Modeling and Simulation of BitTorrent

Gábor Balázsfalvi,János Sztrik
University of Debrecen, Hungary

Abstract—**BitTorrent is a distributed system designed for sharing large amount of data among many peers. In recent years the data traffic produced by BitTorrent has overloaded the internet. It would be crucial to decrease and diminish this load although at least the recent performance should be kept in balance. We propose a rather new investigation using the network coordinates of peers to estimate the latency among them. Based on this latency peers can select closer peers to download from. We enumerate some methods that can give us the required coordinates. We model BitTorrent and performed simulations by our discrete event simulator working in both the coordinated mode and the conventional one, and the performance of the two methods are compared and demonstrated. Finally, the benefits of the network coordinates are analyzed.**

Index Terms—**Protocols, Computer network performance, Discrete event simulation, Peer to peer, File sharing, BitTorrent**

I. Introduction

The aim of this paper is to point out a kind of extension of BitTorrent, that is a protocol for sharing large data among a number of clients. For peer to peer systems like BitTorrent, mathematical models are very difficult to build up and solving them is difficult if we require high details from the model. For example in [1] Qiu and Srikant modeled BitTorrent but they overgeneralized. We could also analyze real traces, such as Izal did in [2], or Epema in [3]. The drawback of this way is the long period of time needed to collect all data. In addition, it is undesirable that the inspected protocols are not modifiable, i.e. we cannot check them if slightly changed. Simulation of such a system is another possibility to acquire its property; this can be done by discrete event simulators, see for example [4]. Up to now, there are already a couple of simulators made purposely to model peer to peer systems. Bharambe has illustrated a tool of this kind in [5], [6]. We have also chosen this method to try out the inspected extension, namely when peers predict the latency helped by computed internet coordinates. We anticipated it to improve the performance of BitTorrent. We possess a few methods for computing this kind of coordinate. We also know that low dimensional coordinate data are satisfactory to reflect the experimental measurements on the latency between peers.
The structure of this paper is as follows. First, we introduce BitTorrent and we outline some methods for computing coordinates. Then we present our discrete event simulator software and a way to build the structure of events to simulate BitTorrent. Finally, we show some simulation results and analysis concerning BitTorrent, and we try to draw a conclusion.

Research is partially supported by Hungarian Scientific Research Fund-OTKA K 60698/2006, OTKA T049409 and grants of Xerox Foundation UAC (No. 1478-2004),U.S.A..

II. BitTorrent

BitTorrent is a protocol for peer to peer file sharing. It was designed in 2001 by Cohen in [7]. BitTorrent is for distributing great amount of data spread on a large scale and keeping costs down by using those clients that serve data they have already received.
A BitTorrent client is a tool that implements the BitTorrent protocol. First, it cuts into small pieces the data to be shared, and creates a metafile called torrent. The torrent contains hash information about the original data, the pieces, and the tracker,which is the computer that coordinates the file distribution. In general, the number of pieces is very high and their size is very small, usually a few hundred kilobytes. Generally, BitTorrent is used to share data ranging between a couple of megabytes up to gigabytes.
The original owner of the data, after creating the torrent, should put it to a well-known web site where other people can find it. If somebody wants to download the data he has to download the torrent file first. Any BitTorrent client software can read this torrent file then. The software contacts the tracker for a list of peers already taking part in the torrent; they are called neighbors. The neighbors can provide those pieces of data, from which the downloader selects the rarest one, except for the very first time, when the piece is selected at random. After fully receiving a piece of data, one performs a hashing on it, and compares the result with the hash value found in the torrent. If they match, the next piece of data will be selected for download. The peer that downloads is sometimes called leech, while peers having the full data are called seeders. The most important seeder is the original owner.
According to the tit for tat policy, the clients usually prefer uploading to neighbors that also send data back. However, a part of their bandwidth is reserved for a random neighbor. This policy is called optimistic unchoking and it is to help new peers to download their first pieces.
It is important in a torrent's life that the availability of the data should be more than one. This is the number of full copies of the data, i.e. a seeder counts one in this number, while another peer having e.g. 30% of such data that only seeders have, counts 0.3.
According to a recent extension of BitTorrent, the tracker can be implemented as a distributed hash table (DHT). In this case, the protocol is completely decentralized.

III. Creating peer coordinates

In this section, we present a short survey of methods to create peer coordinates used for latency estimation. There are two basic approaches: in the first case, the peers compute their own coordinates, while in the second one, the coordinates are centrally computed and they are distributed among the peers. One way of such a distribution is shown in [8].

A. Triangulated heuristic

One of the first algorithms is called triangulated heuristic and it is shortly described by Eugene and Zhang in [9]. In this algorithm a fixed set of N nodes denoted by $B_1..B_N$ is used as a base. When a new node arrives, it measures its network distance, i.e. the latency related to all B_is, and creates an N-tuple $(d_{HB_1},..,d_{HB_N})$ from the measurement. It uses these data as relative coordinates. We assume for the moment that triangle inequality holds for network distances. Given two nodes H_1 and H_2, we can fix the limits of the distance between them by L and U, computed as:

$$dist(H_1,H_2) \leq \min_{i\in\{1..N\}}(d_{H_1B_i}+d_{H_2B_i})(=U),$$

$$dist(H_1,H_2) \geq \max_{i\in\{1..N\}}(|d_{H_1B_i}-d_{H_2B_i}|)(=L).$$

We could use any convex combination of U and L as a distance between H_1 and H_2. We have to mention that no global coordinates are created, but this is not a problem since only the distances are used.

B. Global Network Positioning

The Global Network Positioning (GNP) is also described by Eugene and Zhang in [9]. This method uses a fixed set of N nodes, called landmarks. Denoting the dimension of the coordinates by D, the number of landmarks must be greater than the dimension, i.e. $N > D$. First of all, the landmark nodes compute their own coordinates by minimizing the sum of the squared error between the computed distances and the measured ones. A new node entering the group has to measure its distance to all landmark nodes and it has to minimize the sum of the squared error. According to Costa's extension described in [10], the landmarks can differ from peer to peer. This method has some important disadvantages, for example, the need for the high computational power for solving the minimizing problem. Furthermore, a large number of landmark nodes are needed because the coordinates correlate to each other. The dimension of the coordinates can be decreased only in a later phase when all the coordinates are known. The set of landmarks and as well their coordinates have to be refreshed as the number of nodes increases.

C. Vivaldi

Recently the latest technique is called Vivaldi, which is a distributed spring system, described by Cox et al. in [11]. Whenever two nodes communicate with each other, they measure their latency and refresh their coordinates according to the result. They try to minimize the error between the predicted and measured values. Each node receives the coordinates of the neighbors as well as latency measurements during the communication, and it slightly adjusts its own coordinates according to neighbor's push. If two nodes occupy the same position, they push each other in a random direction (this also controls the initial condition when all the coordinates are 0). The algorithm is based on a δ parameter, which is a real number determined by the peer and is changed adaptively according to local and remote errors. As Cox et al. showed in [11], the convergence of the algorithm is mainly based on this parameter, and the algorithm can work with low dimensional coordinates.
The experimental usage of Vivaldi can be read in [12], where Ledlie describes the Azureus software as their test bed. Azureus is an implementation of the BitTorrent client protocol, and it is capable to use network coordinates.

IV. Discrete event simulator

A. Framework

In writing our framework, we have followed the suggestions of Jain presented in [4]. We have built an event and process oriented discrete event simulator in Java language. Although software products running on Java Virtual Machine need more resources than their native analogues, a Java program can run on a large scale of platforms without any modifications. Additionally, Java has extensive graphical library to serve as a basis for graphical representations. The components of the simulator can be seen in figure 1. Its main part is the scheduler, which can schedule events at a specific time or after a concrete time. It is able to schedule processes too, the expiration of which (the moment when they are finished) varies dynamically according to other events and processes. The *Event* class is the ancestor of every event in our framework. The task of the events is done in their *run()* method in compliance with the later descriptions. They can be timed at a well defined instant or at some point after a given moment. The former is called well defined, and the latter is called badly defined schedule time event, because its timing is dynamically determined by other events; we can only say that the event does not fire before the given point of time, but it fires before the next well defined schedule-time event with a latter schedule-time. We can schedule processes in addition to events. The ancestor of every process is the *Process* class of which *finishTime()* method is invoked at every event's fire time and every process' expiration moment with the current time as an actual parameter. Then these processes can do their own work, i.e. their task that was to be done since the last invocation. They must prognosticate their moment of expiration according to the actual conditions (these conditions can be changed only by other processes or by an event, so the correct point of time cannot be missed).

B. Modeling BitTorrent

We have modeled BitTorrent by the following series of events. A peer is always born by a *BornEvent*. We can set this object to create the peer as a seeder or as a leech, and we have to give the tracker object to this event object. The *run* method of the peer schedules a *GetPeerListEvent* object, which is responsible for the neighbors of its owner. These neighbors can be produced according to the network coordinates if they are present. The *GetPeerListEvent* can schedule another *GetPeerListEvent* at a later time, because the peer may have to renew its list of neighbors, e.g. after becoming seeder.

It also schedules a *SelectPieceEvent* object that implements the different kinds of piece selection policies, i.e. the rarest first policy and the random one. The *GetPeerListEvent* is responsible for considering the network coordinates too. If the selected piece is not available for download then it schedules another *SelectPieceEvent* after the next event, otherwise an *UpdateLinksEvent* object is created to renew and refresh the download processes. After all the pieces are downloaded, a *BecomeSeederEvent* object is created turning the peer into seeder. The death of a peer is modeled by the *DeathEvent* class.
At present, the only process that can be modeled in our system is the download of a piece, the duration of which depends on the actual bandwidth that is dynamically computed by the *Link* class.

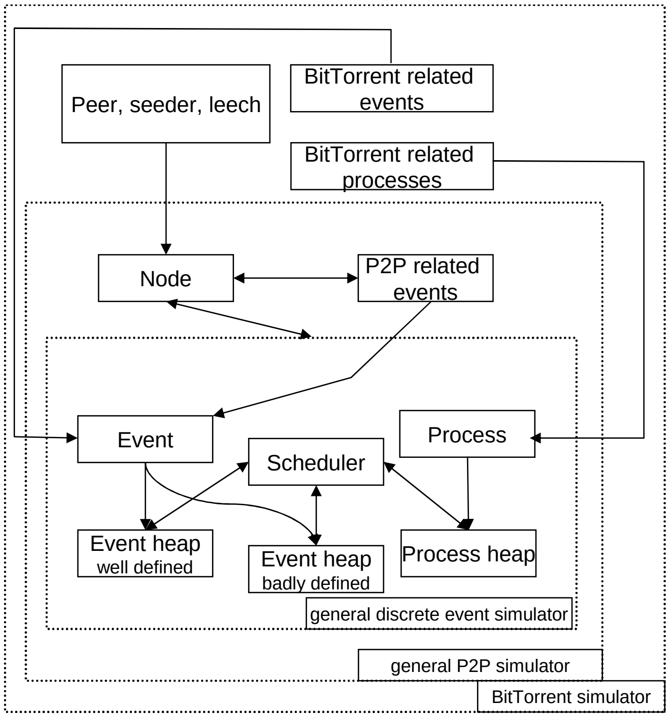


Fig. 1. Simulator model: associations.

V. Peers with coordinates

Our exploration was to dissect the topic of giving latency and bandwidth information to the peers. In real life, we could do this by giving coordinates to peers by their help they can guess the bandwidth and latency concerning other peers. The Vivaldi system described in [11] is a distributed system doing exactly this, i.e. it computes multidimensional coordinates for each peer taking their communication speed into account. By having the coordinates of other peers, a peer can compute the distance (e.g. the Euclidean distance) from other peers and it guesses the bandwidth without even sending them a bit of information on the network, thus saving a lot of resources. We have not modeled the details of the coordinating system yet. We have just assumed that we know the distances which are supposed to be uniformly distributed. Thus we can build a formula estimating the download time.

For the sake of analysis, let the bandwidth between the x_1 and x_2 peers be denoted by $B(x_1, x_2)$, and let us denote the distance between them by $t(x_1, x_2)$. Since the number of peers is finite, t can be chosen so that it maps into the real interval $[0, 1]$. If x_1 and x_2 were identical, $t(x_1, x_2)$ should be zero. The largest distance between two peers is 1. We can also scale the B mapping in such a way that its highest value (which is the maximal bandwidth) is equal to 1, but B must map into the domain of positive real numbers, because peers can always reach each other in our context. Let the number of neighbors of a peer x_0 be n, denoted by $x_1..x_n$. We can determine B by:

$$B(x_0, x_i) = 1 - (1 - u)t(x_0, x_i),$$

where u is some real constant between 0 and 1.
In general, the expected value of a function f of the distance t can be determined by the probability density function ($p(t)$) of the distance:

$$E[(f(t)] = \int_{-\infty}^{\infty} f(t)p(t)dt = \int_{0}^{1} f(t)p(t)dt.$$

This means that we can get the average bandwidth by the equation:

$$B = E[1 - (1 - u)t] = \int_{0}^{1} (1 - (1 - u)t)p(t)dt.$$

In our scenarios we assumed that the distances $(t(x_i, x_j)|i, j \in 1..n, i \neq j)$ were uniformly distributed in the range $[0, 1]$, so $p(t) = 1$ in that range. This means that the resultant bandwidth (which is denoted by B_1) is in direct proportion to $0.5(1+u)$. If the tracker used coordinates, when it gives the list of neighbors to the peers, we would have the distribution function (distribution among the neighbors)

$$p(t) = \begin{cases} \frac{m-1}{n} & t \in [0, \frac{n}{m-1}], \\ 0 & \text{otherwise}, \end{cases}$$

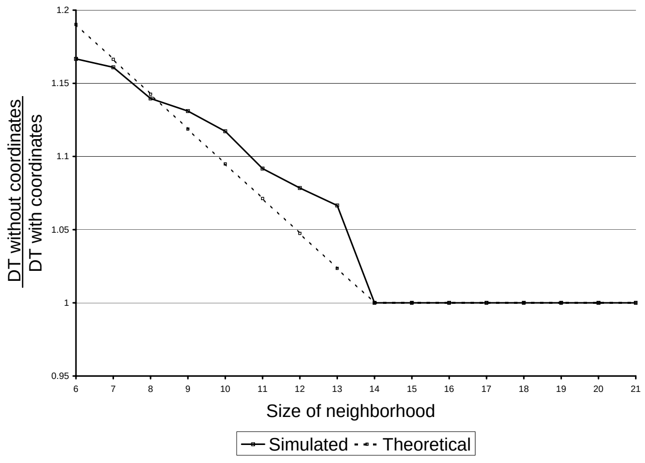
where m is the total number of peers in the system, and $m \geq n$. Then

$$B_2 = \frac{m-1}{n} \int_{0}^{\frac{n}{m-1}} 1 - (1-u)\, t dt = \frac{2m - 2 - n + ne}{2(m-1)},$$

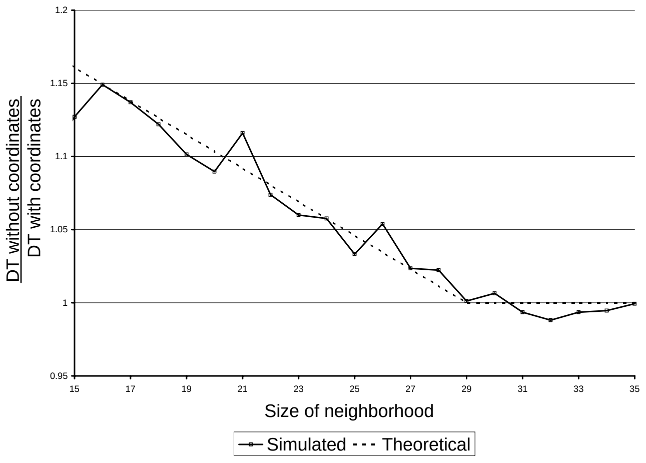
which shows the improved bandwidth. For the sake of comparison of B_2 with B_1 we computed their quotient showing the improvement produced by the investigated method.

$$\frac{B_2}{B_1} = \frac{2m - 2 - n + nu}{(1 + u)(m - 1)}.$$

We ran two scenarios with uniformly distributed distances. The results can be seen in figure 2. The first one shows the scenario for a fixed set of 15 peers one of which was seeder and the others downloaded the data. We set the size of the maximal neighborhood between 6 and 21. The distances were uniformly distributed in the real interval $[0, 1]$, and the parameter u was set to 0.5. In order to compare the two methods, we ran the simulator 30 times for each setup. We computed the



(a) Scenario 1



(b) Scenario 2

Fig. 2. The fraction of download times, theoretical and measured values.

average download time considering random list of neighbors and distance directed lists as well. After obtaining the results we divided the former by the latter one, expecting the result to be similar to $\frac{B_2}{B_1}$, since the download time was inversely proportional to the bandwidth. We found that the same effect appeared when a larger set of peers were present. This result can be seen in figure 2b. Here the number of peers was set to 30, and the size of the neighborhoods varied from 15 to 35. The same effect can be recognized as before, when the size of the neighborhoods exceeded the number of leeches.
The two methods performed in a similar way when the neighbors could exceed the peers downloading in number (14 in figure 2a and 29 in figure 2b).

VI. Conclusion

It was shown that applying network coordinates the average download time could be reduced. This observation was justified both by theoretical and simulation results. The simulation was made by our discrete event simulator, which was described as well. We demonstrated a way of modeling BitTorrent by events. Although BitTorrent is not standardized with network coordinates, creators of different kinds of client software should think of using some types of network distances and filtering the list given by the tracker, thus the above studied approach could be of use.

References

[1] D. Qiu and R. Srikant, “Modeling and performance analysis of bittorrent-like peer-to-peer networks,” in *ACM SIGCOMM 2004*, 2004, pp. 367 – 378.

[2] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. Al Hamra, and L. Garc’es-Erice, “Dissecting bittorrent: Five months in a torrent’s lifetime,” in *Proceedings of the 5th Passive and Active Measurement Workshop*, 2004, pp. 1 – 11.

[3] J. A. Pouwelse, P. Garbacki, D. H. J. Epema, and H. J. Sips, “The bittorrent p2p file-sharing system: Measurements and analysis,” in *In Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS’05)*, vol. 3640, 2005, pp. 205 – 216.

[4] R. Jain, *The art of computer systems perfomance analysis*. New York: John Wiley & Sons, 1991.

[5] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, “Analyzing and improving bittorrent performance,” Microsoft Research, Technical Report MSR-TR-2005-03, 2005.

[6] A. R. Bharambe, V. N. Padmanabhan, and C. Herley, “Analyzing and improving a bittorrent network’s performance mechanisms,” in *IEEE Infocom 2006*, 2006, pp. 1 – 12.

[7] B. Cohen, “Incentives build robustness in bittorrent,” in *Workshop on Economics of Peer-to-Peer Systems*, 2003, pp. 251 – 260.

[8] R. Fonseca, P. Sharma, S. Banerjee, S. Lee, and S. Basu, “Distributed querying of internet distance information,” in *Proceedings of 8th IEEE Global Internet Symposium*, 2005, pp. 2876–2881.

[9] T. S. Eugene Ng and H. Zhang, “Predicting internet network distance with coordinates-based approaches,” in *IEEE Infocom 2002*, vol. 1, 2002, pp. 170 – 179.

[10] M. Costa, M. Castro, A. I. T. Rowstron, and P. B. Key, “Pic: Practical internet coordinates for distance estimation,” in *International Conference on Distributed Computing Systems*, 2004, pp. 178 – 187.

[11] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris, “Practical, distributed network coordinates,” *ACM SIGCOMM Computer Communication Review*, vol. 34, pp. 113 – 118, 2004.

[12] J. Ledlie, P. Gardner, and M. Seltzer, “Network coordinates in the wild,” in *Proceedings of NSDI 2007*, 2007, to appear.

[13] S. H. Kwok, “P2p searching trends: 2002-2004,” *Information Processing and Management*, vol. 42, pp. 237 – 247, 2006.

[14] G. Zihui, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley, “Modeling peer-peer file sharing systems,” in *IEEE Infocom 2003*, vol. 3, 2003, pp. 2188 – 2198.