File Delivery with Longest Processing Time First Scheduling in P2P Networks

Chun-Hao Wen[1,2], Steve W. Haga[1] and Richard Chun-Hung Lin[1]

[1]*The Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan*
[2]*Department of Information Technology, Meiho University, Taiwan*
jaywen@mail.cse.nsysu.edu.tw, lin@cse.nsysu.edu.tw, stevewhaga@cse.nsysu.edu.tw

Abstract

Peer-to-peer (P2P) File delivery systems of distributed computing are designed with the understanding that any peer can leave the network at any time, often right after completing its download. The conventional approach to P2P scheduling, shortest processing-time first, is not well suited to peer-leaving situations. We therefore propose a new scheduling algorithm that significantly reduces average finish time in such cases. We further note, at the end of the paper, that our method is also suitable for the more general problem of a dynamic network of peers that may leave early or enter late.

Keywords: *P2P, scheduling, data sharing*

1. Introduction

Most P2P schedulers (*e.g.*, [1, 2, 3, 4, 5, 6]) prefer those peers with shortest processing times (*i.e.*, higher upload bandwidths). This is called SPT scheduling [7, 8]. But peers often disconnect once their downloads finish, to save power or bandwidth [9]. If this happens then SPT scheduling performs poorly, with the total bandwidth of upload (BW_k) rapidly dropping off in each time interval. It is instead better to do the opposite, by favoring peers with longest processing times (LPT scheduling [5]). But this also is non-ideal: since fast peers receive data slowly, their contribution BW_k is limited.

Alternatively, we might use SPT scheduling but withhold the fast peers in the network to service others. But that would be unwise, because all peers would finish together, and increase the average finish time, T_A. T_A is an important metric of overall performance, but minimizing it is considered NP-complete [10, 11, 12].

We instead propose a novel scheduler, Reverse Store-and-Forward (RSF), which unicasts data and uses inverse-multiplexing [13] with divide-and-conquer approach. By increasing the time at which the last peer finishes (T_L), RSF substantially decreases the T_A of networks with peer-leaving.

2. System Model

A single server (P_0) with an upload bandwidth (or capacity) of C_0 services n peers (P_j, j= 1, …, n) with capacities C_j . Since the labeling of the peers is arbitrary, we may choose an ordering such that the slowest is P_1 and the fastest is P_n (*i.e.*, $C_1 \le C_2 \le \ldots \le C_n$), such that BW_k is maximized when peers are leaving sequentially from P_1 to P_n. Furthermore, at time τ_0, each of the n peers requests a data file (of size $|F|$) from P_0. We adopt the assumptions of [1]: no new peer arrivals, no delays in data redistribution, no repeated data transmissions, a fully connected network, and no constraints on download (retrieving) capacity. We also add the assumption that each peer leaves immediately after it obtains a complete copy of the file. A proof for our more-difficult problem of peer-leaving is harder, however. Still, given that files sent in P2P networks tend to be large, this assumption often holds.

As Figure 1 shows, RSF scheduling creates time intervals, each ending when some peer leaves. Figure 1 also labels the time intervals. During the k_{th} such interval, peer P_k is at the head of the scheduling queue. Then, at time τ_k, this peer P_k finishes. Only at these discrete times does BW_k reduce, being constant within each interval. The higher BW_k gives short time interval.

Moreover, since peers are ordered by capacity, Figure 1 also shows that slower peers finish first: after k-1 peers have left, only those peers of higher number remain (P_k, P_{k+1}, …, P_n). In this way, the finishing time (τ_k) of each P_k is, in turn (and in order of slowness), greedily minimized without considering its effect on future intervals.

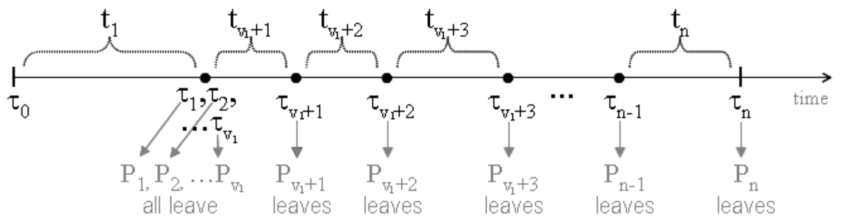

Figure 1. The timeline of peer-leaving events in the proposed RSF Scheduling

Such notation is useful because we apply the Min-Min heuristic of [1], which simplifies the problem of minimizing T_A across all peers to the smaller problem of minimizing only t_k ($=\tau_{k+1}-\tau_k$). Note that some t_k can be zero (*e.g.*, t_2 or t_{v1} of Figure 1). Once P_k finishes, the goal then changes to minimizing t_{k+1}. Then, the relationship between T_A and t_k is:

$$T_A = \frac{\tau_1+\tau_2+...+\tau_n}{n} = \frac{\sum_{k=1}^{n}(n-k+1)t_k}{n} \tag{1}$$

We also use the following notations:

- C_0: the capacity of the source provider
- C_i: the *capacity* of peer P_i
- BW_k: the *total* capacity ($C_0 + C_k + C_{k+1} + \ldots + C_n$) of P2P network in the time interval of t_k. This constrains the performance of P2P file delivery.
- $r_{ij}(t_k)$: the transmission rate from P_i to P_j during interval t_k. (From which it follows that $\sum_{i=k}^{n} r_{ij}(t_k) \le C_i \mid i \ne j$.)
- $R_j(t_k)$: the total bandwidth which P_j receives during interval t_k.. Since the total receiving capacity cannot exceed the total upload capacity, it follows that $R_j(t_k) \le BW_k$. For an example, $\sum_{j=k}^{n} R_j(t_k) = r_{0j}(t_k) + \sum_{i=k}^{n} r_{ij}(t_k)$).

A single-server system can be either slow ($C_0 \le \sum_{j=1}^{n} \frac{C_j}{n-1}$) or fast ($C_0 > \sum_{j=1}^{n} \frac{C_j}{n-1}$) [14]. It is known that a slow-server condition is trivial to schedule, because all peers can receive data at the rate C_0 [15]. We therefore only consider the fast-server condition.

3. Reverse Store-and-Forward Scheduling

As noted above, the RSF scheduler seeks to minimize t_k, for each value of k. This goal can be equivalently restated as minimizing the time needed for the slowest remaining peer (P_k) to finish. One way to obtain this goal would be, of course, to have P_0 only send to P_k. The problem here is that the P2P network would be significantly underutilized; P_{k+1} could receive forwarded data from P_k but only at the rate of C_k (*i.e.*, the smallest capacity). This is the pitfall of all greedy algorithms – that minimizing the current step can make later steps take longer.

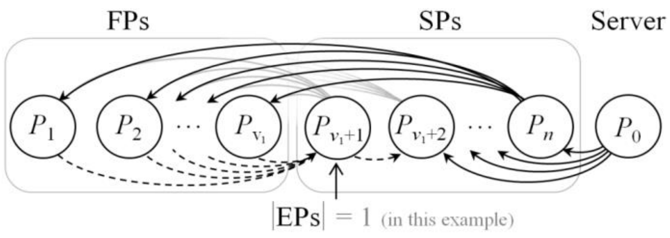

Figure 2. Communication pathways allowed during the first time interval

We overcome this pitfall by hybridizing divide-and-conquer method. For RSF scheduling, the configuration obtains the same minimum value for t_k, but with better network usage. RSF divides peers into two partitions: those which are receiving data from P_0 and those which are receiving forwarded data from other peers. There are therefore two constraints in RSF scheduling: the just-mentioned partition-restriction, and the aforementioned requirements to greedily minimize t_k. Subject to these constraints, RSF scheduling optimizes the total P2P network bandwidth within each time interval. In other words:

$$\text{minimize: } t_k = \frac{|\text{data missing from } P_k \text{ at } \tau_{k-1}|}{\text{transmission rate to } P_k \text{ on } t_k} = \frac{|F| - \sum_{i=1}^{k-1} t_i \times R_k(t_i)}{R_k(t_k)}$$

$$\text{subject to: } \quad t_1 \geq \frac{F}{C_0} \quad \text{and} \quad \sum_{j=k}^{n} R_j(t_k) \leq BW_k$$

$$\text{given that: } \quad R_k(t_k) = \begin{cases} C_0 & k = 1 \\ \sum_{i=k+v_k}^{n} \frac{C_j}{v_k} + \sum_{i=k+1}^{k+v_k-1} C_i & k > 1 \end{cases}$$

where v_k is the partition boundary: $(P_k \ldots P_{k+v_k-1}), (P_{k+v_k} \ldots P_n)$

The above equations raise questions about the partition scheme. One question regards how the partition boundary, v_k, is chosen. Another question regards the purpose that these partitions serve. To simplify these questions, we will now begin by considering the pathways of the first time interval.

ON THE FIRST TIME INTERVAL, k =1, so the partitions are (P_1, P_2, ... P_{v1}) and (P_{v1+1}, P_{v1+1} ,... P_n), as Figure 2 shows. This figure labels these partitions as FPs (for finishing peers) and SPs (for supporting peers), respectively. Additionally, there is a possibility of having EPs (for excess peers), shown as a subset of the SPs.

The distinctions between these sets are in the allowed communication pathways. From Figure 2, the primary pathway is right-to-left, as shown by the solid lines in the figure: P_0 sends data to the SPs, while these SPs send data to the FPs. But these two transmissions differ: whereas P_0 unicasts data and uses IMUX to send *unique* data to each SP (except for the EPs subset), yet an SP sends the *same* data to each FP.

The dashed lines in Figure 2 show left-to-right pathways. We do not wish to waste the upload bandwidth of the FPs (although it is small). Since the FPs still have the upload bandwidth, they instead forward the received to the slowest SP.

Let us now make a brief aside to consider the value of τ_1 for the above configuration. First, consider any one peer P_j, where $P_j \in$ SPs and $P_j \notin$ EPs. Note that, since P_j multicasts to each FP, Since P_j cannot transmit any faster than C_j/v_1, it does not need to receive any faster, either. Second, suppose we choose the SP set to be large enough so that $\sum_{i\in SPs}\frac{C_i}{v_1} \geq C_0$. It then follows that P_0 transmits unique data as fast as it possibly can (since P_0 sends unique data to each SP). Consequently, the capacity of the server is multiplied by v_1 and each FP receives unique data at a rate of C_0. As intended, this configuration is not only greedy for the finish time of P_1 (τ_1 is at its theoretical minimum), but it is also effective at speeding up other peers[3].

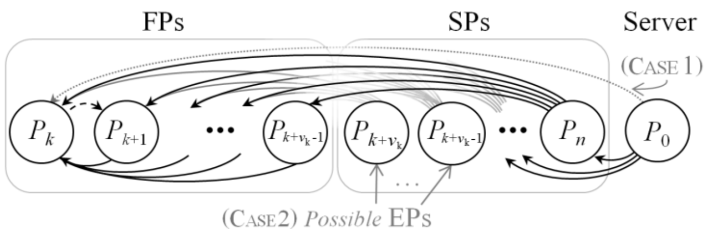

Figure 3. Communication pathways allowed on later time intervals

Now consider the choice of v_k, on each time interval t_k. The optimum value of v_k should minimize any mismatch between the number of FPs and the available capacity among the SPs:

$$v_k = v`, \text{ for that } v` \in (1\ldots n\text{-}k) \text{ that minimizes } \left|\sum_{i=v'+k}^{n} C_i - v`\times C_0\right| \qquad (2)$$

This equation has an absolute value because the difference may be negative. If negative (which we call **CASE 1**), it means that the SP and FP sets overlap: P_0 has extra capacity (to give directly to P_k). If positive (which we call **CASE 2**), then the SPs have excess capacity – meaning that the EPs set will not be empty. (The third case – that of a perfect match – is considered as a trivial form of **CASE 2**; nothing more will be said of it.) The first interval is special because **CASE 1** is disallowed.

Note that the number of EPs may not be an integer. In this case, EPs contribute their capacities to servicing the SPs. Common sense dictates the way, we hereafter simplify the discussion (without loss of generality) to assume that |EPs| (the number of EPs) is an integer.

ON LATER TIME INTERVALS, scheduling is more complex, as Figure 3 shows. These lines indicate that there are no EPs and thatP_0's extra bandwidth goes to P_k. This is because all of the FPs finish at the same time as P_1. Each FP still receives from the SPs. But all capacities of the FPs (except for redundant data) go to P_k. With BW_k fully utilized, t_k is minimized and the requirement of greediness is met. There are now two cases. In CASE 1, the additional green dotted lines are allowed. As for the dashed path in the Figure, P_k's capacity goes to P_{k+1}. The Figure also has additional (dot line) path that are only valid in **CASE 1**.

Most importantly, these pathways are heuristic that simplifies the problem to find v_k. Equation (2) finds an approximate answer, for which the optimal answer of the specific pathways for each peer would be prohibitively difficult to make an algorithm faster, especially when the number of peers is large. When concerning a single time interval, t_k is minimized optimally, because the pathways allow P_k to receive data as fast as possible, and the problem of resource relocation is solved with a divide-and-conquer approach.

Peer Remaining Supported Approaches, these are enhanced versions of RSF for which peers can register themselves as the designated remaining peers (RPs). In this case, peers may not leave immediately after finish receiving data. Even if most peers do leave, those that stay can have a large effect on T_A. We model the likelihood of whether a peer does remain as the PRm (*i.e.*, $PRm = |RPs| / n$), and server has no need to send any data to RPs, the capacity of RPs do not count in (2). Additionally, we assume RPs remain till all peers finish.

Since RPs will remain, our first strategy, eRSF, is to place the RPs at the front of the queue. Thus, if P_m is a RP, then we number FPs such that C_m, C_k, C_{k+1}, ... , C_{k+vk}. After RPs finish receiving data, they are marked as EPs`. EPs` are part of EPs to serve FPs, but they will not turn into FPs. A user of eRSF is rewarded for remaining in the network, by getting less waiting time.

Additionally, uRSF is introduced for the incentive of large capacity. To reward a user for investing in larger capacity, we now modify eRSF by numbering all of FPs with SPT scheduling, thus $C_k \geq C_{k+1} \geq ... \geq C_{k+vk}$. Then, faster RPs (*i.e.*, RPs with larger C_j) would finish first by place them at the front of the queue.

4. Results

We compare our method against two others. The first is SPT, the scheduling algorithm of [1]. SPT perfectly minimizes T_A, if peers do not leave. The second is Even-Share Scheduling, ESS, in which every peer shares equally with all others, so that they all finish together. ESS perfectly minimizes T_L, the time for the last peer to finish. This minimum T_L is constrained by the physical requirement that $T_L^* \geq max\ (|F|/C_0, n|F|/\sum C_{j\in 0,1,\ldots,n})$.

Figure 4 shows that RSF performs well if peers leave, and acceptably if peers remain (given that SPT is, in this case, optimal). The results are for both peer-leaving and peer-staying scenarios. For comparison, the network is the same as in [1]: $n = 4$, $BW_k = 25$, and $|F| = 144$. The peer capacities are C_0=12, C_1=6, C_2=4, C_3=2, C_4=1. Also notice, there is no difference between the finish times of ESS, because all peers finish together.

We next compare the effect of peers having various distributions of capacities, with the same BW_k=25 and C_0=12 that were used in Figure 4. There are three basic types of distributions. The peers of RSF-extreme has capacities that differ greatly: C_1=1, C_2=1, C_3=1, C_4=10. On the contrary, the peers of RSF-similar have capacities that distribute lesser: C_1=3, C_2=3, C_3=3, C_4=4. The RSF-reference is reproduced from the RSF (peer-leaving) in Figure 4. In Figure 5, the RSF has the worst performance when the differences between peers' capacities is the small. Our simulations will base on RSF-similar condition to avoid over-optimistic assessment.

Subsequently, we evaluate the effect of some peers staying upon the performance of the various scheduling strategies, at varying peer sizes. The experiments assume that peer are selfish, and all C_i have an asymptotically chi-square distribution with 1 degrees of freedom, such that we always have fast-server condition and more than 90% of $C_i \approx 1$ for lesser distributions of capacities. In addition, this experiment models the situation where peers do not always leave: by assuming that 80% of the peers are randomly selected to leave after finishing. Then, the experiment is run 1000 times. With confidence interval is 90%, we refer the mean of the results by solid lines, and the standard deviation (σ) by different types of dotted lines. Note, the result for SPT was also intended, but was too poor to graph into the same plot area, for n>10.

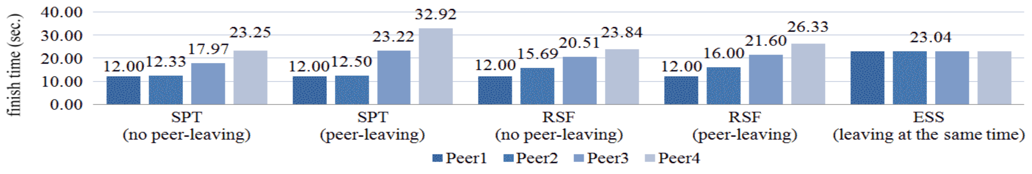

Figure 4. Comparison between RSF, SPT, and ESS

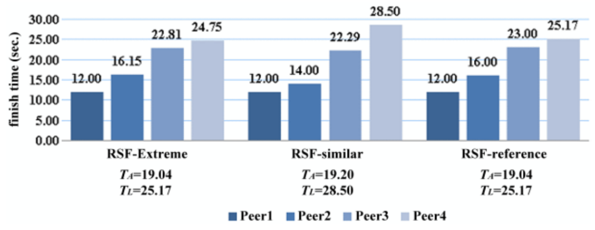

Figure 5. Comparison between RSF, SPT, and ESS.

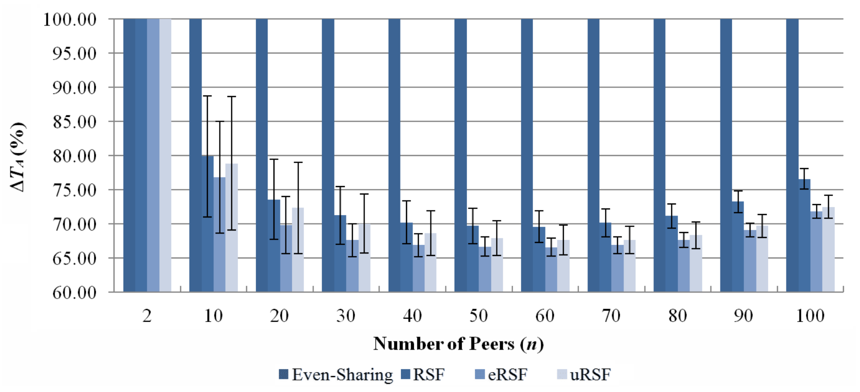

Figure 6. Performance of Normalized T_A vs. Number of Peers

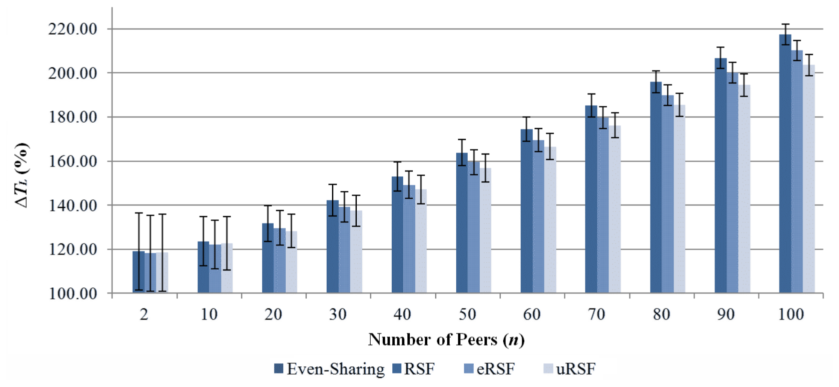

Figure 7. Performance of Normalized T_L vs. Number of Peers

In Figure 6, T_A is normalized to ESS as ΔT_A, and $\Delta T_A=(T_A- T_L^*)/T_L^*$. Not surprisingly, the figure shows that ESS is an upper bound for ΔT_A. RSF, as expected, performs worse (*i.e.*, smaller ΔT_A is better) than eRSF and uRSF. This is because RSF cannot benefit from peer-remaining. Also, the unexpected, uRSF (which is the only method to consider SPT scheduling) has worse result of ΔT_A than eRSF. This happens because removing some fast peers from the tail of the SPs queue will weaken the bandwidth of getting the data from the server and τ_1 is irreparablly belated. Although this is unfortunate, recall that the purpose of the uRSF schedule is to encourage helpful user behavior. Perhaps, without these incentives, those 20% of peers would not invest in larger capacity, too. And, if more users remained, the improvement would increase further.

In Figure 7, T_L is normalized to ESS as ΔT_L, and $\Delta T_L=(T_L- T_L^*)/T_L^*$. Accordingly, the figure shows that ESS is an lower bound for ΔT_L. In this case, the uRSF achieves the better ΔT_L than eRSF and RSF, but the improvement is not signficant. This is because uRSF can benefit more from faster RPs, and the supplementary capacities of the faster RPs are not substantial to have greater influence.

Looking at these figures, the results show that ΔT_A of RSF is reduced significantly when the number of peers is about 60. The improvement reduces about 2/3 finish time of conventional ESS file delivery. Specifically, ΔT_A is reduced down to 69.5%. Nevertheless, ΔT_L continues to rise, and we observe a very interesting phenomenon, where RSF seems to have a scalability wall. This happens because the simulations have RSF-similar condition, and the continues rising ΔT_L has greater impact on the performance. The lesser distributions of capacities, the lesser BW_k in the scheduling queue of RSF. To solve the problem, the peers could be divided into smaller size of clusters with RPs as server (*i.e.*, similar to super peers in Skype). And, if there are resources, service providers could offer a fast dummy peer to be the one who finish last, and use uRSF to offers better ΔT_L (but ΔT_A is slightly worse than eRSF). The optimal number of peers for BW_k is a novel insight deserving future study.

5. Conclusion

Our approach discretizes P2P scheduling, then solves each time interval with a min-min heuristic. Although the scheduler is sub-optimal, this is rarely a concern in practice, because: 1) it is more practical (P2P scheduling is NPcomplete where as RSF is fast), 2) close to optimal is as good as optimal, and 3) real-world networks will not entirely conform to our model, so optimizing it is not profound. Another group [15, 16, 17, 18, 19, 20] has developed an alternative scheduler, which mainly resulting from the tit-for-tat mechanism, for power saving, fairness, and robustness improvement of P2P networks. Their approaches do not include our innovations of partitioning and pathway restrictions.

We have also presented an adaptation that delivers good performance even if peers remain. Future work will extend this with a matrix of historical record to predicit peer behavior and support full peer churn situation as well. Our scheduler is well-suited to this, because it operates in real-time – deciding on the allocation for t_{k+1} while the data of t_k is being sent. Any dynamic changes of system state can therefore be rolled into the schedule.

References

[1] G. Ezovski, A. Tang and L. Andrew, “Minimizing average finish time in p2p networks”, Proceedings of IEEE International Conference on Computer Communications, **(2009),** pp. 594–602,

[2] X. Shen, H. Yu, J. Buford and M. Akon, “Handbook of peer-to-peer networking”, Springer Publishing Company, Incorporated, **(2009)**.

[3] P. Brucker, “Scheduling algorithms”, Springer, **(2007)**.

[4] G. M. Ezovski, A. Tang and L. L. H. Andrew, "Minimizing Average Finish Time in P2P Networks", Proceedings of IEEE International Conference on Computer Communications, Rio de Janeiro, Brazil, **(2009)**.
[5] R. L. Graham, E. L. Lawler, J. K. Lenstra and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey", Annals of Discrete Mathematics, **(1979)**, vol. 5, no. 2, pp. 287-326.
[6] K. V. Shakhbazyan and T. A. Tushkina, "A survey of scheduling methods for multiprocessor systems", Journal of Mathematical Sciences, vol. 15, no. 5, **(1981)**, pp. 651-669.
[7] S. Deb, A. Ganesh and P. Key, "Resource allocation between persistent and transient flows", IEEE/ACM Transactions on Networking, vol. 13, no. 2, **(2005)**, pp. 302-315.
[8] G. Huang and Y. Zhou, "A Request Handling Mechanism with the Shortest Processing-Time First in Web Servers", Proceedings of IEEE Computer Science and Software Engineering, **(2008)**.
[9] G. Wei, Y. Ling, Y. Gu and Y. Ge, "A Dependable Cluster-based Topology in P2P Networks", Journal of Communications, **(2010)**.
[10] J. Leung and H. Zhao, "Minimizing mean flowtime and makespan on master-slave systems", Journal of Parallel and Distributed Computing, vol. 65, no. 7, **(2005)**, pp. 843–856.
[11] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness", A Series of Books in the Mathematical Sciences, WH Freeman and Company, San Francisco, **(1979)**.
[12] J. D. Ullman, "NP-complete scheduling problems", Journal of Computer and System Sciences, vol. 10, no. 3, **(1975)**, pp. 384-393.
[13] M. Mehyar, "Distributed averaging and efficient file sharing on peer-to-peer networks", Ph.D. Dissertation, California Inst. of Technology, **(2007)**.
[14] J. Mundinger, R. R. Weber and G. Weiss, "Analysis of peer-to-peer file dissemination amongst users of different upload capacities", ACM SIGMETRICS Performance Evaluation Review, vol. 34, no. 2**, (2006)**, , pp. 5-6.
[15] A. Sucevic, L. Andrew and T. Nguyen, "Powering down for energy efficient peer-to-peer file distribution", Proceedings of the ACM SIGMETRICS on Measurement and Modeling of Computer Systems, **(2011)**.
[16] B. Fan, J. Lui and D. Chiu, "The design trade-offs of BitTorrent-like file sharing protocols", IEEE/ACM Transactions on Networking, **(2009)**.
[17] Y. Qiao, F. E. Bustamante, P. A. Dinda, S. Birrer and D. Lu, "Improving Peer-to-Peer Performance Through Server-Side Scheduling", ACM Transactions on Computer Systems, vol. 26, no. 4, **(2008)**, p. 10.
[18] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy and A. Venkataramani, "Do incentives build robustness in bittorrent?", Proceedings of USENIX Symposium on Networked Systems Design and Implementation, **(2007)**.
[19] B. Fan, D. Chiu and J. C. S. Lui, "The delicate tradeoffs in bittorrent-like file sharing protocol design", Proceedings of IEEE International Conference on Network Protocols, **(2006)**.
[20] F. L. Piccolo, G. Neglia and G. Bianchi, "The effect of heterogeneous link capacities in bittorrent-like file sharing system", IEEE International Workshop on Hot Topics in Peer-to-Peer Systems, **(2004)**.