Performance analysis of high-performance file transfer systems for Grid applications

Cosimo Anglano, Massimo Canonico
Dipartimento di Informatica, Universitá del Piemonte Orientale
Alessandria, Italy
{cosimo.anglano,massimo.canonico}@unipmn.it

Abstract

Data-intensive Grid applications require the availability of tools able to transfer very large files in the shortest amount of time. Many file transfer tools, based on solutions aimed at overcoming the limitations imposed by the TCP protocol, have been recently developed. In this paper we experimentally compare the performance of some of these tools in various network scenarios by running experiments on PlanetLab, an open platform for the development, deployment, and access of planetary-scale services, that comprises hundreds of hosts scattered across the globe. Our results show that solutions based on UDP and adopting rate-based algorithms result in better performance than other alternatives in most cases, while solutions based on TCP achieve similar performance only under specific circumstances.

1 INTRODUCTION

Data-intensive applications are increasingly become commonplace in the Grid. The scientific exploration in many disciplines, like High Energy Physics, Climate Modeling, and Life Sciences, requires indeed the processing of massive data collections. For instance, the amount of data that must be processed in many High-Energy Physics experiments is in the order of Terabytes (and sometimes even Petabytes) [5, 6, 7], while typical climate modeling applications generate output datasets whose size is in the order of hundreds of Gigabytes [11]. To achieve satisfactory performance, data-intensive grid applications require the availability of a system able to transfer potentially huge files in the shortest possible amount of time [11], as their completion time is often dominated by the time taken to transfer data [26]. An apparently straightforward way to reduce data transfer times is to increase the bandwidth of the networks interconnecting the resources in a Grid. Recent advances in communication technology are indeed enabling the deployment of Gigabit-per-second networks on geographic scale at a reasonable cost, and a few testbeds of this type have been already deployed. These networks, that are often called *high Bandwidth Delay Product (BDP) networks*, are characterized by very high round-trip latencies because of the distances covered by their links. Despite the very high capacity provided by high BDP networks, empirical and analytical evidence shows that the performance achieved by file transfer tools based on TCP, the *de facto* standard transport protocol used in the Internet, is rather unsatisfactory. As a matter of fact, the windowing mechanism used by TCP imposes a limit on the amount of data sent before waiting for an acknowledgment. This in turn implies that, given the typically large delays of high BDP networks, the sender will spend an inordinate amount of time waiting for acknowledgments, with the consequence that only a small fraction of the available network capacity is actually exploited. Therefore, simply increasing the available network bandwidth not necessarily translates into proportional improvements in data

transfer performance, unless optimized TCP versions are used.

Of course, by properly setting the TCP window size to a value matching the BDP of the network, it is possible to exploit a large fraction of the available network capacity [13, 25]. Unfortunately, this operation not always can be performed by users with standard privileges [10] (for instance, in many Unix variants, the maximum buffer size is a system-wide value that can be modified only by the superuser). Moreover, it may be necessary to dynamically change the window size, since both the bandwidth available on a given network path, and the corresponding latency, may greatly vary over time because of the dynamic nature of the cross traffic usually present in production networks (like the Internet). These limitations make manual tuning of TCP parameters inadequate for the purpose of achieving very high file transfer performance over production networks.

In response to the need of overcoming these limitations, new transfer techniques, working either at the system or at the application level, are being developed. System level solutions, that usually require modifications to the operating system of the machines, of the network equipment, or of both, include automatic tuning of TCP window sizes [1], (sometimes complemented by other sophisticated mechanisms [8]), and the development of new TCP versions (e.g., Selective Acknowledgment TCP [24], High Speed TCP [17], and Scalable TCP [23]). This approach can yield very good performance, but usually requires significant updates of the networking infrastructure, so its practical applicability seems, at the moment, rather limited. Conversely, application level solutions do not require any modification to the networking infrastructure, since they attempt to circumvent TCP problems by exploiting techniques not needing any special privilege, and are therefore readily usable in current networks. Application-level solutions can be classified according to the transport protocol (TCP or UDP) they use to transfer data among machines. TCP-based solutions (e.g., GridFTP [28], bbFTP [4], and bbcp [20]) use TCP connections to move data, and attempt to circumvent TCP's window size problems by using parallel streams, so that an aggregate congestion window matching the BDP of the network path used to carry out the transfer, is obtained. Conversely, UDP-based solutions (e.g., FOBS [15], Tsunami [29], UDT [18], and SABUL [19]), use UDP to move data, and employ rate-based control algorithms to match the transmission speed with the available network bandwidth, sometimes coupled with congestion and flow control algorithms in order to keep as low as possible the loss rate during transmissions.

Given the relatively large number of application-level high-performance transfer tools that have been developed in the recent past, the question about the effectiveness of each of them arises naturally. However, although prototypes of many of the above systems have been around for a while, an experimental study aimed at comparing them is still lacking in the literature. To the best of our knowledge, the only study of this type has been carried on LambdaGrids [22], whose peculiarities make the corresponding results inapplicable to more conventional Grid networking infrastructures, like those considered in this paper. This paper aims at filling this gap by presenting the results we obtained in an experimental study aimed at answering the following questions concerning application-level high-performance file transfer tools:

1. Are TCP-based and UDP-based solutions equivalent, or one approach yields better performance than the other one, when production networks are used?
2. Is there a data transfer tool that works better than the other ones?
3. How do these tools work for production networks with relatively low BDP values?

Our study was performed by selecting, among the tools mentioned before, those that are, in our opinion, representative of their class, and by repeatedly using them to transfer rather large files among machines belonging to the PlanetLab [14] infrastructure. The results we obtained indicate that, while for relatively low BDP networks both TCP-based and UDP-based approaches achieve similar performance, for higher BDP networks UDP-based tools are definitely the only viable option.

The rest of the paper is organized as follows. In Section 2 we briefly describe the main characteristics of the tools we compared. In Section 3 we discuss the experimental methodology we used, report the results we obtained, and highlight the differences among the various tools. Finally, Section 4 concludes the paper.

2 FILE TRANSFER TOOLS

As anticipated in the Introduction, many high-performance application-level file transfer tools have been developed in the recent past. To the best of our knowledge, the exhaustive list of these systems includes FOBS [15], Tsunami [29], SABUL [19], UDT [18], RBUDP [21], GTP [30], GridFTP [28], bbcp [20], bbFTP [4], and PSockets [27]. In our study, however, we considered only a subset of the above tools, namely FOBS, UDT, RBUDP, and bbFTP, for the following reasons. Tsunami seems to be a very promising tool, but its current implementation suffers from a bug [12] that makes it hang (and slow down also other processes) from time to time, making it hard to perform an extensive experimentation. UDT is the successor of SABUL, so it should exhibit higher performance, while GTP is specific for Lambda Grids and focuses on multipoint-to-point transfers. Among the TCP-based tools, we selected bbFTP as representative of the whole class, since we believe that all of them offer similar performance. Furthermore, GridFTP requires the installation of various Globus components, we did not have access to any PSockets implementation, and bbcp posed us some problems in using it through scripts, so these tools were not considered in our evaluation. In the rest of this Section, we will describe in more detail the tools we considered in our study.

2.1 bbFTP

bbFTP is a file transfer system that uses multiple TCP streams between the same endpoints to speed-up data transfers when the maximum TCP window size is lower than the BDP of the network path used to transfer data. The rationale of using N independent TCP streams, each one characterized by a maximum window size of B bytes, lies in the fact that it roughly corresponds to using a single TCP stream having a windows size of $N \cdot B$ bytes. Therefore, by properly choosing the number N of parallel streams, it is possible to match the BDP of the network. Of course, the larger the number of streams, the larger the overhead experienced by the sending and receiving machine, so N should be set to the highest value that makes the aggregate window size smaller than (or equal to) the path BDP. It is worth to point out that bbFTP does not provide any automatic means of choosing the number of streams, whose responsibility is entirely left to the user. In order to further speed up data transfers, bbFTP uses several aggressive optimization techniques, like on-the-fly data compression and automatic sizing of individual TCP windows. Finally, bbFTP provides a set of advanced functionalities concerning authentication and interfacing with various I/O subsystems. We tested version 3.0.2, downloaded on Jan. 27th, 2004.

2.2 Reliable Blast UDP (RBUDP)

The Reliable Blast UDP system (RBUDP), a part of the Quality of Service Adaptive Networking Toolkit (QUANTA) [3], belongs to the family of UDP-based solution. With RBDUP, hosts exchange data packets via UDP, and control packets via TCP. More specifically, a transfer is carried out in the following way. In the first data transmission phase, the source machine sends the whole file, at a constant, user-specified rate, to the machine requesting it. At the end of this phase, the receiver sends back (via a TCP connection) a list of lost packets, that are retransmitted again as a "blast". This process is repeated until no more packets need to be retransmitted. RBDUP requires the user to specify the rate at which data are sent, and this rate is kept constant for the whole transfer duration. RBUDP's authors recommend to specify a sending rate not larger than the bandwidth of the bottleneck link of the path connecting the two hosts, that must be determined prior to the transfer by using a suitable tool like Iperf [2] (or its extended version app_perf developed as part of RBUDP). We tested the version distributed with QUANTA 0.3, downloaded on Feb. 18th, 2004.

2.3 FOBS

FOBS is a user-level, UDP-based data transfer mechanisms in which, unlike RBUDP, there is some degree of interaction between the sender and the receiver during the transfer. More specifically, the sender works by transmitting (as fast as possible) a batch of fixed size data packets, that includes both unsent and retransmitted packets. Periodically, the receiver sends acknowledgment packets to the sender, containing a list of unreceived packets, that is used by the sender to decide how many lost packets will be resent in the the next batch. The version of FOBS used in our experiments includes also a congestion control mechanism [16] used to adjust the sender's speed to the conditions of the network. We tested the version downloaded on Jan. 12th, 2004.

2.4 UDP-based Data Transfer protocol (UDT)

The UDT protocol, the latest version of the Simple Available Bandwidth Utilization Library (SABUL) [19], uses UDP to transfer data between pairs of hosts, and combines rate-based, window-based and delay-based control mechanisms to deliver high throughput and low loss data transmission. UDT implements slow start and AIMD control schemes for flow control (which makes it more TCP friendly than the other rate-based schemes), and window-based control for controlling the number of outstanding packets in flight. UDT employs also rate adjustment based on delay monitoring, providing improved performance over classical AIMD schemes. More specifically, the sender calculates the inter-packet time, which is updated by the rate-control algorithm. Packets are sent out every inter-packet time, but the number of outstanding packets cannot exceed the threshold imposed by the window size, that in turn is dynamically adjusted to match the current network conditions. The receiver reorders, if necessary, the packets it receives, and uses selective acknowledgments, sent at a constant rate, to request lost packets. We tested version 1.1, downloaded on Jan. 12th, 2004.

3 EXPERIMENTAL EVALUATION

As anticipated in the Introduction, the goal of our study was to compare the above tools from the perspective of a user that has the need of transferring large files over production networks. Therefore, in our evaluation we considered only user-perceived performance indices, and in particular the throughput achieved during the transfer, and not other figures of merit (e.g., fairness w.r.t. other traffic flows, fast adjustment to network capacity, etc.) that could give insights into the behavior of each tool.

3.1 Experimental testbed

In order to carry out our comparison, we set up a testbed comprising a set of machines belonging to the PlanetLab infrastructure. All these machines use the Linux 2.6.x kernel, suitably extended with the functionalities needed to run the PlanetLab services, and have the same software configuration (the hardware characteristics may instead be different from a machine to another). In order to compare the various tools in a variety of scenarios, we used both machines connected by paths with a relatively high BDP value (from 2.5 to 3.75 MB), and machines connected by paths with lower BDP values (from 250 to 800 KB). The machines included in our testbed are listed in Table 1, where we report both their Fully Qualified Domain Names and the corresponding short names we used in order to keep the various tables within the page margins.

3.2 Evaluation methodology

Our evaluation has been carried out by repeatedly transferring rather large files among pairs of machines, and by measuring the throughput achieved during each transfer. In order to evaluate the performance of the various tools in different operational condition, we performed experiments transferring both 100 MB and 500 MB files. Since the performance of tools using parallel TCP streams strongly depends on the number of TCP connections used to

Table 1: Machines belonging to the testbed

Short name	Fully Qualified Domain Name	Location
Mont	planet1.montreal.canet4.nodes.planet-lab.org	Montreal (Canada)
Otta	planet1.ottawa.canet4.nodes.planet-lab.org	Ottawa (Canada)
Atla	planetlab1.atla.internet2.planet-lab.org	Atlanta (USA)
Hous	planetlab1.hstn.internet2.planet-lab.org	Houston (USA)
LosA	planetlab1.losa.internet2.planet-lab.org	Los Angeles (USA)
Sidn	planetlab1.it.uts.edu.au	Sidney (Australia)
Taiw	planetlab1.im.ntu.edu.tw	Taiwan
Kais	cplanetlab1.kaist.ac.kr	Korea

carry out the transfer, for bbFTP we ran experiments using 5, 10, 15, and 20 parallel streams.

Performing a comparison based on measurements collected on production networks is not an easy task, since the presence and the variation of cross traffic may differently affect the various tools, thus biasing the results. In order to minimize these effects, we organized our experiments in *rounds*, each one consisting in the execution (in a "back-to-back" fashion) of all the file transfers between the same pair of machines. The rationale of this choice lies in the consideration that dramatic changes in the cross-traffic between the same pair of machines are unlikely to occur during relatively short periods of time (in our experiments, the duration of each round was less than 30 minutes). In order to take into account longer term variations of the cross traffic, we performed a sequence of rounds, spanning about two weeks (for the 100 MB file transfers), and used the the measurements collected during the various download to compute, for each tool, its average throughput for each machine pair. In order to quantify the possible effects of cross traffic flows, we computed also the standard deviations of the measured values. Intuitively, higher standard deviations indicate larger variations of cross traffic, while smaller values indicate a relative stability of network conditions across the various rounds we performed.

For the evaluation involving 500 MB files we adopted the same approach, but, in order to reduce the total amount of traffic transferred over the PlanetLab infrastructure, we performed less transfer experiments than in the 100 MB case. If the number of experiments had not been scaled down, the resulting amount of generated traffic would have violated the PlanetLab's "Acceptable User Policy". Therefore, for 500 MB files we considered a smaller number of machine pairs, and performed a smaller number of rounds.

As stated in Section 3.1, PlanetLab machines may differ in their hardware and software configuration. In order to avoid possible biases in our results because of these differences, the results obtained from the various machine pairs considered in our evaluation have been kept separated. Since the hardware and software configurations of the machines did not change across the various experiments, the results obtained for the various tools do not depend on specific characteristics of the involved hosts.

3.3 Experimental results: 100 MB files

The results obtained with 100 MB files are reported in Table 2, where each row corresponds to a particular $< sender, receiver >$ pair and reports the average throughput value (expressed in Mbit/s.), and the respective standard deviation, for each of the tools we considered. For each row, the tool that achieved the best performance is highlighted by using bold fonts for the corresponding throughput value. For each $< sender, receiver >$ pair we report also the corresponding Bandwidth-Delay-Product (BDP) value (expressed in MB), computed as the product of the

Table 2: Experimental results for 100 MB files. Average throughput values are expressed in Mbit/s.

Hosts			FOBS		RBUDP		UDT		bbFTP		
Sender	Receiver	BDP (MB)	Avg	St. Dev.	Avg	St. Dev.	Avg	St. Dev.	Avg	St. Dev.	TCP Streams
Hous	Atla	0.25	9.34	0.47	**86.29**	**2.47**	84.28	3.3	87.94	0.51	5
Mont	Otta	0.29	9.71	0.22	77.74	19.77	**90.4**	**5.59**	89.4	1.76	15
Otta	Atla	0.6	8.82	0.81	**86.25**	**3.63**	81.03	3.06	79.70	3.29	10
Atla	Otta	0.61	9.17	0.43	75.87	19.24	**86.02**	**3.17**	80.37	5.15	15
Mont	Atla	0.61	9.66	0.2	**86.71**	**1.64**	82.31	2.86	86.04	0.14	10
Hous	Otta	0.8	9.27	0.4	76.15	19.13	**82.96**	**4.49**	81.35	5.11	5
Hous	Taiw	2.77	7.18	2.48	**21.73**	**8.71**	10.06	1.87	1.65	0.26	15
Atla	Taiw	2.78	7.1	2.56	**22.36**	**9.28**	11.17	1.85	1.78	0.22	15
Taiw	Taiw	2.85	7.17	0.51	**10.34**	**1.82**	10.11	1.43	9.62	0.56	15
Mont	Taiw	2.92	6.48	2.98	**25.29**	**6.93**	10.65	2.98	1.72	0.2	15
Otta	Taiw	3.75	5.7	2.8	**25.06**	**8.78**	11.18	2.1	1.63	0.29	15
Taiw	Otta	3.64	7.24	0.42	**10.58**	**0.98**	9.98	1.64	9.34	0.92	15

path capacity (measured by means of pathrate [9] and properly scaled to take into account possible bandwidth limitations on PlanetLab nodes) and the round-trip-time.

As shown by the results in the table, the best tool for paths characterized by a relatively small BDP value (from 250 to 800 KBytes), was either RBUDP or UDT. More specifically, RBUDP provided better results when the receiver was Atla (its throughput was from 2% to 6% higher than those obtained by UDT), while UDT achieved better performance when the receiver was Otta (its throughput was from 8.9% to 16.2% higher). The performance of bbFTP for these paths can be considered quite good as well, as indicated by the respective throughput values we measured. However, as indicated by the values reported in the TCP Streams column, it is rather hard to properly choose the number of parallel streams that result in the best performance for a given machine pair. Network paths with similar BDP values required indeed a different number of TCP streams (for instance, 5 for the $< Atla, Hous >$ pair and 15 for the $< Otta, Mont >$ pair). Finally, the performance of FOBS on lower BDP paths was quite disappointing, as in these cases it achieved a throughput 8 or 9 times smaller than UDT, RBUDP, or bbFTP, showing that UDP-based solutions that do not adopt a rate-based approach in order to match the transmission speed to the available network bandwidth (as instead done by RBUDP and UDT) may fail to achieve adequate performance. The small values for the standard deviations computed for these paths indicate also that the results were not polluted by cross-traffic variations.

For paths with larger BDP values (i.e., from 2.77 to 3.75 MB), the results are different. In particular, when Taiw was the sender, RBUDP and UDT exhibited very similar performance, while the other tools obtained a smaller throughput, but were more or less equivalent to each other. However, when Taiw was the receiver, the differences among the various tools became very evident. In particular, the best tool was by far RBDUP that, with 25 Mbit/s., showed improvements from 124% to 137% w.r.t. UDT, from 219% to 394% w.r.t. FOBS, and from 1156% to 1437% w.r.t. bbFTP. We hypothesize that this behavior is due to the effects of the bandwidth-limiting mechanisms used on some PlanetLab nodes (for the Taiw machine a 10 Mbit/s. bandwidth limit was set for each account). This hypothesis seems to be supported by the fact that, not considering RBUDP, the throughput achieved by all the tools is very close to

the bandwidth limit of 10 Mbit/sec. imposed on the Taiw machine. These mechanisms seem to be effective only for those tools that employ some form of congestion control (in practice, all the tools but RBUDP), while they appear ineffective for RBUDP that, as already discussed, does not encompass any congestion control feature and hence might be able to unhinge them. As additional evidence to this hypothesis, consider that for all the tools the observed standard deviation was quite low, indicating the stability of the network conditions across all the experiments, while for RBDUP was considerably higher, possibly indicating that sometimes the limiting mechanisms had some effects also on RBUDP, but that in most cases they were ineffective.

3.4 Experimental results: 500 MB files

In order to verify whether the results obtained with 100 MB files allow one to draw general conclusions about the performance of the tools we considered, we ran a set of experiments in which the size of the transferred files was set to 500 MB. In these experiments, however, we did not consider all the tools mentioned before, as we excluded (for different reasons) FOBS and RBUDP. FOBS was not included since, as shown by the results obtained with 100 MB files, it is much slower than the other tools, so it would have probably taken too much time to transfer a 500 MB files, thus increasing the likelihood of significant cross-traffic variations during the execution of individual rounds. RBUDP was instead excluded from this set of experiments because of its inability to complete the transfers, probably due to a memory allocation problem. More precisely, we have experimentally observed that, especially for network paths characterized by lower BDP values, the resident memory size of the RBUDP receiver process quickly grows until the available RAM is saturated, and this makes it crash. We suspect this problem may be ascribed to the Selective Acknowledgment mechanism used by RBUDP, that forces the receiver process to keep into its buffer a packets until all its predecessors have been correctly received. This effect did not show up for 100 MB files, as in that case a smaller buffer was sufficient to store these packets, while for 500 MB files the amount of memory required to buffer packets exceeded the memory limit associated with the receiver process.

The results obtained with 500 MB files are reported in Table 3, whose structure is identical to that of the table shown in the previous section. The results we obtained confirm the conclusions we draw from the experiments involving 100 MB files. In particular, we observe that, while for network paths characterized by a low BDP value bbFTP achieves performance similar to, or better than, UDT, for higher BDP values UDT represents a better choice. Again, it is worth to point out that it is rather hard to choose the optimal number of TCP streams to be used with bbFTP, as indicated by the fact that for the highest BDP value the best performance were obtained with 10 streams, while 20 were required in all the other cases.

4 CONCLUSIONS

In this paper we presented a comparison of available high-performance data transfer tools for data-intensive Grid applications. Our results indicate that, while for relatively low BDP networks both TCP-based and UDP-based approaches achieve similar performance, for higher BDP networks UDP-based tools are definitely the only viable option. However, simply using UDP does not automatically result in good performance, as demonstrated by FOBS that, not using any rate-based mechanism, was not able to achieve performance comparable with those attained by the other UDP-based tools. As a final consideration, we note that using UDT is definitely simpler than using RBUDP (not requiring any estimation of the available bandwidth) and bbFTP (not requiring any estimation of the proper number of TCP streams). Moreover, UDT has proved to be stable even for 500 MB files, while in this case RBDUP did not allow us to complete the transfer experiments.

Although these tools have been developed for very high BDP networks (characterized by BDP values much higher than those considered in this paper), and hence when used on relatively low BDP networks may show unexpected and unplanned behaviors, we

Table 3: Experimental results for 500 MB files. Average throughput values are expressed in Mbit/s.

Hosts			UDT		bbFTP		
Sender	Receiver	BDP (MB)	Avg	St. Dev.	Avg	St. Dev.	TCP Streams
Sidn	LosA	0.23	5.07	0.9	**9.98**	0.23	20
Sidn	Atla	0.27	4.34	1.35	**9.75**	0.44	20
Atla	LosA	0.64	83.11	1.14	**87.7**	1.15	20
Atla	Sidn	2.43	**10.76**	1.94	8.34	0.68	20
Atla	Kais	3.25	**17.52**	0.75	13.55	1.91	10

believe that our comparison makes sense from a practical point of view. We indeed believe that the vast majority of Grid users will not have access to very high-speed networking infrastructures, so a comparison of available high-performance transfer tools on production networks may provide them with valuable information.

References

[1] Automatic TCP Window Tuning and Applications. http://dast.nlanr.net/Projects/Autobuf_v.10/autotcp.html.

[2] Iperf: The TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects/Iperf.

[3] QUANTA Home Page. http://www.evl.uiuc.edu/cavern/quanta. Accessed on March 28th, 2004.

[4] The bbFTP – Large Files Transfer Protocols Web Site. http://doc.in2p3.fr/bbftp.

[5] The Grid Physics Networks (GriPhyN) project. http://www.griphyn.org.

[6] The Large Hadron Collider (LHC) project. http://lhc.web.cern.ch/lhc.

[7] The Particle Physics Data Grid Project. http://www.cacr.caltech/edu/ppdg.

[8] The Web100 Project. http://www.web100.org/.

[9] The BW-meter project Home Page. http://www.pathrate.org, 2004.

[10] The TCP Tuning Cookbook. http://proj.sunet.se/E2E/tcptune.html, 2004.

[11] B. Allcock, J. Bester, J. Bresnahan, A.L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data Management and Transfer in High-Performance Computational Grid Environments. *Parallel Computing*, 28(5):749–771, May 2002.

[12] S. Ansari. Tsunamy - A Study. http://www-iepm.slac.stanford.edu/bw/Tsunami.htm.

[13] B. Tierney. TCP Tuning Guide. http://www-didc.lbl.gov/TCP-tuning/.

[14] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communications Review*, 33(3), July 2003. http://www.planet-lab.org.

[15] P.M. Dickens. FOBS: A Lightweight Communication Protocol for Grid Computing. In *Proc. of Europar 2003*, Klagenfurt, Austria, August 2003.

[16] P.M. Dickens and V. Kannan. Application-Level Congestion Control Mechanisms for Large Scale Data Transfers Across Computational Grids. http://www.cs.iit.edu/ pmd/FOBS/fobs_papers.html, 2004. Submitted for publications.

[17] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649, Dec. 2003.

[18] Y. Gu and R.L. Grossman. Using UDP for Reliable Data Transfer over High Bandwidth-Delay Product Networks. http://www.dataspaceweb.net/papers.htm, 2003. Submitted for publication.

[19] Y. Gu, X. Hong, M. Mazzucco, and R.L. Grossman. SABUL: A High Performance Data Transfer Protocol. http://www.dataspaceweb.net/papers.htm, 2003. Submitted for publication.

[20] A. Hanushevsky, A. Trunov, and L. Cottrell. Peer-to-Peer Computing for Secure High Performance Data Copying. In *Proc. of the 2001 Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP 2001)*, Beijng, China, September 2001.

[21] E. He, J. Leigh, O. Yu, and T. DeFanti. Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. In *Proc. of 5th Int. Conf. on Cluster Computing*, 2002.

[22] R. Huang and A. Chien. Benchmarking High Bandwidth Delay Product Networks. Technical report, Concurrent Systems Architecture Group, University of California, San Diego. Retrieved on March 28th, 2004.

[23] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. In *Proc. of First Int. Workshop on Protocols for Fast Long-Distance Networks*, CERN, Geneva, Switzerland, February 2003.

[24] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment. RFC 2018.

[25] R.S. Prasad, M. Jain, and C. Dovrolis. Socket Buffer Auto-Sizing for High-Performance Data Transfers. *Journal of Grid Computing*, 1(4), December 2003.

[26] K. Ranganathan and I. Foster. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[27] H. Sivakumar, S. Bailey, and R.L. Grossman. PSockets: The Case for Application-Level Network Striping for Data Intensive Applications using High Speed Wide Area Networks. In *Proc. of Supercomputing 2000*, 2000.

[28] The Globus Team. GridFTP: Universal Data Transfer for the Grid. http://www.globus.org. White Paper.

[29] S. Wallace. Tsunami File Transfer Protocol. In *Proc. of First Int. Workshop on Protocols for Fast Long-Distance Networks*, CERN, Geneva, Switzerland, February 2003.

[30] R.X. Wu and A. Chien. GTP: Group Transport Protocol for Lambda-Grids. In *Proc. of 4th ACM/IEEE Int. Symp. on Cluster Computing and the Grid*, Chicago,USA, 2004.