U.P.B. Sci. Bull., Series C, Vol. 76, Iss. 3, 2014 ISSN 2286-3540

AN APPLICATION LEVEL MULTI-PATH ALGORITHM FOR FAST DATA STREAMING

Dan SCHRAGER[1]

Data intensive computing applications which require access to geographically distributed data are increasingly important in the grid and cloud computing field. Therefore I designed a new algorithm capable of transferring bulk data between remote hosts with support for multi-path at application level, based on the fork-join queueing model, by combining the use of classical pipes as a mean of serializing data locally with multiple parallel TCP connections which ensure high bandwidth throughput. My experiments in 100 Mbps, 1 Gbps and up to 8 Gbps high bandwidth networks have proven the validity of algorithms in static as well as in dynamic conditions. Transfers of arbitrary data streaming were done effectively, efficiently, in parallel, between pairs of distributed processes connected via extended fast pipes.

Keywords: pipeline, parallel data streaming, multi-path
MSC2000: 53C 05.

1. Introduction

My research is placed in the context of grid and cloud computing where the ability to transfer large amounts of data over wide area networks is increasingly important. For example, the Atlas experiment [2] which is transferring 10 PB of data a year; or the VIRGO collaboration [26] that has storage needs in the order of many hundreds of TB, and very low latency requirements.

The task of moving large files among distant hosts is usually handled by using modified enhanced versions of TCP [20] (e.g. SACK [23], Vegas [21], HighSpeed [25], FAST [8], MulTCP [19]), active queue management algorithms in routers (e.g. RED [12], BLUE [13], SFB [14]), non-TCP protocols (e.g. XCP [9], SCTP [10], DCCP [11]), or dedicated applications and libraries which either use networking parallelism (e.g. XFTP [22], PSockets [18], GridFTP [27]) or UDP [28] as a transport means (e.g. RBUDP [17], SABUL [29]). While TCP modifications are backward compatible they typically require long standardization times and are difficult to disseminate; non-TCP protocols require usually expensive changes in routers or dedicated or at least rewritten applications; libraries have to be included in new applications, while existing programs may be too specialized.

[1]PhD Student, University *Politehnica* of Bucharest, Romania, e-mail: `danschrager@gmail.com`

My approach is based on a pipeline mechanism, of high speed and capacity, which interconnects, at application level, pairs of data producer/consumer processes running on different hosts in a high speed network. It is worth emphasizing the generality and elegance of the approach which manages to exploit, through encapsulation, the pipe programming paradigm - specific to modern operating systems (Unix) - based on reuse of existing system components (e.g. *cat, dd, tar*) or dedicated storage and data access tools (e.g. Castor [4] at Cern [5], *rfcat* [6], *rftar* [7]), as opposed to building new monolithic applications.

In terms of its ability to use efficiently multiple paths of different bandwidth, my application resembles to Multipath TCP [1] which is however implemented in the operating system (Linux) kernel and so suffers, as the majority of TCP extensions, from difficulties of dissemination and lenghty standardization times.
Compared to the class of applications or library functions for networking transfers using multiple concurrent TCP connections, which is a member of, my application differs from previous achievements by targeting high rate streaming and therefore not being limited to file transfers of known size.

In this paper I present experiments made in high throughput networks of 100 Mbps, 1 Gbps and up to 8 Gbps which continue the line of research approached in [31]. I was able to evaluate the streaming performance of my transfer algorithms and I have also improved the multi-path weighting techniques meant to increase the overall throughput and transmission stability.

The following section presents the fork-join queueing model that is underlying my research. The main features of the weighted transmission algorithms are included in Section 3. Section 4 presents methodology and experimental results and their analysis is done in Section 5. Finally, conclusions and future research directions are presented in Section 6.

2. The fork-join queue model

The architecture of my system, presented in detail in [15] and here in Fig. 1, is based on the model of synchronized queues, also known as fork-join queues [16], specific to parallel and distributed processing in general. Thus, data read by the parent process from a pipeline, which makes a single point of interface with the underlying (Unix-type) OS, is split, in order, among N child processes, via a shared memory segment, containing FIFO networking data buffers of size B (fork phase). The join happens actually at the receiver, with a symmetrical structure, where data is recomposed, in the same order, from its parts, and leaves the system through another writing pipeline.

The performance of my application is determined by its sojourn time, defined as time elapsed between the fork and join moments, or, in other terms, by its global transmission rate R.
Evidently, it depends on individual TCP transmission rates, r_i, with $i \in [1, N]$.

Denote $r_{min} = \min\limits_{i=1,N}\{r_i\}$, $r_{max} = \max\limits_{i=1,N}\{r_i\}$ and let $t_i = \frac{B_i}{r_i}$ with $B_i = B$, $t_{max} = \frac{B}{r_{min}}$ and $t_{min} = \frac{B}{r_{max}}$.
Then given the in order transmission and streaming synchronization constraints:

$$R = \frac{\sum\limits_{i=1}^{N} B_i}{t_{max}} = \frac{NB}{\frac{B}{r_{min}}} = N r_{min} \tag{1}$$

The above equation poses a problem for unequal transmission rates, so I decided to adapt the utilization degree of network buffers depending on measured throughput, according to formula:

$$B_i = B\frac{t_{min}}{t_i} = B\frac{r_i}{r_{max}} \tag{2}$$

Since now:

$$t_{max} = \max_{i=1,N}\{t_i\} = \max_{i=1,N}\{\frac{B_i}{r_i}\} = \frac{B}{r_{max}} \tag{3}$$

equation (1) becomes:

$$R = \frac{\sum\limits_{i=1}^{N} B_i}{t_{max}} = \frac{\frac{B}{r_{max}}\sum\limits_{i=1}^{N} r_i}{\frac{B}{r_{max}}} = \sum_{i=1}^{N} r_i \tag{4}$$

which makes an ideal approximation of R as sum of all transmission rates, provided that the individual measured times t_i are accurate.

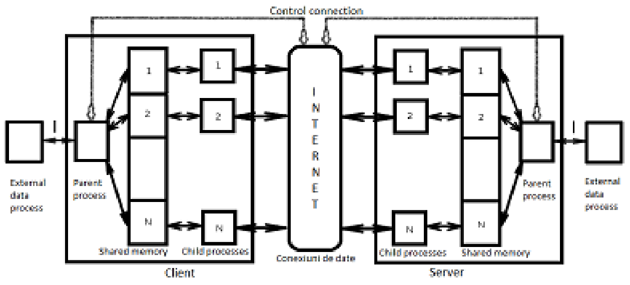


Fig. 1. Architecture of Parallel Streaming System

3. **Streaming algorithms with multi-path support**

The overall structure of my transmission algorithms is formed out of:

- an infinite transmission loop with independent synchronization regions at each individual transmission/reception buffer level.
- an agreed data streaming ordered over N parallel TCP connections.

It is seen as a global fork-join queue which interconnects with speed, through a generalized long and large pipeline, pairs of remote data producer/consumer processes.

To accommodate multi-path support at application level I superposed transmission weighting strategies, implemented, for simplity and efficiency, exclusively at the sender side, and having three main phases:

- measurement of per connection/queue sojourn times, $\{t_i\}$, when all buffers are of equal size, followed by adjustment (weighting) of buffer sizes according to formula (2) above.
- a short waiting period to attain steady-state TCP transmission.
- a feedback loop, determining eventual reweighting when some paths change or/and transmission rates are throttled dynamically, in time.

These strategies are meant to obtain the shortest global sojourn time, setting the overall throughput closer to that computed by equation (4).

From a control systems theory perspective, the weighted transmission algorithm is a proportional only (P-only) type of control [30], as shown in Fig. 2 below. Thus, the sensor computes the average of individual transmission times t_i which is compared to the initial reference value and if the average value has increased with more than 33%, the controller triggers a reevaluation of the degree of use of buffers that in turn sets new individual transmission rates and transfer times, ensuring an optimal global transmission rate given the existing disturbance caused by the connection rate variablility.

To better support a larger class of network bandwidths, I identified, during experiments presented in the next section, a few improvements to my multi-path algorithms, leading to an improved performance and overall stability.

3.1. A more precise sojourn times measurement

I averaged the transmission times over a number of steps varying inversely proportional to N (but no less than 2 steps), instead of a fixed number of steps. I used the mean and standard deviation of the N times as indicators whether the weighted buffer times are better than those initially measured with equally sized buffers: when both mean and standard deviation were bigger, I decided to continue with the equally sized buffers; when only the mean was bigger, an early reweighting was triggered, without entering the feedback loop at all. I also considered an alarm mechanism, able to trigger a clocked reweighting, correcting rare cases when the global transmission rate was wrongly driven by the slowest connection.

These strategies helped with reducing rate measurement errors, due to the characteristically uneven TCP rate over time.

3.2. Stability improvements

I managed to reduce unnecessary reweighting phases, and got better stability and throughput, by considering only significant (more than 33%) increases in the mean sojourn times as a reason for triggering reweighting, discounting decreases.

In the same time, I employed a moving reference to the feedback loop - a rolling average (over last three values) of the mean sojourn times, instead of a fixed initial mean value of them. This approach makes the control system a proportional and integral one (PI), having the error function summed over time. It follows that the offset specific to P-only controllers gets eliminated when operating levels change.

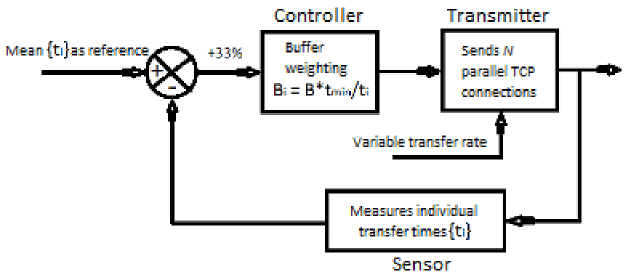

Controller
Transmitter
Mean {ti} as reference
+
-
+33%
Buffer weighting $B_i = B^* t_{min}/t_i$
Sends N parallel TCP connections
Variable transfer rate
Measures individual transfer times {ti}
Sensor

FIG. 2. P-only Control of Weighted Transmission

4. Methodology and experimental results

To evaluate the performance of the new transfer algorithms I created an application (bbftpPRO [3]) that implements the fast data streaming model (using pipes forked/joined over parallel TCP connections), with multi-path support and without it - used as a term of comparison. Integrating both algorithms in the same application has allowed accurate configuration of networking parameters of the TCP protocol, in the same way for each of the experimented transfer modes. In addition, instrumentation was also common, which gave confidence in the results obtained measuring the transfer rate for each of the cases considered.

I compared the average throughput of the weighted streaming algorithm with its unweighted form. I used a special testbed made out of a number of independent channels with different transfer rates, overlapping the underlying physical network. I configured the network topology using the traffic control program *tc*. I set three channels in the 100 Mbps network, of 10, 20, and 30 Mbps each; three channels of 100, 200, and 300 Mbps maximum throughput in the 1 Gbps network; and similarly, another three channels of 1, 2, and 3 Gbps maximum rate, in an IPoIB, 8 Gbps network.

4.1. Static mode

My static tests were of type memory-to-memory and lasted for 60 seconds each, so to achieve TCP stability. I varied the number of concurrent TCP connections, their distribution per channel and the number of channels used in the same time. I allocated 1 connection per channel in the 100 Mbps and 1 Gbps network, in a round-robin mode, until all connections got alloted. Same allocation strategy was used in the 8 Gbps network, except that the number of connections per channel was 2, at each step, in order to provide enough TCP connections to cover for the

available bandwidth. For example, out of a total of 3 connections (in the 1 Gbps network), 2 went to the 200 Mbps channel and 1 to the 100 Mbps one; out of 8 connections (in the 8 Gbps network), 4 went to the 3 Gbps channel, 2 to the 2 Gbps channel, and another 2 to the 1 Gbps one, etc.
Figure 3 presents the average throughput over two channels (20 + 10 Mbps) and 2 to 8 parallel TCP connections; and Fig. 4 the case of three channels (30 + 20 + 10 Mbps) and 3 to 9 parallel TCP connections, in the same 100 Mbps network.
Figure 5 presents the average throughput for two channels (200 + 100 Mbps) and 2 to 8 parallel TCP connections; and Fig. 6 the case of three channels (300 + 200 + 100 Mbps) and 3 to 9 parallel TCP connections, in the same 1 Gbps network.
Finally, Fig. 7 presents the average throughput for two channels (2 + 1 Gbps) and 4 to 24 parallel TCP connections; and Fig. 8 the case of three channels (3 + 2 + 1 Gbps) and 6 to 24 parallel TCP connections, in the same 8 Gbps network.

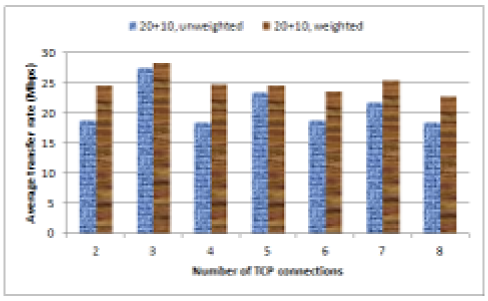


Fig. 3. Two channels, 20+10 Mbps

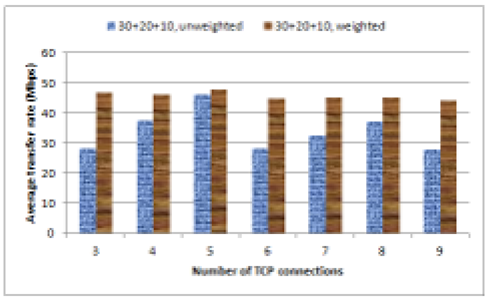


Fig. 4. Three channels, 30+20+10 Mbps

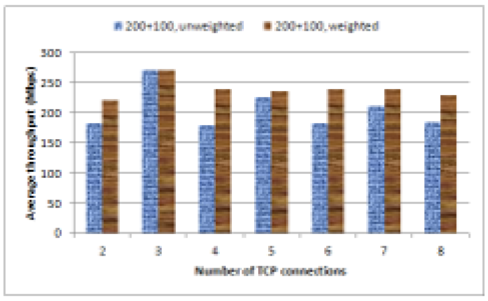


Fig. 5. Two channels, 200+100 Mbps

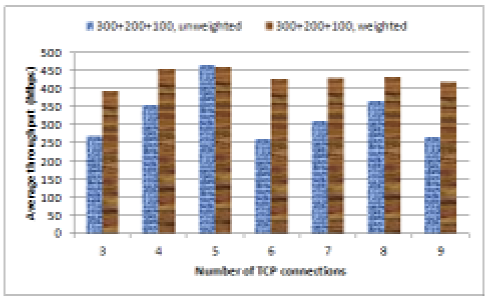


Fig. 6. Three channels, 300+200+100 Mbps

4.2. Dynamic mode

I was also interested to evaluate the dynamic response of my streaming algorithms when the maximum throughput of the participant connections fluctuates. Thus, I reconfigured alternatively, every 20 seconds, the maximum rate of individual connections, by associating them with channels having different bandwidth, in a continous memory-to-memory transfer loop.

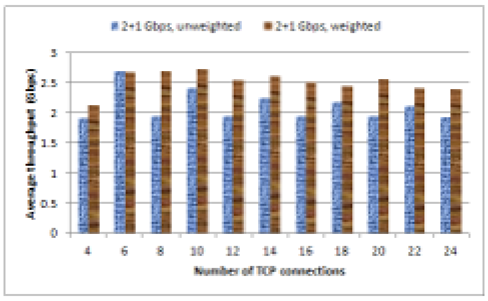


FIG. 7. Two channels, 2+1 Gbps

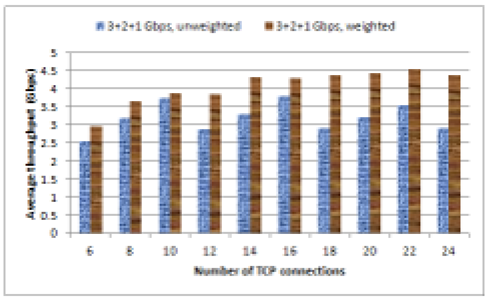


FIG. 8. Three channels, 3+2+1 Gbps

I captured networking snapshots of the *System Monitor* program, for experiments in the 100 Mbps and 1 Gbps network. Figure 9 shows 4 unweighted parallel connections, 2 in a 10 Mbps channel, and the other 2 in a 20 and then 30 Mbps channel, alternatively. Snapshot in Fig. 10 illustrates the weighted transmission, in the same context.
Figure 11 shows 8 unweighted parallel connections, 4 in a 100 Mbps channel, and the other 4 in a 200 and then 300 Mbps channel, alternatively. Snapshot in Fig. 12 illustrates the weighted transmission, in the same context.
In the 8 Gbps network I instrumented my dynamic experiments by intermediary piping the transferred data through the *pmr* [24] program and collecting its instantaneous rate reports once every second. Figure 13 shows both the weighted and unweighted transmission of 12 parallel connections, 6 in a 1 Gbps channel, and the other 6 in a 2 and then 3 Gbps channel, alternatively.

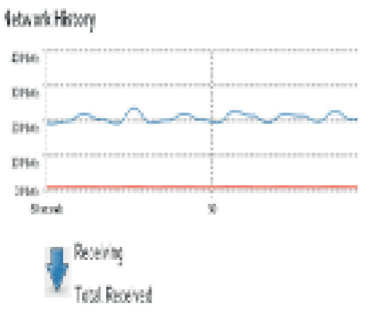


FIG. 9. Unweighted

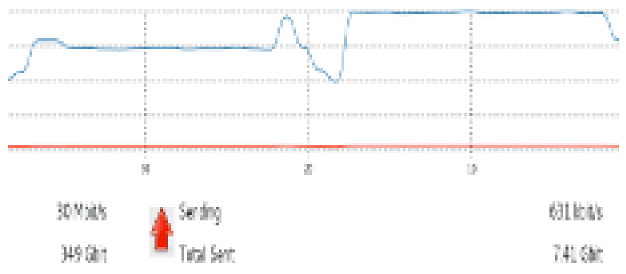
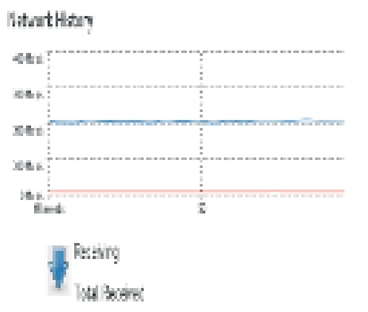


FIG. 10. Weighted, 10+20|30 Mbps

5. Analysis of Experimental Results

The static and especially the dynamic mode performance evaluation shows that the weighted transmission algorithm uses to a greater extent the available overall bandwidth and has a superior responsiveness to dynamic bandwidth changes than the unweighted one.



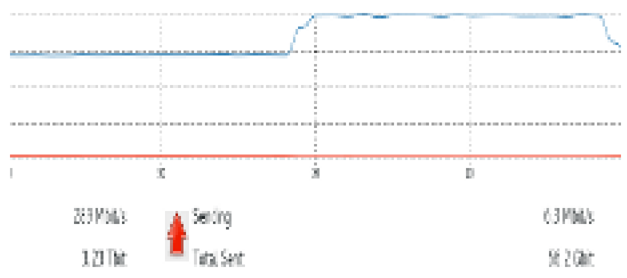


FIG. 11. Unweighted

FIG. 12. Weighted, 100+200|300 Mbps

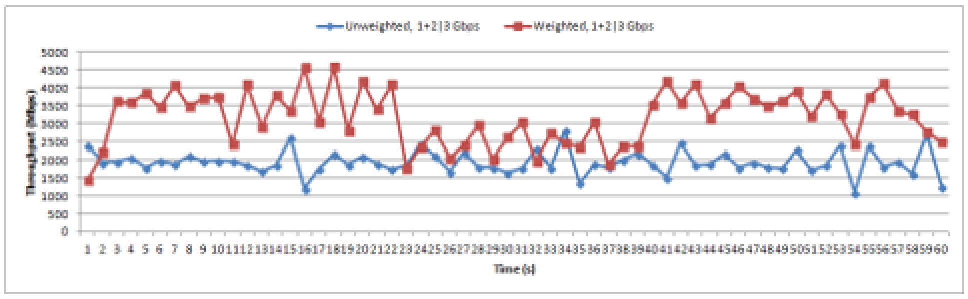


FIG. 13. Un/weighted, 1+2|3 Gbps

5.1. Static Analysis

Thus, in the static 100 Mbps case: data presented in Fig. 3 shows, for two channels (of 20 and 10 Mbps), that the average throughput is 18.1% higher, with a peak of +33.8% for 4 concurrent TCP connections, when using the weighted algorithm. The utilization degree of the available bandwidth (equal to 30 Mbps) averages 82.9% in the weighted mode, while only 70.1% in the unweighted mode.
Data shown in Fig. 4, for three channels (of 30, 20, and 10 Mbps), computes an average throughput 34.5% higher, peaking at +66.9% for 3 parallel connections, when weighting is in use. The utilization degree of the available bandwidth (equal to 60 Mbps) averages 76.3% in the weighted mode, while only 56.6% in the unweighted mode.

In the static 1 Gbps network case: data presented in Fig. 5 shows, for two channels (of 200 and 100 Mbps), that the average throughput is 16.6% higher, with a peak of +32.7% for 4 concurrent connections, when using the weighted algorithm. The utilization degree of the available bandwidth (equal to 300 Mbps) averages 79.9% in the weighted mode, while only 68.5% in the unweighted mode.
Data shown in Fig. 6, for three channels (of 300, 200, and 100 Mbps), computes an average throughput 31.8% higher, peaking at +64.5% for 6 parallel connections, when weighting is used. The utilization degree of the available bandwidth (equal to 600 Mbps) averages 71.8% in the weighted mode, while only 54.5% in the unweighted mode.

In the static 9 Gbps network case: data presented in Fig. 7 shows, for two channels (of 2 and 1 Gbps), that the average throughput is 19.4% higher, with a peak of +39.1% for 8 connections, when using the weighted algorithm. The utilization degree of the available bandwidth (equal to 3 Gbps) averages 84% in the weighted mode, while only 70.3% in the unweighted mode.
Data shown in Fig. 8, for three channels (of 3, 2, and 1 Gbps), computes an average throughput 27.2% higher, peaking at +50.5% for 18 connections, when weighting is in use. The utilization degree of the available bandwidth (equal to 6 Gbps) averages 67.9% in the weighted mode, while only 53.4% in the unweighted mode.

It is noteworthy, from a qualitative standpoint, the clear distinction between the rates achieved by the weighted algorithm as compared to the unweighted one. While the former maintains approximately the same higher throughput regardless of TCP connections distribution to channels, the latter is obviously affected by it, hence the sawtooth shape of its throughput line.

5.2. Dynamic Analysis

In the absence of weighting, throughput stays unmodified, at around 20 Mbps as shown in Fig. 9 for the 100 Mbps network experiments; at about 200 Mbps as shown in Fig. 11 for the 1 Gbps network experiments; and at about 1.95 Gbps as shown in Fig. 13 for the 8 Gbps network. These values correspond to the rate of the slowest connection, multiplied by the number of connections, as predicted by theory.
In contrast, the weighted algorithm is sensible to the ±50% bandwidth variation happening every 20 seconds. Figure 10 shows, for the 100 Mbps network case, two average throughput levels, one of about 28 Mbps and the other of about 38 Mbps, both approximating the bandwidth sum of the channels in use. Figure 12 shows, for the 1 Gbps network case, two average throughput levels, one just below 300 Mbps and the other just below 400 Mbps, both approximating the bandwidth sum of the used channels, as expectedly. In Fig. 13, for the 8 Gbps network case, the two average throughput levels are about 2.62 Gbps and 3.48 Gbps, both representing about 87% of the channels' bandwidth sum, in their respective states.

Finally, these results also demonstrate that the design of my multi-path algorithms, described in Section 3 above, has indeed contributed to a better overall transmission stability.

6. Conclusions

This paper has presented the features and behavior of improved data streaming algorithms over an extended range of networks capable of high bandwidth of 100 Mbps, 1 Gbps and up to 8 Gbps. I managed to preserve the advantages of the pipe programming model by extending it between fast data reading/writing processes distributed over the Internet.

The experiments and results described and analyzed in the previous sections demonstrate that my application implements improved algorithms that handle the multi-path transmission case efficiently, in both static and dynamic regimes, yielding better throughput and stability.

Based on these results, I am confident that the proposed model and the new fast transfer algorithms meet the modern requirements of today's research domains interested in transmitting massive bulk data elegantly, efficiently, effectively and at parallel speeds.

6.1. Future Research Directions

I consider further improving the stability and efficiency of transfer algorithms, and also automating transmission parameter choosing, like optimal number of parallel connections, buffer sizes, or transparent choosing of participating paths at local IP level. I appreciate that the expected diversification and continuous development of networks in the future would certainly benefit from further research extending my current work.

REFERENCES

[1] *A. Ford, C. Raiciu, M. Handley and O. Bonaventure*, TCP Extensions for Multipath Operation with Multiple Addresses draft-ietf-mptcp-multiaddressed-09, IETF draft, Jun. 2012.

[2] *ATLAS Experiment*, Available: `http://www.atlas.ch/`

[3] *bbftpPRO*, Available: `http://bbftppro.myftp.org/`

[4] *CASTOR*, Available: `http://castor.web.cern.ch/`

[5] *CERN*, Available: `http://public.web.cern.ch/public/`

[6] *LCG Grid Deployment Team*, rfcat, Available: `http://linux.die.net/man/1/rfcat`

[7] *D. Schrager*, rftar, Available: `http://castorold.web.cern.ch/castorold/DIST/CERN/savannah/CONTRIB.pkg/Dan.Schrager@weizmann.ac.il/`

[8] *C. Jin, D. Wei and S. Low*, FAST TCP: Motivation, Architecture, Algorithms, Performance, Proceedings of IEEE Infocom, Hong Kong, Mar. 2004.

[9] *D. Katabi*, Decoupling Congestion Control and Bandwidth Allocation Policy with Application to High Bandwidth-Delay Product Networks, PhD thesis, Massachusetts Institute of Technology, Mar. 2003.

[10] *A. Jungmaier, M. Schopp and M. Tuexen*, Performance Evaluation of the Stream Control Transmission Protocol, in 3rd International Conference on ATM (ICATM2000), Heidelberg, Germany, Jun. 2000.

[11] *E. Kohler, M. Handley and S. Floyd*, Designing DCCP: congestion control without reliability, in Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications SIGCOMM '06, New York, Oct. 2006.

[12] *S. Floyd and V. Jacobson*, Random Early Detection gateways for Congestion Avoidance, in IEEE/ACM Transactions on Networking, Aug. 1993.

[13] *W. Feng, D. Kandlur, D. Saha and K. Shin*, BLUE: A New Class of Active Queue Management Algorithms, University of Michigan, CSE-TR-387-99, Apr. 1999.

[14] *W.-c. Feng, D. D. Kandlur, D. Saha i K. G. Shin*, Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness, in INFOCOM, Apr. 2001.

[15] *D. Schrager and F. Radulescu*, Efficient Algorithms for Fast Data Transfers Using Long and Large Pipes in WAN Networks, 19th International Conference on Control Systems and Computer Science (CSCS-19), May 2013

[16] *L. Flatto and S. Hahn*, Two parallel queues created by arrivals with two demands, SIAM J. Appl. Math. Vol 44, No 5, 1041-1053, 1984.

[17] *E. He, J. Leigh, O. Yu and T. DeFanti*, Reliable Blast UDP : Predictable High Performance Bulk Data Transfer, Proceedings of IEEE International Conference on Cluster Computing, 2002.

[18] *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, Proceedings of the IEEE/ACM SC2000 Conference, 2000.

[19] *J. Crowcroft and P. Oechslin*, Differentiated End-to-End Internet Services Using a Weighted Proportional Fair Sharing TCP, ACM SIGCOMM Computer Communication Review, Vol. 28 Issue 3, Pag. 53 - 69, Jul. 1998.

[20] *J. Postel*, Transmission Control Protocol, RFC 793, Sep. 1981.

[21] *L. Brakmo and L. Peterson*, TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE Journal on Selected Areas in Communications, Vol. 13, No. 8, Oct. 1995.

[22] *M. Allman, S. Ostermann and H. Kruse*, Data Transfer Efficiency over Satellite Circuits Using A Multi-Socket Extension to the File Transfer Protocol (FTP), Proceedings of the ACTS Results Conference, NASA Lewis Research, 1995.

[23] *M. Mathis, J. Mahdavi, S. Floyd and A. Romanow*, TCP Selective Acknowledgment Options, RFC 2018, Oct. 1996.

[24] *pmr*, URL: `http://zakalwe.fi/~shd/foss/pmr/`

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

[26] *VIRGO Detector*, URL: `https://wwwcascina.virgo.infn.it/`

[27] *W. Allcock, J Bester, J Bresnahan, A Chervenak, L Liming and S Tuecke*, GridFTP: Protocol Extensions to FTP for the Grid, Global Grid Forum Recommendation Document GFD.20, 2005.

[28] *J. Postel*, User Datagram Protocol. RFC 768, Aug. 1980

[29] *Y. Gu and R. Grossman*, SABUL: A Transport Protocol for Grid Computing, Journal of Grid Computing, Volume 1, Issue 4, pp. 377-386, 2003.

[30] *K. J. Astrom and T. Hagglund*, PID Controllers: Theory, Design, and Tuning, Instrument Society of America, 1995.

[31] *D. Schrager*, New Solutions for Fast Data Transfers in HBDP Networks, UPB Scientific Bulletin, Series C, Electrical Engineering and Computer Science, ISSN-L 2286-3540, 2014.