University of Würzburg
Institute of Computer Science
Research Report Series

Relying on Randomness PlanetLab Experiments with Distributed File-sharing Protocols

Ilkka Norros[1], Vesa Pehkonen[1], Hannu Reittu[1], Andreas Binzenhöfer[2], and Kurt Tutschku[2]

Report No. 407 April 2007

[1] VTT Technical Research Centre of Finland
P.O. Box 1000, 02044 VTT
Finland
{ilkka.norros, vesa.pehkonen, hannu.reittu}@vtt.fi

[2] University of Würzburg
Institute of Computer Science
Germany
{binzenhoefer, tutschku}@informatik.uni-wuerzburg.de

Relying on Randomness
PlanetLab Experiments with Distributed File-sharing Protocols

Ilkka Norros, Vesa Pehkonen, Hannu Reittu
VTT Technical Research Centre of Finland
P.O. Box 1000, 02044 VTT
Finland
{ilkka.norros, vesa.pehkonen, hannu.reittu}@vtt.fi

Andreas Binzenhöfer, Kurt Tutschku
University of Würzburg
Institute of Computer Science
Germany
{binzenhoefer, tutschku}@informatik.uni-wuerzburg.de

Abstract

In this paper we present and evaluate a fully distributed file-sharing system architecture. Initially, a seeder node splits a large file into moderate-sized chunks and offers it for download. The seeder and all peers interested in downloading the file join a Chord-based overlay and contact each other randomly for chunk transfers. We report and discuss PlanetLab test on this system as well as on our modified Chord implementation which it is based on. We also describe algorithms for finding uniformly random peers in the overlay and for estimating the distribution of chunk copies and show that both have a significant effect on the performance.

1 Introduction

BitTorrent [1] introduced a very effective technique for distributing a large file to a large number of recipients. The basic idea is that the file is chopped into small chunks that the recipients can immediately upload further. With optimal coordination, the number of copies of each chunk grows exponentially fast until it saturates. The main point in this paper is that essentially as good performance is in fact possible without any centralized coordination, indicating that the "tracker" functionality of the original BitTorrent may not be necessary.

The experimental prototype of a BitTorrent-counterpart with completely distributed architecture, considered in this paper, was originally designed in the Finnish research project PAN-NET in 2005. This system is based on random encounters, i.e., the peers contact each other randomly and download in each encounter one or several chunks if such are available by the contacted peer. The PAN-NET prototype uses Chord [2] as the underlying overlay network structure where random contacts can be realised. The prototype was developed further in the REDLARF project of the Euro-NGI Network of Excellence (`http://www.eurongi.org`), aiming at field testing in PlanetLab [3]. Two kinds of tests were run. First, we wanted to obtain experience and insights about how well the Chord protocol works in real implementations and what improvements in it are recommendable. These tests were performed using a Java implementation written at the University of Würzburg. The second set of tests was made with the PAN-NET prototype after improving its Chord implementation according to the experience from the first set.

Some theoretical and simulation-based work already exists, posing interesting questions for tests with a practical implementation. Massoulie and Vojnovic [4] studied a similar system as an abstract model obtaining some remarkable results and insights. In particular, they found that a file sharing system based on random encounters can reach a balanced state where it works very effectively and, moreover, does not even need strategies like "transfer rarest chunk first". Moreover, the peers were assumed *non-altruistic* in the sense that they stay in the system only as long as they have not downloaded all the chunks. In contrast to this steady-state scenario, Norros *et al.* [5, 6] focused on the so called *flash crowd* scenario, where all nodes except the original seeder join the file distribution overlay simultaneously and empty-handed. In this scenario, the chunk selection strategies and other auxiliary algorithms turned out to have much more significant impact than in the steady-state scenario. Purely random chunk selection policy easily results in at least one chunk becoming rare in the system, but the situation can be improved without need of centralized algorithms.

A detail which is trivial in models and simulations but highly non-trivial in real implementations is the selection of uniformly random peers. One of the main contributions of this paper is the very significant effect of the Random Forward Address algorithm which was proposed but not yet implemented in [5].

The remainder of this paper is organized as follows. Section 2 gives a brief overview of related work. The design and the architecture of our file distribution system which is based on random encounters is described in Section 3. A general description of the emulation environment as well as a detailed look at the Würzburg Chord prototype is given in Section 4. Sections 5 and 6 summarize the results obtained from the practical evaluation of the two prototypes, regarding different aspects of the system. Section 7 concludes the paper.

2 Related Work

Fundamental performance limits of file sharing were obtained in [7, 8]. The performance of distributed file sharing systems has been studied both with simulations and with mathematical models. Since the large number of parameters and complicated algorithms make accurate mathematical models intractable, articles such as [4], [9] and [10] modelled the systems in limiting regimes. Simulation studies such as [11] and [12] allow to consider the effect of various parameters (e.g., the chunk selection policy, the peer selection policy, and heterogenous upload and download bandwidths) on the performance of the system. The phenomenon of one chunk becoming very rare in random chunk selection policy was observed in [11]. Several simulation studies have shown that the *rarest first* policy outperforms other policies such as *random* selection [11], [12], and hence could be one of the main reasons for the good performance of BitTorrent.

The BitTorrent tracker builds bounded degree random graphs that are well suited for the file dissemination. However, it was shown recently that such graphs can be obtained in a distributed manner, resulting in a 'truly trackerless BitTorrent' [13].

Recently, it was proposed to apply Network Coding [14] to content distribution in order to improve the throughput utilization and to deal with well known problems such

as the last chunk problem. Peers exchange encoded versions of the k initial chunks, whereas k linearly independent blocks suffice to recreate the original file. A summary of the advantages and disadvantages can be found in [15]. Note that our algorithm deals with the simple distribution of chunks and is thus independent of whether network coding is applied at a higher layer or not.

ChunkCast [16] is similar to our approach in that it also relies on a DHT in order to make the distribution of large content more efficient. However, in this case the DHT is used to find sharing peers which are physically close to the downloading peer. The evaluation was performed on a cluster of 40 computers using Modelnet [17] to emulate a wide-area physical topology with realistic latency and bandwidth constraints. The study did not consider the overhead introduced by the DHT layer, even though it is frequently used to store and share information about which peer possesses which chunks.

In this context, PlanetLab [3] is often used to study different aspects of DHT implementations under realistic conditions. The most prominent example is OpenDHT [18], which builds the basis for a variety of different applications. However, in contrast to this work, most evaluations of DHT implementations in testbeds concentrate on the redundant storage of resources. The OceanStore Project [19], e.g., tries to build a highly-available and persistent storage utility based on an infrastructure which is comprised of untrusted and unreliable peers. In this paper, we follow a different approach and utilize the underlying DHT to achieve random lookups. That is, we do not store any resources in the DHT, but solely use the overlay structure as an efficient way to keep the participating peers connected.

3 The File-Sharing System "PAN-NET" Based on Random Encounters

Basic scenario: One peer, called the seeder, has the complete file, chopped into chunks, and stays in the system as long as it wants to distribute the file. All peers, including the seeder, run the same software. The seeder initiates an overlay network, in our implementation a Chord. A peer interested in the file gets from the seeder access to the overlay and thus the possibility to contact other members of the overlay. A new member remains invisible for Chord searches (a "parasite") by delaying its first `stabilize` command until it has downloaded one chunk.

Encounter procedure: Peer A, the initiator, sends to a randomly selected peer B its chunk-bitmap (0 for missing, 1 for present chunks) and an optional list of specially wanted chunks. A transfer rate (interpreted as upper bound) can be specified as well. Peer B answers with its own bitmap and, optionally, its list of "recommended" chunks and its opinion on the maximal transfer rate. Peer A then sends its decision on the chunk to be downloaded, and B starts to upload that chunk. After a succesful download, peer A can either repeat the above procedure to get the next chunk, or select another random peer. The PAN-NET client can run several independent downloading processes in parallel.

Upload restrictions: Even when only one download process is running per peer,

several simultaneously active upload processes are allowed. Their number is however bounded by a system parameter.

Leaving procedure: When a peer has downloaded all chunks of the file, it either leaves the system ("non-altruistic behavior") or changes to a seeder mode for a random time T ("altruistic behavior"). In this paper we study only the non-altruistic case.

Random Forward Address (RFA) procedure: Finding a *uniformly* random peer from an overlay is a non-trivial problem that has not been considered much in literature (see, however, [20]). In Chord, for example, it is not sufficient to issue a `find_successor(`x`)` command with a random argument x, because the nodes are not equidistant on the Chord ring. We propose the following algorithm. Each peer, say A, maintains a storage for one address, say r_A, where it initially stores its own address. When a peer B wants to find a random peer, it issues `find_successor(`x`)` with random x and gets the address of, say, C. Peer B then contacts C with a RFA request and receives the content of r_C, whereas C stores the address of B in r_C to wait for the next request. In this way, the address of peer B is forwarded with the same frequency as B is itself initiating contacts. If all peers have the same downloading speed, the forwarded addresses are distributed almost uniformly.

Although the seeder does not make downloads, it must also issue RFA messages in order to be found. In our implementation, this happens at regular intervals as long as the number of the seeder's upload processes is below the allowed maximum. The interval length is a system parameter. When an altruistic node has collected all chunks, it changes to a seeder mode and sends out RFA messages with the same rule as the original seeder.

Chunk distribution estimation: Each peer can maintain an estimate $(f_1, \ldots, f_p)$ of the relative frequencies of the number of copies of each chunk in the overlay. The initial values are all zeroes. In each encounter, upon reception of a peer's bitmap $(\iota_1, \ldots, \iota_p)$, the estimate is updated as

$$f_i^{(\text{new})} := \gamma f_i^{(\text{old})} + (1-\gamma)\iota_i, \quad i = 1, \ldots, p, \tag{1}$$

where p denotes the number of chunks and γ is a system parameter. We used the value $\gamma = 0.95$ and guess, without having studied this question closer, that the optimal value probably depends somewhat on p but not much on the size of the overlay. The chunk selected for transfer is the seemingly rarest possible, according to the downloader's distribution estimate.

4 Description of the Emulation Environment

To backup our theoretical results we conducted several emulation experiments in PlanetLab [3], a large overlay network testbed composed of many computers which are distributed around the globe. The intentions of our prototype studies in PlanetLab are two-fold. On the one hand, we intend to investigate the functionality of our algorithms and see in how far we can exploit the Chord structure to realize random encounters. On the other hand, we want to study the practicability of using Chord as the basis for

our prototype by investigating the involved overhead as well as the robustness against churn and flash crowd behavior. To that end, we implemented two different prototypes, one at VTT and one at the University of Würzburg. Both prototypes are based on the same architecture described in Section 4.2 but investigate the different aspects mentioned above.

4.1 Lessons Learned form PlanetLab

At the time of our studies PlanetLab consisted of 718 machines, hosted by 345 sites, spanning over 25 countries. However, we were only able to disseminate our prototype to 451 of the 718 machines. During the following emulations, we furthermore had permanent problems to deploy the necessary configuration files to about 80 hosts which were frequently unreachable due to downtimes and other issues. This left us with a total of 370 successfully initialized nodes in our slice.

After requesting all peers of the slice to register for the next emulation run, the central administration entity was usually contacted by about 300 to 315 peers. Further emulations revealed that many peers in PlanetLab are unable to reach each other. Simple ping messages in a fully meshed overlay showed that 30 peers could not be contacted by 75 percent of the peers in our slice. Another 30 peers had a very bad connection quality, DNS problems, a very high load, varying SSH-keys, or simply not enough disk space to write log files.

Considering the above, we concluded that 250 peers is the maximum number of peers which can be used in our emulations. However, to be able to produce comparable emulation runs and achieve reproducible results, we limited the emulation environment to the 150 peers with the best performance. This number is also backed up by the monitoring statistics for PlanetLab from the CoMon project [21]. The CoMon project provides a node-centric view of all nodes in PlanetLab which can be queried based on user-provided criteria. During the times of our emulations about 475 peers were online on average from which about 275 had more than 50 slices running and about 115 produced a 5 minute system load average of more than 10. If we also consider nodes which have permanent DNS problems, a current CPU utilization of over 99 percent, or not enough disk space, this adds up to a total of about 330 unusable peers. In fact, a query for usable nodes running less than 5 slices and having a system average load of less than 5 returned approximately 160 peers which agrees with our findings.

4.2 Architecture of the Prototypes

The original description of Chord [2] gives a good theoretical overview of the system but does not suffice to maintain the stability of the overlay, nor the redundancy of the stored resources under churn [22]. Based on our experience with Chord, we applied appropriate modifications which are able to cope with the kind of user behavior expected for our application. Since our algorithm does not need any documents, our major modification is to omit anything related to replication or redundancy. Other modifications concern the join, stabilize, and lookup routines.

Join: Each time a peer joins the overlay it needs to obtain a list of its current overlay neighbors and additionally has to be introduced to them. In our implementation a joining peer p contacts a random peer in the overlay which in turn initiates a search for the direct successor s of p. Peer s answers this query by directly sending a stabilize message to peer p and additionally notifies all appropriate neighbors about the arrival of peer s. To initialize its routing table, peer p downloads the current finger list from one of its new overlay neighbors.

Stabilize: The stability of the overlay network heavily depends on the correctness of the pointers to the overlay neighbors [23]. To keep the neighbor lists up-to-date, each peer periodically exchanges neighbor lists with its direct successor and predecessor every t_{stab} seconds. In order to improve the robustness of the overlay, we use symmetric neighbor lists instead of maintaining just one single predecessor. In addition to its periodic stabilize messages, a peer immediately shares newly gained information about the status of other peers [24]. That is, each time a peer detects an offline node (*notify down*) or gets to know a new peer (*notify up*) it notifies its neighbors accordingly. The result section will show that the involved overhead is negligible but has a great effect on stability.

A major problem of the original Chord algorithm is that of oscillating offline entries. That is, when a peer detects and removes an offline entry from its list of neighbors, there is still the risk that another peer, which has not yet learned about the offline status of the peer, re-introduces this peer at the next stabilize instant. This way offline entries may indefinitely circulate the neighbor list of the peers. Therefore, each peer maintains a *downlist*, a list of peers which it knows to be offline. Entries of this list will only be removed after three stabilize instants or if a falsely assumed offline entry directly contacts the peer.

Lookup: The original Chord algorithm applies a recursive routing scheme, where a query is passed from one peer to another peer until the final destination is reached. Since the initiator of the search does not get any progress report during the search process, it is hard to set timeouts or pinpoint the exact location where the search failed. As an alternative, routing can be performed iteratively, where each peer involved in the search directly reports back to the initiator of the search. While this avoids the above mentioned problems it significantly slows down the entire search process. Therefore we use a hybrid routing scheme [25], where the query is passed recursively from peer to peer but each peer reports back to the initiator of the search instead of to the peer which relayed the search. This combines the advantages of recursive and iterative routing, while it does not introduce any additional overhead.

Churn: In literature, there are two predominant ways to model churn. The first assumes churn per network by specifying a global join and leave rate [22], whereas the global join process is modeled by a Poisson process with rate λ. One of the main problems of this model is that the number of nodes joining the system within a given time interval is independent of the current size of the system. The second way is to model churn by specifying a distribution for the time a peer spends in the system (*online time*) or outside the system (*offline time*), which makes it comparable in networks of different size. In this paper, we follow the latter approach. To be able to model the offline time

of a peer, we assume a global number of n peers, each of which can either be online or offline. Joins are then modeled by introducing a random variable T_{off} describing the duration of the offline period of a peer. Accordingly, leaves are modeled by a random variable T_{on} describing the online time of a peer. To remain comparable to other studies, T_{on} and T_{off} are exponentially distributed with mean $E[T_{on}]$ and $E[T_{off}]$, respectively.

Administration of the Emulation Runs: Finally, to conduct experiments in PlanetLab one has to carefully organize and manage the individual emulation runs. Our emulation environment is based on the Nixes tool set [26] which provides a set of scripts to install, maintain, control, and monitor remote applications via SSH. In this context, we implemented the *Chord-Admin* tool which resides on a central entity as shown in Figure 1. It is responsible for the organization and administration of the individual

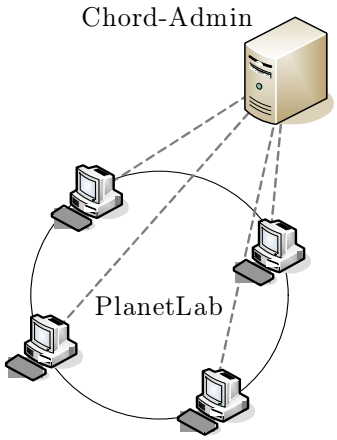

Figure 1: Basic architecture of the testbed environment

experiments, which were conducted as follows:

1. At first we installed the prototypes on all 451 machines of our PlanetLab slice.

2. Once every day we determined a list of the 200 best peers. This was done by sending a simple ping message from each peer to all other peers of the slice. We then chose those peers which could be reached by the most other peers and had the smallest 5 minute average load.

3. At the beginning of each experiment, the Chord-Admin sends a request to all of the 200 peers asking them to participate in this particular emulation run.

4. The invited peers register with the Chord-Admin.

5. Once there are enough peers to conduct the experiment, the Chord-Admin distributes the individual configuration files to the corresponding peers. Each configuration file contains information about how long the emulation is running and when a peer should join or leave the overlay.

6. After having distributed the configuration files, the Chord-Admin sends a starting signal to all participating peers. This way the accuracy of the time synchronization is roughly in the order of one overlay hop, which is sufficient for the kind of statistics we are interested in.

7. In the final step the Chord-Admin automatically collects all log files from the participating peers and prepares them for evaluation.

A typical disadvantage of distributed prototype studies is the lack of a global view of the system. We therefore implemented two different possibilities to generate a global snapshot. In the first approach, all participating peers write their current state to a log file at the predetermined times. However, due to high load and other problems on some of the PlanetLab machines it is difficult to verify if all log files were written at approximately the same time. Therefore in the second approach the Chord-Admin queries all participating peers in parallel at the given time in order to generate a global snapshot. While this approach does clearly not scale, it sufficed for the size of our emulations. A practical algorithm to generate snapshots of larger scale overlays can be found in [27].

5 Evaluation of the Practicability of Chord as the PAN-NET overlay

At first we want to investigate whether it is practicable to use Chord as the basis for our file-sharing algorithm. Therefore we have a closer look at the overhead created by the Chord layer and verify if our modified overlay mechanisms are able to handle the expected user behavior.

In our first scenario, we let x peers join the Chord ring, wait until the overlay is stable and then let the individual peers perform searches for random identifiers. This resembles the search of a downloading peer for a random peer from which it will download the next chunk. Figure 2 shows the mean search time (c.f. left y-axis) and the percentage of

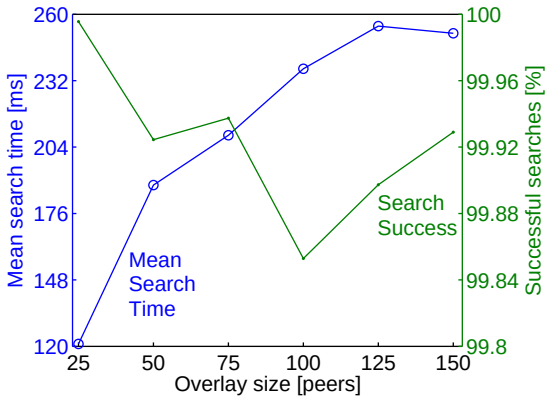

Figure 2: Average search time and percentage of successful searches

successful searches (c.f. right y-axis) for overlay sizes between 25 and 150 peers. In agreement with theoretical results [28], the mean search time increases approximately logarithmically with the size of the overlay. The absolute values for the mean search time are significantly less than one second, which indicates that the search time is negligible in comparison to the expected time it takes to download an entire chunk. Furthermore, while in a stable overlay network all searches should be successful, we experienced some failed searches. This reflects the background churn in PlanetLab, which arises due to high loads, short downtimes, or missing routes between two hosts.

To study the general behavior of the prototype under churn, we configured 100 peers to have exponentially distributed online and offlime times with $E[T_{on}] = E[T_{off}]$ in order to keep the mean number of online peers stable. Figure 3 shows the overhead

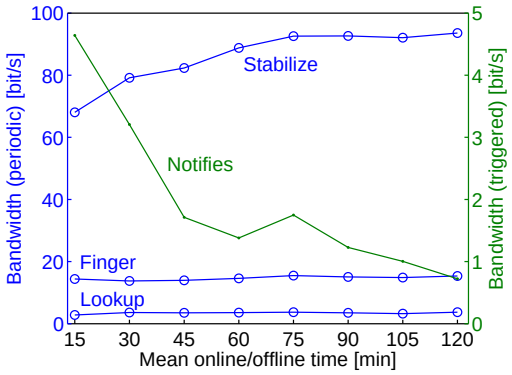

Figure 3: Consumed bandwidth in dependence of the mean online time.

generated by the different functions of the Chord layer in dependence of the mean online time. The left y-axis shows the mean bandwidth generated by lookups, finger updates, and the periodic stabilize calls. In our implementation we update one finger every 30 seconds, which makes the generated overhead independent of both the current size of the system and the current churn rate. The main part of the overhead traffic is used for the periodic stabilization of the overlay structure. In general it can be said that the overhead traffic required to maintain the overlay is negligible compared to the traffic which will be generated while downloading chunks. Finally, the right y-axis shows the bandwidth required by the triggered notify down/up messages. The additional overhead introduced by this modification is negligible compared to the periodic stabilize traffic. It furthermore shows a self-organizing behavior as the amount of overhead adapts to the current churn rate in the system.

To analyze the impact of a flash crowd behavior, we let x peers join the network simultaneously and study the time it takes until all successor and predecessor pointers are accurate. Figure 4 shows the mean time which passed before steady state was reached for the successor and predecessor pointers and different flash crowd sizes. The results represent the mean over 10 emulation runs, whereas the error bars show the 95 percent confidence interval. It takes slightly longer to stabilize the successors than the

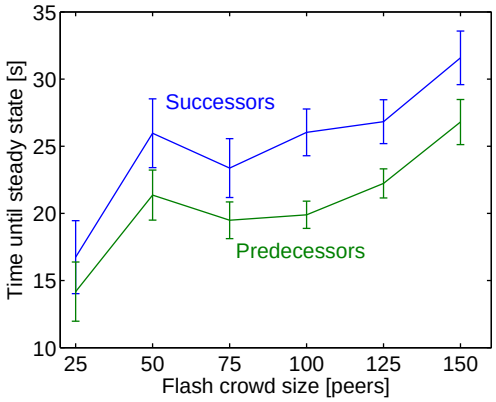

Figure 4: Time until the overlay is stable in the flash crowd scenario.

predecessors as searches in a Chord ring are performed in a clockwise direction. Thus predecessors tend to contact their successors more often than vice versa.

A more problematic aspect of churn are unfriendly leaves. To study the correctness of neighbor pointers in such scenarios, we look at mass exits which might e.g. happen at the end of a work day. Figure 5 looks at the mean percentage of offline entries

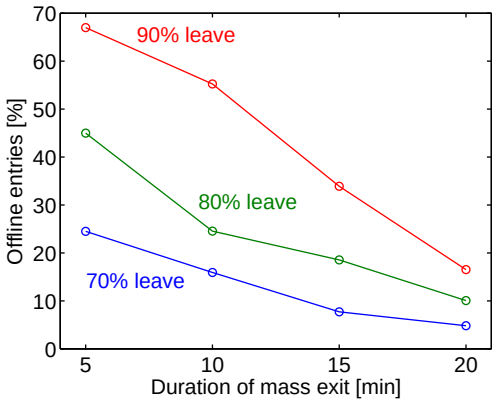

Figure 5: Percentage of offline entries after a mass exit.

in a peers neighbor list after a certain percentage of all peers left the overlay within x minutes. In general, the longer the duration of the mass exit, the more time there is for stabilization and the less offline entries arise. While almost 70 percent of all neighbor pointers are invalid if 90 percent of all peers leave within 5 minutes, the overlay will still be connected. Thus, similar to the previous scenario, it will take longer to search for other peers (which might not necessarily be random), but it will still be possible to find file sharing partners.

6 Evaluation of the Performance of the Peer and Chunk Selection Algorithms

A number of test runs of the PAN-NET client software was performed in PlanetLab with the following scenario:

File size	51 MB
Chunk size	512 kB
Number of chunks	102
Number of nodes	50-52
Download bitrate	30 kB/s
Nominal chunk download time	17 sec
Node activation interval	16 sec
Max # paraller uploads	3

In order to avoid complications possible in a flash crowd scenario of Chord (such problems were in the focus of the previous section), the nodes were activated one in turn with intervals of 16 seconds. Thus, the last activation happened about $50 \times 16 = 800$ seconds from the start of the test.

Results on the following three algorithm combinations are reported in this paper:

test IDs	RFA	distribution estimation
27, 28, 29	NO	YES
48, 49, 56	YES	YES
61, 62, 63	YES	NO

The following table presents summarizing characteristics of the test results:

test	fs_resp	tot_av	upl_lim	Cho_err
27	2.88	3686	1210	169
28	2.11	3977	1273	188
29	2.28	3876	1306	171
48	0.98	3264	162	204
49	0.93	3286	185	263
56	0.57	2794	240	182
61	0.64	2971	212	187
62	0.71	2846	205	169
63	0.56	2827	205	144

The entries are the mean `find_successor` response time (fs_resp), mean download time of the whole file (tot_av), the number of download requests rejected because of the parallel upload limit (upl_lim), and the number of Chord error messages (Cho_err) (times in seconds). These numbers do not yet tell much about the performance of the algorithms we are interested in. The differences in `find_successor` response times may

be due to small improvements made into the software or simply to load variations in PlanetLab.

However, the nodes also logged all their chunk transfers and other events of interest. Since the PlanetLab hosts have reasonably well synchronized clocks, the logged events could be easily combined to produce pictures of the growths of (i) the number of copies of each chunk, (ii) the number of chunks in possession of each peer, and (iii) the number of uploads performed by each peer. These plots provided interesting qualitative insight into each of the three algorithm combinations. On the other hand, the three runs with each combination produced quite similar pictures, except for the behavior of some individual hosts probably suffering from PlanetLab overload.

Let us start with the case that both RFA and distribution estimation were applied. Fig. 6 shows that it took about 1600 seconds before all chunks were copied from the seeder at least once. After a chunk has been uploaded from the seeder, the number of its copies grows very fast, until it reaches an average level. After 2000 seconds, the distribution of chunk copies is roughly uniform. Fig. 7 shows that the nodes collect chunks very regularly, except for one node which lags behind in the last phase. The upload picture Fig. 8 reveals that in the "end game" phase only the seeder performs uploads. The seeder is more active in uploads than the other nodes because it tries to maintain a maximal number of parallel uploads, in our case three (see Section 3). The upload processes are very regular, thanks to the RFA.

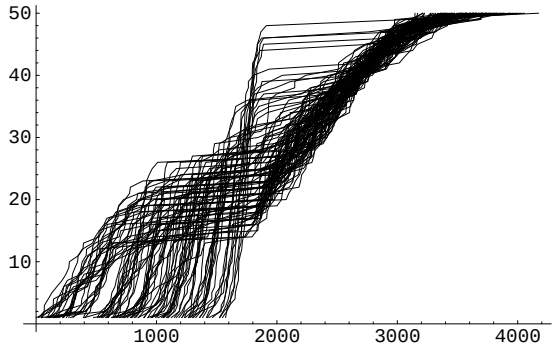

Figure 6: Growth of chunk copies, with RFA and distribution estimation. (Test number 56.)

Let us now look how the performance changes when RFA is switched off, and the peers encounter each other directly according to `find_successor(`*random*`)` commands. The difference of the upload curves in Fig. 9 and those of the previous picture is dramatic. Without RFA, the upload curves have different slopes, depending on their ID's distance from their predecessor's ID. Interestingly, even the distribution estimation (1) seems to lose its effect, as indicated by Fig. 10. Since the seeder does not announce itself as it does with RFA, the peers may have difficulties in finding it, if its Chord-position happens to be unlucky. Moreover, we observe a rare chunk phenomenon "in nature".

Our third combination focuses on the role of the distribution estimation. The difference between Fig. 11 and the earlier Fig. 6 is very clear: the selection of the rarest

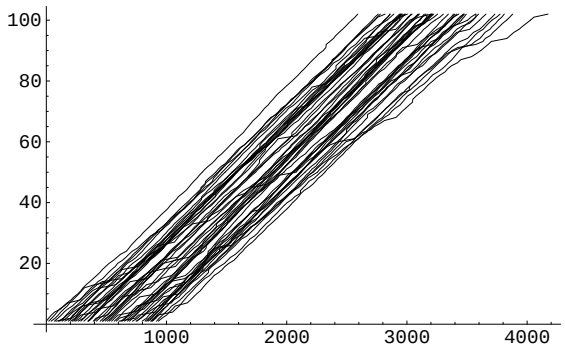

Figure 7: The nodes' download processes, with RFA and distribution estimation. (Test number 56.)

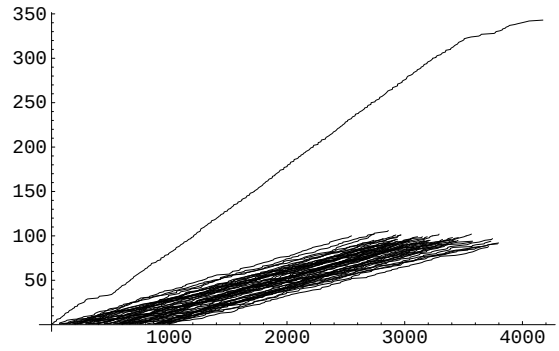

Figure 8: Uploads with RFA and distribution estimation. (Test number 56.)

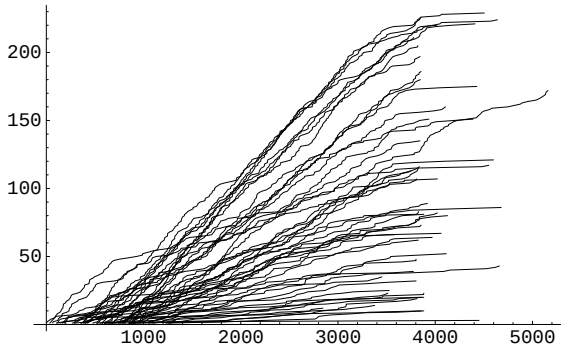

Figure 9: Uploads without RFA but with distribution estimation. (Test number 29.)

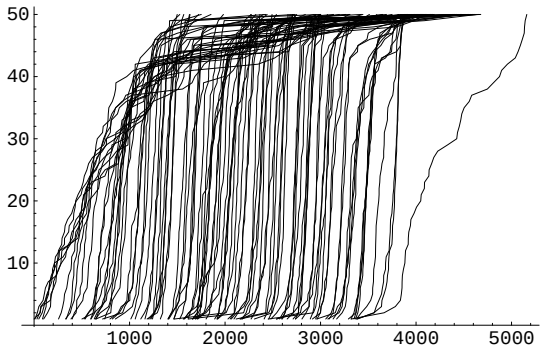

Figure 10: Growth of chunk copies, without RFA but with distribution estimation. (Test number 29.)

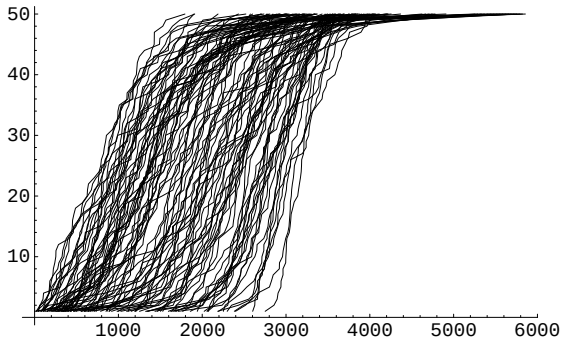

Figure 11: Growth of chunk copies with RFA, without distribution estimation. (Test number 61.)

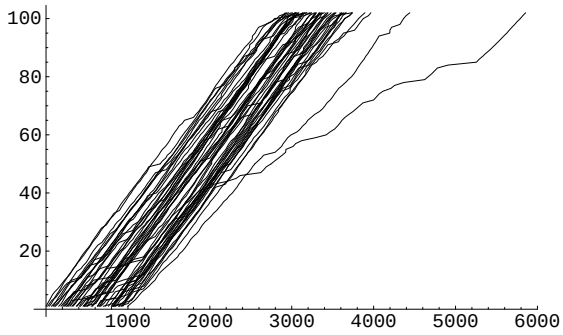

Figure 12: The nodes' download processes, with RFA, without distribution estimation. (Test number 61.)

chunk on the basis of distribution estimation causes a much faster run-out of all chunks from the seeder and a more balanced chunk distribution after that than the random chunk selection. However, the download picture of test 61 (Fig. 12) shows good performance except for two nodes, whose problems are probably caused by PlanetLab-related phenomena rather than our algorithms.

Still more insight into the performance differences in our three scenarios is obtained by considering the amount of unsuccessful encounters, where the encountered peer does not possess any chunks that the initiator of the encounter does not already have. Fig. 13 shows the cumulative number of unsuccessful encounters in each of the nine tests. The numbers differ in orders of magnitude so that we had to use a logarithmic plot. On the other hand, curves belonging to same scenario are very similar. Note also that when both RFA and distribution estimation are applied, there are extremely few unsuccessful encounters after all chunks have been copied at least once, whereas the scenario without estimation has another burst of them in the end game phase.

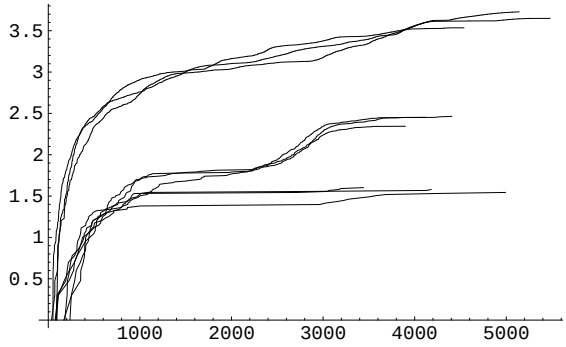

Figure 13: Cumulative number of unsuccessful encounters in each test. Logarithmic plot with base 10. Top three: 27, 28, 29. Middle three: 61, 62, 63. Bottom three: 48, 49, 56.

7 Conclusion

Although our tests on the PAN-NET client were not extensive and rather had a preliminary character, the results were encouraging. Indeed, it seems to be possible to obtain good performance in BitTorrent-like file-sharing without the use of any tracker functionality, just relying on randomness. Our Random Forward Addressing algorithm (which increases the uniformity of the random encounters) and distribution estimation algorithms (which provides better-than-random chunk selection) turned out to have significant positive effect on the performance.

Our studies focusing on the practical functioning of Chord showed that Chord applies well to the role of a file-sharing overlay. It is, however, important to improve the stability of Chord with certain additions to the standard protocol.

In a very big flash-crowd scenario, the bootstrap server (in our case identical to the seeder) providing access to the Chord could become a bottleneck for building the overlay.

Some random entry point mechanism could be the solution.

We plan to continue performance tests with different scenarios, e.g. with heterogeneous transfer rates per peer. Another important dimension, security issues, have so far been neglected in this study, but they will require serious consideration before a concept like ours could be recommended for open use.

Acknowledgements

The authors would like to thank Prof. Phuoc Tran-Gia for the help and the insightful discussions during the course of this work.

References

[1] B. Cohen, "BitTorrent specification," 2006. `http://www.bittorrent.org`.

[2] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *ACM SIGCOMM 2001*, (San Diego, CA), August 2001.

[3] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay Testbed for Broad-Coverage Services," *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 00–00, July 2003.

[4] L. Massoulie and M. Vojnovic, "Coupon Replication Systems," in *Proc. ACM SIGMETRICS*, (Banff, Canada), 2005.

[5] I. Norros, B. Prabhu, and H. Reittu, "Flash crowd in a file sharing system based on random encounters," in *Inter-Perf*, (Pisa, Italy), Oct. 2006. `http://www.inter-perf.org`.

[6] I. Norros, B. Prabhu, and H. Reittu, "On uncoordinated file distribution with non-altruistic downloaders," 2007. Submitted for publication.

[7] J. Mundinger, R. Weber, and G. Weiss, "Analysis of peer-to-peer file dissemination," *To appear in Performance Evaluation Review, Special Issue on MAMA 2006*, 2006.

[8] C.-H. Kwon and K.-Y. Chwa, "Multiple message broadcasting in communication networks," *Networks*, vol. 26, pp. 253–261, 1995.

[9] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks," in *Proc. ACM Sigcomm*, (Portland, OR), 2004.

[10] F. Clévenot-Perronnin, P. Nain, and K. W. Ross, "Multiclass P2P Networks: Static Resource Allocation for Service Differentiation and Bandwidth Diversity," *Performance Evaluation*, vol. 62, pp. 32–49, October 2005.

[11] P. Felber and E. Biersack, "Cooperative Content Distribution: Scalability through Self-Organization," in *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations* (O. B. *et al*, ed.), vol. 3460 of *Lecture Notes in Computer Science*, Springer Berlin, 2005.

[12] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Network's Performance Mechanisms," in *Proc. IEEE INFOCOM*, (Barcelona), 2006.

[13] C. Fry and M. Reiter, "Really truly trackerless BitTorrent," tech. rep., School of Computer Science, Carnegie Mellon University, 2006. `http://reports-archive.adm.cs.cmu.edu/anon/2006/CMU-CS-06-148.pdf`.

[14] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *IEEE Infocom 2005*, (Miami, USA), March 2005.

[15] R. Rodrigues and B. Liskov, "High availability in dhts: Erasure coding vs. replication," in *4th International Workshop on Peer-to-Peer Systems (IPTPS'05)*, LNCS, Springer, February 2005.

[16] B.-G. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz, "Chunkcast: An anycast service for large content distribution," in *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS06)*, (Santa Barbara, CA), February 2006.

[17] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 271–284, 2002.

[18] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: A public dht service and its uses," in *SIGCOMM*, August 2005.

[19] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGPLAN Not.*, vol. 35, no. 11, pp. 190–201, 2000.

[20] V. Vishnumurthy and P. Francis, "On Heterogeneous Overlay Construction and Random Node Selection in P2P and Overlay Networks," in *Proc. of IEEE INFOCOM*, (Barcelona), 2006.

[21] The CoMon Project, "http://comon.cs.princeton.edu/."

[22] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *2004 USENIX Annual Technical Conference*, (Boston, MA), June 2004.

[23] A. Binzenhöfer, D. Staehle, and R. Henjes, "On the stability of chord-based p2p systems," in *GLOBECOM 2005*, (St. Louis, MO, USA), p. 5, November 2005.

[24] G. Kunzmann, A. Binzenhöfer, and R. Henjes, "Analyzing and modifying chords stabilization algorithm to handle high churn rates," in *MICC & ICON 2005*, (Kuala Lumpur, Malaysia), November 2005.

[25] G. Kunzmann, "Iterative or recursive routing? hybrid!," in *GI/ITG-Workshop Peer-to-Peer-Systeme & Anwendungen within the scope of KiVS 2005*, (Technische Universität Kaiserslautern, Germany), March 2005.

[26] The Nixes Tool Set, "http://www.aqualab.cs.northwestern.edu/nixes.html."

[27] A. Binzenhöfer, G. Kunzmann, and R. Henjes, "A scalable algorithm to monitor chord-based p2p systems at runtime," in *Third International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P) in conjunction with the IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006)*, (Rhodes Island, Greece), April 2006.

[28] A. Binzenhöfer and P. Tran-Gia, "Delay Analysis of a Chord-based Peer-to-Peer File-Sharing System," in *ATNAC 2004*, (Sydney, Australia), December 2004.