Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 2255 – 2266

doi:10.3906/elk-1402-304

Improving fairness in peer-to-peer networks by separating the role of seeders in network infrastructures

Alireza NAGHIZADEH*, Reza EBRAHIMI ATANI
Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran

Received: 27.02.2014 • **Accepted/Published Online:** 08.07.2014 • **Final Version:** 15.04.2016

Abstract: Fairness is one of the most important challenges that should be considered as a priority when designing a P2P file-sharing network. An unfair P2P network may attract free-riders, frustrate the majority of users, and consequently shorten the longevity of files. When BitTorrent protocol was first introduced, by suggesting new algorithms like tit-for-tat or rarest-first and combining them with methods like choking and unchoking, it pushed fairness one step forward. However, BitTorrent did not bring a plenary solution and has its own shortcomings. For instance, neither this protocol nor other P2P protocols that we are aware of precisely indicate how seeders should be treated in their networks. Most of the time they dictate that seeders upload as much as possible. With this approach, seeders can easily be abused by free-riders. It gets even worse when we realize that for lack of a good infrastructure a lawful user may become a free-rider unknowingly. The purpose of this paper is to address this problem by suggesting a novel and universal method that can be implemented in current P2P networks. We show that by separating nodes into two groups of seeders and leechers and then implementing a simple pattern for seeders, we can improve fairness in P2P networks. Even though our algorithm is universal and not limited to a specific protocol, by implementing it on BitTorrent we show how it can be used in practice and how effective it is.

Key words: BitTorrent, P2P, seeder, tit-for-tat, free-riding, fairness

1. Introduction

Scalability and cost efficiency are the two most important factors in distributed computing. Peer-to-peer (P2P) networks as distributed solutions address these two factors perfectly. A P2P file-sharing network is a potent distributed solution that, instead of requiring extra servers, functions based on resources of its own users. In comparison with the traditional client/server paradigm, a P2P network is as cost-effective as it can be. In contrast with traditional networks the resource capacity of P2P networks is dynamic in nature. In these networks, as users join the overlay, they share their own resources with the whole network, and as they leave, they take their resources with them. This means that by joining and leaving, the capacity of the whole network will adapt automatically and it gets bigger or smaller based on its user base [1]. The scalability of P2P networks comes from this dynamic nature. Since they are autoadaptive they can scale themselves with a new status perfectly [2].

As we said earlier, in P2P we do not have access to extraneous servers. These networks should only rely on resources of their own users. Thus, overall performance of a P2P network is measured solely by the contributions of its users. However, some clients may not share as much as they take from others. These kinds

*Correspondence: alireza.naghizadeh.a@ieee.org

of behaviors are called free-riding, and outlaw nodes that practice them are free-riders. Free-riding and fairness in P2P share a strong bond. If a network is weak against free-riding we cannot call it a fair network.

Unfortunately, first-generation of P2P networks like Napster [3] and GnuTella [4], which were introduced prior to BitTorrent protocol, did not implement proper mechanisms to have a fair network. These networks could not prevent free-riding and because of the nature of their designs they could even encourage this act. For instance, in GnuTella, users could even get a better download speed by not contributing to other users, as shown in [5,6].

BitTorrent, by using an eminent solution for the recursive prisoner dilemma in game theory, known as tit-for-tat, tried to prevent free-riding or at least make it more difficult [7]. However, it still has some flaws that need to be addressed. This protocol has a tendency to attract and nourish free-riders. It was proven several times that BitTorrent is weak against clients intending specifically to be free-riders. By manipulating clients and changing the rules, free-riders can get even better results than lawful users, as shown in [8–12].

In this paper, by proposing a novel method, we show how we can have a much smarter network by using a global approach for fairness. By combining it with a standard set of simple rules that define the role of seeders and separate them from leechers, we increase the fairness for the whole network.

In Section 2 we give a background of our work. First we present other works on fairness in P2P networks, including what has been done and suggested so far. We then give a quick review of BitTorrent protocol. This gives readers a better understanding of the infrastructure of BitTorrent and prepares them for Section 5, in which we substantiate our method by giving experiments based on BitTorrent and show how it can be implemented in real-world applications. In Section 3 we explicitly define what the problem with current methods is. In Section 4, by taking a universal approach, we introduce our proposed method. Even though this method is by no means specific to BitTorrent and we think that it can be implemented in other structured (or unstructured centralized) P2P networks, in Section 5.1 we show how it can be implemented in BitTorrent protocol. At the end in Section 5.2 we present the results of our experiments and the effectiveness of our method with simulations.

2. Previous works and background

2.1. Related works

Apart from what we present in this paper there has been much work done to improve fairness for P2P networks. Adar and Huberman in 2000 [5] proved that GnuTella was highly vulnerable to free-riding. Their work was continued and in 2005 Hughes et al. [6] showed that after some years the situation for GnuTella had become even worse.

In 2001 Cohen finished version 1 of BitTorrent and in 2003 published it officially [7]. The substantial point in this protocol was to divide data into smaller pieces and use tit-for-tat to speed up downloads and provide fairness in P2P file-sharing networks. Andrade et al. [13] claimed that BitTorrent is actually better than its rivals in preventing free-riding.

However, it was shown several times that BitTorrent is not foolproof against free-riding. Just like in GnuTella, one can even get a better performance without sharing with others. Jun and Ahamad in [8] showed that free-riders could finish their downloads (in a PlanetLab swarm) by only contributing a little. As we have mentioned, tit-for-tat in BitTorrent was implemented to work with a local only approach. Tracker, which is a global manager, does not contribute to preventing free-riding. Liogkas et al. [9] used this shortcoming and showed that by finding seeders and rejecting other nodes one can achieve free-riding. Works in [10–12] used a similar mechanism and we can put them in the same group. They tried to give their free-rider clients a bigger view of the whole swarm and, with that, gain a better download speed than lawful users.

Sirivianos et al. [14] suggested a centralized reputation-based system. In their work, by uploading for other peers, nodes get credits, and by not uploading they cannot achieve more. These credit managements are done by a central server. Izhak-Ratzin et al. in [15] suggested a team incentive system. The main idea is that peers with similar uploading bandwidth should form a group, collaborate, and help each other for a better download. They had to change the original protocol to implement this mechanism. Buddy incentive by Izhak-Ratzin [16] used a similar approach as team incentive but left the BitTorrent protocol intact. Treat-before-trick [17] takes a cryptographic approach and suggests that by using a distributed key algorithm we can force users to contribute back. Works done by Lei Xia and Muppala [18] are similar to [17] in that they block free-riders before they abuse others. This is done by marking them in the network, so before they can cheat they will be identified. Garbacki et al. [19] proposed a tit-for-tat system based on bandwidth instead of reliance on data. This accentuation on bandwidth can be similar to Team Incentives [15], in which users opt for their comrades and make a group based on their bandwidths.

3. Quick review of BitTorrent

BitTorrent is a file-sharing P2P protocol introduced by Cohen [7]. In this protocol data have to be divided into smaller pieces. A swarm will then be made for each specific datum. Nodes collaborate by joining swarms and share the pieces with each other.

The main idea of BitTorrent is that when we want to share specific data, we have to first divide them into smaller pieces. With the collaboration of nodes that are registered to that swarm (by getting help from a tracker), these pieces will be shared sporadically in that swarm. As a result, one specific datum can be downloaded from different nodes with different capacities. In this way we give users flexibility to find new and better nodes and so resource management is much more efficient.

In BitTorrent we have these important concepts:

Metainfo-A file with a .torrent extension that will be created for the first time when someone wants to share his/her data with BitTorrent protocol. Other nodes have to use this file to join the swarm. It consists of the address of a tracker, name, length, and related information about the data to be shared.

Seeder- A node that does not need to download and has the whole data. It is residing in a swarm for helping and uploading for other people.

Leecher- Nodes that do not have the whole data and their primary goal is to finish their downloads.

Tracker- Nodes for joining the swarm and finding other nodes that need to connect to a centralized server. This centralized server is called Tracker.

Swarm-Consists of a group of nodes that collaborate with each other for a specific torrent file. There are two main categories of algorithms in BitTorrent: piece downloading strategy and mechanisms for choosing a node.

Piece downloading strategy refers to a group of algorithms that help nodes for opting for packets. The aim is to help balance the swarm and share packets between nodes more efficiently.

1. Random Order: At first, when a node does not have anything to share, it has to get a piece as soon as possible so that it can upload for others quickly. When a node joins the swarm and does not have anything to share it has to take a piece randomly. This piece is probably repeated in the whole swarm more than the rarest piece (which, as we will show, every other node wants to download). In this way it will have a shorter download time and consequently it can begin to upload for others as soon as possible.

2. Rarest First: This refers to a piece that is copied less than others in the swarm. BitTorrent emphasizes

that nodes should always (after Random Order phase) choose the rarest piece first. If there is more than one rarest piece it can choose one randomly.

3. End Game: This algorithm was suggested for nodes that have almost finished their downloads. In this state there is a tendency for slow speed. To speed up the process, a node sends a request for all of its missing blocks to all of its peers. The node should also send a cancel signal every time a new piece from anyone else arrives.

Mechanisms for choosing a node are about algorithms for selecting new nodes (for downloads/uploads). The aim is to bring fairness for everyone and enhance efficiency. As we have mentioned before BitTorrent does not use a central approach and each node is responsible for increasing its downloads by only relying on its local view. The process when a node bans uploading for its neighbors is called choking and when that node removes the ban and exchanges data for that particular node, is called unchoking. A good mechanism for choosing a new node should follow these three rules:

1. It has to encourage users to participate in uploading for others.
2. It should use all resources (bandwidth).
3. It should thwart free-riding.

Tit-For-Tat: The main idea is that a node can have some connections and data should be exchanged with them simultaneously. In tit-for-tat on BitTorrent, peers should dedicate more than 90% of their bandwidths and upload only for those that are currently uploading for them. The other 10% will be used for finding new nodes.

Optimistic unchoke: The tit-for-tat algorithm works great when all nodes are in the middle of their downloads, but when a node joins the swarm and does not have anything to share, it will never be unchoked. To remedy this problem, BitTorrent uses optimistic unchoke. Optimistic unchoke says that every node should periodically unchoke a randomly selected node. In this way, they may find better nodes to exchange data with and also help newcomers to the swarm.

4. What is the problem?

Much work has been done to improve fairness in P2P networks in general but there are still two problems with current fairness solutions. First, most of the solutions and especially those that use BitTorrent as a base rely too much on a local approach. For instance, even though BitTorrent uses trackers as central supervisors, it does not involve them in detecting and punishing free-riders or helping fairness. Second, they did not implement a good set of rules for seeders. In their solutions seeders are in the network to upload as many packets as possible.

This approach can bring at least three security flaws to networks. First, it makes a good opportunity for free-riders to find and abuse seeders. Second, some nodes may not be free-riders, but by downloading too much from seeders they may freeload unintentionally. Third, when there is not a good set of rules, seeders may not be treated equally, so one seeder may upload much more than its colleagues, which is not fair.

We think that the roles of seeders that already have the whole data and leechers that do not should be separated so that they have to follow two different rules. The duty of seeders should be to make the longevity of data as long as possible, not help leechers to have a free ride. On the other hand, leechers that are in the network not for altruistic reasons but to follow their own desires should be forced to upload as much as possible. With this approach, seeders only upload when it is needed and leechers share back as they are using resources.

5. The proposed method

In this section we introduce our method, which makes seeders upload less and leechers upload more and consequently brings fairness to the whole network. First we propose it with a universal approach without considering the implementation process. In the next section we will show how it can be implemented in a real world P2P protocol, such as BitTorrent.

Our primary purpose is to completely separate two groups of nodes in the overlay and set proper rules for each group on their sharing mechanisms. These two categories are nodes that do not have access to the whole data, which as an analogy to BitTorrent we simply call leechers, and nodes that have access to the whole data. They do not need to download any pieces. They reside in the network only for altruistic reasons. Their job is to help other nodes and increase the longevity of data. We call them seeders.

In this method each datum should have a separate overlay, which gets divided into two, as internal and external parts. Seeders will reside in the external part and leechers will be in internal part. In contrast with traditional networks, the only duty of nodes in the external part is to support the internal part, not upload actively. Thus, seeders will never get involved in the uploading process unless it is needed for data longevity. Figure 1 gives us a better view of this process.

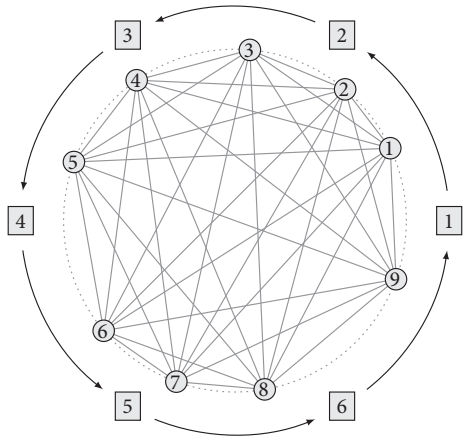

Figure 1. Figure of a P2P overlay that is separated in two parts for specific data. Squares depict seeders and circles depict leechers.

As we can see in Figure 1, the overlay for specific data is separated in two parts. Seeders reside in the outer part and their only duty is to watch over the inner part. When there is a missing piece of data, a seeder will be chosen, which uploads the missing piece for the inner part. In this way, nodes in the inner part are forced to only depend on their own capacities and cannot abuse the generosity of seeders. Consequently, the traffic of the network goes mostly on the shoulders of leechers. With fewer resources to spare, seeders will be motivated to support data much longer than before. It is clear that for free-riders it is much more difficult to live in a network with this paradigm because they cannot abuse seeders anymore. As for the inner part of the overlay, nodes can follow current anti-free-riding algorithms such as tit-for-tat.

For this method to work a global supervisor is required to keep track of some variables such as missing pieces in the inner part of overlay. In a centralized network such as BitTorrent this job obviously falls on the shoulder of trackers. In a structured DHT-based P2P network like Chord [20] it should be possible for seeders to take on this duty one at a time in a periodic order.

Supervising management needs to go through two main stages with two queues for each stage to manage

such a design. The first stage is for missing pieces. The supervisor needs a list of packets that are already being exchanged between leechers in the inner part. If there is any missing piece, this stage helps the supervisor to choose which piece should be uploaded for leechers. As we can see in Figure 2 there are two queues. With the first queue the supervisor realizes which pieces are missing and with the second queue it chooses which piece should be uploaded for leechers in the inner part of overlay.

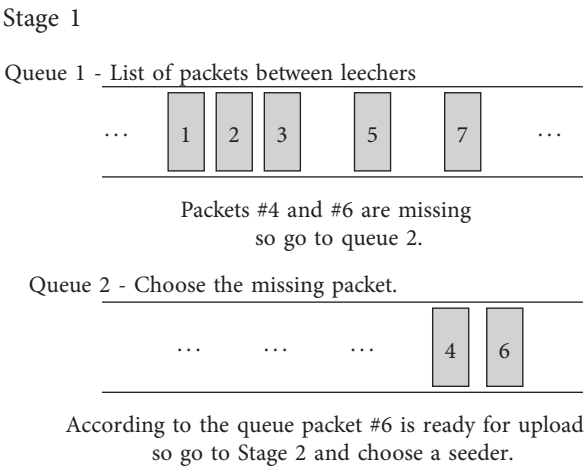

Figure 2. A representation of choosing the missing packets by the supervisor.

When there is a missing piece between leechers (inner part) and it is chosen which missing part should be uploaded, then the supervisor should refer to seeders (outer part). It seeks their help to upload the missing piece for leechers. This is shown in Figure 3. As we can see, it also has two queues. It uses the first queue to choose a seeder. In the second queue it has a list of leechers and chooses a leecher with the lowest number of packets. The reason for choosing a node with the lowest number of packets is to help balance the whole overlay.

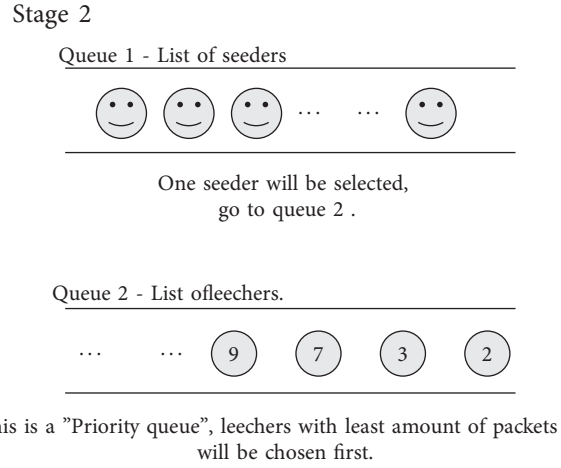

Figure 3. A representation of choosing a seeder and a leecher by supervisor.

In stage 2 each seeder, after uploading a missing piece, should go to the end of the queue and wait until its time comes again. The main reason for this is to make a resting time before it is time for their next upload so seeders would not work continuously; in this way, they will be treated equally.

6. Implementation and analysis

6.1. Implementation on BitTorrent protocol

As we mentioned in Section 2.2 in BitTorrent we have a global server (the tracker) that nodes need to connect to first if they want to join a swarm. For our implementation the tracker gets the role of a supervisor. The obvious drawback for using a centralized infrastructure is that it makes a point of failure for this method.

BitTorrent is a protocol that relies only on local data that peers gather from their neighbors, but for our algorithm to work we need to access information about the swarm globally. To achieve this we have to change BitTorrent protocol so that clients send the exact amount of their downloaded pieces to the tracker. We tried to be as loyal to BitTorrent protocol as possible and the only change we had to make was adding this new rule to the original protocol:

'peer_packets': This is REQUIRED to be sent to the tracker with a bit field representing the pieces that the sender has successfully downloaded, with the high bit in the first byte corresponding to piece index 0. If a bit is cleared it is to be interpreted as a missing piece.

For implementing this method in real-world applications we need these two steps: send peer_packets from clients to the tracker, and implement a tracker that divides the swarm into two groups, an inner part and outer part. We then use peer_packets to implement the proposed method.

We have reverse-engineered both Enhanced CTorrent and BtTransmission (open source BitTorrent clients) and implemented our changes so they could send information to the tracker. This shows how feasibly and simply this can be done. In Table 1 we can see a sample of our results. To make it more intelligible, instead of a bit field we use clear String format. The tracker should get these data and follow the algorithm mentioned in Section 4.

Table 1. Sample of data that manipulated client sent to tracker.

Parameter	Value
peer_id	2d5452323736302d6f62616164357a7939383567
info_hash	db15e84fb4007e2636c5a7ba04e52b6098bd2998
user_agent	Transmission/2.76
ip_address	192.168.1.2
key	9aa3e1604a2c98ca6f59a9f94b1a654c19412f39
port	51413
uploaded	0
downloaded	0
left	723517440
peer_packets	01000001000100.....00000001000110

There were some small differences in the design decision of BitTorrent that we could adapt easily. For instance, in BitTorrent, trackers do not call for seeders, but it is seeders that contact trackers for new leechers.

We should also mention that if we want to be completely loyal to our method, the tracker should send a separate signal to a seeder and determine which pieces exactly a seeder is allowed to upload for other leechers. However, to do this we have to further manipulate BitTorrent protocol, which was not desired. We therefore let it be determined locally. This did not much difference for us since in our implementation the tracker would only introduce one leecher to each seeder at a time and give priority to the leechers with the smallest amount of downloaded packets. It may be true that seeders have to upload a little more, but it also helps the swarm to be more balanced.

After the tracker has received peer_packets, it has to transition from two essential levels: save received data into a database or update the saved data with new data, and depending on the node being a seeder or

leecher it has to get the right data from the database and send it back to the contacted node. The tracker should save the information about the inner part of the swarm into the database. It should check if leechers have all pieces of the data and send back nodes based on these data. In Figure 4 we have shown how a tracker should interact with the database. For making a real-world tracker one can follow this paradigm and make a compatible tracker that follows our method easily.

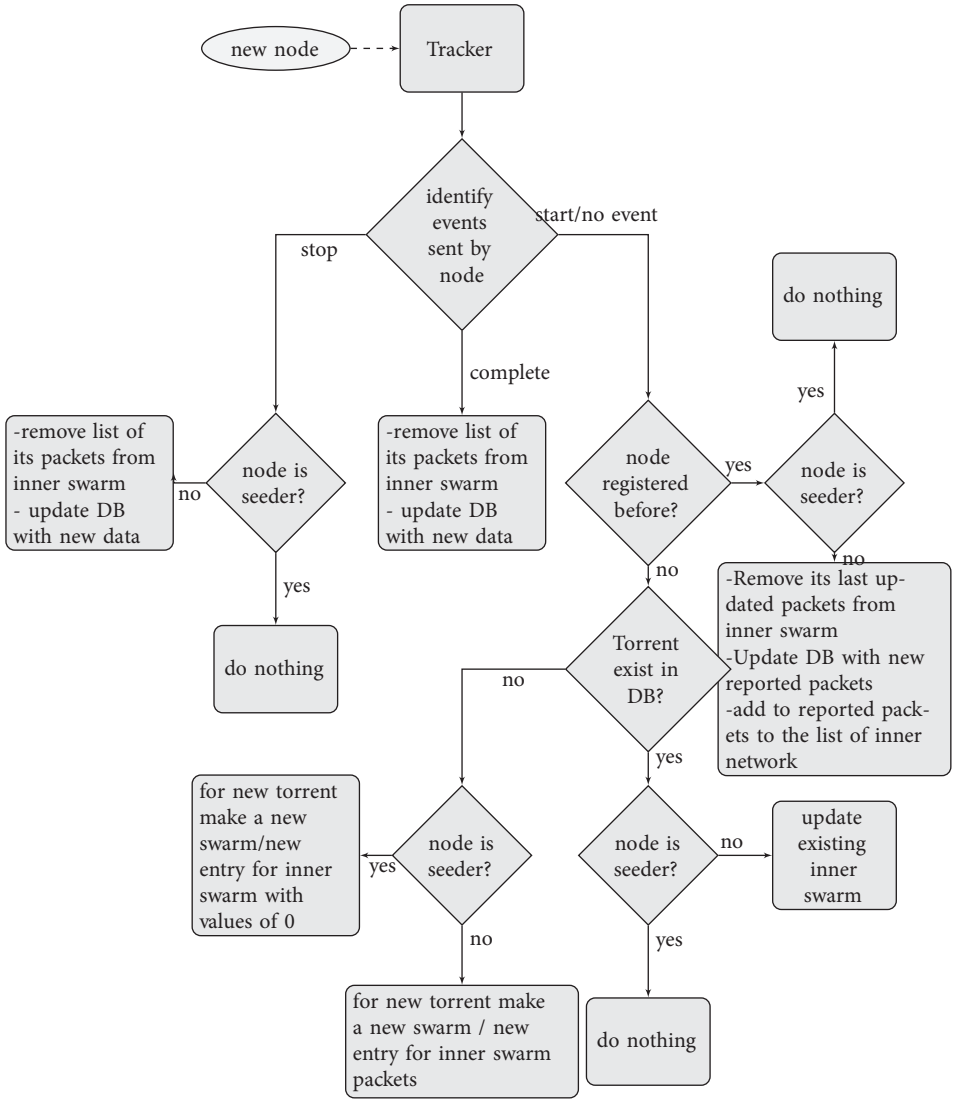

Figure 4. Flowchart for interaction between tracker and database with new data coming from peer_packets.

To see the whole process and to give a more clear view, we made another flowchart, which can be seen in Figure 5. As we can see, by considering existing pieces in the inner part of the network and type of the connecting node, the tracker decides on proper action.

6.2. Simulation and experiments

To do our experiments we used PeerSim [21]. It is an exclusive simulation environment for P2P protocols. PeerSim uses the Java programming language as a base for its component-driven structure. In PeerSim, protocols

can be implemented easily with the help of components and Java objects. Simulations can follow two different models, the cycle-based model and a more traditional event-based model. The former is a simplified model that can be used for high scalable simulations, but it lacks the transport layer and concurrency. We will use a more accurate, event-based model that fits our scenarios better.

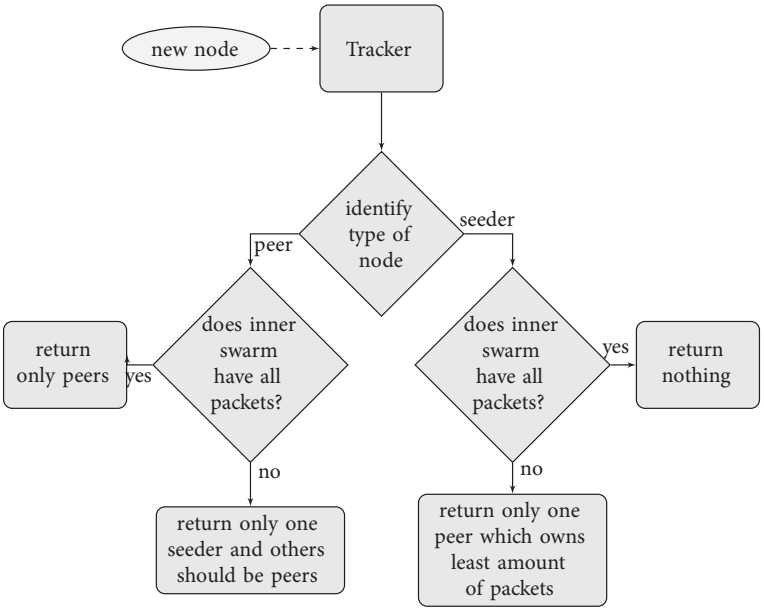

Figure 5. Diagram of overall interaction between tracker and connecting nodes.

Fortunately, BitTorrent was implemented in PeerSim before so we used our experiments based on already established codes. Modification was made when needed, but except for implementing our method and negligible changes, the original code remained intact.

The main purpose of our experiments is to answer these questions: How much traffic will be lifted from shoulders of seeders? Will seeders work in a harmonic way and be treated equally? How much does it compel leechers to upload more? What negative effects does it have on the network and what are the solutions?

The first two questions will be answered in our first experiment. For our first scenario we used a static swarm in which new nodes cannot join or leave the network. In this way we can show how in a perfect situation our proposed method could be beneficial for seeders. The swarm was initialized with the parameters of Table 2. As we can see we have a swarm of 60 nodes with 13 seeders. When nodes have finished their downloads they do not become seeders and they leave the swarm altogether.

Table 2. Parameters of P2P network simulation.

Parameter	Value
File size	100 megabytes
Piece size	256 Kbyte
Number of pieces	390 pieces
Dynamicity	No churn
Swarm size	60 nodes
Leecher set size	50 nodes
Seeder distribution	13
Simulation time	Until all leechers finish downloads

The simulation stopped when all leechers finished their downloads and left the swarm. Figure 6 depicts the results. All 13 seeders that used our method are juxtaposed with their rivals that used traditional BitTorrent. Except for using our method, seeders were exactly in the same situation. As we can see, by using our method, uploads are significantly lower from their rivals. Seeders also have much better harmony in that no one uploads more than at most one or two packets from other seeders.

In our second experiment we want to answer the third question of how it compels leechers to be more active and how changing the infrastructure of our method forces them to upload more for other leechers. This time we used the exact configuration of Table 2 but made our network a little smaller, with 30 nodes as a whole and 22 leechers (7 seeders and 1 tracker). The diagram in Figure 7 is a ratio of downloads to uploads, the average of all leechers that downloaded and uploaded in specific interval times. It shows how much leechers have uploaded for others while downloading in each particular interval time. As we can see the red plot as a whole is in a higher level of uploads.

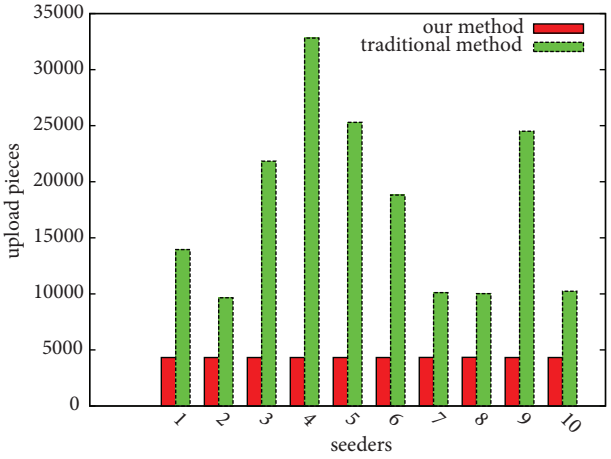

Figure 6. Number of uploads for seeders.

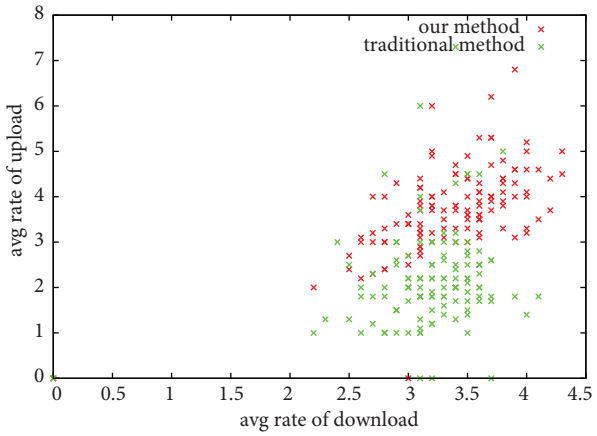

Figure 7. Downloads to uploads for all leechers at specific interval times.

It can be concluded that with this method seeders upload fewer packets, they are treated equally, and it forces leechers to upload more. However, there is one problem that should be considered. By taking out seeders and letting leechers to do more uploads, how much can leechers (by uploading more) recover the job of seeders, and how much negative impact does it have on network? To answer these questions in our last experiment we wanted to be as close to the real-world situation as possible.

The swarm, like a real-world situation, had a dynamic nature, so nodes could leave and join the network. We used a swarm initialized with 30 nodes, which could increase up to 200 and down to 10 nodes. Initially the swarm was configured to be 25% seeders and 75% leechers. The exact configuration setting is shown in Table 3.

On the negative side, as we can see in Figure 8, by using our method leechers usually have to spend more time to finish their downloads. We should remember, though, that by managing the right configuration of the network and making a balance between seeder satisfaction and number of uploads, we can manage to allay this negative side. The other fact is that by lightening resource allocations for seeders, it can exhort other leechers to join seeders and not leave the network after they are finished. This extra force may alleviate this negative impact even further.

Table 3. Parameters of P2P network simulation.

Parameter	Value
File size	100 megabytes
Piece size	256 Kbyte
Number of pieces	390 pieces
Dynamicity	Yes
Add	2
Remove	2
Initial swarm size	30 nodes
Max swarm size	200
Peer set size	50 nodes
Seeder distribution	25%
Initial number of leechers	20
Simulation time	Until all initial leechers finish downloading

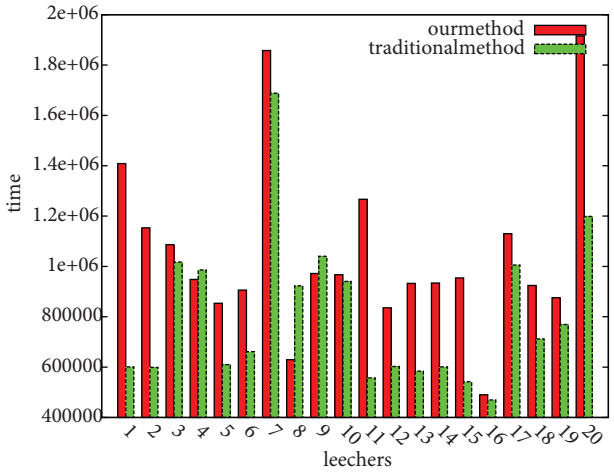

Figure 8. Time required for initial leechers to finish their downloads.

7. Conclusion

The main propose of our paper was to increase fairness in P2P networks by separating the roles of seeders and leechers. Instead of uploading as much as possible for seeders we defined a management model. We had three main premises: seeders should upload less, seeders should be treated equally, and leechers should upload more. Our method was first implemented in real-world applications and then examined by the PeerSim simulation environment and proved to be effective. We discussed a negative side effect, too. Leechers, by uploading more, may not be able to completely fulfill the current status of seeders and nodes may need more time to finish their downloads. We discussed that by better management and getting help from the extra force of new seeders we may alleviate this negative impact.

References

[1] Buford J, Yu H. Peer-to-peer networking and applications: synopsis and research directions. In: Shen X, Yu H, Buford J, Akon M, editors. Handbook of Peer-to-Peer Networking. New York, NY, USA: Springer, 2010. pp. 3-45.

[2] Bondi A. Characteristics of scalability and their impact on performance. In: Workshop on Software and Performance; 2000. pp. 195-203.

[3] Napster Protocol. April 2000. Available from drscholl@users.sourceforge.net.

[4] Clip2 Distributed Search Services. Gnutella Protocol v04. Available from www.clip2.com.

[5] Adar E, Huberman B. Free riding on Gnutella. First Monday 2000; 5: 10.

[6] Hughes D, Coulson G, Walkerdine J. Free riding on Gnutella revisited: the bell tolls? IEEE Distributed Systems Online 2005; 6: 6.

[7] Cohen B. Incentives build robustness in BitTorrent. In: P2PEcon; 2003.

[8] Jun S, Ahamad M. Incentives in BitTorrent induce free riding. In: ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems; August 2005; Philadelphia, PA, USA. pp. 116-121.

[9] Liogkas N, Nelson R, Kohler E, Zhang L. Exploiting BitTorrent For fun (but not profit). In: 5th International Workshop on Peer-to-Peer Systems; February 2006; Santa Barbara, CA, USA.

[10] Locher T, Moor P, Schmid S, Wattenhofer R. Free riding in BitTorrent is cheap. In: 5th Workshop on Hot Topics in Networks; November 2006; Irvine, CA, USA.

[11] Piatek M, Isdal T, Anderson T, Krishnamurthy A, Venkataramani A. Do incentives build robustness in BitTorrent? In: 4th USENIX Conference on Networked Systems Design & Implementation; December 2006; Berkeley, CA, USA.

[12] Sirivianos M, Park J, Chen R, Yang X. Free-riding in BitTorrent networks with the large view exploit. In: IPTPS; 2007.

[13] Andrade N, Mowbray M, Lima A, Wagner G, Ripeanu M. Influences on cooperation in BitTorrent communities. In: ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems; 2005; New York, NY, USA. pp. 111-115.

[14] Sirivianos M, Yang J, Jarecki S. Dandelio: Cooperative content distribution with robust incentives. In: USENIX Annual Technical Conference; 2007. pp. 157-170.

[15] Izhak-Ratzin R, Liogkas N, Majumdar R. Team incentives in BitTorrent systems. In: 18th International Conference on Computer Communications and Networks. pp. 1-8.

[16] Izhak-Ratzin R. Improving the BitTorrent protocol using different incentive techniques. PhD, UCLA, Los Angeles, CA, USA, 2010.

[17] Shin K, Reeves D, Rhee I. Treat-before-trick: free-riding prevention for BitTorrent-like peer-to-peer networks. In: IPDPS; 2009; Rome, Italy. pp. 1-12.

[18] Lei Xia R, Muppala J. Discovering free-riders before trading: a simple approach. In: 16th International Conference on Parallel and Distributed Systems; 2010; China. pp. 806-811.

[19] Garbacki P, Epema D, Steen M. An amortized tit-for-tat protocol for exchanging bandwidth instead of content in P2P networks. In: Self-Adaptive and Self-Organizing Systems; 2007. pp. 119-128.

[20] Stoica I, Morris R, Liben-Nowell D, Karger D, Kaashoek MF, Dabek F, Balakrishnan H. Chord: a scalable peer-to-peer lookup protocol for Internet applications. IEEE/ACM T Netw 2003; 11: 17-32.

[21] Jelasity M, Montresor A, Paolo G, Voulgaris S. PeerSim: A Peer-to-Peer Simulator. Available at https://sourceforge.net/.