Gradus Vol 3, No 2 (2016) 67-72
ISSN 2064-8014

# On maximum throughput in BitTorrent

*Elvira Dobjánné Antal[1]*, and Tamás Vinkó[2]*

[1]Department of Natural Sciences and Engineering, Faculty of Mechanical Engineering and Automation, Pallasz Athéné University, Hungary
[2]Institute of Informatics, University of Szeged, Hungary

***Keywords:***
resource allocation, flow network, BitTorrent, linear programming

***Article history:***
Received 07 Sept 2016
Revised 10 Nov 2016
Accepted 11 Nov 2016

***Abstract***
*The resource allocation problem requesting maximum throughput in BitTorrent networks is investigated. It is known from the literature that the total throughput of BitTorrent is far from the theoretical maximum. However, direct implementation of the unconstrained maximum flow model into BitTorrent is not desirable, due to the necessity of further social considerations. This paper aims to produce more realistic upper bound for maximum throughput in BitTorrent networks by suggesting new model variants to move the maximum flow model closer to the rules of a typical BitTorrent community. Numerical experiments are done on those modified models, introducing lower bounds and balancing constraints on the amount of downloading and uploading, to verify their impact. Additionally, computational results are presented for comparing the network linear program model and the standard algebraic model of the maximum flow problem in AMPL.*

## 1 Introduction

BitTorrent is a computer network protocol for content sharing based on peer-to-peer technology [3]. Those users, who want to download a file $F$, and hence called as *leechers*, get pieces of $F$ from those users, named as *seeders*, who entirely have $F$. Since the file is divided into pieces, the leechers can also exchange them between each other without the contribution of a predetermined central unit – that is the peer-to-peer aspect of BitTorrent. A set of leechers, seeders and files is called *BitTorrent community*. These communities can be modeled by graphs, which allows to analyze different optimization problems related to resource allocation. A simple and natural model is the users–files bipartite network. However, if we need to investigate optimization problems involving bandwidth, then the usage of the tripartite flow network model introduced by Capota *et al.* [4] is advised. One possible problem is maximizing throughput. In that case we would like to maximize the overall happiness in the whole community, while not necessarily take into account the happiness of the participating individuals. Using the tripartite flow network, this problem reduces to the classical graph-theoretical problem of finding the maximum flow. Once it is calculated, maximum flow gives a theoretical upper bound of throughput in the community at the given time instance represented by the graph. Although, this particular problem was investigated up to some degree in earlier papers [4, 8], it is still not fully understood how far is the BitTorrent performance from *realistic* upper bounds.

## 2 Datasets

We used two datasets which are traces of BitTorrent communities. The same datasets were used in [4, 8]. They contain the tripartite graph representation of the two communities, where each graph

*Corresponding author. Tel.: +36 76 516 438
E-mail address: antal.elvira@gamf.kefo.hu

corresponds to an actual status of the online seeders and leechers and shared files. The BitSoup dataset contains ten graphs; it was originally collected in [1]. The FileList dataset contains eight graphs; it was originally collected in [2]. The number of nodes and edges in the BitSoup and FileList graphs are shown in Table 1 and 2, respectively. We can notice that the densities of the graphs are different within a community. Moreover, comparing FileList to BitSoup, we have less number of nodes, but more edges in the former. Hence, we can expect different results for the two datasets.

# 3 Network flow model

One of the most straightforward and easy way to deal with network flow problems is the usage of AMPL [5]. This language enables to model flow networks as network linear programs. In this section we demonstrate the difference of the computational running times between standard (algebraic) model and network model using the `node` and `arc` keywords in AMPL [6].

For testing the two models we used two state-of-the-art LP solvers: Gurobi and MOSEK. Although both solvers accept the two models, only MOSEK is capable to fully utilize the network model as it has a specific LP method to solve network linear programs. The stand-out performance of this specialized method can be clearly noticed in the numerical results below. All the computations were done on a 24-core Intel Xeon 2.27 GHz computer with 24 GB memory.

## 3.1 BitSoup graphs

Table 1 shows the running times in seconds of the two tested solvers. It is interesting to see that while Gurobi is 4.54 times faster on average in case of solving the algebraic model, MOSEK solves the network model even 3.22 times faster on average. It is also beneficial to use the network model for Gurobi, but the increase in the running time is not that much. The clear winner is MOSEK with the network solver.

*Table 1. Running times (in seconds) for BitSoup graphs*

| *graph* | | | *Gurobi* | | *MOSEK* | |
|---|---|---|---|---|---|---|
| *name* | *nodes* | *edges* | *algebraic* | *network* | *algebraic* | *network* |
| b1 | 37 266 | 1 254 129 | 26.20 | 20.26 | 100.33 | 6.22 |
| b2 | 41 243 | 1 526 799 | 32.50 | 21.55 | 133.40 | 8.28 |
| b3 | 33 651 | 934 852 | 16.00 | 13.61 | 74.90 | 4.49 |
| b4 | 33 261 | 739 941 | 13.73 | 12.12 | 62.04 | 3.50 |
| b5 | 34 075 | 1 184 615 | 21.21 | 16.04 | 105.92 | 5.52 |
| b6 | 36 849 | 775 404 | 14.27 | 12.28 | 76.03 | 3.29 |
| b7 | 33 493 | 611 714 | 9.72 | 6.99 | 52.91 | 2.41 |
| b8 | 29 426 | 682 401 | 12.30 | 8.89 | 48.14 | 2.48 |
| b9 | 30 242 | 508 129 | 8.60 | 6.95 | 33.11 | 1.87 |
| b10 | 37 504 | 927 018 | 17.49 | 13.83 | 83.34 | 4.58 |

## 3.2 FileList graphs

The results are shown in Table 2. First of all, we can see that the overall running times are longer than those for BitSoup graphs. Similar trend can be noticed as before, MOSEK is the fastest one if it is running on the network model.

*Table 2. Running times (in seconds) for FileList graphs*

| *graph* | | | *Gurobi* | | *MOSEK* | |
|---|---|---|---|---|---|---|
| *name* | *nodes* | *edges* | *algebraic* | *network* | *algebraic* | *network* |
| f1 | 23 705 | 1 963 645 | 42.58 | 30.92 | 120.96 | 19.34 |
| f2 | 25 516 | 5 673 353 | 141.24 | 89.81 | 668.28 | 41.20 |
| f3 | 27 715 | 4 341 702 | 101.62 | 86.36 | 293.90 | 54.84 |
| f4 | 26 178 | 3 331 577 | 66.19 | 47.60 | 188.50 | 22.87 |
| f5 | 26 731 | 2 997 262 | 58.60 | 43.98 | 178.28 | 19.45 |
| f6 | 26 215 | 6 168 009 | 135.78 | 101.17 | 601.97 | 68.39 |
| f7 | 24 719 | 3 562 633 | 74.86 | 39.66 | 234.56 | 28.93 |
| f8 | 27 796 | 4 671 512 | 86.96 | 97.43 | 382.22 | 42.10 |

Since MOSEK with its network LP solver (called `NETWORK_PRIMAL_SIMPLEX`) performs way much quicker than the others, in the rest of the paper we will use this configuration for the experiments.

# 4 Flows at individuals

We seek the answer to the question: what is the quality of the result of the maximum flow problem at the level of the individuals on these investigated BitTorrent graphs? The paper of Vinkó and Botyánszki [8] gives partial answers, focusing on the seeders. Using the function `maxflow` from the `igraph` package in R, which implements the Goldberg–Tarjan algorithm [7], it turned out that most of the uploading edges have zero flow in the optimum. Moreover, if we require to have positive flow values on the uploading edges, which moves the optimization model closer to BitTorrent, then we get lower value in the total flow compared to the unconstrained maxflow problem. In this paper we investigate the flow values at the leechers' side.

## 4.1 Zero download

First, we have a look on the download edges in the graphs. In case of the BitSoup graphs we obtain zero flow values on the 0.1%–0.3% of the download edges. This means that the corresponding leechers got no data according to the configuration found for the maximum throughput. The FileList graphs give different results: 1.5% – 15% of the download edges get zero flow.

It seems to be a good idea to modify the model in such a way that we require some little amount of flow to be put on each and every download edge. We investigated two constraints: requiring (1) to have flow at least 1% of the capacity of the actual leecher, and (2) to have flow at least 5% of the capacity of the actual leecher. The results we obtained are somewhat interesting: the problems turned out to be unsolvable under the 5% condition. If we require 1%, then two networks was solved in BitSoup and six networks in FileList. In these cases the values of the maximum flow were the same as in the unconstrained version.

## 4.2 Zero download per sessions

In the BitTorrent networks we studied it is usual that leechers are participating in multiple files, i.e. downloading different contents at the same time. These downloads are treated as different edges in the graph representations. Moreover, these edges give more fine-grained picture about the download at a leecher node. Hence, it is worth have a look on the flow values on these edges too. The results are shown in Table 3. We can see that in the network model we obtained larger number of edges with zero flow, which is a disadvantage of that model.

*Table 3. Average percentage of leeching edges with zero flow*

| | *BitSoup* | | *FileList* | |
|---|---|---|---|---|
| *constraint* | *algebraic* | *network* | *algebraic* | *network* |
| w/o | 37% | 40% | 23% | 32% |
| 1% | – | 38% (2 cases) | – | 26% (6 cases) |

On the other hand, when the 1% constraint from Section 4.1 is added to the model (second row in Table 3), then, in the cases when the problems were feasible we got slightly better results.

### 4.3 Balancing at download

In these experiments we introduced another kind of constraints which force balanced flows on the leeching edges of the individual leechers. More specifically, for a given downloading node assuming that it is leeching in $k$ different torrents and gets $F$ amount of total flow in the optimal configuration it is required that each and every leeching edge must have $F/k$ amount of flow. Is it a solvable model?

For the BitSoup graphs we obtained feasible solutions for all the ten cases. On average the value of the maximum flow got decreased to 90% of the unconstrained case's value. Moreover, less than 1% of the downloading edges got zero flow value. Regarding the leeching edges we also have less than 1% of them with zero flow. Note that if we additionally require to have flow value on the downloading edges at least 1% of downloading capacities, then we get feasible solution only for one graph.

For the FileList graphs we again obtained feasible solutions for all the eight cases. On the one hand, on average, the value of the maximum flow got decreased only to 97% of the value of the unconstrained case, which is much better than BitSoup. On the other hand, we got up to 6% of the downloading edges with zero flow, and up to 5% of the leeching edges with zero flow. If we add the 1% constraint on the downloading edges then we obtained feasible solutions on six graphs.

Summarizing the results of these experiments we conclude that it is possible to give equal downloading service per leecher by having a slightly lower overall throughput and by not serving at all some tiny amount of leechers.

### 4.4 Positive flow on leechers uploading

In the last model we get even closer to BitTorrent. It is done by combining the balancing constraints together with trying to put positive flow values to the uploading edges corresponding to the leechers. Note that this experiment was already done in [8] without the balancing constraints. In this model we essentially try to capture the tit-for-tat mechanism of BitTorrent which requests leechers to upload to other leechers (hence they are exchanging pieces of content).

The results we got for the BitSoup graphs are the following. As in Section 4.3 we got 90% of the total flow of the unconditioned model. In most of the cases only less than 10% of the leecher uploading edges got positive flow values. This means that more than 90% of the leechers do not contribute as uploader, which is against the rules of BitTorrent.

In case of FileList, similar to the result of Section 4.3, we have 97% of the total flow value of the original maxflow problem. However, less than 1% of the leecher uploading edges got positive flow, which is even worse than in BitSoup.

An interesting fact to report is that the sum of the flow values on the leecher uploading edges in these experiments are usually rather high. In other words, only a few leechers play role in the uploading process, but they carry a heavy load. In the model examined here we do not aim for maximizing the *number* of leecher uploading edges with positive flow. That would lead to a mixed-integer linear programming problem which might be causing difficulties to solve. Instead, we did a simple modification of the model: the maximum amount of flow on the leecher uploading edges must be less than 1.

This modification resulted in a big difference, as in BitSoup the ratio of zero flow went below 57% (in one of the cases it is as low as 21%). In FileList we could notice also a massive improvement, as at least 25% of the leecher downloading edges got positive flow in every case. Nevertheless, the throughput of the network was decreased only by a maximum of 0.4% for each test case because of this newly introduced constraint.

Finally, we need to mention here that in case we enforce positive flow values on all leecher uploading edges then the problem becomes unfeasible, assuming that the balance constraints are used. This is, however, not that surprising, as in the tripartite graph model this means that a leecher $\ell$ must upload to *all* other leechers participating in the sharing of files which are being downloaded by $\ell$. This is a very restrictive constraint.

# 5 Conclusions

In this paper we did some numerical investigations of the maximum throughput problem in BitTorrent networks, which aims at maximizing the happiness of the community, to see what do we obtain at the individual levels in the optimal configuration. It is known from the literature that the resource allocation algorithm of BitTorrent gives around 60–80% of the theoretical maximum throughput [4]. However, that allocation corresponding to the upper bound does *not* take into account the individuals. It is clear that the direct implementation of the unconstrained maximum flow model into BitTorrent is not desirable. This is due to the fact that it might end up in an allocation where great percentage of the users obtain no service from the system.

We created and investigated three new model variants to move the maximum flow model closer to the rules of a typical BitTorrent community. According to the obtained results, introducing the new constraints leads to decrease of the total throughput compared to the unconstrained case's value. However, our experiments did never show more than 11% decrease, assuming equally good service level for individual leechers in their parallel downloads, but not necessarily requiring leechers to upload. Furthermore, a new linear programming constraint was introduced to involve more leechers to the uploading process without significant downgrade of the total throughput of the network.

To sum up, our results presented in this paper help to understand better BitTorrent in regard to the maximum throughput aspect. It would be worth developing and studying distributed algorithms based on the proposed model variants to verify their advantageous properties compared to the existing BitTorrent protocol. We confirmed with numerical tests that network linear program representation for maxflow problem in AMPL results in significantly shorter running times than running times achieved for the standard algebraic model. Further work could verify whether application of the network LP model for other practical problems is beneficial too.

# Acknowledgement

T. Vinkó was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

# References

[1] N. Andrade, E. Santos-Neto, F. Brasileiro, M. Ripeanu, "Resource demand and supply in BitTorrent content-sharing communities," *Computer Networks*, vol. 53, no. 4, pp. 515–527, 2009.

[2] J. Roozenburg, "Secure decentralized swarm discovery in Tribler," Master's thesis, Delft University of Technology, 2006.

[3] B. Cohen, "Incentives build robustness in BitTorrent," in: *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.

[4] M. Capota, N. Andrade, T. Vinkó, F. Santos, J. Pouwelse, D. Epema, “Inter-swarm resource allocation in BitTorrent communities,” in: *IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2011, pp. 300–309.

[5] R. Fourer, D.M. Gay, B.W. Kernighan, “AMPL,” Boyd & Fraser, Danvers, 1993.

[6] R. Fourer, D.M. Gay., “Expressing special structures in an algebraic modeling language for mathematical programming,” *ORSA Journal on computing*, vol. 7, no. 2, pp. 166–190, 1995.

[7] A. V. Goldberg, R. E. Tarjan, “A new approach to the maximum-flow problem,” *J. ACM*, vol. 35, no. 4, pp. 921–940, 1988.

[8] T. Vinkó and B. Botyánszki, “Empirical investigation of BitTorrent community graphs,” *Computing*, vol. 98, pp. 567–582, 2016.