Improving Peer-to-Peer File Distribution: Winner Doesn't Have to Take All

Ben Leong, Youming Wang, Su Wen, Cristina Carbunaru, Yong Meng Teo
National University of Singapore
13 Computing Drive
Singapore 117417
{benleong, youming, suw, ccristina, teoym}@comp.nus.edu.sg

Christopher Chang and Tracey Ho
California Institute of Technology
1200 E. California Blvd.
Pasadena, CA 91125
{cswchang, tho}@caltech.edu

ABSTRACT

Recent work on BitTorrent has shown that the choke/unchoke mechanism implements an auction where each peer tries to induce other peers into "unchoking" it by uploading more data than competing peers. Under such a scenario, fast peers tend to trade with one another and neglect slower peers. In this work, we revisit the peer-to-peer (p2p) file distribution problem and show that this does not have to be the case. We describe a p2p file distribution algorithm, the Tit-For-Tat Transport Protocol (TFTTP), that is able to achieve faster download performance than BitTorrent by employing a new mechanism called a *promise*. Our experiments show that the average throughput for TFTTP is some 30% to 70% higher than that for BitTorrent under controlled and realistic network conditions. We also show that TFTTP exhibits fairer sharing behavior and avoids the situation where "winner takes all".

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*

General Terms

Algorithms, Design

1. INTRODUCTION

BitTorrent (BT) [4] is undoubtedly the most popular peer-to-peer (p2p) file sharing application on the Internet today. It has attracted significant research interest in recent years [10, 7, 2]. Among them, Levin et al. recently showed that BT's *peer selection* mechanism, namely the choke/unchoke mechanism, is analogous to an auction: peers auction their bandwidth by offering to upload data to other peers, i.e. by unchoking them, in the hope that the peers will reciprocate and unchoke them in return [7].

APSys 2010, August 30, 2010, New Delhi, India.

Because BT clients periodically unchoke only a small set of peers that upload the most file blocks to them, bidders with inherently low upload bandwidths have a significantly lower chance of being unchoked when competing with higher bandwidth peers. We consider this phenomenon a "*winner takes all*" situation, where the losers get nothing in return for uploading blocks. Low bandwidth peers often end up trading among themselves and are less likely to be reciprocated by peers with higher upload bandwidths [6, 2].

Previous studies have also shown that notwithstanding the claim that BT has the right incentives to achieve robustness [4], its incentive mechanism is not robust to strategic clients [9]. While the addition of a block-for-block constraint to the choke/unchoke mechanism can improve fairness, such a mechanism will degrade download performance significantly [1]. In this paper, we show that not only is a block-for-block exchange mechanism not inherently bad or inefficient, it can improve both fairness and efficiency with a different peer selection algorithm. We argue that it is time to consider the design of algorithms beyond the BT choke/unchoke mechanism.

To this end, we present a new p2p file distribution algorithm, called the *Tit-For-Tat Transport Protocol (TFTTP)*. TFTTP removes the uncertainty in reciprocation associated with the BT choke/unchoke mechanism, and instead allows a pair of nodes to set up a *guaranteed* data exchange as long as they both possess blocks of interest to each other. The efficiency of the sharing is further improved with a new mechanism called a *promise*, which is an agreement between a pair of nodes to exchange blocks. The promise allows nodes to trade not only the blocks they already possess, but also blocks that are expected to be downloaded in the near future. For nodes that do not have many blocks at the beginning of a download session, the promise improves the availability of file blocks since it allows nodes to set up trades and start downloading blocks even before they have received promised blocks.

Our experimental evaluations show that TFTTP outperforms BT by 20% to 40% in average finish time and 30% to 70% in average transfer rate for controlled and practical network environments. The goal of our work is not so much to prove that we can perform better than BT, but to show that there is still room for the development of p2p algorithms beyond BT, and that the tit-for-tat mechanism implemented with promise is a viable alternative to the choke/unchoke mechanism for p2p file distribution.

2. RELATED WORK

BitTorrent has been studied extensively in the literature [10, 5, 1, 9, 7]. Previous studies have analyzed various aspects of the BitTorrent protocol [10, 5, 1]. Qiu and Srikant [10] were first to model BT dynamics using a fluid model and showed that block availability, which they referred to as the *effectiveness of sharing*, is high for BT using a combinatorial argument. We verified in our experiments that block availability is only a transient issue at the beginning of p2p file transfer for TFTTP.

Bharambe et al. showed that BT utilizes uplink bandwidth relatively well and is quite efficient, but there is significant "unfairness" in terms of the amount of content uploaded per node under heterogeneous conditions [1]. In particular, higher capacity nodes tend to upload significantly more blocks than what they download. Their attempts at enforcing a block-for-block-based tit-for-tat policy in BT resulted in a decrease of upload bandwidth utilization. In this paper, we demonstrate that a block-for-block exchange policy does not necessarily cause performance degradation compared to BT under a different peer selection and trading mechanism.

Levin et al. showed that BT implements an auction where clients attempt to induce peers to unchoke them by uploading more data than other peers [7]. Under such an auction, clients that win the auction receive roughly the same amount of data regardless of how much they uploaded to win the auction. To improve fairness, they proposed *PropShare*, where the winning peers get a number of blocks proportional to the amount they uploaded to win the bid. While this makes the system "fairer" to the winners, the losers still get nothing in return even though they had "paid" in advance. TFTTP addresses this issue of fairness by ensuring that all clients are reciprocated by their peers.

3. OVERVIEW OF TFTTP

In TFTTP, the clients contact a server to obtain information about the file and a list of peers. Subsequently, they bootstrap the sharing process by downloading file blocks from the server. In other words, in our current implementation of the TFTTP, the server performs the functions of both the BT tracker and seed. These two functions can be decoupled relatively easily to improve scalability, if needed.

The file distribution process can be divided into three stages: ramp up, steady state, and end stage. During the ramp up stage, many nodes do not have blocks to trade with other peers and the *availability* of file blocks is the key factor affecting efficiency. Block availability during the ramp up stage is improved with promises, because the promise helps to bootstrap the system by making peers without many blocks become more "attractive" trading partners to the other peers, including the higher bandwidth ones.

The transfer of several blocks at a time (referred to as *pipelining* for BT [4]) can often improve the efficiency of the transfers [8], so TFTTP nodes trade in *sectors*. A sector consists of several consecutive blocks and is traded on a block-for-block basis. To improve system efficiency, TFTTP nodes monitor their upload rates and will open just enough connections to their peers to fully utilize their upload bandwidth.

The basic idea of a *promise* is straightforward: when a node finds a peer that has a sector of blocks that it needs and realizes that it has blocks that the peer may want, it sends a trade request for the sector. The peer can either reject the request, or accept it by requesting the same number of blocks from the proposing node in return. After the nodes both upload the mutually-agreed sectors, they can initiate a new trade with each other if both have blocks that the other wants. The financial analogy for the promise is a *forward contract*, where two parties negotiate a deal that is completed at a future point in time. The scheduling of trades using promises provides peers with a degree of certainty about the blocks that they will receive in the future. This helps to ensure that nodes have a pipeline of data that they will send and receive to achieve high sustained bandwidth utilization.

To utilize available bandwidth efficiently, we can show with a queuing theory argument that a node should attempt to download from a peer a number of blocks that is proportional to the available bandwidth to the peer. However, in practice, it is hard to estimate available bandwidth. The available bandwidth also tends to vary over time. As it turns out, trading with promises has a nice *self-clocking* property where the process naturally approximates a bandwidth-proportional download. A node can only have one outstanding trade with each peer and a new trade can only be initiated after the previous one is complete. Since the trades with slower peers will take longer to finish, trades will be established with slower peers at slower rate and fewer trades will be made with them overall.

After a client has downloaded most of the blocks, it goes into the end stage where we reduce the sector size to one block. We call this the *packing* mode. The rationale for the packing mode is as follows: near the end of a download, it is important for a peer to avoid requesting too many blocks from other peers, since a slow peer can significantly delay the completion of the download. TFTTP also implements an *end-game* mechanism similar to BT, where a node requests the same block simultaneously from several peers.

The TFTTP protocol is naturally divided into two processes: the metadata exchange and the block exchange. Due to space constraints, only a brief overview of the TFTTP protocol is described in this section.

3.1 Metadata Exchange

The protocol for the transfer of metadata from nodes to a server is achieved with the following message exchanges.

File Request. When a node first joins the system to download a file, it sends a request for the file to the server together with its current state. We track the state of a file with a data structure called a *file ring*. The file ring is a bitmap of the file blocks arranged in a ring.

Server Accept. If a server decides to accept a node's request, it replies with information about the file, a list of peers and their latest file ring information, and starts to upload a sector of 5 blocks to the node.

Server Reject. Since a server can only service a finite number of requests at any one time, rejections are possible when a server is overloaded. If a client's request is rejected, it will wait a random period of time before issuing the request again. Further rejections will cause it to perform binary backoff.

File Ring Update. A node will periodically send an update of its file ring information to the server. This ensures that the peer list information at the server is up-to-date.

Nodes with complete file rings will be removed from the peer list that is given to the new nodes, because they have no incentive to trade with new nodes.

3.2 Block Exchange

The file ring at the client maintains the status of the blocks it currently possesses. Blocks can be in one of three states: downloaded, "promised" or uncommitted. We represent these states with the colors *black*, *grey*, and *white*, respectively. A downloaded block is one which a node has fully downloaded from the server or a peer; a "promised" block is one which a node is currently downloading from either the server or a peer; an uncommitted block is one which a node has yet to confirm where it can be obtained from.

After a node receives the list of peers, it uses the file ring information contained in the list to decide on a peer to trade with (see Section 3.3). The node then sends a REQUEST message to a peer with whom it wants to trade. The REQUEST message contains the requesting node's file ring and the requested sector of blocks. If the recipient of the REQUEST message does not have an ongoing trade with the requesting peer and determines that a trade can be established, it will reply with an ACCEPT message that contains the sector of blocks it wants in return. Otherwise, a REJECT message is sent.

A node can trade with multiple peers at the same time, but it will only maintain at most one trade with each peer. Each node will also periodically send an update of its file ring to the server. A node may receive blocks from the server if its file ring update contains white blocks. Since the server has a global view of the blocks in the system from the file ring updates, it will attempt to send the rarest white blocks. In this way, a TFTTP server plays a more active role in the distribution of file blocks than a seed in BT, which merely serves requests.

3.3 Trading Algorithm

In our implementation, a sector may consist of up to 5 blocks, because our experiments suggest that this works well in practice. After a node obtains a promise for a sector of blocks from the server, it will incrementally set up trades with other peers and monitor the upload (outgoing) bandwidth. When the measured upload bandwidth no longer increases, the peer concludes that it has saturated its upload bandwidth and will not propose new trades. New trades may however be proposed when the existing trades are completed.

Each peer employs the following algorithm to determine a list of peers and the corresponding sector to request from each peer. Each node estimates the rarity of the blocks that it is currently missing from the file ring information of its peers. Each node first computes the possible trades involving only black blocks, starting with the rarest blocks. If after considering all the black blocks available from its peers, a node is still unable to set up a trade, it will try to set up trades involving both black and grey blocks. Black blocks are preferred to grey ones because trades for grey blocks will fail if the intermediate forwarding node fails. Typically, a node will attempt to propose trades in sectors of 5 contiguous blocks, though smaller trades will be proposed if such a sector cannot be found.

When the number of remaining white blocks in its file ring falls below a threshold, a node will enter the *packing mode*. When packing mode is activated, the sector size for each trade is reduced to one block and only black blocks are requested. The end-game mode kicks in when all blocks in the file ring are either black or grey. When this happens, the node will try to download some grey blocks from several peers in parallel. This is analogous to the end-game mode for BT.

Table 1: Summary of results for EC2.

Algorithm	Download Time (s)	Throughput (kB/s)
BT	2,062	53
TFTTP	1,571	70
TFTTP (without packing)	1,598	68
TFTTP (without promise)	1,706	65
TFTTP (without end-game)	1,731	61

4. EVALUATION

The complete TFTTP implementation incorporates all three mechanisms: the promise, the packing mode, and the end-game mode. To better understand the contribution of each optimization technique, we also implemented variants of TFTTP that are each missing one technique: (i) without promise, (ii) without packing, and (iii) without end-game. Our current implementation of TFTTP is written in Java and in our experiments, we use one server (or seed) to distribute a 100 MB file. All peers join the swarm at approximately the same time and a peer leaves as soon as it downloads the entire file and fulfills all its outstanding trades. In other words, a peer that has completed the download will not propose or accept new trades.

We compared the performance of TFTTP and its variants to a BitTorrent-4.4.0 (Python) implementation. We modified the BT client to also quit as soon as the file download is complete. Our experiments were conducted on two platforms: the Amazon Elastic Computing Cloud (EC2) and PlanetLab.

4.1 Amazon EC2

EC2 provided us with a controlled network environment to compare the effect of the various optimization techniques on TFTTP. We set up dedicated EC2 instances to run Fedora 8 with a 1.7 GHz CPU. Each instance's uploading bandwidth is capped with the unix `tc-htb` command. We set up 25 nodes, consisting of 24 peers and 1 server. The server was configured with a maximum upload rate of 300 kB/s. The clients were configured in three groups of eight nodes each, with upload bandwidths capped at 50 kB/s, 100 kB/s, and 150 kB/s.

Table 1 and Figure 1 summarize the results from the EC2 experiments. In Table 1, we present the average download time and effective average throughput of the experiments. We found that the TFTTP variants outperformed BT by 16% to 24% in terms of average finish time, and by 15% to 32% in terms of average throughput.

From Figure 1, we see that the end-game and promise mechanisms achieved the most significant performance improvements. End-game mode improves the performance of fast nodes, while the promise improves the performance of slow nodes. The packing mechanism only marginally improves the overall performance. We note that the fast nodes

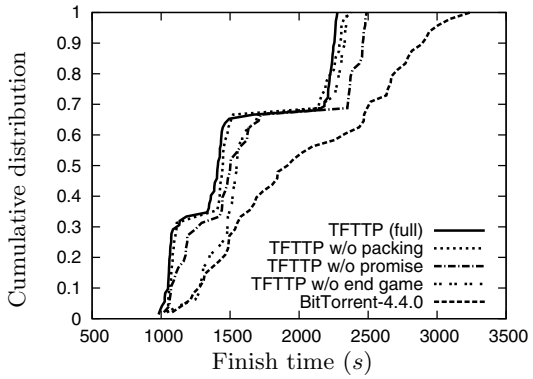


Figure 1: Cumulative distribution of finish times.

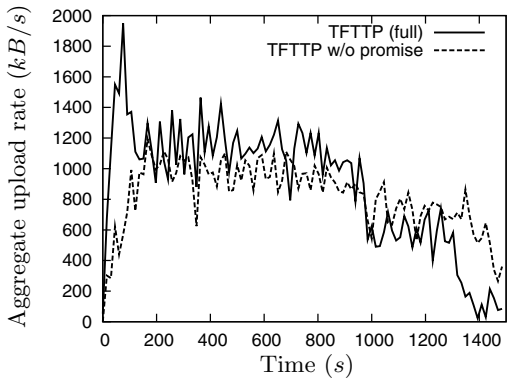


Figure 2: System aggregate upload rate over time.

finished at around the same time for both BT and TFTTP, but the medium and slower nodes ended up significantly slower for BT. This supports our claim that slower peers are disadvantaged in BT.

The end-game mechanism improves the finish times of fast nodes most significantly by allowing them to avoid waiting on slow peers for the last few blocks. The promise mechanism improves the bootstrapping at the beginning of a session because it allows the nodes to set up a pipeline of block transfers even before any of them downloads a single block from the server. Figure 2 shows that the aggregate system upload rate for TFTTP increases much more rapidly at the beginning of the session than that for TFTTP (without promise). In addition, Figure 2 also confirms that TFTTP can maintain a high sustained upload bandwidth utilization during the steady state. The drop at around 1,000 s is caused by fast peers finishing their downloads and leaving the system.

4.2 PlanetLab

For PlanetLab, each experiment is conducted as follows: we select a random set of geographically-dispersed nodes, and pick one of them at random as the server. First, the client nodes use BT to download the file. After all the clients have completed their download with BT, we repeat the same process with TFTTP using the same set of server and client nodes. This procedure allows us to minimize variations in the network conditions and provide us with a fair basis for comparison. We ran the experiments with about a hundred different sets of nodes and the number of clients nodes ranging from 5 to 44. We ran BT with default parameters and a block size of 256 kB, and TFTTP with the same block size.

Table 2: Summary of results for PlanetLab.

Average Value	TFTTP	BT
Download Time (s)	173	305
Throughput (kB/s)	973	547

Table 2 shows that TFTTP achieves on average download times 45% faster than BT on PlanetLab. We observed that the fast nodes for BT are comparable to those for TFTTP, but the slow nodes are often significantly slower. In particular, when we consider the performance of individual nodes, TFTTP is faster than BT about 80% of the time, and is more than twice as fast 40% of the time, and more than three times faster about 20% of the time.

4.3 Clustering for TFTTP

Legout et al. observed a clustering behavior for BT in a heterogeneous environment [6]. By examining the unchoke durations of every pair of peers, they showed that some slower BT peers unchoke faster ones but will not get similar unchoke time in return. Also, peers with similar upload bandwidths showed a clear preference to unchoke similar peers.

We first consider an optimal p2p distribution schedule under a heterogeneous network environment. The optimal schedule is computed centrally using a linear program (LP) that minimizes the average finish time subject to upload bandwidth constraints of each peer and the constraint of reciprocity (i.e. each pair of peers must send the same amount of data to each other) [3]. We run simulations on a network consisting of 25 nodes with 1 server and 24 peers, with the upload bandwidth constraints for server and peers as stated in Section 4.1. Figure 3 shows the data exchange matrix from the LP computations and that generated by a experiment with our implementation of TFTTP on EC2. We found that TFTTP exhibits a similar clustering distribution as the optimal case, in terms of the relative amounts of data exchanged among and within groups, which is comparable to the results for BT in [6].

In Figure 4, we plot the clustering index for the period when all nodes are active in the system (i.e. the period before

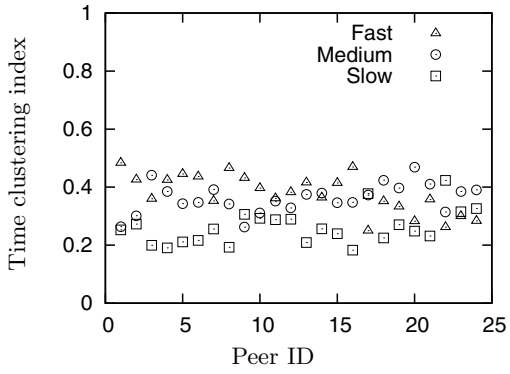
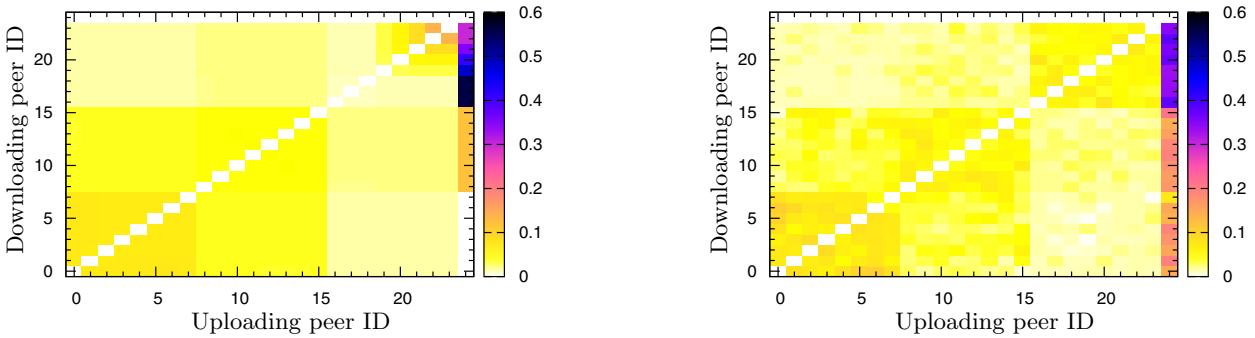


Figure 4: Distribution of time clustering index for different classes of peers with a well-provisioned server. Nodes 0 to 7 are fast peers, nodes 8 to 15 are medium peers, and nodes 16 to 23 are slow peers.



(a) Optimal solution obtained by linear programming.

(b) Distribution for TFTTP on EC2.

Figure 3: Plots of total data uploaded among peers. Node 24 is the server. Nodes 0 to 7 are fast peers, nodes 8 to 15 are medium peers, and nodes 16 to 23 are slow peers. The right color bar shows the amount of data uploaded in terms of the fraction of the whole file.

any of the peers finish downloading the file). The clustering index for a node is the fraction of total upload time the node spends in uploading to each class of peers. Unlike BT, where nodes tend to upload about 65% of the time to other nodes within the same class [6], our results shows that for TFTTP, the clustering index is between 20% to 45% for all nodes and all classes of nodes. The situation where "winner takes all" does not happen because fast nodes are willing to trade with slow nodes and not only among themselves. Also, all nodes, even the slow ones, tend to upload for a longer duration to fast nodes, rather than to the nodes within the same class.

5. CONCLUSION & FUTURE WORK

In this paper, we present the TFTTP protocol for p2p file distribution. TFTTP substitutes the peer selection algorithm of BT, i.e. the chock/unchoke algorithm, with a new block-for-block trading strategy that is naturally fair. Another key contribution in the design of TFTTP is a new promise mechanism. Promises improve the efficiency of trades between peers by allowing nodes to trade blocks to be received in the near future, and allow us to implement a block-for-block mechanism without sacrificing block availability.

Our experiments show that TFTTP can achieve average throughput that is 30% to 70% higher than that for BT and that TFTTP exhibits a fairer sharing behavior than BT. While we have an implementation of TFTTP that works well, we have not fully explored and evaluated the design space for TFTTP. The tuning of parameters like sector size and peer size can possibly be improved. There might also be other server mechanisms that can potentially improve the peer trading performance.

It is clear that block-for-block exchange will not allow us to fully exploit the available bandwidth. This is because once the fast nodes have downloaded most of the file, they have little incentive to upload to the rest. Even with multiple concurrent downloads in end-game mode, it might not be enough to saturate their upload bandwidth. We plan to look into how we can relax the block-for-block mechanism by introducing altruism while keeping the algorithm incentive compatible. We are also working on mechanisms to enforce/police the promises and to address the problem of possible Sybil attacks.

ACKNOWLEDGMENT

This work was supported by the Singapore Ministry of Education grant R-252-000-348-112.

6. REFERENCES

[1] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *Proceedings of IEEE INFOCOM '06*, April 2006.

[2] D. Carra, G. Neglia, and P. Michiardi. On the impact of greedy strategies in BitTorrent networks: The case of BitTyrant. In *Proceedings of P2P '08*, September 2008.

[3] C. Chang, T. Ho, M. Effros, M. Medard, and B. Leong. Issues in peer-to-peer networking: A coding optimization approach. In *Proceedings of NetCod '10*, March 2010.

[4] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the P2P Economics Workshop*, 2003.

[5] M. Izal, G. Uroy-Keller, E. Biersack, P. A. Felber, A. A. Hamra, and L. Garces-Erice. Dissecting BitTorrent: Five months in a torrent's lifetime. In *Proceedings of PAM '04*, April 2004.

[6] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in BitTorrent systems. In *Proceedings of the ACM SIGMETRICS'07*, pages 301–312, New York, NY, USA, 2007. ACM.

[7] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an auction: Analyzing and improving BitTorrent's incentives. In *Proceedings of SIGCOMM '08*, August 2008.

[8] P. Marciniak, N. Liogkas, A. Legout, and E. Kohler. Small is not always beautiful. In *Proceedings of IPTPS '08*, February 2008.

[9] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *Proceedings of NSDI '07*, April 2007.

[10] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proceedings of SIGCOMM '04*, September 2004.