Peer-To-Peer Information Systems

Designing a DHT for low latency and high throughput

Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, Robert Morris

Write-up by
Robert Vollmann

08.12.2004

1 Introduction

A main idea of peer-to-peer information systems is a decentralized, shared global storage infrastructure. But who determines where the data is stored and how can the data be accessed? An efficient approach to this problem are distributed hash tables (DHTs). The paper discusses several designs to optimize DHTs with regard to latency and throughput.

1.1 Distributed hash tables

A hash table is a data structure that uses a hash function to map keys to hash values. With the hash value one can determine where a key should be stored. The search for keys the so-called lookup can be efficiently implemented so that a fast access to keys is supported.
A distributed hash table is a special variant of a hash table where the entire hash table is distributed over all participating nodes. Here the lookup algorithm determines which participating node is responsible for which key.

1.2 Requirements

The requirements a distributed hash table needs to fulfill are to name and to find data. The data should be available as much as possible and it should be distributed equally over all participating nodes. Moreover if nodes join and leave the data should be redistributed as fast as possible among the remaining nodes. Therefore low latency and high throughput is required.
These requirements are more difficult to achieve because of slow link connections of nodes. The physical distance of nodes also plays a role. If two nodes are far away from each other the packets have to travel a long distance through many networks and routers. If a network is congested it may take very long until the packets arrive at their destination. Congestion appears if a router in a network gets more packets than it can process. Thus the packets are buffered and more time is needed until they can be processed. If the number of unprocessed packets increases over a long time and the buffer is entirely filled packets have to be dropped and need to be retransmitted.

2 Background

The main application described in the paper is a distributed hash table called Dhash++.

2.1 Dhash++

DHash++ is an application that can be seen as a network storage system with a shared global infrastructure. In DHash++ values are mapped to keys using the SHA-1 hash function. After this the key/value pairs which will be called blocks

from now on are then stored on different participating nodes. DHash++ also moves the data to the new/remaining nodes if nodes join/fail.
DHash++ uses Chord and Vivaldi as its key components that will be described in the following two sections.

2.2 Chord

Chord is a lookup protocol, which is designed to find data with runtime O(log *N*). It can also be used to determine on which node which block should be stored. The data blocks are mapped to keys and the nodes are mapped to IDs. If a nodes are participating the entire ID-Space range ranges from 0 to $2^{\log((a))}$. In addition every node maintains a finger table holding IDs with the corresponding IP addresses. The ith finger table entry of a node x lies in the range of $x+2^i$ to $x+2^{i+1}-1$ and points to the first available node within this range. Moreover, every node holds a list of its s immediate successors.
If now a key is searched, the originator contacts the node in its finger table with the highest ID lower than the key. The contacted node also searches its finger table and sends the ID back to the originator who then contacts the received ID. This procedure is repeated until the closest predecessor of a key is found. The originator then contacts the predecessor and receives its successor list and the lookup is finished.

2.3 Vivaldi

Vivaldi is a decentralized network coordinate system which calculates and manages synthetic coordinates. Whenever two DHash++ nodes communicate they exchange synthetic coordinates. These coordinates for example are stored in the finger table entries together with the IP address of an ID. So if on a lookup the originator receives a finger table entry he also gets the corresponding synthetic coordinates with which it can predict a latency without having ever communicated with this node.
Other advantages are that Vivaldi is decentralized which fits to the idea of peer-to-peer information systems. Moreover the coordinates can be attached to DHash++ packets and no additional packets are needed.

3 Evaluation methods

Different designs intended to optimize latency and throughput were tested using a test-bed consisting of 180 hosts which are connected via Internet2, DSL, cable or T1. Moreover three DHash++ instances are executed on each host. Furthermore, the testing included a network simulation of 2048 nodes. It uses a matrix with a rank of 2048 called King data-set. The i-th row and the j-th column consists of the round-trip time of the i-th and j-th node. The round-trip time (RTT) is the time it takes to send a packet to a host and receive the response. The round-trip time is usually employed to measure the congestion in networks. Also a second data-set with the round-trip times of the test-bed nodes was used. The latency between each pair of nodes has a median of 76 ms and an average of 90 ms, which is low in comparison to the respective values of 134 ms and 154ms observed on the King dataset.

4 Achieving low latency

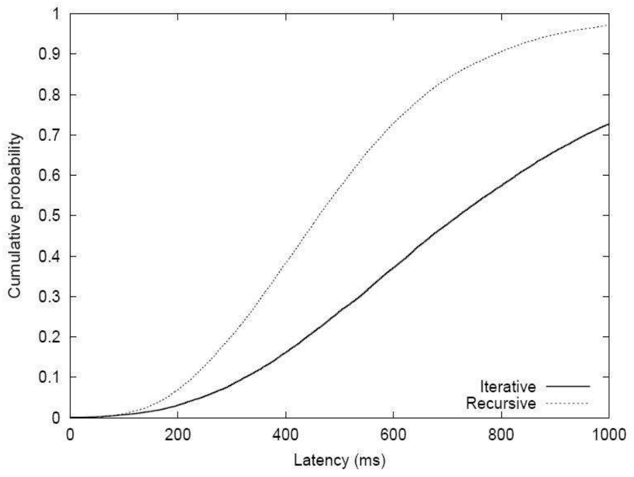
4.1 Data layout

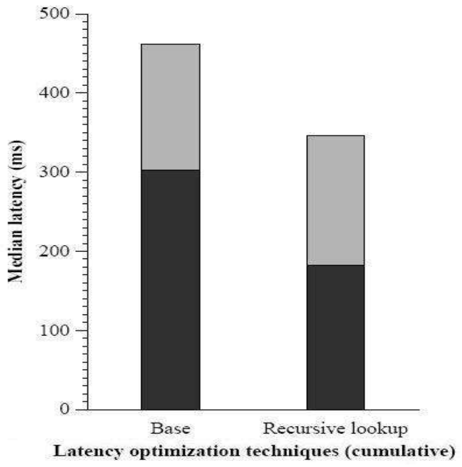
DHash++ is designed for applications that read data most of the time and thus demand low latency with latency being the time needed for a packet to travel from one node to another node. Examples of such applications are SFR (Semantic Free Referencing System) and UsenetDHT.
SFR is designed to replace DNS (Domain Name System). When nodes are communicating to other nodes they use IP addresses to determine the destination where the packets should be sent. But because an IP address is not easy to remember many nodes have aliases that allow numbers and letters. To obtain an IP address for an alias the node first has to contact a DNS server which then sends back the corresponding alias. Because these name bindings are relatively small they can be stored in small blocks.
The second example is UsenetDHT which is a distributed variant of Usenet which contains a lot of articles. Large articles are split and distributed over different nodes while small articles are stored in a single block.
Because of these two examples DHash++ uses small blocks of 8kb length.
Another layout decision is the distribution of the blocks. If some blocks are at most fetched in a specific area it would make sense to distribute these blocks to nodes in this area. But power and network failures could decrease the availability of these blocks drastically. Thus DHash++ distributes the files randomly over the participating nodes using its hash function.

4.2 Lookup routing

The next design decision regards lookup routings. Iterative and recursive routing are examined as two possibilities of lookup routing.
With iterative routing the lookup procedure is the same as the Chord lookup described in section 2.2. The advantage of this method is that failed nodes can be detected very easily. If a lookup query is not answered by a node the originator supposes that the node fails and simply contacts another node. But the disadvantage is that the originator has to wait for every response before it can send the next query.
The second routing possibility is recursive routing. Here the originator sends the first lookup query to the node which is closest to the key according to iterative routing. But instead of answering the contacted node sends the lookup directly to the closest ID in his finger table. This step is repeated until the predecessor is

found. The predecessor then sends his successor list back to the originator.
The advantage in this kind of routing is that there is no need to wait for a response. the lookup query can be sent to the next node immediately. Second only a little more than half of the communication of iterative routing is used because there is no need for an answer from each node except the predecessor. But the big advantage lies in the detection of failed nodes. Because no node answers on node failure one cannot determine which node in the lookup chain failed.
In order to determine which lookup routing is the most efficient both were tested in the simulation and in the test-bed. The figure below show the result of the simulation and the test in the test-bed.



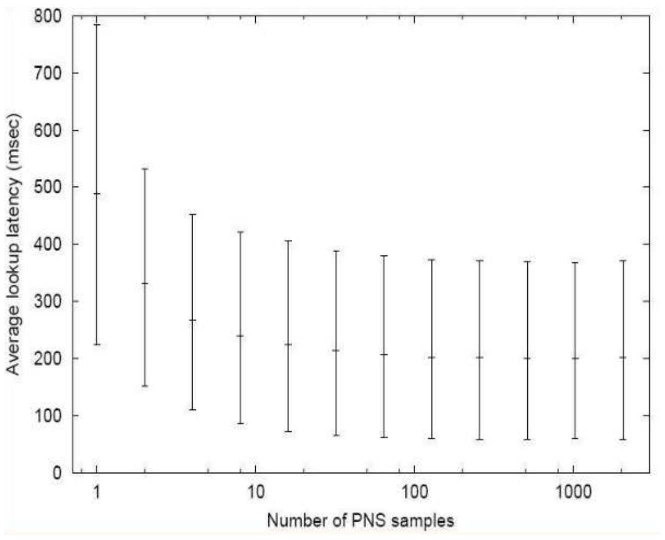


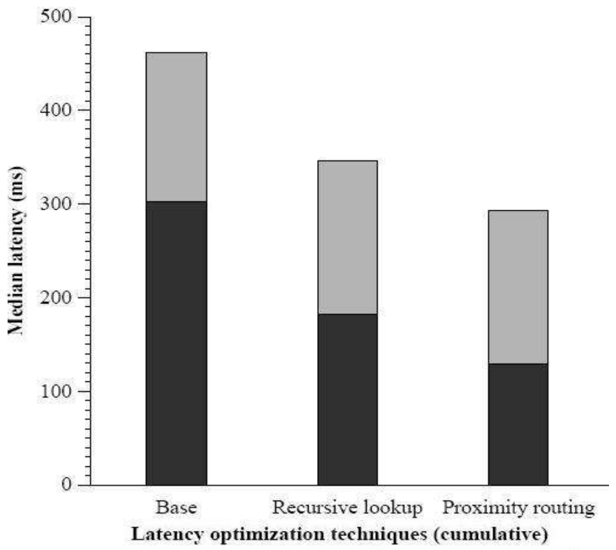
The left figure presents 20,000 lookups with random hosts for random keys were performed. The median and average of iterative routing were 720 ms and 822 ms while the median and average of recursive routing were only 461 ms and 489 ms. This shows that recursive lookup routing only takes 0.6 times as long as iterative routing.
The right figure confirms the result from the simulation. The dark portion of each bar represents the lookup time while the right portion represents the time needed to fetch a block.
DHash++ uses recursive routing because it is faster than iterative routing but switches to iterative routing in case of persistent link failures.

4.3 Proximity neighbor selection

This design decision refers to the lookup procedure itself. The idea behind proximity neighbor selection is to choose nearby nodes to decrease latency. The idea is realized in the following way. As mentioned earlier the ID-Space range of the i-th finger table entry of node is between $a+2^i$ to $a+2^{i+1}-1$ and every finger table entry points to the first available node. Now on lookup a node contacts the node closest to the key for the first x entries in his successor list.

The node then searches a successor with the lowest latency in the same ID range and routes lookup through this node. While in the real implementation x cannot be greater than the length of the successor list the simulation can use any value from 1 to the number of all nodes in the network for x. The next two figures show the result of using different values for x and the test in the test-bed.





In the left figure the 10th, 50th and 90th percentile of the lookup latency for all x values are shown. The values range from 1,2,4,....2048 where 2048 means that the latency of all nodes in the entire network was taken. This shows that with low values of x the latency is highly decreasing while later hardly any optimization is achieved. Thus DHash++ uses a value of 16 for x because it is relatively low and achieves a high decrease of latency.
As a summary the right figure shows an improvement in the test with the real network.

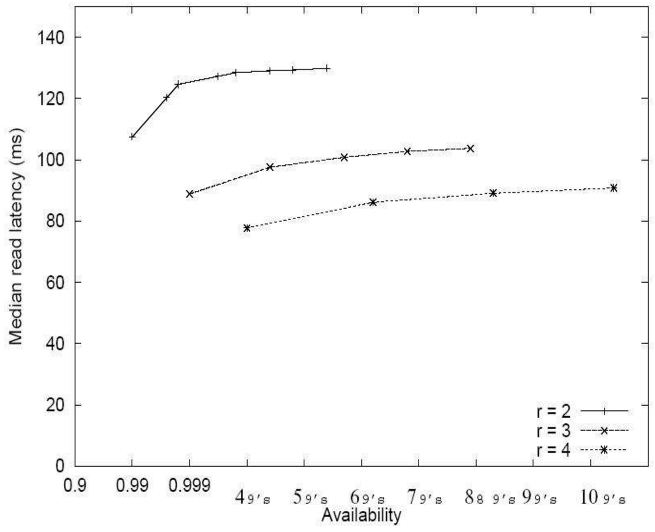
4.4 Erasure-coding vs. replication

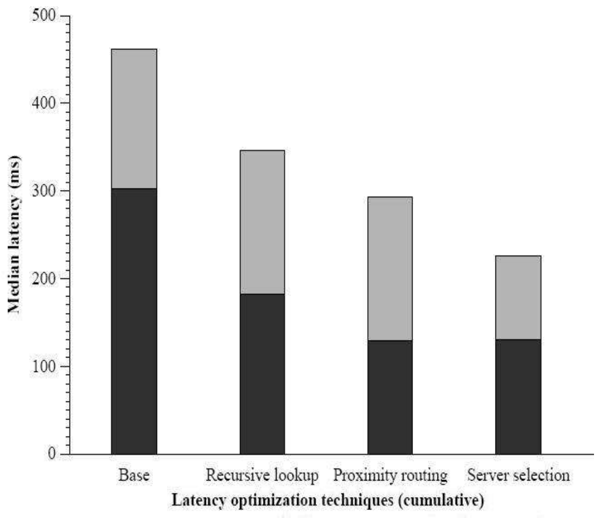
This design choice affects the data blocks. Two methods namely erasure-coding and replication are presented. With erasure-coding a data block is split into l fragments but only m blocks are needed to reconstruct the block. Thus redundant data is stored in each block.
The other method is simple replication where each host stores entire data blocks. Therefore m equals 1 and l represents the number of replicas in the entire network.
To compare both methods $r = l / m$ is defined as the amount of redundancy and the probability that a block p is available is denoted as:

$$p_{avail} = \sum_{i=m}^{l} \binom{l}{i} p_0^i (1-p_0)^{l-i}$$

Where p_0 is the probability that a node is available.

Again a simulation was conducted with different values for l, m and r. The value for p_0 is set to 0.9 here but the authors do not explain why they exactly used this value. Probably the value is set arbitrarily though a different value obtained (e.g. through testing) could probably lead to different results.



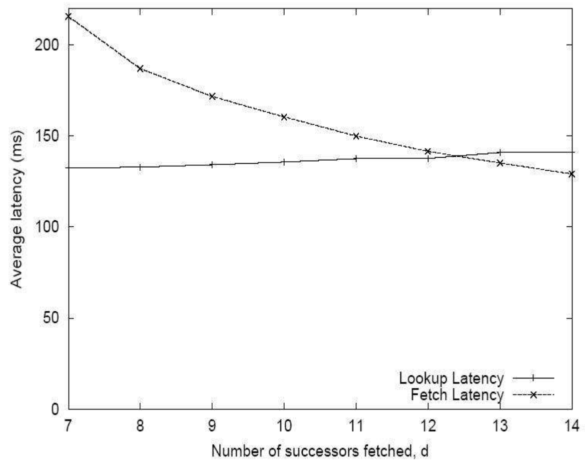


The left figure shows the results of the simulation. The three lines correspond to the different values for r with r = 2,3,4. The points on each line correspond to the values for m with m beginning with 1 which stands for replication. As one can see the latency difference between erasure-coding and replication decreases with higher values of r so that this disadvantage of erasure-coding is very small. Furthermore, with higher m the availability which is an important factor of peer-to-peer systems increases. A disadvantage of erasure-coding is that on joining and leaving nodes much more data has to be redistributed which causes more network traffic.
The right figure represents the test within the test-bed which shows that erasure-coding reduces the fetch latency by nearly one third.
Because of higher availability and lower fetch latency DHash++ uses erasure-coding with m = 7 and l = 14.

4.5 Integration

The advantages of Proximity neighbor selection could hardly be used in the last steps because with every step towards the key the ID-Space range decreases and the choice between nodes is highly decreased**.** Mainly in the last step there is only one direct predecessor of the key which returns the successor list. But the entire successor list is not needed at all because from a successor list with length s only l successors have a needed fragment and from these l successors only m are needed to reconstruct the block. Thus s-m predecessors of the key store a successor list with at least m nodes which have stored a needed fragment. With integration the expensive last hop to the direct predecessor could

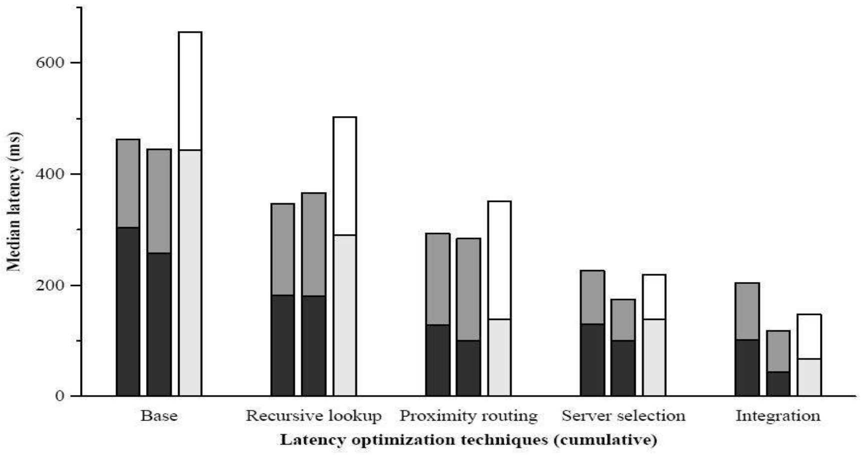
be avoided.
The remaining question is what value *d* representing the minimum number of successors which store a needed fragment which should be fetched is suitable. If d is very small one hop could probably avoided but the choice between successors with low latency is smaller than with large *d*.
The effects of different *d*s are tested in the simulation again:



The figure shows that with increasing d the lookup latency hardly increases while the fetch latency decreases. Because there is an overall increasement the highest possible *d* with a value of 14 is chosen by Dhash++.

4.6 Summary

The following figure summarizes the used design choices and the resulting effects and compares the different testing methods.



The three bars represent the test with the test-bed, the simulation with the test-bed data-set and the simulation with the King data-set. As seen previously the dark portion of each bar represents the lookup latency while the light portion represents the fetch latency. Although the results from the different test methods differ the trend is the same. Each design method reduces the lookup latency respectively the fetch latency by a certain amount. All methods combined result in an optimization factor up to 2.

5 Achieving high throughput

Throughput simply means the amount of data processed in a certain time. It is, e.g., measured in kilobyte per second (kb/s).
The requirements which are made on DHash++ to optimize the throughput are parallel sending and receiving of data as well as efficient congestion control to avoid packet loss and the resulting re-transmission. If packet loss occurs it should recover as fast as possible.
One possible solution, which will be examined in the following section, is to take an existing transport protocol. The alternative is to take a specialized transport protocol fitted to the needs of the application.

5.1 TCP

An advantage of TCP (Transport Control Protocol) is that is already has its own congestion control implemented but it takes a certain amount of time till this control works efficiently. In addition TCP also imposes a start-up latency and the number of simultaneous connections is limited due to storing of many state variables for each connection.
Nevertheless there is an approach to use TCP with DHash++ and is implemented as follows. A node holds a small number of TCP connections open to all finger table entries and to its successors in the successor list. All DHash++ related communication is then routed through these connections in both directions. If a node wants to fetch a data block the predecessor of a key fetches the needed fragments, reconstructs the block and sends it over one of his neighboring TCP connections back to the originator.
In my opinion this approach is not very well thought-out because if DHash++ already gets the predecessor of a key it could easily send his successor list back through his neighbors or open a new short-lived TCP connection to the originator to transfer the successor list. Also the originator could then open a small number of new connections to the successors if he already knows the IP addresses. But with this approach it is possible do fetch a file from a node that is probably somewhere in the neighbourhood over TCP connections to nodes which are distributed all over the world.

5.2 STP

STP (Striped Transport Protocol) is taken by DHash++ to avoid the disadvantages of TCP. STP allows to receive and transmit data directly to other nodes without circumstantial connections to the neighbors. There is also only one single instance running at a time which saves resources. Furthermore all decisions of STP are based on the current network behavior and Vivaldi.

5.2.1 Congestion window control

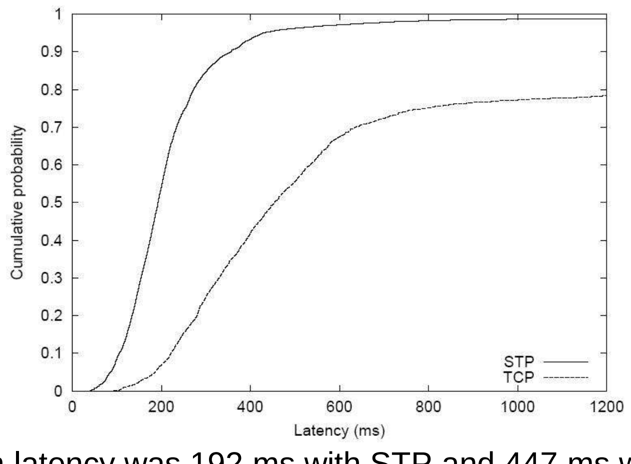
STP has congestion window control similar to TCP. STP can always send w RPCs(Remote Procedure Calls) simultaneously.
RPCs are procedure calls from one node on a different node in the network. An example here is the lookup. A node sends a RPC to another node. This RPC calls a procedure on this node which searches the ID which is closest to the key and sends it back to the originating nodes. If now such a RPC is answered a new RPC can be send and w is increased by $1/w$. If a RPC is lost w is decreased by $w/2$.

5.2.2 RPC Timeout

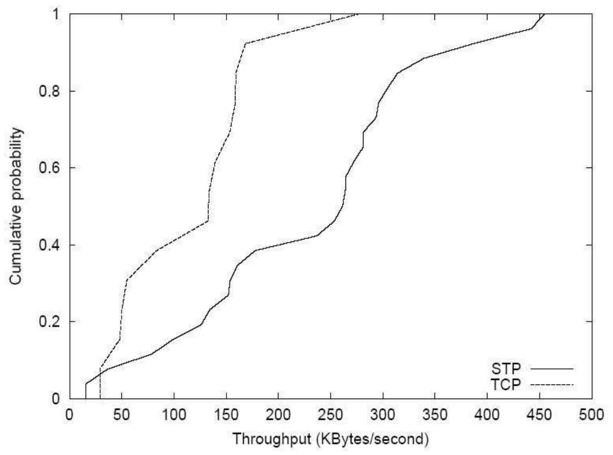
Every time a RPC is sent a timer is activated. If this timer runs out before the RPC is answered the RPC has to be resent. TCP calculates the timeout with the help of deviation of old round-trip times. This means that first some packets have to be sent to measure the round-trip times and calculate a good timeout. The problem with DHash++ is that it does not send two successive RPCs so it must predict the round-trip times to a node before ever having communicated with this node. Therefore synthetic coordinates and Vivaldi are used.
Nevertheless if packet loss occurs the RPC has to be resent. STP does not directly try to send the RPC again but reports the loss to DHash++ which on lookup sends the RPC to the next-closest finger and on fetch queries the successor in the successor list which is next-closest in predicted latency.

5.3 Comparison of TCP and STP

The following figure shows a sequence of single random fetches of 24 nodes.



The median fetch latency was 192 ms with STP and 447 ms with TCP. This shows that the optimization factor of STP is up to 2.5. On average 3 hops were needed to complete lookup , which serves as an explanation for the higher latency of TCP. TCP has to route the entire successor list from a predecessor of a key back through the direct neighbors (here direct neighbors mean the finger table entries) of the nodes while STP can send the successor list directly from the predecessor to the originator of the lookup.



The same observation is made in the following figure.
Here all 26 nodes made simultaneous fetches of random data blocks. While the median throughput of STP was 261 KB/s the median throughput of TCP was 133 KB/s which means a decrease of latency by factor 2.
The reason for this is the same as for the last figure. The data block has to be sent over the neighbors of all nodes with TCP instead of direct send with STP. Although STP shows a better performance than TCP its result is not optimal. Since all 26 nodes transmitted and received data simultaneously and the slowest access link was 0.4 Mb/s the total expected throughput would be 0.4 * 26 = 10.4 Mb/s or 1.3 Mb/s throughput for a fetching site which has such a fast access link. But in this test the fastest site achieved less than half of the possible throughput with STP. Thus, STP is not yet prefect and can still be improved.

6 Summary

The latency optimization designs described in section 4 achieved an optimization up to factor 2. Furthermore STP reduced fetch latency by a factor of 2.5 and optimized throughput by a factor of 2.
Nevertheless, there are several factors which still have to be explored.
First this designs were fitted to protocols with a runtime of O(log N). Protocol runtime could be increased or constant hop protocols could be used.
Also this DHT was designed for read-heavy applications. Write-heavy applications could be explored in future. In addition to that different block sizes than 8kb could have different not necessarily better effects used with DHash++. Nothing was said about the robustness of these design and what effect increased robustness could mean for the performance. Moreover STP and the use of TCP could be optimized further as previously mentioned.

Bibliography

Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, Robert Morris: *Designing a DHT for low latency and high throughput.* In Proceedings of the USENIX Symposium on Networked Systems Design and Implementation, March 2004.
www.pdos.lcs.mit.edu/papers/dhash:nsdi/paper.pdf

Frank Dabek, Russ Cox, Frans Kaashoek, Robert Morris: V*ivaldi: A Decentralized Network Coordinate System.* In Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications, 2004.
http://www.pdos.lcs.mit.edu/~rtm/papers/p426-dabek.pdf

Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, Hari Balakrishnan: *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications.* IEEE/ACM Trans. Netw., 2003.
http://portal.acm.org/citation.cfm?doid=638334.638336

James F. Kurose, Keith W. Ross: *Computer Networking: A Top-Down Approach Featuring the Internet.*Addison-Weslay, Second Edition, 2003.