Fast Freenet: Improving Freenet Performance by Preferential Partition Routing and File Mesh Propagation

Hans-Emil Skogh, Jonas Haeggström, Ali Ghodsi and Rassul Ayani
KTH/Royal Institute of Technology
School of ICT
Email: {hansemil, jonasha, aligh}@kth.se

Abstract

The Freenet Peer-to-Peer network is doing a good job in providing anonymity to the users. But the performance of the network in terms of download speed and request hit ratio is not that good.

We propose two modifications to Freenet in order to improve the download speed and request hit ratio for all participants. To improve download speed we propose Preferential Partition Routing, where nodes are grouped according to bandwidth and slow nodes are discriminated when routing. For improvements in request hit ratio we propose File Mesh propagation where each node sends fuzzy information about what documents it posesses to its neigbors.

To verify our proposals we simulate the Freenet network and the bandwidth restrictions present between nodes as well as using observed distributions for user actions to show how it affects the network.

Our results show an improvement of the request hit ratio by over 30 times and an increase of the average download speed with six times, compared to regular Freenet routing.

1 Introduction

Freenet is a fully decentralized anonymous P2P (Peer-to-Peer)-network which provides freedom of speech. Even though Freenet manages to provide anonymity for all its users, it suffers from a poor ability to locate requested files (low request to hit ratio) and low download speed for found files.

Much research has been done on structured P2P-systems, such a Chord, Pastry, Kademlia, which provide high guarantees in terms of lookup length, data availability, and low maintenance cost. We believe, however, that unstructured P2P-networks are desirable in terms of anonymity, as the inherent randomness leads to less information leakage. Therefore, we believe unstructured P2P-networks to be viable when building large scale anonymous systems, given that it can provide adequate service and performance.

Freenet, nonetheless, suffers from grave performance problems, impairing its anonymity advantages. In this paper, our main motivation is to improve the performance of Freenet. Specifically download speed and request to hit ratio will be improved, as they are two important factors in a P2P-network from a user perspective. We substantially improve on both factors.

The Freenet network is simulated with network bandwidth restrictions and we observe the effects of the amendments proposed.

We will not discuss the affects of churn as this would add considerable complexity and make the results difficult to compare with other work done in the area. We will also not focus on the anonymity aspects of these changes. Even though we believe the methods are coherent with Freenets design objectives, more in depth studies are needed to make assumptions about their impact on anonymity in Freenet.

The rest of the paper is structured as follows. In Section 2, we describe how Freenet works. Section 3 details the two problems in Freenet that we want to solve. Section 4 contains our proposed solution to the problems. In Section 5 we describe how we intend use simulations to verify our proposal. Section 6 presents and discusses the results from our simulations in depth. Section 7 summarizes related work and Section 8 briefly discusses possible future research. Finally Section 9 wraps it up with conclusions and a brief summary of the results.

2 A brief review of Freenet

Freenet[8] was initially designed by Ian Clarke as a system for storing and retrieving data with emphasis on providing anonymity and deniability for all participants. To

achieve these goals the system was designed as a P2P-network utilizing encryption and data forwarding. Freenet does not provide any guarantees on persistent storage, nor does it provide any search functionality.

Because of Freenet's way of routing, where requests and answers are passed from node to node (proxying), a user in Freenet can request and insert documents without the risk of being identified. Each user runs a Freenet node which contributes to the network by providing data storage for the network (a "datastore") and Freenet routing capability.

There is a subtle difference between making documents publicly available in Freenet and other popular file sharing networks. Instead of the documents that the node will be sharing, one must *insert* the documents into the network. Once the documents are in the network there is no way to know who will be storing them, and therefore no way to remove them. A user does not know what documents its node is storing because of encryption, thus cannot affect what documents it is hosting. The user only decides how much space to dedicate to the datastore, and the Freenet node decides what documents to save and when to delete them.

2.1 Under the hood

Each Freenet node provides two basic services for the network: routing and data storage. Routing is done by maintaining routing tables, associating with each node a set of file keys. The data is stored in the datastore together a corresponding file key.

Each node has a routing table in which known routes to file keys are stored. Every time a node is involved in a successful request or insert it becomes aware of the node that requested or inserted the document. The IP (Internet Protocol) address and port to that node is stored in the routing table together with the file key. The size of the routing table is limited. If the table is full and a new route is to be added, the least-recently-used (LRU) item in the table is removed in favor of the new one. By handling the routing table in this manner, popular routes stay in the table while obsolete and invalid routes are replaced.

As with the routing table, the datastore implements a LRU caching scheme, limited in the size of all stored documents. Hence, many small documents might have to be removed to make room for a single large document if the datastore is filled. Again, the LRU-scheme helps to keep more popular documents in the network longer than less popular documents, and because of Freenet's caching mechanism the popular documents will be more widely spread in the network. All documents in the datastore are encrypted individually and the node storing them does not have the descriptive string needed to decrypt them. It does, however, have a matching file key for each document in the datastore, enabling it to identify all its stored documents during routing.

2.2 Routing queries and insertions

Routing in Freenet is handled by recursively sending request and insert messages between the nodes. In order to stop the network from being overloaded by requests for documents that cannot be found, the recursion must be terminated at some point and be interpreted as a failure. This is solved by appending a HTL-counter[1]

In order to get a document one has to obtain the file key of that document by out of band means, as Freenet offers no search functionality. This key is transmitted to a Freenet node, which starts the Freenet query routing algorithm. The node first checks if a document with the corresponding key is located in its own datastore. If it is not found in the datastore, the node calculates and compares the distance (binary closeness) between this key to all the other keys in the routing table and passes the request on to the node that has a key with the closest match. If the query is returned and the requested document has not been found, the node will choose the second closest match, and so on, until either the document is found or the HTL (Hops To Live) of the query reaches zero. Every contacted node then repeats the procedure. As each query is assigned a pseudo-unique identifier, nodes will reject queries that they have seen before. If the document is found in the local store, it will be passed back along all the nodes in the request chain until it reaches the requesting node. All nodes that the document passes will add the originator of the document to their routing table and also cache the document to speed up possible future requests. Each node on the chain may change the ownership of the document to itself or any other node in order to confuse potential malicious nodes. This does, however, mean that if only one of the nodes on the request chain is under heavy load, or has very limited bandwidth, it will be a bottleneck for the entire transfer chain.

When inserting a document the network will be probed for key collisions. This is done in exactly the same way as a document request (including the recursive caching if a document is found), except that the path is "remembered" for this insert query. If no key collision is found, the new document will be propagated and stored along the pre-routed path. If a node is out of storage space it will make space as described in Section 2.1. By doing a search to determine the insertion path, documents will be stored close to documents with similar hashes which will speed up future searches. As with requests, any node on the chain may confuse potential malicious nodes by claiming that it (or any other node) is the inserter of the document.

[1]The HTL in Freenet is equivalent to the more familiar time-to-live counter, to all request (and insert) messages.

3 The Freenet performance problem

We have identified two main areas of concern when it comes to the performance of Freenet. First, a high probability of not finding existing documents, i.e. Freenet suffers from a low hit-ratio. Second, low download speeds for found documents.

Freenet utilizes a non-deterministic request routing that, by design, may fail to satisfy some requests to existing documents in the network. Even though this is by design there is an obvious problem that users will start to mistrust the network if the ability to find existing documents is not good enough. A request failure in Freenet will leave the user unsure of whether the document ever existed, has existed but vanished, or does exist but simply could not be found. Improving the mechanism that locates documents should be a top priority in optimizing Freenet from a user perspective.

Freenet achieves a good part of its anonymity by proxying requests and transfers, and because of that it will be quite susceptible to poor download performance due to limited node bandwidth. Nodes acting as proxies for many other nodes will fill their bandwidth with the numerous transfers and thereby act as bottlenecks in all transfers that they are participating in. Therefore, improving download speeds should be a top priority in optimizing Freenet as well.

4 Our proposal

We are proposing two different solutions to remedy the two identified problems from a user perspective in Freenet.

4.1 The file mesh

As described in section 2.1, Freenet nodes find out about the existence of a document by serendipity when they are involved in a successful document request or insertion. Hence, a node's knowledge about the documents in the system is restricted to the requests and insertions that it has been involved in. Using this specific, but incomplete information about each neighbor, the routing mechanism tries to make the best decision when routing a request.

We suggest improving the request hit ratio in Freenet by making use of a *File Mesh*. Intuitively, the File Mesh compactly represents the set of all files a node has. This data structure is similar to a Bloom filter[5] in that it can give false positives, i.e. believing a node has a file when it has not. A Bloom filter can, however, not be used for our purposes since it does not provide the ability to find if a node has key *similar* to another key. This ability is important, as Freenet tries to insert documents to the nodes with keys similar to the document's key.

Technically the file mesh consists of a bit field of the length N. Each bit in the field represents a $1/N$-sized part of the key space. If the part contains any files, the bit will be set to 1, else it will be set to 0. This gives us the benefit to have some information about the state of the whole datastore of the node, but with a fraction of the space needed to transmit a complete list of keys. This approach also has anonymity benefits, as you can not say anything about the presence of a specific file, you can only say with certainty if it is missing. If searching for a document, whose respective bit is set to 1 in the File Mesh, a node knows that that node at least has some similar documents.

With our proposed solution the nodes pass a file mesh in addition to the document source. Hence, a node will have approximate knowledge about all documents its routing neighbors are storing.

A challenge in the file mesh solution is to keep the meshes on each node fairly up to date without excessive data transfer overhead. To solve this we will employ a form of update difference compression in combination with allowing a certain staleness of the meshes. Each time a node sends a query response to another node, it will check if it has sent a file mesh to it before. If no mesh has been sent, the complete mesh is piggy-backed on the response message and a copy of the mesh is saved and tagged as the last mesh sent to that node. If a mesh has been sent, the node checks if the mesh that has been sent earlier differs from the current mesh. If it differs it sends information about what positions in the file mesh that have been changed, and updates the local copy of the mesh for that node to the state of the current mesh. In this way, only a minimal amount of extra bandwidth will be used and *no* extra messages. The nodes will also get recent file meshes in a way that will guarantee them not to use old and possibly outdated routing information for a node more than once.

4.2 Preferential partition routing

Since documents are routed through multiple nodes in Freenet before they reach their destinations these transfer chains are obviously not stronger than their weakest links. The ability to distinguish faster nodes from slower is a simple method to avoid the obvious potential bottlenecks in order to speed up transfers. Our proposal to solve the bottleneck-problem is called *Preferential Partition Routing*, and we will refer to it as partitioning or partitionizing.

We suggest that all nodes should be divided into a number of different partitions (groups), where each partition contains nodes with similar bandwidth restrictions. During the requesting and inserting phase, the nodes have the possibility to use the information the partitions give to optimize their routing by avoiding slow nodes.

One obvious problem with this approach is to determine

what partition a node belongs to. Our suggestion is that each user will set his own partition (by specifying the available bandwidth) and then the node will know what partition it is in. In order to make the users set the correct bandwidth one must assure that the user will not benefit from cheating (or simply making the wrong setting by mistake), as P2P-users are prone to misreport their real bandwidth if there is something to gain[11].

The two cases are that the node might become member of a higher or lower partition than it really belongs to. A node that becomes a member of a higher partition than intended will be punished by design. Fast nodes will choose it in their routing and it will become overloaded, not getting much bandwidth for its own queries. A user might want to become a member of a lower partition than it should be in order to avoid receiving requests by fast nodes, but still send all of its own requests to fast nodes. This can be avoided by limiting transfer speeds to nodes that claim to be in a slow partition.

When nodes try to determine which neighbors to send a query to, they will avoid choosing nodes that are participants in a lower partition. This is done by modifying the distance each node is assigned in the routing process by making sure that all nodes in a faster partition have a shorter distance than all nodes in a slower partition, while maintaining the mutual order within all partitions. By discriminating the lower partitions the chance of getting a low bandwidth node in the transfer chain is minimized. The sacrifice of the optimum path length routing gives the benefit of trying to ensure a high bandwidth "download path" for the document. This does not guarantee that all nodes in the transfer chain will be high speed nodes since a node will start to query slower nodes when all fast nodes it knows have rejected the query. One obvious downside to this approach is the potential "over shot" that is created when a large number of fast nodes has to be visited before even trying the slower ones, no matter if they are likely to have the file or not. This is possibly remedied by the fact that inserts will follow the same pattern and popular files should be prone to end up on a fast node.

5 Simulations

To test and evaluate our proposed changes ot Freenet we decided to simulate them.

We evaluated the two "official" Freenet simulators (Aurora[2] and Serapis[3]) as well as the simulator developed by Zhang, Goel and Govindan for their paper [12]. We also considered the generic P2P-simulator p2psim[1]. As none of these simulators offered a satisfying method for bandwidth simulation we decided to write our own simulator, named Eos.

Though we strived to model the Freenet network as closely as possible, we made some simplifications to make it possible to finish the simulations within our timeframe. No actual encryption is done anywhere since it does not affect the routing in any way (except contributes to CPU load). No files are actually transferred, instead the simulator keeps track of how much data would have been sent in each transfer. We decided to simulate an abstract network level above the packet level. Requests are instantaneous, atomic and non-concurrent. Document transfers are concurrent and duration will depend on available network resources that the transfer can consume during its lifetime. We are not handling joining or leaving/failing nodes during the duration of the simulation.

The simulator is time step based and parameters such as how often nodes should request or insert files are set per time step. The time span of a single time step can be varied and we have used 10 seconds per time step in our simulations.

To accurately simulate node behavior, we studied several recently published papers on P2P measurements. The results in these papers were incorporated in the simulator as aproximated functions for node up- and downstream bandwidth[2], popularity of any given file[3], size of any given file[4] and relation between inserts and requests in Freenet[5].

We have chosen not to simulate network transfers at packet level, but to see all transfers as flows. This will allow the model to scale with an increasing number of nodes and transfers. The model we have used for all our simulation results is named *minimum share allocation* and is proposed by Ahn and Danzig in [4] as a part of the Narses network simulator. We also implemented the Naïve network model used in Aurora[2] to validate Eos.

We started the network creation in each simulation with a set of nodes connected in a bi-directional ring, to ensure that all nodes are part of the network. To speed up the initial convergence, we also added two more neighbors to each node by connecting the nodes in two rings, where each node has a random position in each ring. This added possible shortcuts through the network.

The simulator continuously meassures several selected properties of the nodes present in the simulated network. These properties are evaluated and logged on each time step. We present the collected statistics from these logs in the Results section.

6 Results

All simulation data originates from the Eos simulator described in Section 5. All graphs represent an average of 10

[2] See [11] by Saroiu, Gummadi and Gribble in 2002.

[3] See [6] by Chu, Labonte and Levine in 2002.

[4] See our own study of over 500 arbitrarily chosen DirectConnect nodes between March 2003 and October 2004 described in [9].

[5] See our own survey in late 2004 described in [9].

runs using different random seeds.

The simulation duration is set to 5000 time steps. We performed a number of longer test runs to observe how the simulated network evolved over a longer time span and to support our choice of simulation duration. We did not see any significant changes after somewhere around the 5000:th time step, and therefore we have chosen this as simulation duration.

All partition based simulations use two partitions. The partition membership is decided by putting all nodes with lower upload bandwidth than the median in the slow partition and the nodes with higher upload bandwidth in the fast partition.

From here on abbreviations will be used for the different algorithms. We have chosen the name V (Vanilla) for the original Freenet implementation detailed in [8] to avoid confusion with current Freenet implementation. The name symbolizes that it is the originally proposed flavor of Freenet. The name M (Mesh) is our routing optimization. The partition algorithms are used in combination with the other algorithms, where the combinations are named PV (Partitioning Vanilla) and PM (Partitioning Mesh).

6.1 Results at a glance

By logging several properties of the simulation every time step, we got the ability to see how they evolve over time. Figure 6.1 and Figure 6.1 are graphs showing how the values changes with time. Each data point on the curve represents an average of 100 time steps to avoid noisy data and visualize trends more easily.

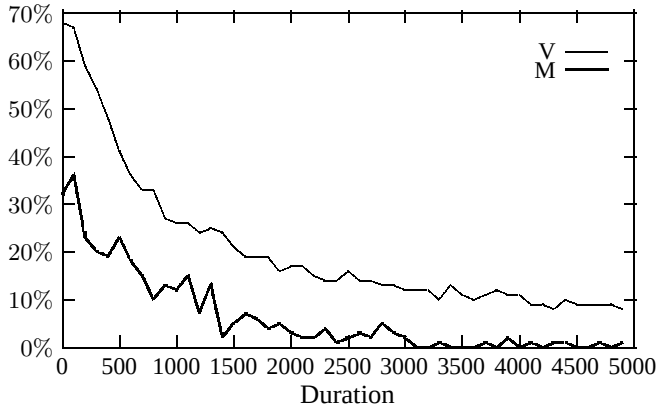

Figure 1. Percent unsuccessful requests over time

By measuring how many of the requests that did not manage to find its document we get the graph in Figure 6.1. As one can tell from the graph the network needs some time to converge before it learns how to route queries. This behavior is unavoidable, but the time needed for the network to learn can be reduced. Both the algorithms have the same trend but our Mesh version does start in a far better position than Vanilla. Mesh also manages to keep the failure rate close to 0% while Vanilla stabilizes with about 10% failures.

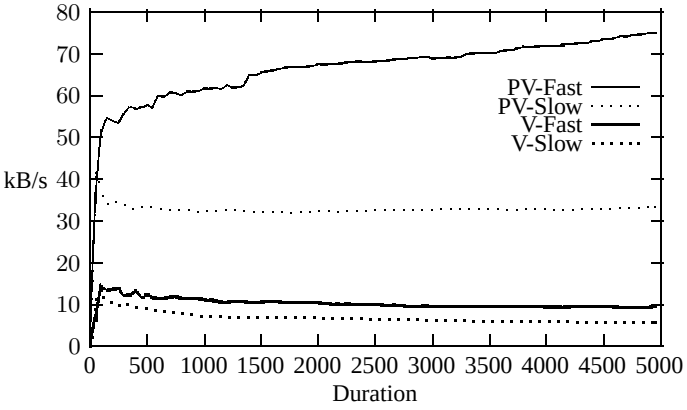

Figure 2. Average download speed over time

Moving on to download speed shown in Figure 6.1, where the lines suffixed with “Fast” and “Slow” represent the faster and slower partitions. It becomes very clear that the partition optimization manages to increase the download speed for all nodes, both fast and slow. The graph also shows that the faster nodes that are using PV are not slowed down as much by the slower ones because of the partitioning. Neither do they level out as fast and definite as the others, but keep increasing.

6.2 Evaluating the results

Figure 6.1 and 6.1 gives a detailed image of how the algorithms perform over time, but it makes it more difficult to see and compare the actual performance of the different algorithms. To avoid this clutter of many fluctuating curves in the same graph, the remaining figures are bar charts that simplify comparison of the different tested algorithms. To avoid the chaotic convergence phase of the network, and thereby get a fair average of every algorithm, each bar represents an average of the data between the time steps 1500 and 5000.

All confidence intervals are calculated using a sample size of 10 simulations per graph and we have chosen a confidence level of 95%. The size of the confidence interval is visualized as a bracket located to the right of each bar.

The number of hops needed to get to a source in a request has earlier been used as a metric to describe the performance of the Freenet network, for example in [8] and

[10]. We did also monitor the request pathlength during our simulations in order to be able to use it to evaluate the different solutions we tested.[6] As seen in Figure 6.2, our results show that the differences in average request pathlength are varying very little between the solutions, once the network has converged. The relatively low pathlength combined with the narrow confidence interval indicates that the efficiency is quite high in all solutions, whenever a document is found.

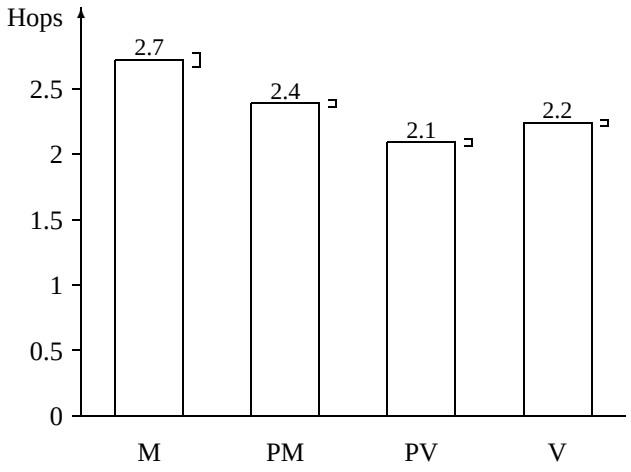

Figure 3. Average successful request pathlength

As we described earlier the number of unsuccessful requests is very important to the user experience. This metric is shown in Figure 6.2. Since all requests in our simulation are issued for documents that we know have previously been successfully inserted into the network at some point, we would expect an optimum of near zero percent failed requests. What could influence this metric is the presence of *orphan documents* in the simulator. These orphans are documents that no longer are present in the network as they have been dropped by every node that had them. Our simulation shows that none of the algorithms produced any orphans during the simulation.

When comparing the different algorithms, all of them show an improvement over Vanilla, but PM stands out clearly as the best. Where Vanilla fails more than every tenth request (that should have succeeded), PM show an improvement of over thirty times with approximately one failure for every two hundred requests. In our simulation this means that, on average, when using Vanilla, 650 requests by

[6]Note the difference in measurement technique. The original Freenet paper measures the hops via timed probes of the network using a number of request packets with a very high HTL, where we are measuring continually on the participating nodes.

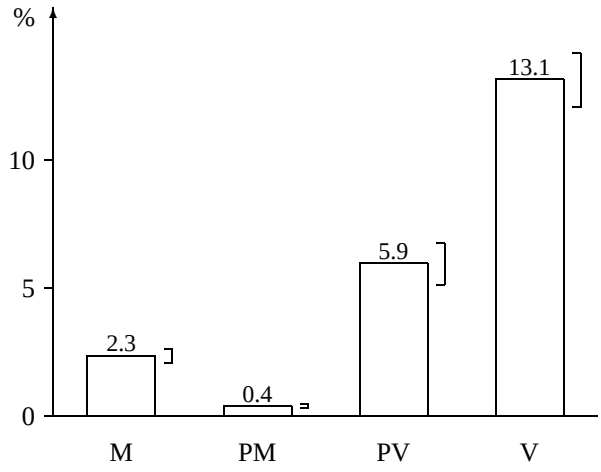

Figure 4. Percent unsuccessful requests

users would be affected by getting false failures, and when using PM this is reduced to 20 requests.

Download speed in Figure 6.2 is not simply measured by averaging all incoming transfers to a node. The only transfers that are taken into account are those where the user initiated the requested. These results are interesting because this is the transfer speed which a user will experience when requesting and downloading documents.

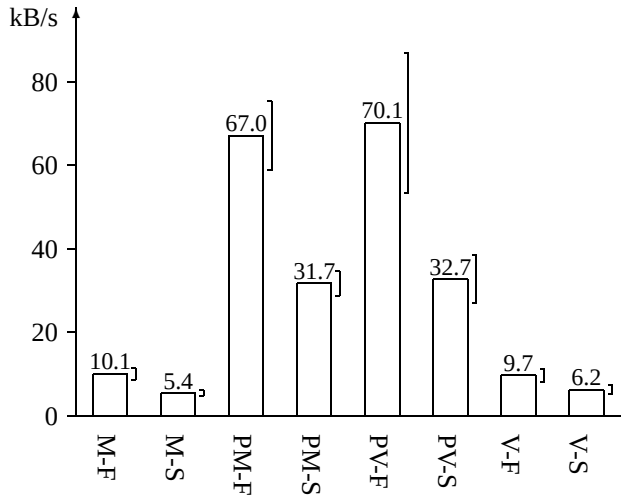

Figure 5. Average download speed

Figure 6.2 clearly shows how all our partition algorithm succeed in increasing the download speed. Even nodes with slow connections manage to perform better with the partition algorithm than fast connection nodes do with Vanilla routing. The pure mesh routing stays close to Vanilla performance but when it is combined with the partition algorithm, its speed increases drastically.

Both slow and fast nodes benefit from the partition routing since the slow nodes do not get overloaded by the faster ones, and the fast nodes do not have to route thru the slower bottlenecks.

Another important issue is how much of the nodes bandwidth that is consumed on average. This is shown in Figure 6.2 and Figure 6.2.

It is easy to think that the low values on the fast nodes mean less utilization than high values on the slow nodes because of the relative scale. But since the capacity is so much higher on the fast nodes, they still manage to outperform the slow ones.

Again, Mesh and Vanilla stay close to each other when it comes to transfer speeds. With Mesh the upload bandwidths for the slow nodes is slightly higher than with Vanilla but other than that they are about equal. The small advantage in speed with Mesh is probably due to the fact that it manages to find more content than Vanilla.

One interesting observation is that the partition algorithm has managed to inverse the upload utilization between fast and slow nodes as shown in Figure 6.2. By inverse we mean that the fast nodes have higher values and the slow have lower. This means that the overall speed will be higher with the partition algorithm since the fast nodes have more capacity.

Another interesting effect of the partition algorithm is that it manages to even out the utilization between fast and slow nodes. This is good because there is not a particular group of nodes that have to pull the weight, but all nodes will get a relatively small part of the total load. This also increases the total performance since the fast nodes get more load than with Vanilla.

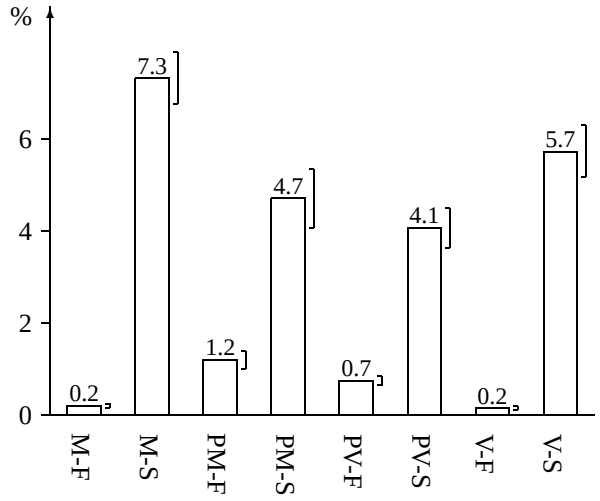

Figure 6. Download bandwidth utilization

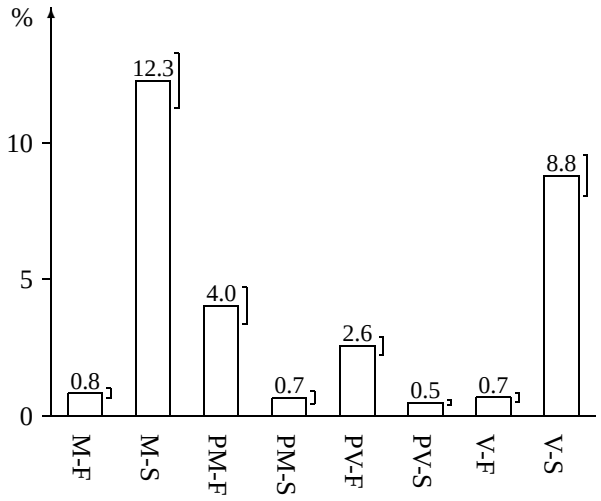

Figure 7. Upload bandwidth utilization

7 Related work

The work done in the area of improving Freenet is limited, probably due to Freenets rather unique design and design objectives.

In [12], Zhang, Goel and Govindan show that by enforcing the small world model on Freenets cache, the routing performance can be improved, compared to using LRU (Least Recently Used). The algorithm influences the node to make the documents in the cache cluster around an arbitrarily chosen key. When discarding a document from the cache, the document that is furthest from the chosen "specialization key" is removed. Simulations in [12] indicate a significant decrease in the number of failed requests compared to LRU and a slight improvement compared to clustering without random shortcuts. The small world method renders close to 0% fails compared to LRU with about 65% fails.[7]

In [10] the authors argue that Freenets approach of updating the routing table gets inefficient if the number of requests is much greater than the number of inserts. The authors propose a better way of solving this problem by spreading the news in a different manner. Instead of simply passing the note along the search path, they let the initial requester spread the news. This is done by the requester by means of sending a message to a random neighbor containing the note (which contains the information about which node satisfied the request of what key) and that neighbor will send the note to one of its neighbors and so on.

Another way of modifying the routing is presented by Ian Clarke in [7] where he proposes the NGR (Next Gen-

[7] The notably bad performance of LRU in this simulation is most likely due to choices in cache size (50-200) and number of inserted documents (10 per node) combined with an excessively large HTL (40-100).

eration Routing) routing principle. The basic idea behind NGR is that all nodes will collect statistical data about the communication with their neighbors and use that material as a decision-base when they are routing future queries.

8 Future research

There is plenty of research needed in this area. Both on Freenet and unstructured P2P-networks in general and regarding our proposals and results.

The file mesh inherits many properties from the bloom filter, like being useless when saturated. This leads to the topic of finding the optimal size of the file mesh. Another quite interesting possibility is to evaluate different "desaturation" techniques. The implementation of such a desaturation technique would probably be a requirement for actual deployment in a live Freenet system.

There is no lack in directions for future research on *Preferential Partition Routing* either. When looking at using partitioning to optimize download performance, there is the matter of choosing the optimal number of partitions and their sizes. Another area of research is selecting (and verifying) what partition each node should belong to in a more efficient manner.

How these changes affects anonymity in Freenet is also an area where research is needed.

9 Conclusions

In this paper we investigate how to improve performance of Freenet by focusing on two issues: (i) hit-ratio, and (ii) download speed.

Our proposal for improving the hit-ratio is based on propagating some information about the documents avaliable at each node, so called File Meshes, to its neighbors. To increase the download speed we suggest to partition the network based on node's bandwidth.

We have developed a simulator to evaluate the impact of our proposal to the Freeenet performance. The simulation results indicate that applying our improvements to a Freenet, the bandwidth would become more evenly utilized by both fast and slow nodes, and as a side effect the download speeds would increase. Routing would also become more efficient and precise due to the file mesh propagation. Our results give a clear indication that preferential partition routing can be used to improve overall performance of networks that route thru multiple hops. They also suggest that file meshes can be used to improve precision in Freenet like routing mechanisms.

Our experimental results show that combining partitioning with file meshes will increase the request hit ratio by over 30 times and the average download speed by six times, compared to the original Freenet.

Further information and the simulator source code are avaliable at http://www.skoghs.se/eos/.

References

[1] p2psim: a simulator for peer-to-peer (p2p) protocols, August 2004.

[2] Sourceforge.net cvs repository - directory - cvs: freenet/aurora, August 2004.

[3] Sourceforge.net cvs repository - directory - cvs: freenet/serapis, August 2004.

[4] J. Ahn and P. Danzig. Speedup vs. simulation granularity. In *IEEE/ACM Transactions on Networking, vol. 4, no. 5*, pages 743–757, October 1996.

[5] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, July 1970.

[6] J. Chu, K. Labonte, and B. Levine. Availability and locality measurements of peer-to-peer file systems. In *vol. 4868 of Proceedings of SPIE*, July 2002.

[7] I. Clarke. Freenet's next generation routing protocol, 2003.

[8] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46, 2001.

[9] J. Haeggström and H.-E. Skogh. Improving freenet performance by precedencial network partitioning and file mesh propagation, 2005.

[10] J. Mache, D. Ely, M. Gilbert, J. Gimba, T. Lopez, and M. Wilkinson. Modifying the overlay network of freenet-style peer-to-peer systems after successful request queries. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, page 90289c, 2004.

[11] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking*, 2002.

[12] H. Zhang, A. Goel, and R. Govindan. Using the small-world model to improve freenet performance. In *Proceedings of INFOCOM 2002*, 2002.