backup applications require multiple connection attempts to complete backup jobs. Arsenal Digital, a company that provides off-site backup facilities estimated that 94.5% of their successful backup jobs required up to three connection attempts [2]. Further, over 60% of unsuccessful backup jobs were attributed to be caused due to network bottlenecks. Symatec, a company that developed the widely-used Netbackup application estimated that 25-35% of staff time is spent resolving network performance issues [3] to troubleshoot causes of failed backup jobs involving their backup application.

In spite of the above gravity of networking issues in the context of DR operations, most DR planning activities do not spend sufficient time in resolving networking issues. They

[∗∗]This work has been supported in part by The Ohio Board of Regents

primarily focus on DR software applications and storage issues and further assume: (a) sufficient bandwidth is available and (b) all network paths are trouble-free, between the DR-client sites and the off-site backup facilities. In reality, every network path between DR-client sites and off-site backup facilities needs to be pre-assessed using extensive network testing. The network testing could involve tuning network paths to deal with different network path characteristics influenced by heterogeneous routing policies and network health dynamics. Network tuning can be performed using a variety of techniques such as using parallel TCP streams [4] and choosing optimal TCP send-and-receive window sizes [5].

The above pre-assessment benchmarks of optimal large-scale data transfers on network paths between DR-client sites and the off-site backup facilities need to be verified using on-going network monitoring. The on-going monitoring needs to provide periodic assessments to ensure no apparent network bottlenecks arise that could potentially degrade the benchmark values significantly. For on-going monitoring, there are several active and passive measurement techniques [8] that can be leveraged. These techniques can provide continuous updates on the network path status in terms of delay, loss and bottleneck bandwidth.

Based on above discussion, we can realize that it is imperative to understand which of the network tuning and on-going monitoring techniques are relevant in routine DR operations. The motivation for our network tuning and monitoring study presented in this paper comes from a DR project involving OSC's mass storage site and Ohio's academic institutions with DR needs: Wright State University, Cleveland State University and Miami University. As part of this project, we developed automated measurement scripts that use an iterative approach to test different combinations of network tuning parameters on any given network path. In this paper, we present the large-scale data transfer throughput measurements obtained using our automated measurement scripts on a DR pilot testbed. The analysis of measurements addresses the following important questions in the context of network tuning in DR operations:

1. What is the optimum data throughput obtainable using popular data transfer applications such as FTP [13], SCP [14] and BBFTP [9]?

2. What are the improvements in data throughput measurements if specialized TCP stacks such as Reno [10], FAST [11] and BIC [12] are used?

3. How many parallel TCP streams are required to attain optimum data throughput?

4. What are the effects of different TCP window sizes and application buffer sizes on the data throughput?

5. What is the impact of file sizes on data throughput?

The remainder of the paper is as follows: Section 2 presents a background on DR networking issues along with related work on network tuning. Section 3 presents our DR pilot testbed and the network measurement methodology used to measure large-scale data throughput on the testbed network paths. Section 4 discusses the throughput measurement results obtained from the pilot testbed for several test cases. Section 5 concludes the paper.

2 Background and Related Work

Generally, DR data backups are performed over network paths to off-site backup facilities for initially transferring the entire DR-client site's historical data. This is followed by daily (or sometimes weekly) incremental backups of recent changes in DR datasets. The DR data requirements of some DR-client sites could involve large-scale data transfers every few hours for real-time data mirroring of critical datasets. An example of real-time data mirroring can be see in Financial Bank sites. Check images of several thousands of customers, each of size 20-40 KB, would need to mirrored in real-time every few hours. In addition to the offline and online DR data backups explained above, large-scale data transfers occur when recovering DR datasets with "as minimum recovery times as possible" in the event of a disaster. Hence, DR operations deal with large-scale data transfers both in a routine and critical manner.

To deal with such large-scale data transfers on today's high-performance networks, there have been a variety of network tuning (also called "TCP tuning") techniques that have been proposed. Several of these techniques have also been proven to be effective in improving the efficiency of large-scale data transfers over high-performance networks [15] [16] [17]. These tuning techniques basically involve tuning system parameters, which span different Internet layers as shown in Figure 1.

At the physical layer, adequate end-to-end network bandwidth can be provisioned to meet specific large-scale data transfer requirements. At the Local Area Network (LAN) level, there have been proposals to use "Jumbo Frames" [6] [7], which increase the Ethernet Maximum Transmission Unit (MTU) to 9 Kbytes instead of 1500 bytes. Prior work [18] suggested that jumbo packets could more than double the throughput of large-scale data transfers on today's networks compared to using the standard 1500 bytes packets. At the Internet protocol layer, several QoS mechanisms such as Diffserv and IP precedence levels are popular options to prioritize DR traffic flows. In addition, there are DR operations that rely on individual institution-specific Virtual Private Networks (VPNs) for data security and network bandwidth virtualization purposes.

Since tuning parameters that involve any of the above Internet layers requires end-to-end adherence to bandwidth provisioning, MTU sizes and routing policies, it is challenging and time consuming in routine DR operations to rely on above tuning techniques. The most effective and commonly used tuning techniques can be seen at the application layer. At the application layer, tuning can be performed by using specialized TCP stacks such as FAST and BIC versus using the default Reno TCP. Another popular tuning option involves using parallel TCP streams at the application layer to improve the throughput of large-scale data transfers [4]. Other tuning options at the application layer involve choosing optimal TCP send-and-receive window sizes and application buffer sizes [5].

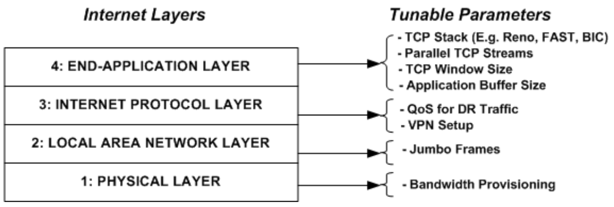

Figure 1. Tunable parameters at different Internet layers

Once a network path has been tuned, it becomes essential in DR operations to monitor whether there arise any performance bottlenecks in the network paths between the DR-client and offsite backup site, that degrade the overall throughput performance. For this purpose, several bandwidth estimation tools such as Iperf [23], Pathchar [25], Pathload [26] and Pathrate [26] have been found to be useful. A few of these tools also provide delay, loss and per-hop bandwidth utilization measurements, which are helpful in identifying performance bottlenecks. Studies in [22] have showed that large-scale data transfer throughputs using popular file transfer applications actually match reasonably well with throughput measurements obtained using bandwidth estimation tools such as Iperf.

3 DR Pilot Testbed

The network tuning and monitoring study presented in this paper is part of a DR project funded by the Ohio Board of Regents. The primarily goals of the DR project are: (i) To implement a DR service which securely copies and retrieves backup files from OSC's mass storage site for Ohio's academic institutions with DR needs: Wright State University, Cleveland State University and Miami University. (ii) To extend the DR service to support large-scale DR data transfers over the Third Frontier Network (TFN)[1] and thus allow efficient and timely completion of real-time data mirroring, daily incremental backups and data retrieval in the event of a disaster.

[1]The Third Frontier Network (TFN) is OSC's dedicated high-speed fiber-optic network backbone spanning over 1,600 miles of fiber linking Ohio colleges, universities, K-12 schools and other academic communities together to promote research and economic development.

3.1 Testbed Setup

As part of the DR project, a pilot testbed was setup between each of the individual DR-client sites and OSC's mass storage site as shown in Figure 2. The pilot testbed had three main goals:

- (Goal-1) Evaluate DR data backup software products such as Veritas Netbackup and IBM Tivoli for handling DR-client data files
- (Goal-2) Evaluate storage solutions such as SUN Solaris Containers [24] for secure storage virtualization and the ZFS file system [21] to allow storage pools to grow dynamically without the necessity of creating file system structures (i.e. formatting a new disk) and mount points
- (Goal-3) Evaluate network tuning and monitoring techniques on the DR testbed paths.

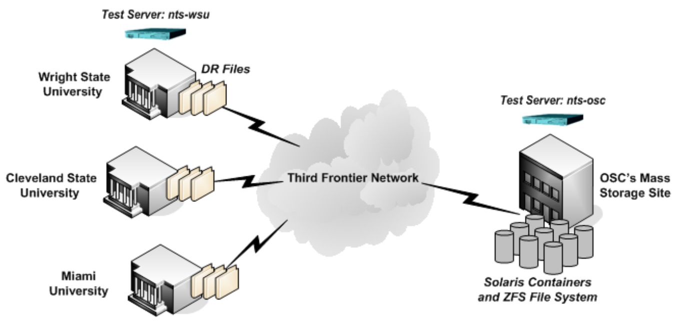

Figure 2. DR Pilot Testbed

Towards fulfilling Goal-3 in the DR pilot testbed shown in Figure 2, the network path between Wright State University and OSC's mass storage site was of specific interest. This was mainly because of the scale of DR data transfer requirements of Wright State University, which demanded an offline large-scale file transfer of over 2TB with daily incremental backups on the order of several hundreds of MB. Further, there was a possibility of real-time data mirroring requirement which would involve over 100 MB of data for every 2-5 hrs pertaining to a medical imaging application. For conducting the network tuning and monitoring study on the network path between Wright State University and OSC's mass storage site, two test servers: *nts-wsu* and *nts-osc* (shown in Figure 2), were deployed at these two sites.

The test servers' hardware configuration included a Dual-Core Intel Xeon Processor (2.4 GHz) with 1GB memory and two 1 GigE cards, both on a 64bit/66MHz PCI bus. The operating system installed on the test servers was Debian Linux with 2.6 kernel instrumented with Web 100 [5]. The test servers were also equipped with kernel modules corresponding to the specialized TCP stacks such as Reno (installed by default), FAST and BIC. In addition, the test servers had several file transfer applications such as FTP (installed by default), SCP and BBFTP. The network path between the *nts-wsu* and *nts-osc* test servers had approximately 32ms round-trip delay (RTT) and was rate-limited to 45Mbps bandwidth end-to-end.

3.2 Design and Execution of Test Cases

Amongst the several network tuning techniques discussed in Section 2, the test cases employed on the DR pilot testbed focused primarily on application-layer network tuning techniques involving: (a) Specialized TCP stacks: Reno, FAST and BIC, (b) Parallel TCP streams, and (c) TCP window size and application buffer size configurations. These network tuning techniques were applied and evaluated for large-scale data throughput measurements obtained using popular file transfer applications such as FTP, SCP and BBFTP, which were made to transfer different file sizes as part of the test cases.

Figure 3 shows the hierarchy used in the different test case combinations. The test cases were automated for obtaining data throughput measurements for different system tuning parameters using a number of Perl scripts and a tests *config* file. The *config* file contained information required by the Perl scripts to iterate through a pre-determined sequence of test cases as shown in Figure 3. The relatively higher-level parameters in the hierarchy formed the outer-loops and the relatively lower-level parameters in the hierarchy formed the inner-loops. The Perl scripts acted as wrappers that interfaced with the multiple TCP stacks, file transfer applications and the TCP window and application buffers in the test case iterations. The Perl wrapper scripts had an additional functionality, which centrally collected the throughput measurements data into a MySql database installed on the *nts-osc* test server.

For creating the different file sizes for the test cases, we used the *dd* utility in Linux to create different size swapfiles. For example, the command- `dd if=/dev/zero of=swapfile.1 bs=1024 count=1024` created a 1024*1024 bytes swapfile. For the TCP tuning test cases, two TCP window size settings were used. The first setting was the default TCP window size (64 Kbytes or 65535 bytes). The second setting was determined based on accurate calculation of the bandwidth-delay product on the network path between *nts-wsu* and *nts-osc* test servers. Given that the end-to-end bandwidth of the network path was 45 Mbps and the round-trip delay (RTT) between the two test servers was 32ms, the optimum TCP window size was set to 184320 bytes given by Equation (1).

$$\begin{aligned} TCP\_WIN\_SIZE &= \text{Bandwidth * RTT} \\ &= \text{(45 Mbps/8)* (32ms/1000)} \\ &= \text{184320 bytes} \qquad (1) \end{aligned}$$

In our earlier experiments with the Linux 2.4 kernel, we had to manually configure the TCP stack using the GRUB boot loader. However, for dynamically changing the TCP stack in the automated scripts for Linux 2.6, we used the *sysctl* application as suggested in [5]. We also followed the key guidelines provided in [5] for configuring the TCP stacks with the appropriate TCP window sizes. For example, [5] suggests that it is necessary to change both the send and receive TCP window size buffers; changing only the send buffer will have no effect because TCP negotiates the buffer size to be the smaller of the two.

4 Results and Discussion

In this Section, we present the throughput measurements obtained between the *nts-wsu* and *nts-osc* test servers on the DR pilot testbed shown in Figure 2. We focused only on unidirectional data transfers with *nts-wsu* as the source and *nts-osc* as the destination. This is because- although, TCP is inherently a full duplex protocol, the protocol dynamics are the same when tested from either directions [15] [19]. The throughput measurements presented in this Section correspond to the results of the various test cases executed by the Perl wrapper scripts as described in Section 3.2. The throughput values reported by the Perl wrapper scripts were determined as shown in Equation (2) -

$$\text{Throughput in Mbytes/s} = \frac{\text{Swapfile Size}}{\text{Swapfile Transfer Time}} \qquad (2)$$

4.1 Effects of TCP Stack Selection

Figure 4 shows the throughput measurements obtained for test case iterations where Reno, FAST and BIC TCP stacks were selected. In these test case iterations, the TCP window size was set to 65535 bytes, the application buffer size was set to 2048 bytes, and the file size for the large-scale data transfer was the 2GB swapfile. We can observe from Figure 4 that Fast TCP stack always outperforms the BIC TCP and Reno TCP stacks. As expected, the BIC TCP performs better than the Reno TCP, which is used by default in Linux distributions. Further, in cases corresponding to the BBFTP tests with one or two parallel streams, BIC TCP performance is almost comparable to the FAST TCP performance. Since BIC TCP is known to more quickly recover from packet loss on high-speed networks than FAST TCP, we concluded that using either FAST or BIC TCP would be pertinent for routine DR operations.

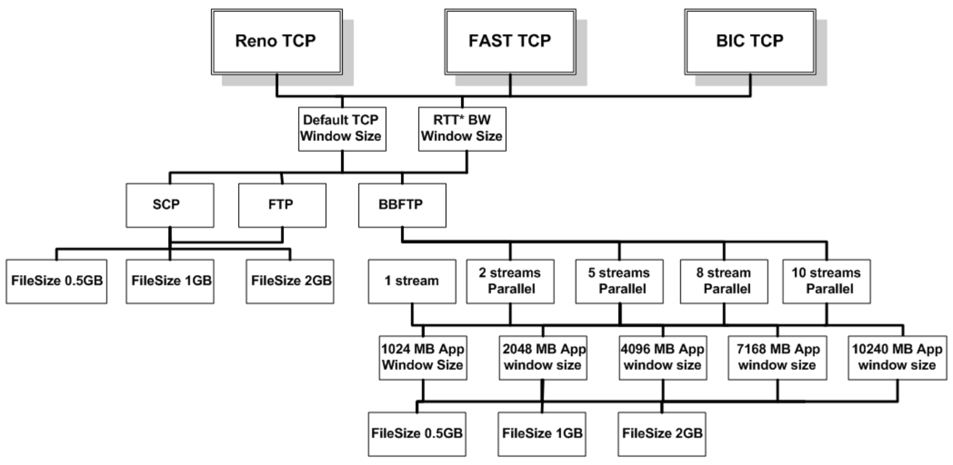

Figure 3. Test cases for automated assessment of optimum tuning parameters

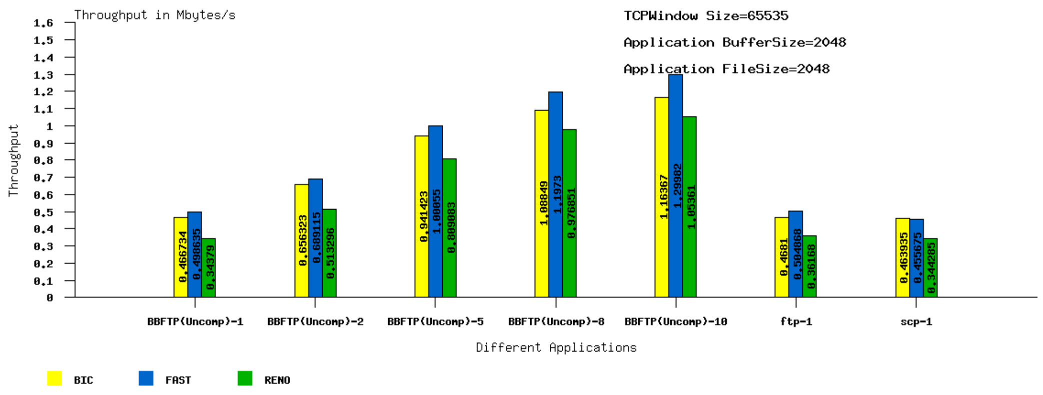

Figure 4. Throughput measurements of Reno, FAST and BIC TCP stacks

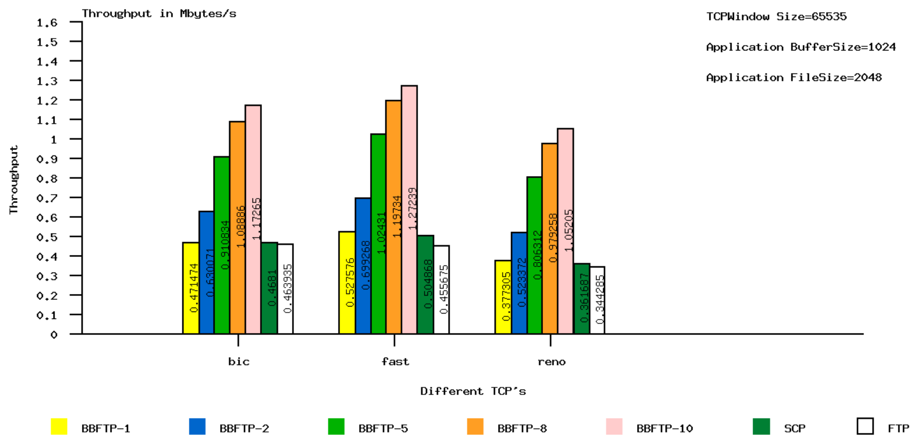

Figure 5. Throughput measurements using parallel TCP streams

4.2 Effects of Parallel TCP Streams

Figure 5 shows the throughput measurements obtained using different numbers of parallel TCP streams for test case iterations involving Reno, FAST and BIC TCP stacks. In these test case iterations, the TCP window size was set to 65535 bytes, the application buffer size was set to 1024 bytes, and the file size for the large-scale data transfer was the 2GB swapfile. We can observe from Figure 5 that the throughput increases with the increase in the number of parallel streams. However, the amount of increase is not as significant ($< 50\%$ improvement) for more than 8 parallel streams, as much it is for less than 5 parallel streams. The throughput levels for 10 parallel streams reached as high as almost 1.3 Mbytes/s (or approximately 10.5 Mbps) before the increase in throughput was insignificant with the increase in the number of parallel streams. We noted this throughput value as the "optimum large-scale data throughput" for the test path between *nts-wsu* and *nts-osc*. We intend to use a similar methodology in future to benchmark the optimum large-scale data throughput values for each of the DR-client sites. These values will be used as part of the on-going monitoring to detect potential performance bottlenecks that could limit the optimum large-scale data throughput on a routine basis.

Although we could have increased the number of parallel streams to very large values such as 40 or more parallel streams to attain throughputs comparable to the end-to-end bandwidth of 45 Mbps, we refrained from doing so. The reason being that the DR pilot testbed path between *nts-wsu* and *nts-osc* was part of a production network carrying other application traffic as well. Hence, to avoid disrupting this production network path, we restricted our number of parallel streams using a trial-and-error method till the point we were able to note the optimum large-scale data throughput value. We remark that dynamically determining the optimum number of parallel streams for any given network path is still an open research problem.

4.3 Effects of TCP Window and Application Buffer Sizes

Figures 6 - 8 show the throughput measurements obtained using different TCP window and application buffer sizes for test case iterations involving Reno, FAST and BIC TCP stacks, respectively. In these test case iterations, both the default TCP window size (65535 bytes) and the bandwidth*RTT TCP window size (184320 bytes) settings were used. The file size used for the large-scale data transfers was the 2GB swapfile. We refer to the former as the "test case setting without TCP tuning" and the latter as the "test case setting with TCP tuning".

We can observe in Figures 6 - 8 that the increase in application buffer size has no significant impact on the throughput measurements. The throughput measurements from BBFTP

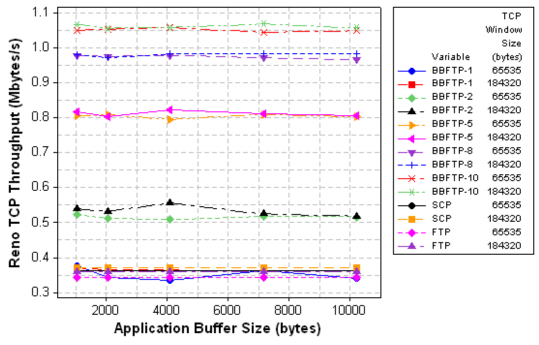

Figure 6. Throughput measurements of Reno TCP for different settings of TCP window size and application buffer size

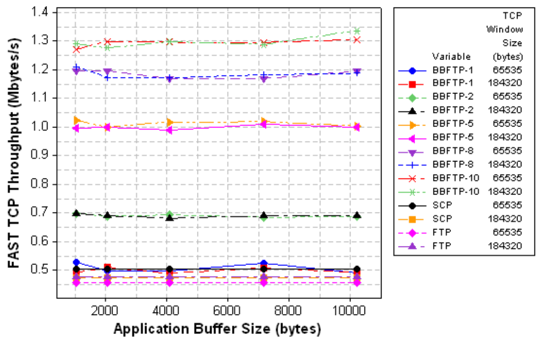

Figure 7. Throughput measurements of FAST TCP for different settings of TCP window size and application buffer size

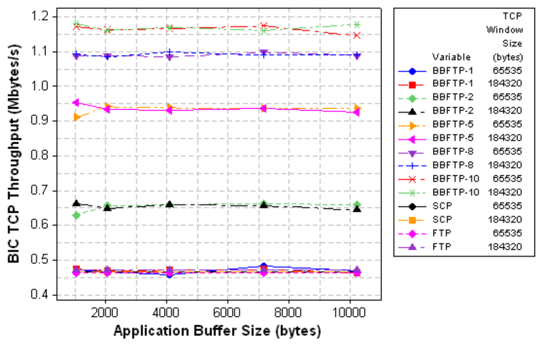

Figure 8. Throughput measurements of BIC TCP for different settings of TCP window size and application buffer size

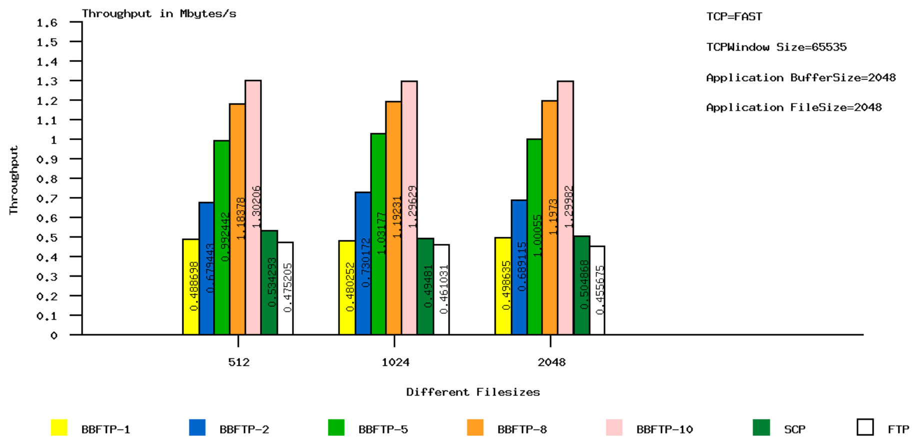

Figure 9. Throughput measurements of Fast TCP for different file sizes

test cases are almost identical but with minor variations for any of the particular application buffer sizes, both with and without TCP tuning. This can be explained as due to the magnitude of the bandwidth*RTT product value between *nts-wsu* and *nts-osc*. As described in [5] [7] [15], the bandwidth*RTT product needs to be in the order of several MB (in our case it was 180 Kbytes) to produce significant improvements in the throughput measurements. Hence, we concluded that tuning TCP window and application buffer sizes was not a matter of serious consideration for our routine DR operations.

Interestingly, the throughput measurements from FTP and SCP test cases showed absolutely no variation with the changes in application buffer size, both with and without TCP tuning. This observation can be attributed to the fact that both SCP and FTP use OpenSSL [14], which uses statically defined internal flow control buffers i.e., it uses its own buffering/windowing mechanisms, which are independent from the TCP flow control buffers. These mechanisms have been known to greatly reduce TCP performance, especially on long-delay, high bandwidth network links [5].

4.4 Effects of File Sizes

Figure 9 shows the throughput measurements while transferring different file sizes (i.e. the 0.5 GB, 1 GB and 2GB swapfiles) using different file transfer applications for test case iterations involving the FAST TCP stack. In these test case iterations, the TCP window size was set to 65535 bytes and the application buffer size was set to 2048 bytes. These test cases were executed on a week-day and during regular business hours of the Wright State University. We can note from Figure 9 that the throughput measurement results are identical for any of the file sizes. Similar results were obtained for test cases involving Reno and BIC TCP stacks.

Above observation can be attributed to the fact that there was no competing application traffic along the network path during all the data transfers. Even the optimum large-scale data throughput values (approximately 10.5 Mbps) were not affected, indicating that our tests were not disrupting the production network path. Thus, we realized that adequate network bandwidth provisioning was critical for the DR operations in order to not affect other application traffic. Further, the amount of bandwidth provisioning would have to meet the worst-case DR data backup and retrieval requirements, which in turn are influenced by the file sizes of daily incremental backups, data retrieval times in the event of a disaster and the scale of real-time data mirroring requirements of each DR-client site.

5 Conclusion and Future Work

In this paper, we presented a DR pilot study that addressed different network tuning and monitoring issues involving OSC's mass storage site and Ohio's academic institution sites. In particular, we described the testing methodology that was used to perform large-scale data transfer throughput measurements using popular file transfer applications such as FTP, SCP and BBFTP on a test path between Wright State University and OSC's mass storage site. The test cases used an iterative approach that produced throughput measurements for popular TCP tuning techniques such as using specialized TCP stacks: Reno, FAST and BIC, parallel TCP streams, and different TCP window size and application buffer size configurations.

The analysis of the measurements demonstrated that large-scale data transfer throughput can be significantly increased using appropriate network tuning techniques. However, aggressively pushing the transfer limits could disrupt other application traffic. Hence, appropriate end-to-end bandwidth between the DR-client sites and the remote DR backup site needs to be provisioned based on the file sizes of daily incremental backups, data retrieval times in the event of a disaster and based on the scale of real-time data mirroring. We also observed that increase in throughput was not proportional to the increase in the number of parallel TCP streams. Increasing up to an optimum number of parallel streams by trial-and-error allows us to benchmark the optimum large-scale data throughput for any given DR-client site.

Another notable result from the testing was that the bandwidth*RTT product values between the DR client sites and OSC's mass storage site were not large enough. As a result, TCP tuning with different TCP window size and application buffer size configurations was proven to be irrelevant for improving the throughput measurements for routine DR operations. Finally, we found that using different file sizes did not influence throughput measurements with any of the TCP tuning techniques, provided there was adequate end-to-end network bandwidth provisioning between the DR-client site and OSC's mass storage site.

As future work, we plan to deploy network measurement servers at each of the DR-client sites. The measurement servers will be equipped with OSC's ActiveMon [20] software. The ActiveMon software supports on-going and on-demand measurements using various bandwidth estimation tools such as Iperf, Pathchar, Pathload and Pathrate. We plan to use the analysis and visualization features of ActiveMon to routinely and pro-actively check for network performance bottlenecks that hinder efficient and timely completion of DR data backup and retrieval jobs at the DR-client sites.

6 Acknowledgements

We would like to thank Dr. Steve Gordon of OSC for providing the motivation and start-up ideas for this study. We would also like to thank Paul Schopis and Weiping Mandrawa of OSC for their helpful discussions regarding networking related issues pertaining to the DR pilot testbed.

References

[1] S. Shalunov, R. Carlson, "Detecting Duplex Mismatch on Ethernet", *Proceedings of Passive and Active Measurement Workshop*, 2005.

[2] R. Whitehead, "Network Analysis and Data Protection Services", *Apparent Networks User Conference*, 2006.

[3] M. Peek, "An Ounce of Prevention: Making the move from Reactive to Proactive", *Apparent Networks User Conference*, 2006.

[4] T. Hacker, B. Noble, B. Athey, "Improving Throughput and Maintaining Fairness using Parallel TCP", *Proceedings of IEEE INFOCOM*, 2004.

[5] B. Tierney, "TCP Tuning Guide for Distributed Application on Wide Area Networks", *LBNL Technical Report 45261*, 2000. http://www-didc.lbl.gov/TCP-tuning

[6] P. Dykstra, "Gigabit Ethernet Jumbo Frames", *WWW Document*, 1999. http://sd.wareonearth.com/ phil/jumbo.html

[7] M. Mathis, "Pushing up the Internet MTU", *Internet2/ESCC Joint Techs Workshop*, 2003.

[8] P. Calyam, D. Krymskiy, M. Sridharan, P. Schopis, "Active and Passive Measurements on Campus, Regional and National Network Backbone Paths", *Proceedings of IEEE ICCCN*, 2005.

[9] G. Farrache, "BBFTP: Open-source File Transfer Software", *WWW Document*, 2005. http://doc.in2p3.fr/bbftp

[10] W. Stevens, "TCP/IP Illustrated: Volume I", *Addison-Wesley Press*, 1994.

[11] C. Jin, D. Wei, S. Low, G. Buhrmaster, J. Bunn, D. Choe, L. Cottrell, J. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh, "FAST TCP: From Theory to Experiments", *IEEE Network Magazine*, 2005.

[12] L. Xu, K. Harfoush, I. Rhee, "Binary Increase Congestion Control for Fast Long-Distance Networks", *Proceedings of IEEE INFOCOM*, 2004.

[13] J. Postel, J. Reynolds, "File Transfer Protocol (FTP)", *IETF RFC 959*, 1985.

[14] D. Barrett, R. Silverman, R. Byrnes, "SSH: The Secure Shell", *O' Reilly Press*, 2001. http://www.openssh.org

[15] E. Weigle, W. Feng, "A Comparison of TCP Automatic Tuning Techniques for Distributed Computing", *Proceedings of IEEE HPDC*, 2002.

[16] J. Semke, J. Mahdavi, M. Mathis, "Automatic TCP Buffer Tuning", *Proceedings of ACM SIGCOMM*, 1998.

[17] H. Bullot, L. Cottrell, R. H-. Jones, "Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks", *Proceedings of PFLDnet*, 2004.

[18] L. Jorgenson, "Size Matters: Using Jumbo Packets on Gigabit Ethernet", *Apparent Networks Technical Whitepaper*, 2003.

[19] L. Zhang, S. Shenker, D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way traffic", *Proceedings of ACM SIGCOMM*, 1991.

[20] P. Calyam, D. Krymskiy, M. Sridharan, P. Schopis, "TBI: End-to-End Network Performance Measurement Testbed for Empirical-bottleneck Detection", *Proceedings of IEEE TRIDENTCOM*, 2005. http://www.osc.edu/oarnet/itecohio.net/activemon

[21] Sun Microsystems, "ZFS: The Last Word in File Systems", *SUN Microsystems Technical White Paper*, 2004. http://www.sun.com/2004-0914/feature

[22] L. Cottrell, "SLAC Bulk Throughput Measurements", *WWW Document*, 2003. http://www-iepm.slac.stanford.edu/monitoring/bulk

[23] A. Tirumala, L. Cottrell, T. Dunigan, "Measuring End-to-end Bandwidth with Iperf using Web100", *Proceedings of Passive and Active Measurement Workshop*, 2003. http://dast.nlanr.net/Projects/Iperf

[24] Sun Microsystems, "Solaris Containers: Server Virtualization and Manageability", *SUN Microsystems Technical White Paper*, 2004.

[25] A. B. Downey, "Using Pathchar to Estimate Internet Link Characteristics", *Proceedings of ACM SIGCOMM*, 1999.

[26] C. Dovrolis, P. Ramanathan, D. Moore, "Packet Dispersion Techniques and Capacity Estimation", *IEEE/ACM Transactions on Networking*, 2004.