

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

Ben Leong, Youming Wang, Su Wen,
Cristina Carburanu, 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 (TFFTP), 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 TFFTP is some 30 to 70% higher than that for BitTorrent under controlled and realistic network conditions. We also show that TFFTP exhibits fairer sharing behavior and avoids the situation where “winner takes all”.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Design

Keywords

peer-to-peer, tit-for-tat, BitTorrent

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 the 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 unchoking other peers and uploading data to them, in the hope that the peers will reciprocate and unchoke them in return [7].

Because BT clients periodically unchoke only a small set of peers that upload the most 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 (peers with low upload bandwidths) often get nothing in return for uploading blocks and are often reduced to sharing a small number of optimistic unchokes. This phenomenon is reflected in separate studies where it is shown that in the long run, a slower peer is less likely to be reciprocated by a peer with a higher upload bandwidth [6, 2].

Previous studies have also shown that notwithstanding the claims that BT has the right incentives to achieve robustness [4], its incentive mechanism is not robust to strategic clients [9] and that while the inclusion 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 under certain scenarios for a file distribution application. We argue that it is time to relook 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 (TFFTP)*. The key difference between TFFTP and BT is in its peer selection algorithm. TFFTP removes the uncertainty in reciprocation associated with the BT choke/unchoke mechanism, and instead allows peers to set up *guaranteed* data exchanges as long as they each possess pieces of interest to the other party. 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 not only allows slow nodes to ensure that they get something in return for each block they upload, it allows nodes to trade not only the blocks they already possess, but also blocks that are expected to be downloaded from other peers in the near future. The latter improves the availability of file blocks at the beginning of a download session when a node does not have many blocks.

Our experimental evaluations show that TFFTP outper-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

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

Copyright 2010 ACM 978-1-4503-0195-4/10/08 ...\$10.00.

forms BT by 20 to 40% in average finish time and 30 to 70% in average transfer rate for the controlled and practical network environments. The goal of our work is not so much to prove that we can perform better than BitTorrent, but to show that there is still scope for the development of p2p algorithms beyond BitTorrent 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 the various aspects of the BitTorrent performance [10, 5, 1]. Qiu and Srikant [10] were first to model BitTorrent dynamics using a fluid model and showed that block availability, which they referred to as the *effectiveness of sharing*, is high for BitTorrent 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 TFTP.

Bharambe et al. showed that BitTorrent 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 BitTorrent resulted in the decrease in 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 BitTorrent implements an auction where clients attempt to induce peers to “unchoke” them by uploading more data than other peers [7]. Under this auction, client that win the auction receive the same amount data regardless 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 would still have paid in advance even though they get nothing in return. To some extent, the promises in TFTP attempt to address this issue of fairness by ensuring that all clients are reciprocated by their peers.

3. OVERVIEW OF TFTP

In TFTP, clients contact a server to obtain information about the file and a list of peers and they subsequently also bootstrap the sharing process by downloading file blocks from the server. In other words, in our current implementation of the TFTP, 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 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” as trading partners to other peers, including the higher bandwidth ones.

Their transfer of several blocks at a time (referred to as

pipelining for BT [4]) can often improve the efficiency of the transfers [8], so TFTP nodes trade *sectors* consisting of several consecutive blocks. Sectors are traded on a block-for-block basis. To improve system efficiency, TFTP nodes monitor their upload rates and will open just enough connections to their peers to fully utilize the upload bandwidth.

The basic idea of a *promise* is straightforward: when a node finds a peer that has blocks that it needs and realizes that it has blocks that the peer may want, it sends a trade request for a set of blocks. The peer can either reject the request, or accept the request and request the same number of blocks from the proposing node. After the nodes uploaded the mutually-agreed blocks, they can initiate a new trade with the same node, or proceed to trade with other peers. If BitTorrent choke/unchoke mechanism is an auction [7], then 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 can 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. If a node has a high bandwidth to a peer, it will finish a trade faster and propose more trades with that peer; conversely, a trade with a slow peer will take a longer time to complete and so fewer trades will be made.

After a client has downloaded most of the blocks, it goes into the end stage where it trades only one block at a time with each peer, in what we call 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 one slow peer can significantly delay the completion of the download. TFTP also implements an *end-game* mechanism similar to BT, where a node downloads the same block simultaneously from several peers.

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

3.1 Metadata Exchange

The protocol for the transfer of metadata from nodes to a server are 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.

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 the node has fully downloaded from the server or a peer; a “promised” block is one which the node is in the process of downloading from either the server or a peer; an uncommitted block is a block that a node has yet to decide where it should 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 subsection 3.3). The node then sends a REQUEST message to a peer with whom it wants to trade. The REQUEST messages contains the requesting node’s file ring and the requested sector of blocks. If the node receiving the REQUEST does not have a ongoing trade with that peer and determines that a trade can be established, it will reply with an ACCEPT message. Otherwise, a REJECT message is sent.

A node can trade with multiple peers at the same time, but it will only maintain one trade per peer. In addition to trading with peers, a node may also receive blocks from the server if the file ring update it sends to the server still contains white blocks. In our current implementation, the server may choose to send the rarest blocks among white blocks to a peer when it receives a file ring update message from a peer if the peer is not currently downloading blocks from it. In this way, TFTP server plays a more active role in the distribution of file blocks than the BT seed, which merely serves requests.

3.3 Trading Algorithm

In the current implementation, a sector may consist of up to 5 blocks. After a node obtains a sector of blocks from a server, it will incrementally set up trades with other peers and monitors 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 will however be proposed when the existing trades are completed.

Each peer employs the following algorithm to determine a list of peers and their corresponding blocks to trade. 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, the node is still unable to set up a trade, it will try to set up trades involving both black and grey blocks. Requests for black blocks are preferred to requests for grey blocks because trades for grey blocks will fail if a 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 a sector of 5 contiguous blocks cannot be found.

When the number of remaining white blocks on its file ring falls below a threshold, a node will enter the *packing*

Table 1: Summary of results for EC2.

Algorithm	Download Time (s)	Throughput (kB/s)
BitTorrent	2,062	53
TFTP	1,571	70
TFTP (without packing)	1,598	68
TFTP (no promises)	1,706	65
TFTP (without end game)	1,731	61

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 the node’s file ring are either black or grey. When this happens, the node will try to download some blocks on its own file ring that are grey from several peers in parallel. This is analogous to the end-game mode for BT.

4. EVALUATION

The complete TFTP 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 TFTP that are missing one optimizations technique: (i) without promise, (ii) without packing, and (iii) without end-game. Our current implementation of TFTP is written in Java and in our experiments, we use one server (or seed) to distribute a 100MB 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 TFTP 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 experimental platforms: the Amazon Elastic Computing Cloud (EC2) and PlanetLab.

4.1 Amazon EC2

EC2 is a controlled network environment that allows us to compare the effect of the various optimization techniques on TFTP. We set up dedicated EC2 instances to run Fedora 8 with a 1.7GHz 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. The server’s upload bandwidth is capped at 300KB/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 all variants of TFTP outperforms 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 provides the greatest performance improvement. 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 packing mech-

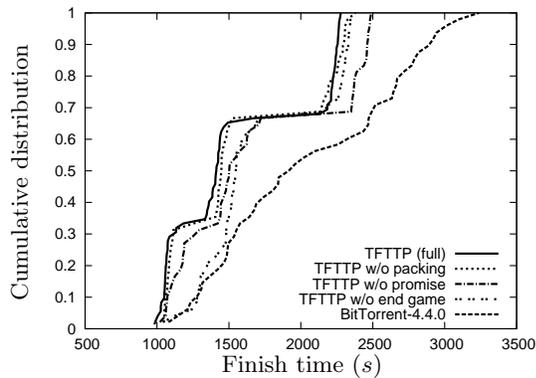


Figure 1: Cumulative distribution finish times.

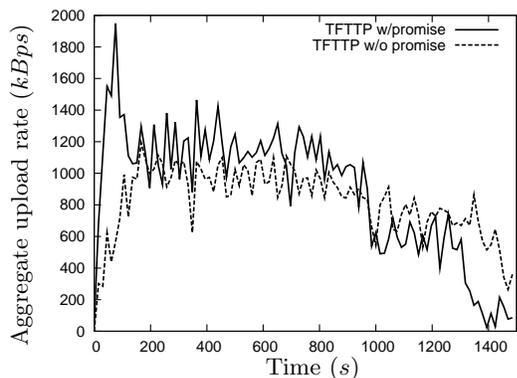


Figure 2: System aggregate upload rate over time.

anism only marginally improved the overall performance. We note that the fast nodes in BT finished close to that of TFTP, but the medium and slower nodes ended up significantly slower. This supports our claim that slower peers are disadvantaged in BT.

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. The effect is clear from Figure 2, which shows the aggregate system upload rate for two variants of TFTP over time that the aggregate upload rate for TFTP increases much more rapidly at the beginning of the session than that for TFTP (no promises). In addition, Figure 2 also confirms that TFTP can maintain a high sustained upload bandwidth utilization during the steady state. The drop 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 $n + 1$ geographically-dispersed nodes, pick one of them at random as the server. All client nodes start downloading the files at approximately the same time. Immediately following the BT download, we repeat the same process with TFTP with the same server and the same set of client nodes. This procedure allows us to minimize variations in the network conditions and have a 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 param-

Table 2: Summary of results for PlanetLab.

Average Value	TFTP	BitTorrent
Download Time (s)	173	305
Throughput (kB/s)	973	547

eters and a block size of 256kB, and TFTP with the same block size.

As shown in Table 2, our experiments on PlanetLab shows that TFTP achieves on average download times 45% faster than BT. We observed that the fast nodes for BT are comparable to those for TFTP, but the slow nodes are often significantly slower. In particular, when we consider the performance of individual nodes, TFTP is faster than BitTorrent 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 TFTP

Legout et al. observed a clustering behavior for BT in a heterogeneous environment [6]. By examining unchoke durations of every pair of peers, they showed that slow BT peers unchoke faster ones but will not get similar unchoke time in return from some faster peers. 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 4 shows the data exchange matrix from the LP computations and that generated by an experiment with our implementation of TFTP on EC2. We found that TFTP exhibits a similar clustering distribution as the optimal case, in terms of the relative amounts of data exchanged within/between the three groups, which is comparable to the results for BT in [6].

In Figure 3, we plot the clustering index for the period

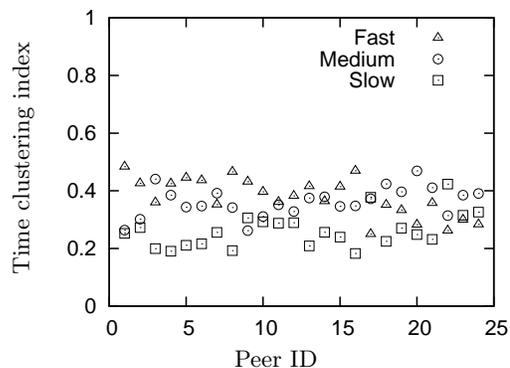
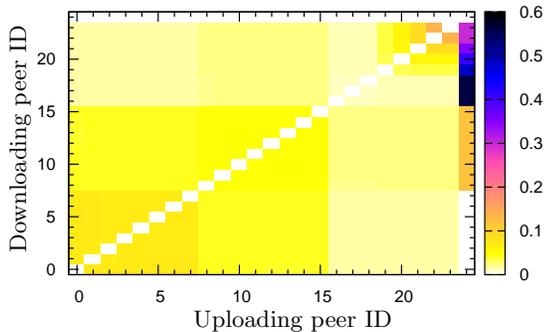
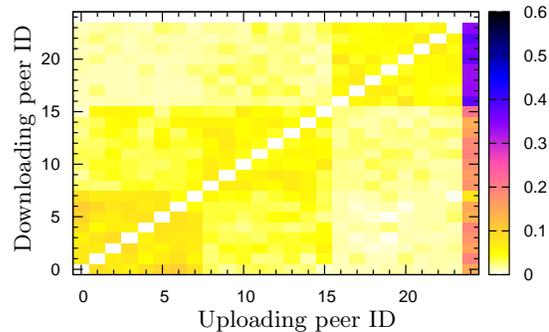


Figure 3: Distribution of time clustering index for different classes of peers. Nodes 0 to 7 are fast peers, nodes 8 to 15 are medium peers, and nodes 16 to 23 are slow peers.



(a) Optimal solution obtained by linear programming.



(b) Distribution for TFTTP on EC2.

Figure 4: Plots of total data uploaded (normalized to file size) between 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 colour bar shows the amount of data uploaded in terms of the fraction of the whole file.

when all nodes are active in the system (i.e. the period before any of the peers has finished downloading). 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 60+% of the time to other nodes within the same class [6], our results show that for TFTTP, clustering index is between 20 to 40% for all nodes and all classes of nodes, and the situation where “winner takes all” does not arise because fast nodes are willing to trade with slow nodes and not only between themselves. All nodes, even the slow ones, tend to upload longer to fast nodes, rather than to nodes within the same class.

5. CONCLUSION & FUTURE WORK

In this paper, we presented the TFTTP protocol for p2p file distribution. TFTTP substitutes the peer selection algorithm of BT, i.e. the choke/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. Promises allow us to implement a block-for-block mechanism without sacrificing block availability.

Our experiments show that TFTTP can achieve average throughput that is some 30 to 70% higher than that for BT and that TFTTP exhibits 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, and 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 incentives compatible. We are also working on a new mechanism 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*, 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. Garcés-Erice. Dissecting BitTorrent: Five Months In 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.