Lookup Performance Impact of Replica Count in Kademlia

Oliver Kabi
Department of Electronics, Computing and Mathematics
University of Derby
o.kabi1@unimail.derby.ac.uk

***Abstract*—Distributed Hash Tables provide a structured method of organising peer-to-peer network resources. One such DHT protocol, Kademlia, is employed by BitTorrent, a popular file sharing network deployed across tens of millions of peers. In this paper we empirically study the effect of replication count on the performance of the key DHT operation, lookup, within our own implementation of Kademlia. Our results show that storage overhead can be increased in order to improve lookup performance of the DHT.**

I. Introduction

PEER-TO-PEER (P2P) applications are distributed networks of peers often without any formal centralisation or hierarchy. By utilising the resources of each peer a system can be created that scales with each new peer to join the network. In order for these networks of peers to effectively participate, overlay networks are utilised in order to organise content and manage peer discovery. Throughout this paper, the terms ”node” and ”peer” will be used interchangeably.

A. Unstructured Overlays

Unstructured overlays consist of a network of peers without any formal topology and loose rules about how peers join or distribute content. While this technique is effective for finding popular content, it's ability to find rare content is much more limited due to the small scope of searches, relative to the overall network size, and the approach is un-scalable as the load on peers increases linearly as the number of queries and peers in the system grows. Gnutella provides an example of an unstructured overlay through its use of flooding [1]. While simple, this method does not guarantee the successful look up of content or peers that resides within the network.

B. Structure Overlays

Structured overlay networks provide an evolution to content lookup in a distributed and decentralised network. In a structured overlay, the topology of the network is controlled and content is distributed to specific locations in a deterministic manner. Such systems use a distributed form of hash table, with peer identifiers corresponding to the keys of content. This structured approach allows peers to efficiently locate the location at which to store or retrieve any given key. In theory Distributed Hash Table (DHT) based systems can guarantee content look up in just O(log N) hops on average, where N is the total number of peers [2]. DHTs provide the ability to consistently assign identifiers that are both uniform and random to new peers within a large space of identifiers. Content objects are also provided keys from within the same space of identifiers. Content is then distributed to peers with identifiers either the same as that of its key or as close as is currently active within the network. DHT-based systems support the development of a large variety of internet applications, well beyond just file sharing, from naming systems (such as DNS) to application-layer multicast, due to their ability to provide scalable, wide-area retrieval of information [2].

While unstructured overlay networks such as Gnutella are resilient to peers joining and leaving the network, due to the lack of structure dictating where content will reside on the network, structured protocols face many more challenges and performance issues in the wake of this ”churn”. Upon leaving the network, the content that a peer was responsible for is no longer available meaning that content unique to this node would no longer reside within the network. There are two broad routes of solutions to this problem, the first being replication of content across more than a single node, providing a level of resilience to the irregular nature of most nodes, the second involves giving nodes the responsibility of offloading the keys (and related values) to the next peer best matching the distance metric of the given overlay network.

The intent of this paper is to assess the impact on lookup performance of varying the level of replication in an implementation of Kademlia. Section II presents an overview of the Kademlia protocol. In order to study the impact of replica count on lookup performance, we will create a basic implementation of Kademlia in which we will vary the number of replicas created across a range of swarm sizes, while recording the number of lookup hops taken, along with the time taken to execute the operations. We have provided an overview of our implementation design in section IV. Section V covers our testing methodology and analysis of both the results and the implementation itself.

II. Background

Kademlia is a widely adopted DHT protocol [3], implementations of which are used in applications such as BitTorrent [4], which is estimated to have between 15-27 million peers [5]. Kademlia uses consistent hashing to provide peers with 160-bit identifiers; this method is also used to generate keys

for content. Kademlia use a prefix-matching method similar to DHTs such as Pastry and Tapestry [6], [7] in which a XOR-based metric is used to determine the closeness of identifiers, allowing nodes to progressively locate closer and closer nodes to the target identifier.

Kademlia was designed with improvements based on observations made of existing structured overlays, one such observation being that the longer a node remains active the higher the likelihood it will remain active for at least another hour [8]; based on this observation, Kademlia places a preference on old nodes, allowing a node to, over time, find and favour more active nodes in the network.

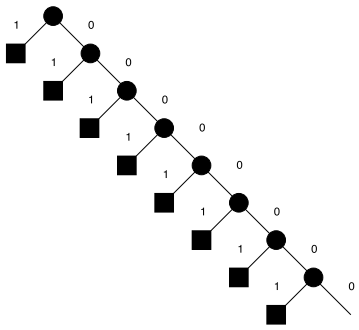

Fig. 1. Contacts are sorted into the tree-based routing table structure based on the bit-wise XOR of the contact's ID and the local nodes ID. The length of the zero-prefix of the XOR determines the height of the sub-tree the contact will reside within the routing table

A. Routing Table

Each Kademlia node maintains a routing table of contacts, other nodes that it has become aware of, in a series of buckets, each of which groups together contacts of a different distance relative to the local node. The distance between two identifiers is determined by the number of zeroes that prefix the bitwise XOR of the two identifiers. Using this XOR result, the distance between two distinct identifiers can be measured between 0 and *B*-1, where *B* is the size in bits of the identifiers used. For each value within the range of 0 and *B*-1, a *k*-bucket (with a predefined maximum size, *k*) exists in the routing table, at which contacts with the given XOR prefix length will reside, creating a tree-like structure as seen in Figure 1. Due to the nature of the XOR distance metric, it is expected that half of all peers would reside in the first bucket. Within a k-bucket, the contacts are sorted based on when each was last seen, with the least seen node at the top. Each time a message is received, the routing table is updated with the message originator, if it a new contact then it is added to end of the appropriate bucket, else an existing contact is moved to the end of the bucket it already resides within.

B. Lookup Procedure

Lookups within Kademlia use the same basic principles as other DHTs. The lookup locates successively closer nodes to the target identifier, *T*, in logarithmically many steps converging on *T*. The process begins with the querying node, *!*, selecting the α closest contacts from it's own routing table, which it then adds to a short-list followed by sending them a FIND_NODE request. The queried nodes respond to a FIND_NODE request with the k closest contacts from their own routing table, which *Q* adds to the short-list and then proceeds to send further FIND_NODE requests to the α closest contacts in the list. The query status of each node in the short-list is recorded, with each successfully responding node marked as contacted, and the routing table updated accordingly. This iterative process continues until the termination requirements have been met. For content lookup, this would be finding the content or failing to find any closer nodes in the responses from queried nodes. For node lookup this would be either a short-list of *k* successfully contacted nodes or as with content lookups, failing to find any closer nodes in successive iterations. The number of lookup steps is in theory limited to O(log *N*) where *N* is the number of nodes in the current Kademlia network. For store operations, a replica is stored at the *r* closest nodes returned by the lookup.

III. Related Work

Early work on DHTs introduced new methods for achieving O(log *N*) lookups, along with new techniques for combating the effects of churn [6], [9], [10]. It has been difficult to compare these various overlay networks directly with one another, due to the various tunable parameters of each adjusting the performance of the respective DHT in given environments. However Jinyang Li et al [11] developed a framework for comparing different DHTs with a performance-versus-cost metric. They observed that for a given bandwidth usage, there is a minimum lookup latency that can possibly be achieved for the entire space of tunable DHT parameters.

Medrano-Chvez et al studied the impact of peer online times for look up performance across Chord and Kademlia [12]. They found that Kademlia's fundamental preference for longer lived peers allowed it to increase dependability to a degree close to one, with only 10% of peers being long-lived. Medrano-Chvez also showed that even with long-lived "benefactors" the dependability of the system decreased with the increment of churn. They suggest that to provide a Quality of Service, Kademlia networks must guarantee a percentage of long lived nodes.

Stutzbach demonstrated experimentally the benefits of using parallel lookups, to reduce the number of hops required while also better dealing with "stale", or inactive, contacts within the routing table of a peer [13]. Their experiments on a real world network using Kad, an implementation of Kademlia, provided empirical results to determine optimal values for the tunable parameters of the protocol. Their results showed that the overhead of the network could be reduced while also increasing the performance of lookups.

Kang et al [14] showed that lookups found only 18% of nodes containing replicated data, revealing the inefficient use of resources within the network. The ineffectiveness of the lookup is shown to be caused by inconsistent lookup results due to high levels of similarity across node routing tables. Due to the similarity seen across node routing tables, very few new close nodes are found per lookup, resulting in a only a subset

of the closest nodes being returned. This leads to get and put operations being performed a distance from the true closest nodes.

Liu et al [15] later performed a similar study and found additional factors impacting the reliability of lookup. Lookup misses due to time-outs after already having found the replica roots but not having had time to perform operations on them as well as misses due to lookups that did not find the replica roots. They suggested changes to the way routing tables are maintained with a focus on the θ-neighbourhood of an identifier, which is defined as the region of space sharing a prefix length of theta. This ensured that nodes are aware of the nodes in their local region, allowing iterative lookups to more effectively locate closer and closer nodes to the target.

IV. System Design

There are two core models in the system, ID and Contact. An ID represents the 160-bit identifiers specified in the Kademlia protocol. The object contains the 20 byte array identifier along with the XOR distance metric logic. The Contact model combines the ID and networking address attributes of a node.

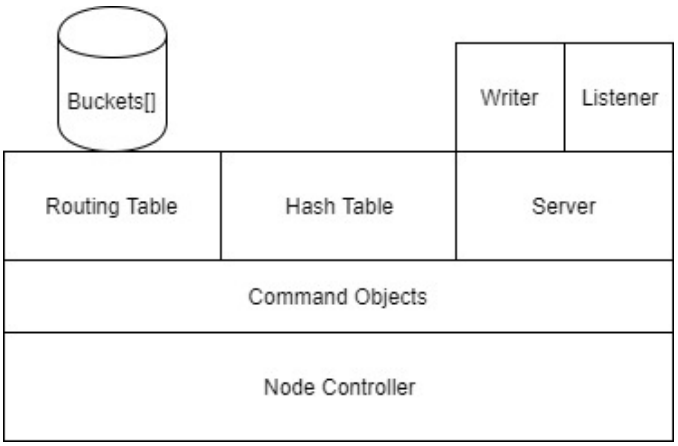

Fig. 2. System Architecture

The system exposes itself to users via the Node Controller, which in turn orchestrates the initialisation and operation of the node. The Node Controller is responsible for constructing the Routing Table, the Hash Table and the Server, along with injecting dependencies into these operational objects where required. The Controller utilises the Command Pattern to execute operations [16].

The Server is responsible for sending messages as part of operations on behalf of other objects, while also accepting and handling messages transmitted to the local node. Operation-specific implementations of the Command Receivers interface are also utilised by the Server's listener thread to handle incoming requests and responses.

For the purposes of our research, the system is not required to handle complex data types, as such we have opted for a simple data store implementation that consists of a hash table to map stored strings to their associated 160-bit identifiers.

The Routing Table is responsible for maintaining the k-Buckets along with providing the k closest contacts to a given target identifier.

V. System Analysis

A. Methodology

To determine the impact of varying the number of replica roots, r, we analysed five values of r, each across six swarm sizes, N. The values used for r and N are shown in Table I. Multiple swarm values were chosen so that it could be determined if the impact of altering r was uniform across different size networks. We simulated the network locally on a single machine using individual ports per node, using standard Java Datagram Sockets, as such negligible networking time was used in each operation.

TABLE I
VALUES TESTED FOR r AND N

r	1	5	10	15	20
N	100	1000	5000	10000	15000	20000

For each test case, a static address bootstrap node was used, along with the swarm of size N, with a final node running the scripted test actions. The actions taken for each test were as follows: The bootstrap node was started, followed by building up the swarm, one node at a time, each of which would perform a bootstrap process of sending a ping to the bootstrap node, followed by a lookup operation for itself, in order to identify its neighbour nodes. Once the swarm was at the desired capacity and activity stabilised, the test node would be started, following the same bootstrap operation as each swarm node. The test node would then perform a **PUT** operation on a generated unique string, followed by a **GET** operation on the ID returned from the previous **PUT** operation. This process of **PUT**, **RESET**, **GET**, was repeated 1000 times for each test case, each using a unique string. It should be noted, in order to ensure a full lookup on the **GET** operation following a **PUT**, a reset of the routing tables was performed, followed by running the bootstrap process. To clarify, the test environment was completely re-instantiated for each test case, with no persistence of content or state in the nodes . During the tests, no churn was present in the peer population of the network, as all simulated nodes remained active for the entirety of a single test case. For each **GET** operation performed, the number of completed FIND_NODE requests until the content was found was recorded, along with the execution time. Additionally, the success of each get operations was recorded in order to determine the successful lookup ratio for each test case. We configured Kademlia with the following parameter values. Keys were set at the standard value of 160 bits. The value of the K-buckets was set at 20. The value of α was set at 3. The values chosen were standard values suggested in the design specification used for implementation [17]. As these values were not part of the test cases, no variance was considered. While the introductory paper of Kademlia does not itself specify a recommended value of replicas, it is usually tied to the value of k, the size of routing table bucket, as that

provides the most efficient lookup relevant to the number of replica roots that will be created. We have varied the number of replicas independently from the size of k so that we can specifically look at the trade off between storage resources and operational resources. Varying the size of k to reflect the number of replicas required would also impact lookup operations for retrieval of content.

B. Results

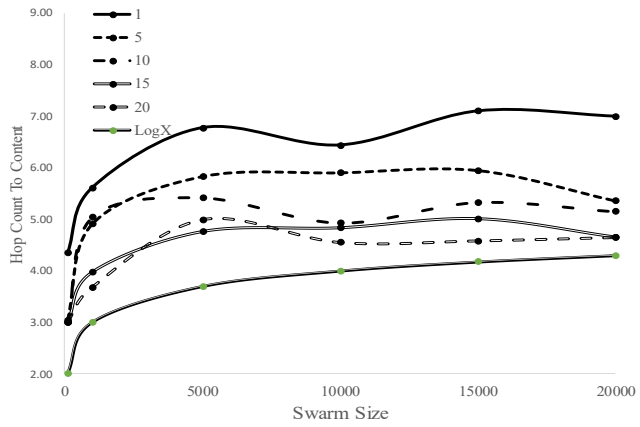

Fig. 3. The system follows the expect Log X curve with an added intercept, for the number of completed Find Node requests to other peers

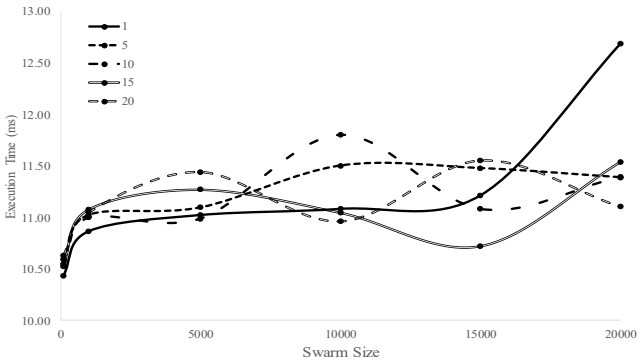

Fig. 4. Execution times for the get operation do not generally vary much across values of r, other than for r = 1 which shows a drastic increase for larger swarms

In Figure 3, 4 and 5 we show the results collected from our testing. For Figure 4 and 5, in the X axis of both is the size of the swarm, while the Y axis represents the execution time of the operations. In Figure 3, the X axis still represents the size of the swarm, while the Y axis represents the number of FIND_NODE requests completed in order to locate the content. The results for each test case were collated into a single average value for each, as such each value plotted represents an average across 1000 operations.

The results shown in Figure 3 are reflective of the complexity expected, log N, and also show the small performance differential of values of r between 10-20 in larger swarms.

In Figure 4 it can be seen that there is very little divergence in the execution times between each value of r. It should be noted that the increase in execution time, relative to swarm size, seems to be linear. As the **PUT** operation is built on the same lookup algorithm as a **GET** operation, it would be expected that the increase would be more closely reflective of the O(log N) complexity of that operation.

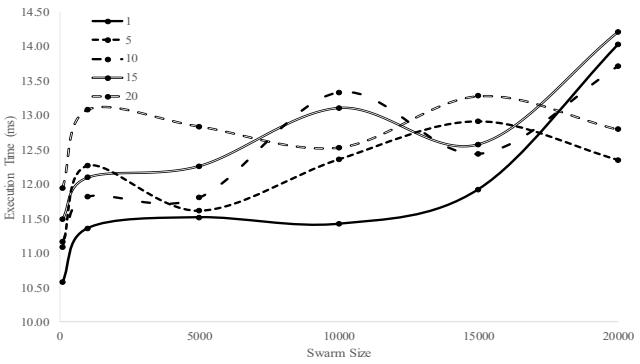

Fig. 5. Put operations follow a similar trend to get operations with execution times similar across the values of r

Figure 5 shows results that more closely follow the expected O(log N) complexity. It is clear that using only a single replica increases the execution time of lookup operations, due to the increased difficulty of finding a replica root. This increased difficulty is also reflected in the success ratio results seen in Table II where results for r = 1 had the only failures. However, curiously the most failures occurred with a swarm size of only 100, a size at which we would not expect there to be difficulty locating the replica root. For values of r greater than or equal to 10, it can be seen in Table II that there were no failures at all.

C. Analysis

Based on our results, replica counts as low as 10 could be suitable for networks of varying sizes. The performance of 10 replicas very closely follows that of 20, however the storage resources used would be half of that required for the latter.

TABLE II
NUMBER OF FAILURES OF GET OPERATIONS ACROSS ALL TEST CASES PERFORMED. IN OUR TEST ENVIRONMENT WITH NO CHURN, NO FAILURES WERE SEEN WITH VALUES OF $r \geq 5$

	1	5	10	15	20
100	20	0	0	0	0
1000	3	0	0	0	0
5000	2	0	0	0	0
10000	9	0	0	0	0
15000	6	0	0	0	0
20000	6	0	0	0	0
Total	46	0	0	0	0
Success %	95.4	100	100	100	100

TABLE III
LOOKUP HOPS TAKEN, ACTUAL & EXPECTED, FOR r = 20

Swarm Size	100	1000	5000	10000	15000	20000
Actual	3.00	3.69	5.00	4.55	4.58	4.65
Expected	2.00	3.00	3.70	4.00	4.18	4.30

It can be seen in Table II, the system showed itself to be able to provide a 100% success rate for lookups on unique items within networks of up to 20,000 peers when $r \geq 5$. Without network factors, it is able to perform an iterative lookup within 13 milliseconds. Table III shows the actual and expected lookup hops taken, the results of which are closer to the expected values than we anticipated prior to the testing. While this study is centred around Kademlia, we believe the results could be applicable to other DHTs that utilise replication to counter churn.

VI. Critical Review

Use of parallelism could impact the number of hops taken during a lookup to a small degree, particularly in instances where the querying node has few contacts populated in it's lookup table, as of the initial α FIND_NODE requests, only 1 may be needed to find closer contacts than the searching node was aware of.

Our testing was performed on a static network of peers, in which after the initial start-up no peers would join or leave the network, thus the system was free of churn. The levels of churn can vastly impact the performance of an Overlay network [18].

The Kademlia protocol specifies that contacts are only ever removed from the Routing Table if they fail to respond to a request, as such, the lack of churn in the system, also results in static routing tables because no nodes will ever be replaced by a more recently seen contact once K-Buckets have been filled. As seen in related works, this can impact the closeness of **GET** and **PUT** operations, relative to the true closest nodes [14].

The lack of churn could be addressed through the use of an Overlay Network simulator. A network simulator would provide the ability to simulate a more realistic environment and also the ability to test the implementation across academic testing networks such as PlanetLab [19].

While we have only used swarm sizes up to 20,000 nodes, some deployed kademlia based systems have swarms that number in the tens of millions, as such further testing with larger networks would be required to assess the impact of such tuning for these massive networks. The use of PlanetSim along with the PlanetLab network would provides an opportunity for such testing. Before such research could be undertaken, the system would first need further development in order to handle the refreshing/expiration of content, along with the heartbeat messaging between peers and their neighbours.

VII. Conclusion

This paper is based on the Kademlia Overlay Network Protocol and examines the impact of tuning the value of r to optimise lookup performance over network storage resources used. We first created a Java implementation of Kademlia, which we then used to test store and lookup operations on unique strings, across various swarm sizes up to 20,000 nodes. Our results show that it is possible to trade storage resources for better lookup performance by increasing the number of replicas created and distributed across the system. We showed that in a static network, a replica count of 5 or greater provided 100% successful lookup across all swarm sizes, however diminishing returns were seen as the replica count was increased past that point. With 20 replicas for each piece of unique content, the number of completed FIND_NODE requests more closely matched the expected results based on the theoretical O(log N) complexity expected of Kademlia lookup operations. Our results could be further improved by repeating the tests with varying levels of churn as this would provide results which would be more relevant to real world use.

References

[1] "The annotated gnutella protocol specification v0.4." [Online]. Available: http://rfc-gnutella.sourceforge.net/developer/stable/

[2] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.

[3] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[4] A. Loewenstern and A. Norberg, "Dht protocol." [Online]. Available: http://www.bittorrent.org/beps/bep_0005.html

[5] L. Wang and J. Kangasharju, "Measuring large-scale distributed systems: case of bittorrent mainline dht," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–10.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2001, pp. 329–350.

[7] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on selected areas in communications*, vol. 22, no. 1, pp. 41–53, 2004.

[8] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "Measurement study of peer-to-peer file sharing systems," in *Multimedia Computing and Networking 2002*, vol. 4673. International Society for Optics and Photonics, 2001, pp. 156–171.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*. ACM, 2001, vol. 31, no. 4.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[11] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil, "A performance vs. cost framework for evaluating dht design tradeoffs under churn," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 1. IEEE, 2005, pp. 225–236.

[12] A. G. Medrano-Chavez, E. Perez-Cortes, and M. Lopez-Guerrero, "On the effect of peer online times on the lookup service of chord and kademlia p2p systems," in *Communications (LATINCOM), 2013 IEEE Latin-America Conference on*. IEEE, 2013, pp. 1–6.

[13] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht." in *INFOCOM*, 2006, pp. 1–12.

[14] H. J. Kang, E. Chan-Tin, N. J. Hopper, and Y. Kim, "Why kad lookup fails," in *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*. IEEE, 2009, pp. 121–130.

[15] B. Liu, T. Wei, C. Zhang, J. Li, and J. Zhang, "Improving lookup reliability in kad," *Peer-to-Peer Networking and Applications*, vol. 8, no. 1, pp. 156–170, 2015.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *European Conference on Object-Oriented Programming*. Springer, 1993, pp. 406–431.

[17] "Kademlia: A design specification." [Online]. Available: http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

[18] S. Rhea, D. Geels, T. Roscoe, J. Kubiatowicz *et al.*, "Handling churn in a dht," in *Proceedings of the USENIX Annual Technical Conference*, vol. 6. Boston, MA, USA, 2004, pp. 127–140.

[19] 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, no. 3, pp. 3–12, 2003.