Proceedings of the Asia-Pacific Advanced Network 2010 v. 30, p. 45-52.
http://dx.doi.org/10.7125/APAN.30.7 ISSN 2227-3026

Reliable Multipath Transfer Scheduling Algorithm Research and Prototype Implementation

Wentao Zhang [1,*], Qian Wu [2], Wang Yang [3] and Hewu Li [2]

1 Department of Electronic Engineering, Tsinghua University, Beijing, PRC
2 Network Research Center, Tsinghua University, Beijing, PRC
3 Department of Computer Science, Tsinghua University, Beijing, PRC

E-Mails: zwt06@mails.tsinghua.edu.cn; wuqian@cernet.edu.cn; yang-w04@mails.tsinghua.edu.cn; lihewu@cernet.edu.cn

* Tel.: +86-10-6277-2375; Fax: +86-10-6277-2375

Abstract: End-to-end data transfer has long been a fundamental service of Internet. With the trend of multi-interface communication terminals and redundant paths between them both in cellular networks and in the Internet, efficient protocols of multipath transfer are needed to gain the benefits of multi-interfaces. In this paper, we propose SAR-RMTP, a Self-adapted Rescheduling Reliable Multipath Transfer Protocol for end-to-end file transfer, which can adaptively reschedule data to different paths according to the current average bandwidth to achieve nearly the same transferring finish time between different paths, and thus results in effective use of overall bandwidth. We implement the prototype of SAR-RMTP in Linux and compare its performance with existing scheduling algorithms in experimental environment. The results show that SAR-RMTP can notably decrease the difference of transfer finish time of different paths and thus shorten the overall file transfer time and increase the overall bandwidth. The results also show that compared with other scheduling algorithms SAR-RMTP can achieve much better performance when bandwidth changes more dramatically.

Keywords: multipath transfer; scheduling algorithm; reschedule.

1. Introduction

Multipath transfer applies in both Wired Internet and Heterogeneous Wireless Networks. In wired networks, Multi-homing and multipath routing have been widely adopted for stub networks. With the rapid advance of wireless technology, most off-the-shelf wireless mobile devices are equipped with multi-radio capability (GPRS, Wi-Fi, 3G, Bluetooth, etc.). The devices are assigned with several IP addresses to enable multipath.

It has been acknowledged that multipath can be utilized to aggregate bandwidth, improve reliability, enhance security and boost quality of service [1]. Since different paths may have different characteristics in terms of bandwidth, loss rate and delay, it is challenging to effectively aggregate bandwidth of different paths.

A main branch of multipath transfer research, including CMT (Concurrent Multipath Transfer) [2,3], Multipath TCP [4,5] proposed by IETF MPTCP working group, focuses on the solution at transport layer, which aims to provide the same service as TCP, i.e. in-order, reliable, and byte-oriented delivery. That means application layer data are all treated as a streaming regardless of its real type, and all the data are delivered in the order of how it comes. However, for file transfer, data transferred on different paths don't have order relevance, which means that disjoint portions of file could be simultaneously sent to the peer on multiple paths

regardless of their order. Consequently, schemes of transport layer multipath transfer protocol like Multipath TCP are usually unnecessary complex for applications of file transfer.

In this paper, we propose an efficient Self-adapted Rescheduling Reliable Multipath Transfer Protocol (SAR-RMTP) which acts as a middleware between transport layer and application layer, and aims at provide service to applications that are based on end-to-end file transfer. Meanwhile, SAR-RMTP is lightweight and efficient for using standard TCP to ensure reliable transfer on each path.

The most important mechanism of SAR-RMTP is how to dynamically and reasonably schedule data among multiple paths according to the behavior of path bandwidth, and make different paths finish transferring at nearly the same time, for the goal of minimizing the overall file transfer time, i.e. maximizing the overall bandwidth of the selected paths. In this paper, we propose a self-adapted rescheduling algorithm whose main thought is to reschedule assigned data from slower paths to faster ones according to average path bandwidth from last assignment to now, which could shorten the idle time that faster paths spend waiting for slower ones to finish data transfer.

We implement the prototype of SAR-RMTP and compare its performance with existing scheduling algorithms in experimental environment. The results show that SAR-RMTP could notably reduce the difference of transfer finish time of different paths and decrease the overall file transfer time, moreover, SAR-RMTP could have more remarkable efficacy improvement than existing scheduling algorithms when path bandwidth suffers severe fluctuation.

The remainder of this paper is organized as follows. Related works of multipath scheduling algorithm are described in section 2. The SAR-RMTP architecture, data scheduling algorithm and protocol prototype implementation are presented in section 3. The experiments and analyses are described in Section 4. Finally, section 5 concludes with future enhancements.

2. Related Works of Multipath Scheduling Algorithm

As mentioned above, the opportunity to obtain higher bandwidth between endpoints is provided by multiple interfaces of communication terminals and multiple reachable paths between them, thus we could establish multiple TCP connections via these paths and transfer data simultaneously. In a similar way, parallel file transfer methods in Data Grids transfer a file to a client from different replica servers, each of which corresponds to a TCP connection and has the complete duplicate of the requested file. The difference is that the multiple TCP connections a client used to receiving file portions from different peers might be established on the same interface. Nonetheless, the data scheduling algorithms on multiple connections of SAR-RMTP and co-allocation algorithms of parallel transfer in Data Grids have similarity and are comparable. A few co-allocation algorithms of parallel file transfer in Data Grids are described below.

A. Static Allocation Algorithm

The Brute-Force Co-allocation scheme in [6] divides file sizes equally among available flows; it does not address bandwidth differences among various client server links. Actually, this primal method could gain more efficiency by a little modification. The modified algorithm is called Static allocation Algorithm, which divides file sizes among paths proportional to initial path bandwidth and doesn't do any adjustment while file transferring. Each path finishes when the assigned portion of file is entirely transferred.

B. Adjustable Single Block Load Balancing Algorithm

The Conservative Load Balancing scheme in [6] divides requested datasets into k disjoint blocks of equal size. Available servers are allocated single blocks to deliver in parallel. Servers work in sequential order until all requested files are downloaded. Loadings on the co-allocated flows are automatically adjusted because faster servers deliver file portions more quickly. Efficiency of this method depends on the size of disjoint blocks. It could make load balancing very approximate to actual bandwidth fluctuation by taking small enough

block size, which, however, would considerably increase the amount of additional information such as block header and the cost of synchronization among paths because the target file would be accessed more frequently by different paths. To make the block size adjustable and reasonable, we modified the Conservative Load Balancing scheme referring to the Recursively-Adjusting Co-Allocation scheme in [7,8]. The modified scheme is called Adjustable Single Block Load Balancing algorithm, which is described as follows.

First let *BlockSize* equals to *FileSize*/*j*, where *FileSize* is the size of file to be transferred and *j* is a certain constant, e.g. 20. Initially, each path was assigned a single block to transfer. After path *i* ($1 \leqq i \leqq n$, *n* is the number of flows) finished transferring the assigned size of data, the scheduling algorithm would calculate a new block size for this path to transfer as follows, where the new block size might equals to the previous one:

$$ParticalSize = \left(\sum_{k=1}^{n} LeftBlkSize_k + LeftFileSize\right) \cdot BW_i \Big/ \sum_{k=1}^{n} BW_k - LeftBlkSize_k \quad (1)$$

where $LeftBlkSize_k$ is the currently unfinished data size of path k, *LeftFileSize* is size of the entire unassigned file, $BW_{i/k}$ is the average bandwidth of path *i/k* during its last block.

- if *ParticalSize* $\leqq$ *LeftFileSize*, new block size for path *i* equals to min[*ParticalSize*, *BlockSize*];
- if *ParticalSize* > *LeftFileSize*, new block size for path *i* equals to min[*LeftFileSize*, *BlockSize*].

For the modified algorithm, if calculated *ParticalSize* for a certain path is less than both *LeftFileSize* and *BlockSize*, the path will take the smallest *ParticalSize* as block size which is more reasonble to avoid being awaited by other paths. However, when the calculated particle size is larger than *LeftFileSize*, and bandwidth of this path would not decline in the future, this path would be one of the earlier finished paths and would be idle awaiting the last finished path.

Performances of algorithm B should be of notable promotion compared with algorithm A. However, all the algorithms mentioned above including that in [7,8] have a common shortage, data that has been assigned to certain paths could not be rescheduled to other paths in the next assignment or adjustment. If after an assignment the bandwidth of one path declines to a certain value that even if all the unallocated file portions such as *LeftFileSize* in algorithm B was assigned to other paths, the declined path would still last for an amount of time transferring data after all the other paths has finished, the overall performance of file transfer would degrade.

In the experiments section, we will compare performance of scheduling algorithm of SAR-RMTP with algorithm A and B.

3. Self-adapted Rescheduling Reliable Multipath Transfer Protocol (SAR-RMTP)

3.1. Protocol Architecture

The architecture of SAR-RMTP is illustrated in figurer 1. At the sender side data of a certain file are simultaneously transferred on multiple paths, meanwhile, the receiver simultaneously receives them from corresponding paths. To ensure reliable transfer, each path uses standard TCP to transfer its data between the two end points.

The most important part of the architecture is the Data Scheduling Model, which determines the outgoing path of each file portion and reschedules data among paths to adapt to path bandwidth variation and fluctuation.

In order to know the initial values of path bandwidth, there is an initial path bandwidth estimation phase before the scheduling algorithm takes over data scheduling. The protocol first enters this phase after file transferring starts, during which each path is assigned single blocks of certain size after it has finished its last block. The initial path bandwidth estimation phase ends when a threshold of assigned data size is reached, and then assignment and adjustment of data among paths is taken over by the data scheduling algorithm. We use $\alpha \cdot FileSize$ ($0 < \alpha < 1$, e.g. 0.1) as the threshold, where *FileSize* is size of the requested file and use average

bandwidth of each path, which is denoted as $BWS_i \ (i = 1,...,n)$, as initial values of path bandwidth for data scheduling algorithm, where *n* is the number of paths used to transfer data.

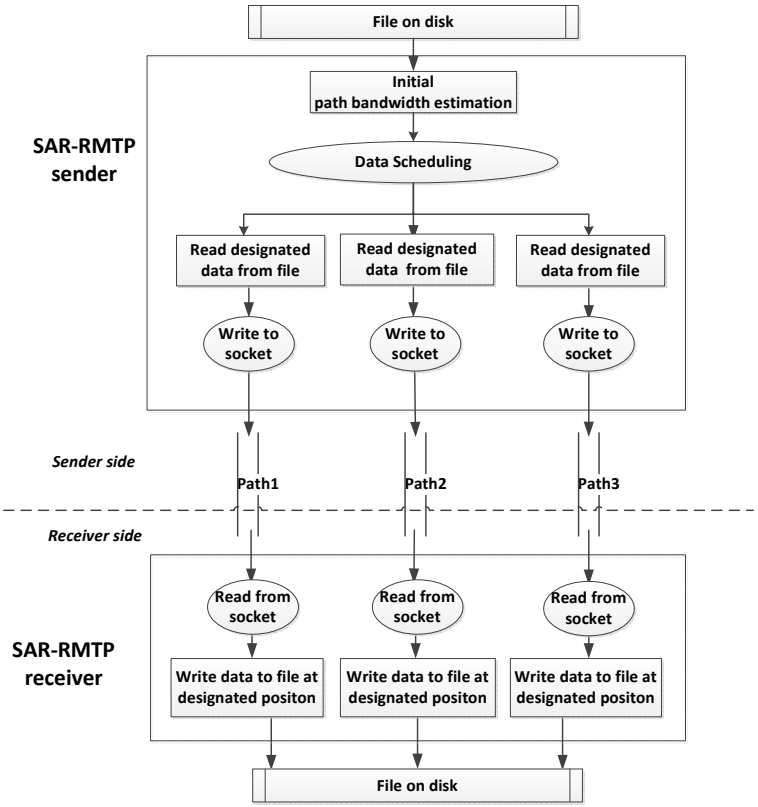

Figure 1. SAR-RMTP architecture

3.2. Data Scheduling Algorithm of SAR-RMTP

As mentioned above, when the threshold of assigned data is reached in initial path bandwidth estimation phase, the data scheduling algorithm is triggered to assign data to all paths for the first time and it will do this job till overall file transfer is finished. We call the data set one path gets from the scheduling algorithm a task. The brief process of data scheduling is that before or when one path finishes its task, the scheduling algorithm is triggered to calculate new task size for each path according to path average bandwidth from last task was assigned to now, and reschedules data among paths if the current unfinished size of a path does not equal to the new task size calculated for it. We call an execution of the scheduling algorithm an adjustment; the continuous adjustments will stop when the expected finish time of all paths reaches a certain threshold.

The exact scheduling algorithm is as follows:

1) The first task of each path after the threshold of assigned data is reached in initial path bandwidth phase is calculated as below:

$$TSK_i(0) = \frac{BWS_i}{\sum_{i=1}^{n} BWS_i} \cdot (1-\alpha) \cdot FileSize, i = 1,...,n \tag{2}$$

where $TSK_i(0)$ is the first task of path *i* in adjust phase. Record the current time as *tstart(0)*.

2) When one certain path, e.g. path i, first finished transferring $\beta \cdot TSK_i(k)$ size data after the k_{th} $(k \geqq 0)$ adjustment, where $TSK_i(k)$ is the task size assigned to path *i* in the k_{th} adjustment and $0.5 \leq \beta \leq 1$, the

$(k+1)_{th}$ adjustment is triggered. Record current time as *tend(k)*. Some parameters are first calculated as follows:

$$BW_i(k) = TRANS_i(k) / (tend(k) - tstart(k)), i = 1, ..., n \quad (3)$$

where *tend(k)* is the time the k_{th} adjustment finished, $BW_i(k)$ is the average path bandwidth of path *i* from *tstart(k)* to *tend(k)*, $TRANS_i(k)$ is transferred data size of path *i* from *tstart(k) to tend(k)*.

$$LTSK_i(k) = TSK_i(k) - TRANS_i(k), i = 1, ..., n \quad (4)$$

$$LT_i(k) = LTSK_i(k) / BW_i(k), i = 1, ..., n \quad (5)$$

where $LTSK_i(k)$ is the unfinished size of path *i* at *tend(k)*, $LT_i(k)$ is the expected remaining time of path *i*. Then, the condition below is judged:

$$\max_{1 \le i \le n}[LT_i(k)] - \min_{1 \le i \le n}[LT_i(k)] \le \min[\delta \cdot TransTimeForNow, LeftTimeThreshold] \quad (6)$$

where *TransTimeForNow* is time that the overall file transfer process has lasted for since it began with start phase, $0 < \delta \le 5\%$, e.g. 1%, and *LeftTimeThreshold* is a threshold that could be assigned manually, such as 2000 microsecond.

- If (6) is satisfied, there would be no adjustments from now on till the overall file transfer is finished.
- If (6) is unsatisfied, it means that the $(k+1)_{th}$ adjustment should be executed and the new task size of each path is calculated as follows,

$$TSK_i(k+1) = \left(\sum_{i=1}^{n} LTSK_i(k)\right) \cdot BW_i(k) / \sum_{i=1}^{n} BW_i(k), i = 1, ..., n \quad (7)$$

where $TSK_i(k+1)$ denotes the task size assigned to path *i* in the $(k+1)_{th}$ adjustment. It doesn't mean additional $TSK_i(k+1)$ size data should be assigned to path *i*. The three possible cases of data reschedule are described as follows:

- If $TSK_i(k+1) == LTSK_i(k)$, it means that the unfinished data of path *i* at *tend(k)* should not be touched in the $(k+1)_{th}$ adjustment;
- If $TSK_i(k+1) < LTSK_i(k)$, it means that $LTSK_i(k) - TSK_i(k+1)$ size of the unfinished data of path *i* at *tend(k)* should be rescheduled to other paths in the $(k+1)_{th}$ adjustment.
- If $TSK_i(k+1) > LTSK_i(k)$, it means that $TSK_i(k+1) - LTSK_i(k)$ size of unfinished data form other paths at *tend(k)* should be rescheduled to path *i* in the $(k+1)_{th}$ adjustment.

After data were rescheduled among paths according to $TSK_i(k+1)$ and $LTSK_i(k)$, record current time as *tstart(k+1)*, go to step 2).

The data scheduling algorithm is as the two steps above. Relying on reschedule operations in adjustment, faster paths could get new data from unfinished data of other paths when their tasks is about to finished, consequently they would not be idle awaiting slower paths to finish. This is supposed to be a method of high efficiency and the exhaustive method to decrease transfer time differences among different paths. We would validate the performance of it in the experimental section.

3.3. SAR-RMTP Prototype Implementation

We have implemented the prototype of SAR-RMTP in Linux with C++, which runs on a sender and a receiver while transferring a file via multipath between them.

At the sender side, simultaneously data transfer on multiple TCP connections is implemented by multiple threads, each of which corresponds to a connection. Each thread reads data at the designated position of file

and then writes what it has read to socket. When it's time to calculate next task for the corresponding path, the thread will execute the data scheduling algorithm described in section 3.2, and then continuing data transfer on this path. Time spent on calculating of scheduling algorithm is short enough to be negligible, thus it wouldn't block data transferring. Because the task structure of paths would be touched both when the corresponding path is transferring data and when other paths are executing the scheduling algorithm, we use a mutex for each path task structure to make protection and a global mutex for the scheduling algorithm execution to make sure when multiple paths want to execute the scheduling algorithm, they will execute it without overlap. Furthermore, to ensure accuracy, unit of time is microsecond.

At the receiver side, the multiple thread architecture is taken as well. An independent thread is created for each path to receive data from the corresponding connection and then data received is directly written to file. As every block received includes a header that contains data position and data length, multiple threads could simultaneously write data to the same file without overlap. After all the paths finish receiving data, the receiving file is complete as well.

4. Experimental Results

To evaluate the performance of SAR-RMTP, we setup an experimental environment illustrated in figure 2, where we could expediently set bandwidth of each path with the switch between sender and receiver. There are two bandwidth fluctuation cases (case 1 and case 2) in the experiments. In each case, we analyze the performance of algorithm A, B in section 2 and SAR-RMTP by comparing the overall transfer time and the idle time which the fastest path spends waiting for the slowest path to finish at the sender side.

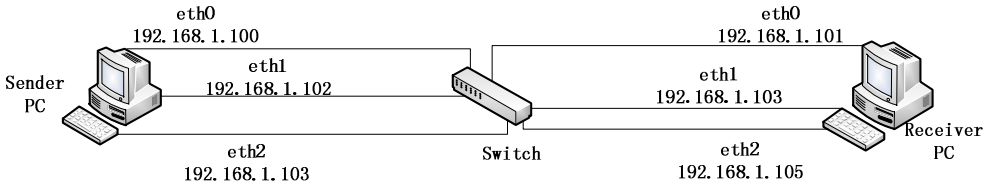

Figure 2. Experimental environment

We respectively set bandwidth of the three paths between sender and receiver to 8Mbps, 16Mbps and 24 Mbps by setting rate limit to corresponding switch ports. In case 1, we change the bandwidth of path 3 from 24Mbps to 8Mbps at time (0.9**FileSize*/48Mbps), i.e. when 90% of the file has been transferred in the set rate condition. In case 2, we change the bandwidth of path 3 from 24Mbps to 2Mbps at the same time as case 1 to make bigger bandwidth fluctuation.

In the two cases, we measured the performance of each algorithm by transferring files of size 101.5MB, 304.3MB, 507.1MB, 710.1MB and 913.0MB.

For case 1, figure 3 shows the overall file transfer time of each algorithm in histogram and the saved time ratio of SAR-RMTP compared with algorithm A and B in line chart, which is defined as the overall transfer time difference of SAR-RMTP and algorithm A(or B) divided by the overall time of A(or B) for the same file size. We can see that SAR-RMTP uses the least time to finish file transferring and averagely saves 11.4% and 8.9% of overall transfer time compared with algorithm A and B.

Figure 4 shows the idle time of each algorithm in case 1 in histogram. The idle time is defined as the maximal finish time difference between three paths. The idle ratio of each algorithm defined as idle time divided by overall transfer time which is shown in line chart in figure 4. We can see that SAR-RMTP could make different paths finish transferring at nearly the same time, while algorithm A and B are respectively 16.2% and 12.4% idle during the overall file transferring.

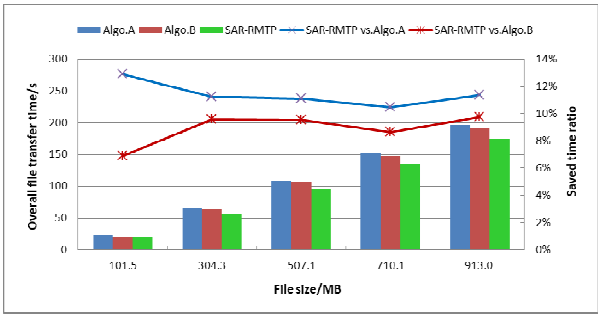

Figure 3. Overall transfer time in case 1

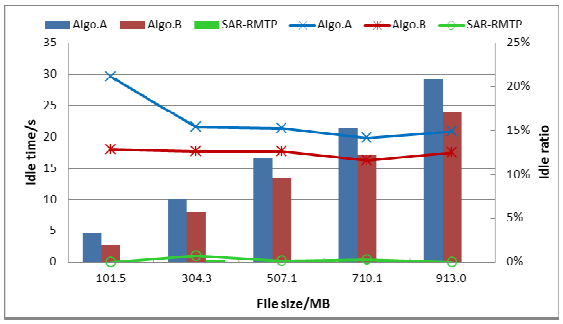

Figure 4. Idle time in case 1

Performance of each algorithm in case 2 is shown in figure 5 and figure 6 in the same way as case 1. We can see that under bigger bandwidth fluctuation circumstance, SAR-RMTP still performs the best, with finish time of three paths nearly the same and averagely 45.6 % and 41.4% of transfer time saved compared with algorithm A and B.

Comparing figure 6 with figure 4, we can see that when path bandwidth fluctuates more dramatically, idle ratio of algorithm A and B increases greatly; the first finished path are idle for about half of the overall transfer time ! However, SAR-RMTP could keep idle ratio always near zero, thus keep stable and high performance.

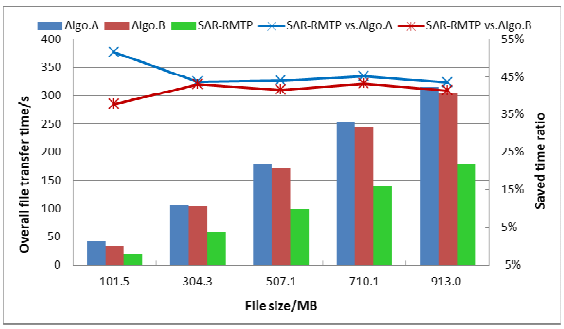

Figure 5. Overall transfer time in case 2

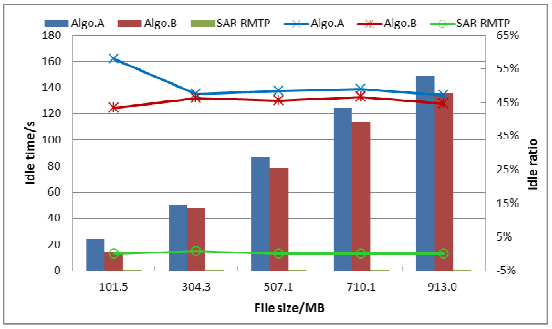

Figure 6. Idle time in case 2

5. Conclusions

With scheduling algorithm that could reschedule data among different paths according to path bandwidth, SAR-RMTP performs much better than existing scheduling algorithms. It could keep finish time of different paths nearly the same, thus shortens the overall transfer time and aggregates bandwidths of multiple paths. It also keeps high performance and efficiency under various bandwidth fluctuation circumstances, which shows higher adaptability. As a middleware between application layer and transport layer, SAR-RMTP could provide an efficient solution to end-to-end multipath file transfer. We will make efforts to put it into practice on the Internet in the future.

Acknowledgements

This work is supported by High-Tech Research and Development Program of China under Grant No. 2008AA01Z212, National Key Technology R&D Program under Grant No. 2008BAH37B09, China Next Generation Internet (CNGI) Project under Grant No. 2008-105 and CNGI 2008-119, and National Sci-Tech Major Special Item under Grant No. 2008ZX03003005.

References

1. D. Sarkar; P. D. Amer; R. Stewart. Concurrent Multipath Transport, *Computer Communications*, 2007, vol. 30, no. 17, pp.3215–3217.
2. Iyengar, J.R.; Amer, P.D.; Stewart, R. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths, 2006, *IEEE/ACM Transactions on Networking*, Vol. 14(5), pp. 951-964.
3. Gaponova, I. Concurrent Multipath Transfer for Heterogeneous Networks, 2008, *master thesis, Technische Universitat Braunschweig.*
4. http://datatracker.ietf.org/wg/mptcp/
5. http://datatracker.ietf.org/doc/draft-ietf-mptcp-architecture/
6. S. Vazhkudai. Enabling the Co-Allocation of Grid Data Transfers, *Proceedings of Fourth International Workshop on Grid Computing*, 2003, pp. 44-51.
7. C.T. Yang; I.H. Yang; K.C. Li; C.H. Hsu. A Recursive-Adjustment Co-Allocation Scheme in Data Grid Environments, *ICA3PP 2005 Algorithm and Architecture for Parallel Processing*, 2005,vol. 3719, pp. 40-49.
8. Chao-Tung Yang; Yao-Chun Chi; Ming-Feng Yang; Ching-Hsien Hsu. An Anticipative Recursively-Adjusting Mechanism for Redundant Parallel File Transfer in Data Grids, *Computer Systems Architecture Conference*, 2008. ACSAC 2008. 13th Asia-Pacific, pp. 1-8.