Improved Lookup Algorithms for Viceroy

Oren Dobzinski, Anat Talmy
The Hebrew University of Jerusalem,
Jerusalem 91904, Israel
orend@cmu.edu, atalmy@andrew.cmu.edu

Keywords: Peer-To-Peer, Lookup algorithms, Distributed Hash Tables, Simulations.

Abstract

We present a new lookup algorithm for Viceroy, a peer-to-peer system, which outperforms the existing algorithm for the system, yet it is much simpler and easier to implement. We also present our fully-functional graphical simulation of Viceroy, and propose several other improved lookup (routing) algorithms for this network. We show experimental results to support our claims, and discuss the implications of a simplified algorithm for the system.

1 Introduction

The Viceroy [9] network is a constant degree, peer-to-peer lookup network, which behaves as a DHT (Distributed Hash Table), i.e. distributes resources stably to nodes in a dynamic, distributed network. Its main purpose is to efficiently look up resources, where no central authority exists and where the network is dynamically changing. The Viceroy network is a composition of an approximate butterfly network and a connected ring of predecessor and successor links. Due to the independent structure of the network, adding or removing any node can be achieved without any global coordination, and only local updates are required. Locating resources is done in an efficient $O(log(n))$ hops, where n is the total number of nodes in the system. Viceroy was the first constant-degree network to achieve these results. Similar results have also been achieved in non-constant degree networks such as Chord [15], Pastry [5] and Tapestry [16].

Our Contribution: While it has many desirable properties, the Viceroy network and its routing algorithm is far from being simple, and this is one of its main weaknesses. We tackle the routing algorithm's complexity problem by suggesting a simple and completely greedy routing algorithm, called FindFast. Experimental results show that it achieves even better performance from the original, complex, algorithm suggested in Viceroy [9]. Its greediness allows it to be completely stateless, which greatly simplifies its implementation.

We provide an applet that gives the user an opportunity to get a graphical view of the Viceroy network; it is a fully-functional system that can dynamically perform all available operations on the peers, including join, leave and lookup. We also provide several graphical views of the system, each demonstrating a different aspect of it.

The remainder of this paper is organized as follows: In Section 2 we describe relevant previous work. Section 3 concentrates on describing the Viceroy network. In Section 4 we describe the existing routing algorithm for Viceroy, and give our improvements and new algorithms for it. In Section 5 we give our experimental results, and discuss their consequences. Finally, we conclude in Section 6.

2 Previous Work

The most widely used approach in Peer-To-Peer systems which distribute and access data is the Distributed Hash Table approach. The main task in such systems is to locate the peer responsible for a given resource. The lookup request is forwarded from peer to peer in the system, until the peer that holds the resource is found. One of the main measures of quality of a routing algorithm in a peer to peer system is the number of hops traversed until the correct peer is found. In most cases logarithmic complexity is achieved (such as in [1, 12, 7, 6, 15, 5, 16, 9]).

Several systems use prefix-routing algorithms in order to route lookup requests. The first to suggest this type of routing were Plaxton *et al.* [13], which introduced a routing algorithm for a static network, rather than a dynamic one. In every step of the routing path one 'digit' is fixed; suppose we look for a resource whose value is 5555. The initiating node will pass the request to a node whose ID's first digit is 5. The second node will pass it to a node whose ID's first two digits are 55, and so forth, until all digits are fixed. This routing algorithm reaches the 'responsible' node in $O(log(n))$ steps, for a network of n nodes. Some implementations that use prefix-routing heuristics are Pastry [5] and Tapestry [16]. A recent system that uses this routing scheme is LAND [2], which also optimizes the distance traveled by the lookup query, and not only the number of hops traversed.

Both Chord [15] and Viceroy [9] use a circular value space with ring links between numerically close nodes, which is used in the final stage of their routing algorithms. Both systems achieve $O(log(n))$ performance using long-range links ('fingers' in Chord, 'children', 'parent' and 'level' links in Viceroy). In Chord, routing requests are forwarded to the closest possible node to the target (numerically), without reaching a node whose numerical value is higher than that of the target. A different approach to forming a DHT is taken by CAN [14], whose nodes form a d-dimensional torus. Routing is done by passing the lookup request to the closest neighbor. Note that when d is chosen to be $O(log(n))$, the performance of CAN [14] is also $O(log(n))$. Another efficient DHT construction is Distance Halving [12]. Two of its main advantages are its elegancy and simplicity.

Some greedy routing schemes have been deployed in Peer-To-Peer systems, inspired by the work of Kleinberg [8], who modeled the Small World phenomenon. This phenomenon was originally introduced by Milgram [11], who claimed that people are connected by surprisingly short chains of acquaintances. His original experiment showed that people were able to deliver letters effectively using these links. Kleinberg used this observation to construct a system in which each node has four close range links and only one long range link. He used greedy routing in order to deliver messages between nodes, an approach which later proved to be optimal by Barriere *et al.* [3]. The recent Symphony [10] network also uses greedy routing in a system that allows the number of outgoing links of each node to be configurable.

3 An Overview Of Viceroy

Viceroy is a Peer-To-Peer system, consisting of peers, or nodes, running on different and remote machines. Every peer has the ability to 'lookup' resources, which can be movies, songs, documents, or any type of file or object. At the same time, every peer holds information about the location of some resources, or as we will refer to it below, 'has responsibility' for the resources. In this sense Viceroy is a Distributed Hash Table (DHT). Resources are mapped to values from the stretch [0;1) by a hash function known to all participating peers. The starting precision of this mapping is 128 bit, but it can be enlarged afterwards. We refer the stretch [0;1) as a closed, circular, range in which values increase as we traverse to the clock-wise direction, except for the connection point of the stretch. We denote the 'clock-wise distance' between two peers as the distance between the peers in the clock-wise direction on the values ring.

Peers' Properties: Every peer that enters the system selects an ID independently and uniformly from the stretch [0;1). The default ID length is 128-bits. Temporarily, two peers with the same ID can exist. If such an ID clash is detected during the operation of Viceroy, a procedure of changing the ID is performed: the peer that detected the clash adds several precision bits to its ID and clash is then resolved. This ID is used to determine the set of resources the peer is responsible for. Another property of every peer is its level, which is used to place the peer in a butterfly-like network. During startup, every peer initiates a level selection procedure, whose goal is to have a balanced network, such that every level contains approximately the same number of nodes. The level selection procedure of the nodes must be consistent with this property. Specifically, when a peer joins the system, it performs an estimation of the number of peers currently active in the system, n. Since no global authority exists, the peer can only use local data, and it bases its estimation of n on the distance to its successor. Then, the level this peer picks is selected randomly and uniformly among $[1 \ldots \log(n)]$. As stated in [9], the distributed procedures of level and ID selection achieve a good dispersal of levels and IDs among Viceroy peers, and result in an approximate balanced network.

Peers' Connections: Each peer in the system has three outgoing links to chosen long-range contacts - **parent**, **left child** and **right child** links, and two ring connections, one to its **successor** and one to its **predecessor**. Parent, left child and right child connections are considered 'butterfly' links. The parent link points to a peer whose level is lower by one from the current peer, while the children links point to peers with a level higher by 1 from the current level. The predecessor and successor links are considered 'ring' links, and they point to the closest peers, in terms of ID - the peer with the lowest ID which is higher than the current one, and the peer with the highest ID which is smaller than the current one. There are two additional level links in the more complex version of Viceroy, which is not yet implemented, pointing to the next and previous peer in a given peer's level.

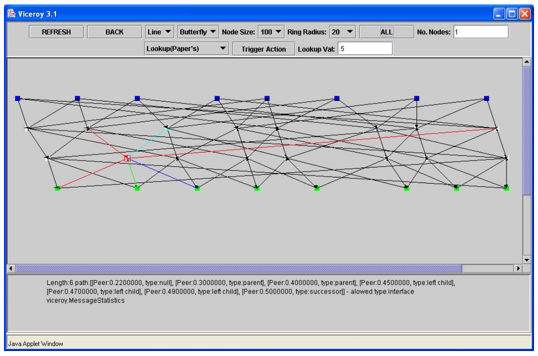
Peers Joining And Leaving: Whenever a peer joins or leaves the network, some local changes in the network occur. When a node joins the network, it is given the responsibility for a stretch of possible resources whose mappings are in the stretch $(predecessor.id \ldots current.id]$. The joining peer must request this data from its successor, which drops these values as soon as the data transfer is over. Having responsibility for a resource merely states that this peer holds information regarding which peer actually holds a given resource. The reverse procedure is performed when a peer leaves, or when a peer discovers that its successor left unexpectedly. These actions usually cause a structural change in the network, as some peers might need to find new butterfly connections, or re-select their level if they have a new successor. The assumption is that joins and leaves are frequent events, and therefore they must cause the smallest and most local change as possible.

The only part missing from the above description is the lookup algorithm - stating how the request for locating a given resource is routed until it reaches its final destination, which is the peer responsible for the resource. We give a full description of the current lookup algorithm and our suggested algorithms in Section 4.

Implementation: We have fully implemented the Viceroy network, (demonstration is available at [17]), and we describe here the simulation we performed based on it. For a complete discussion of the Viceroy system, see [9] and for the details regarding the implementation see [4]. The applet running in the website (see Figure 1) is a simulation of local Viceroy peers, running on one machine. However, the full implementation can be used in a distributed way, where each peer resides on a different machine.

4 Routing Algorithms

Generally speaking, the purpose of all lookup algorithms is to find a peer, which is responsible for a given *Resource*. When looking up a resource we actually apply a given hash function on it, and look up the result, which is in the stretch $[0;1)$. Responsibility for a given resource means that this calculated value is in the stretch of that peer (between the peers



Length:6 path:[[Peer:0.2200000, type:null], [Peer:0.3000000, type:parent], [Peer:0.4000000, type:parent], [Peer:0.4500000, type:left child], [Peer:0.4700000, type:left child], [Peer:0.4900000, type:left child], [Peer:0.5000000, type:successor]] - alowed type:interface viceroy.MessageStatistics

Figure 1: The Viceroy Simulation

ID and its predecessors ID). Note that the peer is not guaranteed to hold this resource, but this is the only possible place to look for it. Thus, the properties of the Viceroy network guarantee that if the resource cannot be found in the peer that is responsible for it, the resource is not in the system.

We have investigated several routing (lookup) algorithms in the context of the Viceroy network and implemented a statistical program that compares the effects of different lookup algorithms on a given Viceroy topology. For a discussion on this analysis see Section 5. In the following subsections we briefly overview the currently implemented algorithms, and present our new algorithms. We give our motivation for each algorithm and explain the expected gain of each change over the original algorithm. As we will later discuss, some of the changes did not show an improvement in performance, but others shortened lookup path lengths drastically.

4.1 Paper Algorithm

The original algorithm presented in Viceroy [9] is a three-phase routing algorithm, which utilizes the unique structure of Viceroy. In the first phase, the lookup query is routed to a root peer (a lowest level peer) by using the **parent** links. This phase is known as Proceed To Root phase. In the second phase, known as Traverse Tree phase, the query traverses the tree by choosing either the **left child** or the **right child** links. This phase continues until the required child does not exist, or until it overshoots [1] the target. In the final phase, Traverse Ring phase, the lookup query traverses the ring using the **successor** and **predecessor** links until the peer responsible for the value is found.

[1]overshoots means: true if $\frac{1}{2^{(peer.level-1)}}$ < ClockwiseDist(peer,val)

In Figure 2 there is a pseudo-code describing the algorithm. The three parameters to the first phase is the current peer ('peer'), the looked-up value ('valueToLook'), whose hash-value can be calculated by any peer, and a boolean state parameter ('viaParent') with an initial value of 'true'.

The algorithm's performance is $O(log(n))$ hops until the destination is reached. The complete proof can be found at [9].

4.2 PapersPlus Algorithm

The PaperPlus algorithm is based on the Paper algorithm, but adds an optimization step that proved to be vital in the simulation. The optimization is a check that is being performed in the TraverseTree phase. There, before doing anything, it checks if the left child or the right child is responsible for the looked up value. If so, then without any additional checks, we go to the traverse ring phase with the child that is responsible for the sought value as its parameter. Figure 3 depicts the algorithm. This enhancement does not change the asymptotical behavior of the algorithm, but rather it saves a few redundant hops if the target of the lookup is accidently encountered. Therefore, this algorithm is guaranteed to perform at least as well as the original Paper algorithm.

4.3 TraverseTreePlus Algorithm

This algorithm (see Figure 4) introduces a change in the TraverseTree phase of the lookup algorithm: If one of the children exists, go to one of them in any case (even if they overshoot the target). Otherwise, go to TraverseRing phase. Currently, there is no threshold and the algorithm proceeds down the tree as long as there are children.

TraverseTreePlus tries to delay the transition to the last phase of the algorithm as much as possible, given that this phase (TraverseTree) does not use long-range links, and as such has a higher chance of adding many redundant hops to the route.

4.4 TraverseTreeAbsolute Algorithm

In this algorithm's Traverse Tree phase we choose the absolute closest node (the shortest between the clockwise and the counter-clockwise distances) to the target out of the current node and its children. If one of the children is closer to the target, then we go to this child and continue in this phase. Otherwise, we go to TraverseRing phase. Note that the original Paper algorithm uses a clock-wise distance for that decision.

```
ProceedToRoot ($AuthorizedViceroyPeerpeer$,
        $Resource\ valueToLook$, $boolean\ viaParent$)

if (peer.getLevel() $\neq$ 1)
    if ( viaParent && peer.parent() $\neq$ null)
        ProceedToRoot(peer.parent(), valueToLook,
            true)
    else
        ProceedToRoot(peer.parent(), valueToLook,
            false)
else
    TraverseTree(peer.parent(), valueToLook)
```

```
TraverseTree ($AuthorizedViceroyPeerpeer$,
        $Resource\ valueToLook$, $boolean\ viaParent$)
if( clockwiseDist(peer, valueToLook) < $\frac{1}{2^{level}}$ &&
            peer.leftChild() $\neq$ null &&
            $\neg$overshoot(peer.leftChild(), valueToLook))
    TraverseTree(peer.leftChild(), valueToLook)

else if ( clockwiseDist(peer, valueToLook) $\geq$
        $\frac{1}{2^{level}}$ &&
            peer.rightChild() $\neq$ null &&
            $\neg$overshoot(peer.rightChild(),
                valueToLook))
        TraverseTree(peer.rightchild(), valueToLook)

    else TraverseRing(peer,
        valueToLook)
```

```
TraverseRing ($AuthorizedViceroyPeerpeer$,
        $Resource\ valueToLook$)
if peer.isResponsibleFor(valueToLook)
    return peer
else
    if(valueToLook.isInStretch(peer, peer.successor())
        TraverseRing(peer.successor(), valueToLook)
    else
        TraverseRing(peer.predecessor(), valueToLook)
```

Figure 2: The Paper Algorithm.

The improvement here is supposed to be achieved if during the normal course of the search the current node finds that it is in the (absolute) vicinity of the looked up peer, or at least more close to it than its children. In that case, there is possibly a higher

```
TRAVERSETREEALTERED(AuthorizedViceroyPeer
        peer, Resource valueToLook)

if( peer.leftChild() $\neq$ null &&
        peer.leftChild().isResponsible(valueToLook))
    TraverseRing(peer.leftChild, valueToLook)

if( peer.rightChild() $\neq$ null &&
        peer.rightChild().isResponsible(valueToLook))
    TraverseRing(peer.rightChild(), valueToLook)

if( clockwiseDist(peer, valueToLook) < $\frac{1}{2^{level}}$ &&
        peer.leftChild() $\neq$ null &&
        $\neg$overshoot(peer.leftChild(), valueToLook))
    TraverseTreeAltered(peer.leftChild(),
            valueToLook)

else if ( clockwiseDist(peer, valueToLook) $\geq$
            $\frac{1}{2^{level}}$ &&
            peer.rightChild() $\neq$ null &&
            $\neg$overshoot(peer.rightChild(),
                valueToLook))
        TraverseTree(peer.rightchild(), valueToLook)

    else TraverseRing(peer, valueToLook)
```

Figure 3: The PAPER PLUS ALGORITHM.

```
TRAVERSETREEPLUS(AuthorizedViceroyPeer
        peer, Resource valueToLook)

if( clockwiseDist(peer, valueToLook) < $\frac{1}{2^{level}}$ &&
        peer.leftChild() $\neq$ null &&
        $\neg$overshoot(peer.leftChild(), valueToLook) )
    TraverseTreePlus(peer.leftChild(), valueToLook)

else if ( clockwiseDist(peer, valueToLook) $\geq$
        $\frac{1}{2^{level}}$ &&
        peer.rightChild() $\neq$ null &&
        $\neg$overshoot(peer.rightChild(), valueToLook) )
    TraverseTreePlus(peer.rightchild(), valueToLook)

else if ( peer.leftChild() $\neq$ null &&
        peer.rightChild() == null )
    TraverseTreePlus(peer.leftChild(), valueToLook)

else if ( peer.rightChild() $\neq$ null &&
        peer.leftChild() == null )
    TraverseTreePlus(peer.rightChild(), valueToLook)

else if ( peer.leftChild() $\neq$ null && peer.rightChild())
    TraverseTreePlus(child, valueToLook)
            //the child that
            //least overshoots. leftChild or rightChild

else TraverseRing(peer, valueToLook)
```

Figure 4: The TRAVERSE TREE PLUS ALGORITHM.

chance that a rigorous scan of the nodes would yield a shorter path, since TraverseTree might direct the query to more distant nodes. As this is only a small change we do not show the pseudo-code for this algorithm.

4.5 FindFast Algorithm

This greedy algorithm compares the absolute distances to the target of all possible inbound and outbound connections and chooses the closest connection. The lookup query is then forwarded to the closest node. Outbound connections are all the connections mentioned earlier (left/right child, successor, etc.), while inbound connections are those in which the current peer participates: if another peer chose a second peer to be its right child, this connection is considered an outbound connection for the first peer, and an inbound for the second peer. Note that the parent-child links are not symmetric, and if a peer was chosen to be a child of a second peer, it does not necessarily mean that the second peer is its parent.[2] Again, this algorithm does not use the structure of the Viceroy network at all, but rather it treats all links equally and only tries to find the closest link to the target. See Figure 5 for the complete pseudo-code.

5 Experimental Results

We have implemented and used a statistical program that compares the results of different lookup algorithms on a given Viceroy topology. Figure 6 shows the results of a simulation for a network containing 1000 peers. We plot the number of lookup steps needed to reach the target (the node responsible for the looked up resource) based on 200 lookups with

[2]Future versions of Viceroy would change this confusing notation

```
FindFast(AuthorizedViceroyPeer
        peer, Resource valueToLook)

if (peer.isResponsibleFor(valueToLook)
    return peer
else
    connections=peer.inboundConnections()+
    peer.outboundConnections()

return
    findAbsoluteClosestTo(connections, valueToLook)
```

Figure 5: The FindFast Algorithm.

FindFast and the Paper algorithm. Obviously, the lower size of path the better, and as can be seen in the figure, the paths of FindFast are shorter, and their variance is smaller. A commutative distribution function comparison of the two algorithms in Figure 7 gives a clearer picture of the distribution of path lengths - many lookup paths are extremely inefficient due to an immature transition to the last routing phase. This can be seen from the heavy tail distribution of the Paper algorithm. FindFast lookup results are not only better, but they also more dense, without the extremely non-optimal lookups that are occasionally found in the Paper algorithm.

Comparing the average path length (see Table 1) results shows a remarkable difference in favor of FindFast: 60.315 lookup steps were needed to reach the target in the Paper algorithm, while for the same network topology and the same number of lookups, which were performed with the FindFast algorithm, the average was 17.01 lookup steps. The median values were 25 for Paper and 10 for FindFast. The results for the PaperPlus show only a minor improvement in the performance, while TraverseTreePlus and TraverseTreeAbsolute show worse performance compared to the Paper algorithm. It seems that these two algorithms caused an immature abandon of the TraverseTree phase, which resulted in a longer, exhaustive, pass over the remaining nodes on the ring. This pass over the ring is the main reason for the performance degradation. As for PaperPlus, its performance improvement is achieved when the algorithm saves a few redundant hops if the target of the lookup is accidently encountered.

As can be clearly seen from the results, the performance of FindFast is remarkably better than the performance of the Paper algorithm. Note that the Paper algorithm is proved to find the target node after $O(log(n))$ hops. Achieving the same bound, or even a better bound on the performance, using a much simpler algorithm may help simplifying the discussion of properties of the Viceroy network. Moreover, the greedy nature of the algorithm makes it a stateless algorithm, which is simpler to develop and to deploy. Using FindFast in the Viceroy system would help to simplify Viceroy's behavior.

Lookup Algorithm	Average Path Size
Paper	60.315
PaperPlus	56.17
FindFast	17.01
TraverseTreePlus	319.67
TraverseTreeAbsolute	82.555

Table 1: Averege Path Size In a 1000 nodes Network And 200 Lookups

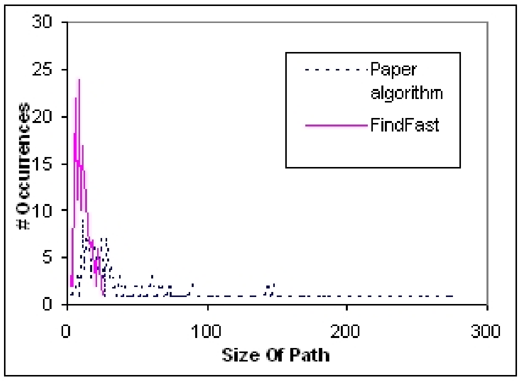


Figure 6: Paper Algorithm vs. FindFast

6 Conclusions And Future Work

Viceroy's original lookup algorithm, as well as some of our improved versions of it, utilize the unique structure of the system. Surprisingly, using FindFast, a simple, greedy algorithm that ignores the differences between the existing types of links (butterfly links, level links, ring links) achieves remarkably better results.

We believe that FindFast's usage of both inbound and outbound links is the main reason for its improved performance. Inbound links are maintained

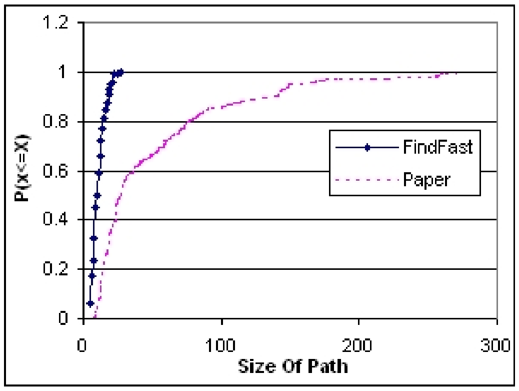


Figure 7: CDF for Paper Algorithm and For FindFast

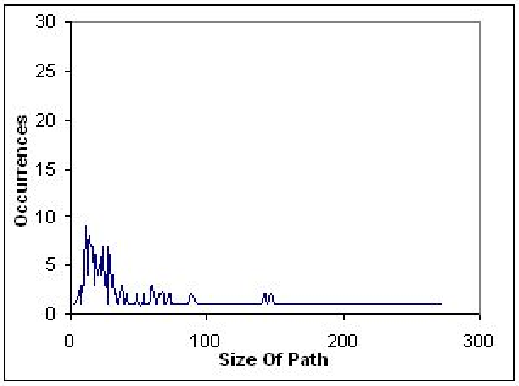


Figure 10: Paper Algorithm Results

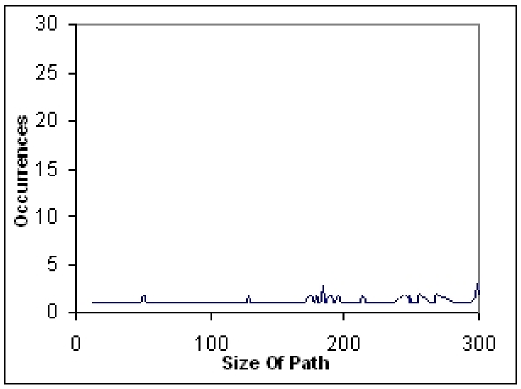


Figure 8: TraverseTreePlus Algorithm Results

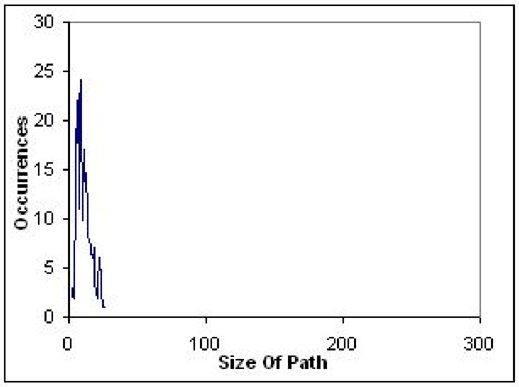


Figure 11: FindFast Algorithm Results

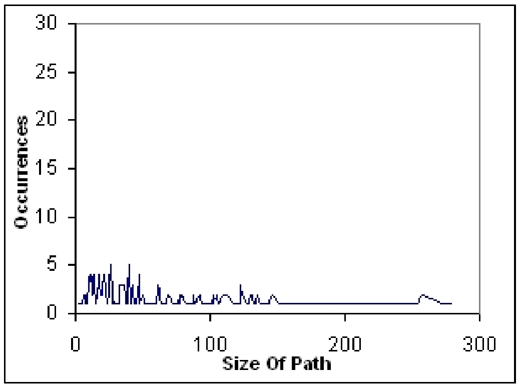


Figure 9: TraverseTreeAbs Algorithm Results

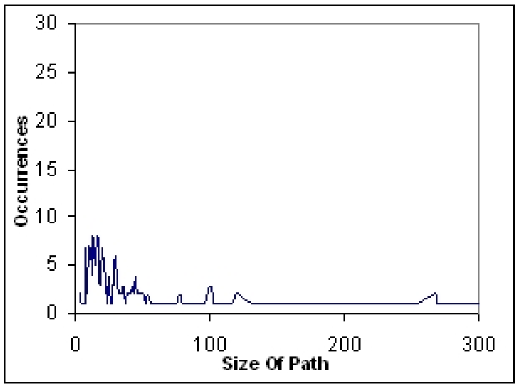


Figure 12: PaperPlus Algorithm Results

and kept alive with the same cost as outbound links, and therefore, an algorithm that does not utilize these links ignores a crucial resource. Therefore, we highly recommend to use FindFast in future versions of Viceroy instead of the original algorithm, Paper algorithm. The remarkably better results it achieves and its simplicity make FindFast superior over the Paper algorithm. A future work is needed for proving our algorithm's performance bounds, an analysis which would allow to perform a detailed comparison with DHT constructions postdating Viceroy, such as [1, 6, 12, 7]. We believe that the simplification and improved performance FindFast brings to Viceroy makes it a worthy alternative to these DHTs.

7 Acknoledgments

We would like to thank Dahlia Malkhi for the helpful guidance and also to the rest of the Viceroy team, in particular to Danny Bickson. Thanks also to Yoav Etsion for proof-reading drafts of this paper. Finally, we would like to thank the anonymous reviewers for their useful comments.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal D. Malkhi and E. Pavlov. “A Generic Scheme for Building Overlay Networks in Adversarial Scenarios”. In *International Parallel and Distributed Processing Symposium* (IPDPS 2003), April 2003, Nice, France.

[2] I. Abraham, D. Malkhi, and O. Dobzinski. ”Land: Stretch $(1 + \epsilon)$ Locality-Aware Networks for DHTs”. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA ’04), January 2004.

[3] L. Barriere, P. Fraingniad, E. Kranakis and D Kriziac. ”Efficient routing in networks with long range contacts”. In *Proc. 15th. Intl. Symp. on Distributed Computing*, pages 270-284, (DISC ’01), 2001.

[4] O. Dobzinski, A. Talmy. ”Viceroy - On The Implementation Of A Peer To Peer Network”. *Technical Report 2003-77, Leibnitz Center of the School of Computer Science and Engineering, the Hebrew University of Jerusalm*, September 2003.

[5] P. Drushel and A. Rowstron. “Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems”. *Proceeding of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[6] P. Fraigniaud and P. Gauron. “The Content-Addressable Network D2B”. Brief announcement in *ACM Symp. on Principles of Distributed Computing (PODC)*, July 2003.

[7] F. Kaashoek and D. R. Karger. “Koorde: A Simple Degree-optimal Hash Table”. In *2nd International Workshop on Peer to Peer Systems* (IPTPS ’03), February 2003, Berkeley, CA.

[8] J. Kleiberg. ”The small-world phenomenon: An algorithmic prospective”. In *Proc. 32nd ACM Symposium on Theory of Computing*, pages 163-170, (STOC ’00), 2000.

[9] D. Malkhi, M. Naor and D. Ratajczak. “Viceroy: A scalable and dynamic emulation of the Butterfly”. In *Proceeding of the 21 st ACM Symposium on Principles of Distributed Computing* (PODC ’02), July 2002.

[10] G.S Manku, M. Bawa, P. Raghavan. “Symphony: Distributed Hashing In Small World”. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[11] S. Milgram. ”The small world problem”. In *Psychology Today, 67(1)*, 1967.

[12] M. Naor and U. Wieder. “Novel Architectures for P2P Applications: the Continuous-Discrete Approach”. In *The Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA ’03), 2003.

[13] C. Plaxton, R. Rajaraman, and A. Richa. “Accessing nearby copies of replicated objects in a distributed environment”. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA ’97), pp. 311–320, June 1997.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker. “A scalable content-addressable network”. In *Proceeding of the ACM SIGCOMM 2001 Technical Conference.* August 2001.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan. “Chord: A scalable peer-to-peer lookup service for Internet applications”. In *Proceedings of the SIGCOMM 2001*, August 2001.

[16] B.Y. Zhao, J. D. Kubiatowicz and A.D. Joseph. “Tapestry: An infrastructure for fault-tolerant wide-area location and routing”. *U.C. Berkeley Technical Report UCB/CDS-01-1141*, April, 2001.

[17] `http://www.ece.cmu.edu/~atalmy/viceroy/`