“Umbrella”: A novel fixed-size DHT protocol

A. D. Sotiriou, P. Kalliaras, P. Papageorgiou, N. Mitrou
Computer Network Laboratory, School of Electrical and Computer Engineering
National Technical University of Athens
9 Heroon Polytechniou Street, Zographou, 15773, Athens, Greece
{dsot, pkal}@telecom.ntua.gr, mitrou@softlab.ece.ntua.gr

Abstract—In this paper we present the Umbrella protocol, a novel distributed hash table architecture that supports key publishing and retrieval on top of an overlay network for content distribution. The Umbrella protocol provides a scalable environment with efficient algorithms based on a distributed routing table of constant size for each node, thus minimizing traffic load and scalability problems. Through our theoretical analysis and simulation results we prove the integrity of our algorithms and their logarithmic requirements as node population increases.

Keywords: P2P architecture, overlay networks, routing infrastructure, distributed hash-table.

I. INTRODUCTION

Recent developments in the telecommunication area have transformed the role of the internet. Even traditional players such as television or radio are now being challenged by the new entertainment media, the home computer. These changes have transformed users from simple consumers to receivers, editors and publishers of information. Numerous attempts have been made during the past, with perhaps the most common examples of Napster [1] and Gnutella [1]. With time, new systems emerged that quickly expanded and formed, along with other applications, the new field of Peer-to-Peer systems. Such systems benefit from the advanced processing, storage and bandwidth capabilities of everyday terminals. These systems must be able to cope with enormous amount of users, content and demands. A solution has been proposed in the form of Distributed Hash-Tables, which distribute users in a unique overlay network and route content distribution and discovery through advanced routing algorithms.

This paper proposes an alternative architecture for content distribution based on a new DHT routing scheme. The proposed architecture is well structured and self-organized in such a way as to be fault-tolerant and highly efficient. It provides users with content distribution and discovery capabilities on top of an overlay network. The novelty of our proposed architecture lies in its routing table which is maintained by each node and is of constant size as opposed to other algorithms that are proportional to the network’s size (usually \(O(\log N)\)). All operations in our architecture are of \(O(\log N)\) steps and degrade gracefully as up-to-date information of the routing table decreases due to numerous node failures. The rest of the paper is organized as follows. In chapter 2 we present related work and ideas that have been thoroughly studied prior to our architecture design. In chapter 3 we present our novel architecture and in the following chapter we provide in detail the algorithms invoked during our routing protocol. Chapter 5 discusses a number of results obtained through our simulation of the system. Finally, chapter 6 offers useful conclusions.

II. RELATED WORK

The firsts to introduce routing algorithms that could be applied to DHT systems were Plaxton, Rajaraman and Richa [3]. The algorithm wasn’t developed for P2P systems, and thus every node had a neighborhood of \(O(\log N)\) and inquires resulted in \(O(\log N)\) steps. It was based on the ground rule of comparing one byte at a time until all (or best compromise) were met. A key feature of their scheme was that the routing table could be transformed as thus the overlay distance between nodes could be of a constant factor of the real distance, when all latencies between nodes are known. Our scheme meets the logarithmic growth of inquires introduced by Plaxton, and even though nodes are not placed within constant distance from each other, this is not an issue as it was only implemented in a theoretical study as opposed to real conditions where knowledge of latencies between all nodes is both impossible and redundant in a p2p environment.

A variation of the Plaxton algorithm was developed by Tapestry [4], properly adjusted for P2P systems. The algorithm once again tackles one digit at a time and through a routing table of \(\beta*\log N\) neighbors routes to the appropriate node, resulting in a search of \(\log N\) maximum steps. Our architecture is based on the fundamentals ideas set by Plaxton and further developed by Tapestry, but is also fine-tuned for P2P systems that are likely to have an enormous amount of population and content, thus transforming the routing table into a constant sized table. Pastry [5] is similar to Tapestry but added a leaf set of neighbors that the node first checks before referring to the routing table. Also a different neighbor set is maintained for tolerability issues. Each node maintains a neighborhood of \(\log_{2\beta} N\) rows with \((2^\beta-1)\) elements in each row and requires a maximum of \(O(\log_{2\beta} N)\) steps for inquires. Proper routing is maintained as long as \((L/2)\) nodes are available in the neighborhood of each node. Once again, the variable size of each node’s table limits the algorithm’s scalability. In addition, our algorithm’s results showed that inquiries can be successful even with less available nodes in the routing table, increasing the system’s tolerability.

In Chord [6] a different approach was applied, placing nodes in a circular space and maintaining information only for a number of successor and predecessor nodes through a finger table. Routing is established through forwarding queries to the correct successor based on their identifier. Even though the basic Chord mechanism only requires the knowledge of one successor, modifications where needed in order for the system to be applicable to a robust environment, introducing a finger table of \(O(\log N)\) size.
Finally, CAN [7] furthered on Pastry’s alternation and implied DHT in a d-dimensional Cartesian space based on a d-tore. The space is constantly divided and distributed amongst nodes, which much maintain information about their neighbors and route by following the Cartesian space. CAN provides a constant O(d) table but, unlike our algorithm, requires O(d^n/d) steps for lookups, which grows faster than our logarithmic requirements.

III. SYSTEM ARCHITECTURE

A. Structure Overview

Our proposed architecture is based on the creation of an overlay network, where all inserting nodes are identified by a unique code, asserted by applying the SHA-1 [8] hash-function on the combination of IP and computer name, which returns an 160-bit identifier. This hash-function has been proven to distribute keys uniformly in the 160-bit space and thus provide the desired load balancing for both the user space and the content space, as the same function is applied to each content destined for distribution in the system. The main objective of the architecture is to insert and retain nodes in a simple and well structured manner, thus querying and fetching of content is both efficient and fault-tolerant. In addition, each node will need only to retain up-to-date information of a limited, constant number of neighboring nodes, such allowing the system to escalate in population of both users and content.

![Image](Umbrella Architecture)

The overlay network is constructed in the form of a loose B-Tree, where each node is placed in a hierarchy tree with a parent node and b child nodes, which in our initial architecture is of the value 16. All nodes are placed along the tree structure, without being required to fulfill predefined ranges as in a proper B-tree structure, and are responsible for updating their connections with neighboring nodes that reside on either the parent, sibling or child level. Thus, each node operates autonomously and no central coordination is needed to maintain the structure’s integrity. Along with obvious connections (parent, child and sibling level of each node), further links to a limited number of nodes in the near vicinity are kept in record for fault-tolerant operations.

![Image](Typical Neighborhood Table for Node 8L1)

Fig. 1. The Umbrella architecture

Routing in the umbrella protocol is simple and constitutes the forwarding of messages to either a parent or child node until the appropriate node is reached. In the rest of the paper, with the term appropriate node we will refer to either the exact or closest match alike.

Each level n of the structure is capable of withholding 16^n nodes. Each node has a unique parent node, which is always one level higher, and a maximum of 16 children at a lower level. The Umbrella overlay network is configured with the following simple rule. The relation between a parent node at level n and a child node (which must by default reside on level n+1) is defined as such and only such that:

- The n+1 first digits of the parent’s identifier are equal with the corresponding of the child’s.
- The n+2 digit of the child’s identifier determines the child’s position in the parent’s child list.

B. Routing Table

As in most DHT systems, a routing table is maintained by each node in order to route incoming messages. Each node is responsible for keeping the table up-to-date by issuing messages to all nodes in its table at different intervals. The routing table in our architecture consists of three different sets, a basic, an upper and a lower set. These three sets constitute the node’s neighborhood table and are presented in Fig. 2, with regard to node 8 of level 1 seen in Fig. 1.

![Image](Basic)

The above elements are sufficient to maintain proper routing in our architecture even in the case of sudden failure of nodes. The upper set allows routing to nodes of higher level (when the parent node is unreachable) and the lower set to nodes of lower level (when child nodes fail). Each node is responsible to modify or fix its routing table when nodes enter/leave the network or a failure to communicate with another node is detected, respectively. Our architecture’s structure and routing table described so far ensure that a published key can be located by an appropriate query within logarithmic overlay steps to the total size of the network. This is stated and proved within the following two theorems:

**Theorem 1.** Given an Umbrella network of N nodes with identifiers of base b acquired by a consistent hash function, the maximum height of the loose B-tree structure is of logarithmic scale.

**Proof:** Let b denote the base of our identifiers, N the total number of nodes and k a particular level in the Umbrella structure. Then according to the Umbrella
protocol, in each level a maximum of $b^k$ nodes can reside, with $b^{0}=1$ as stated for the first node that creates the network. Thus, if $m$ denotes the number of levels required for the above population of nodes, we acquire the following relation with high probability:

$$N = \sum_{k=0}^{m} b^k = \frac{b^0 - b^{m+1}}{1-b} = \frac{1 - b^{m+1}}{1-b} \Leftrightarrow m = \log_b \left( \left\lceil \frac{N}{b^{-1}} \right\rceil + 1 \right) - 1$$  

Thus the maximum height $m$ of our structure is $O(\log N)$. 

Theorem 2. A successful lookup in an Umbrella network requires, with high probability, $O(\log N)$ steps.

Proof : Suppose that a node $p$ that resides at level $l_p$ is seeking for a specific key $k$ that resides within our network in another node $f$ at level $l_f$. If $m$ denotes the number of levels of the current network, $N$ the nodes and $b$ the base of identifiers, then we could argue that the worst case scenario would require both nodes to reside at level $m$ and with maximum distance between them (thus node $p$ is a $m$-depth child of the first child at level 0 and on-and-forth and node $f$ is the $m$-depth child of the $b$ child at level 0 and on-and-forth). In this case, the lookup must first ascend all the way to the top of our structure (thus $m$ steps) and then descend to the bottom ($m$ steps again). In total, a maximum of $2m$ steps are required. Hence, from theorem 1, the required maximum steps for a successful lookup is, with high probability, of $O(\log N)$ steps.

IV. ALGORITHMS AND IMPROVEMENTS

A. Main algorithms

During the creation of the overlay network, the first node to enter creates the new network by placing itself on the top of the system. As new nodes arrive, they are placed according to their identifier, as described in previous section. A node only needs to contact an existing node in the system in order to be inserted (our architecture can embody any of numerous techniques for fetching existing nodes by outside contacts as proposed in [9]). The insertion mechanism is quite simple and consists of the following steps:

- Contacting a connected node and issuing an insertion request. The node checks if the $n+1$ first digits of the identifier match its own, where $n$ is the level it resides
- If not then the insertion message is forwarded to the node’s parent
- If yes then the message is forwarded to the child with the $n+2$ digit matching that of the new node
- If such a child does not exist then the new node is placed as a child to the current node
- The new node is informed of its neighbors and via versa

After the proper insertion, the parent node forwards all keywords that are more close to the new node’s identifiers for it to maintain. Both publish and search procedures are similar to insertion and thus are suppressed. The final mechanism provided by our protocol is that of node departure from the system. When a node issues a departure the following algorithm is followed, as shown in Fig 3.:

1. delete ()
2. if( has_kids() )
3. rand_kid = choose_random_kid()
4. if( rand_kid.has_kids() )
5. rand_kid.delete()
6. else
7. rand_kid.move_published()
8. rand_kid.copy_neighbors()
9. inform_neighbors( rand_kid )
10. disconnect()
11. else
12. this_node.father.move_published()
13. inform_neighbors( this_node.father )
14. disconnect()

Figure 3. Pseudo code for departure mechanism

B. Enhanced algorithms

The algorithms presented in the previous section embody the main mechanisms of our architecture and are capable of maintaining the system stable and fully functional under normal conditions. The system is however liable to sudden node departures, which we will call “node failures”. These are due to either voluntary departure without calling the appropriate mechanism or sudden departures due to client errors or network disconnections.

We treat all of the above cases in the same manner and through changes in the algorithms already presented we allow the system to bypass node failures. Most changes are based on using the upper and lower set of our neighbourhood table to bypass nodes that aren’t responding. The upper set is utilized to forward messages to nodes of a higher level while the lower set for nodes on a lower level. In the first case, when a node is unable to contact its parent node it attempts to forward requests consequently to the parent’s parent node, the node to the right and finally the node to the left of the parent node respectively. Whichever of the above succeeds first will terminate the mechanism. Similarly, in the latter case of a child node failure the corresponding nodes from the lower set are contacted until one succeeds.

C. Repair mechanism

In order to address the problem of node failures even further, we have designed a repair mechanism, which is invoked whenever such a failure is detected. The algorithm utilizes the delete algorithm presented in our main mechanisms section in order to repair a failure to a child node. It can be proven that all other failures can be transformed into a child failure through contacting nodes in the neighboring table and forwarding a repair message to higher or lower levels. Once the appropriate node is reached and informed of the child failure, a variation of the delete algorithm is evoked in order to repair the failure by substituting the failed node with one of its children or by deleting it if none is available. Each node is responsible for checking its neighborhood table periodically by issuing ping messages to all node entries and invoking the repair mechanism whenever a failure is detected. As will be shown in later simulation results, this mechanism increases the system’s stability and fault tolerance tremendously. In conjunction with the simple replication schema that has been implemented, it can support a high percentage of node failures without compromising its efficiency and simplicity.

V. SIMULATION RESULTS

In order to testify our architecture’s integrity and elaborate on its efficiency we have modeled our system and its algorithms using the neurogrid [10] simulator. This
tool allows us to simulate thousands of nodes, controlling all vital parameters. The neurogrid simulator, which we have extended, implements a number of core classes that provide objects such as keywords, documents, messages and nodes, and allows communication between nodes by implementing a message handler for each node, which serves requests serially according to our algorithms. All of our simulations were executed on a 3.2Mhz PC with 512Mb of RAM and based on Java.

A. Results without repair mechanism

In the first set of simulations we aim to prove the integrity of our design under normal conditions, testing all basic mechanisms. For this purpose we have conducted simulations with node populations varying from 10 nodes up to 6000 nodes. We firstly investigate the number of hops required for a successful insertion and lookup with a varying population of nodes. As is seen in Fig. 4, the number of hops grows logarithmically with the node population in both mechanisms. We can also observe that the required steps are even less in the case of the lookup operation, which could prove quite beneficial since a higher ratio of lookup queries is expected throughout the system’s life span. If we further analyze the results from the previous figures we will observe that the average hops required by each operation are given by $2\log_{10}N$. Thus it satisfies theorem 2 and only introduces a constant factor of 2. This factor is due to the way we move along the B-tree structure and also due to the loose structure (no algorithm for compressing the B-tree has been introduced yet).

![Figure 4. Average and max-min number of hops for (a) insert and (b) lookup operations](image)

Next, we investigated the overall traffic generated by our architecture in the form of total messages per request. As can be seen from Fig. 5 the system sustained a low number of messages exchanged, which was expected due to the small number of hops required for each successful request and also due to the limited (constant) number of neighbors maintained by each node. As can be seen, the total number increases linearly with the node population. For the case of 6000 nodes we see that only an average of 25 messages per node are required for the whole duration of the simulation, which is significantly low.

![Figure 5. (a) Total number and (b) per node average number of messages as a function of node population](image)

After completing our first set of simulations, we concluded that our algorithms worked as expected and provided 100% successful insertion, publication, lookup and departure mechanisms regardless of the node population. We therefore conducted a second set or simulations to test the system’s tolerability against node failures, without the use of the repair mechanism. A system degration as node failures increase was expected prior to the simulation runs, since we do not apply in this case any kind of patch for the failing nodes, but it is nevertheless important to observe our system’s response. Firstly we simulated a network of 1000 nodes, where 200 of them published their contents and progressively caused node failures from 0 up to 80% of the total node population, in steps of 5%, and in each case conducted 100 searches and reported the results. The success rate can be seen in Fig. 6(a). We observe that for a high rate of up to 22% of failing nodes, the success rate is kept high (over 80%). The system then slowly degrades up to a mid-point of 50%, where we have correspondingly an almost 50% success rate. We call this ratio the border-point, from where onwards our system becomes unstable and success rates drop dramatically, which can be considered quite high since no repair mechanism is issued.

![Figure 6. (a) Success rate of lookup operation and (b) cumulative results of successful lookups as the node failure ratio increases](image)

B. Results with repair mechanism

In our final set of simulations we tested our repair mechanism in the case of failing nodes in order to evaluate its effect on the success rate and the border-point. We conducted simulations with variant node populations from 1000 up to 10000 nodes and periodically issued random node failures in steps of 10% from 0 up to 80%. Our repair mechanism bypasses the failing node and allocates a neighboring node in its place, while the routing table is obtained through nearby neighbors, to the extent that is possible. Each node invokes the repair mechanism in two cases; either whenever a failure is detected due to inability of communication with a node in the routing table or during the course of a routing table consistency check,
which is issued periodically by each node. The former is constant and issued throughout our simulations. The latter varies as we have conducted simulations with different consistency check periods. In the results presented here we have varied this period by 3T, 6T and 20T, where T is a constant representing communication activity of each node (in our case T equals to 100 messages exchanged by a node). This ensures that an inactive node will not suffocate (in our case T equals to 100 messages exchanged by a node). This ensures that an inactive node will not suffocate. 

We firstly investigate the impact of the repair mechanism on the success rate of lookup operations. As seen in Fig. 7 the repair mechanism dramatically increases the success rate regardless of the node population and the check period. The protocol is able to sustain flawless operation for node failure rates even higher than 50% and a border-point is not reached even in the case of 80% failures. In the case of 10,000 nodes we can observe that node failures impact the system’s efficiency after 60% failure rate and even this results in a sustained performance of higher than 80% successful lookups up to a point of 75% failures. In the case of short check period (3T) the system performs lookups at a success rate of higher than 90% even when node failure reach 80%.

![Figure 7](image)

Figure 7. Successful lookups as a function of node failures for (a) 5,000 and (b) 10,000 nodes and check periods 0, 3T, 6T and 20T

In order to further understand the above results, we will approach the problem through probabilistic analysis. Let us consider a single node $n$ and its routing table. We have a node population of $N$, from which $D$ nodes reside in $n$’s routing table ($D \leq N$). If we cause $m$ node failures ($m \leq N$), thus remove without replacing them, we seek to find the probability of $k$ of these nodes belonging to $n$’s routing table. The above describes a hypergeometric distribution. During our simulations we have observed that an average of $D=8$ nodes reside in each routing table (regardless of node population increase). Let us examine the case of $N=10,000$ nodes and the probabilities of $k=0,1,\ldots,8$ for varying failure rates of 10% up to 80%. The results in Fig. 8 represent the probability of each node having none, one and up to all neighbors that reside in its routing table failing, with regard to the node failure rate. We observe that for a ratio of 60% failure, each node has about 6 nodes in its routing table failing. That is 75% of its routing table consists of failing nodes and still the repair mechanism is able to sustain almost 100% efficiency. Degradation of the protocol’s efficacy is observed for rates higher than 70%, where only one node in its routing table is not failing.

Finally, an important metric concerning the protocol’s efficacy is the total amount of messages exchanged between nodes. This was expected to increase with the introduction of the repair mechanism. In Fig. 9 we see that as the node population increases, the total amount remains almost constant for rate failures of up to 50% and increases linearly from then on. In all cases, the total messages per node average is kept reasonably low.

![Figure 8](image)

Figure 8. Probability of failures in routing table as a function of total node failure ratio

![Figure 9](image)

Figure 9. Total Messages as a function of node failures for node population (a) 5,000 and (b) 10,000 and check periods 0, 3T, 6T and 20T

VI. CONCLUSIONS

Through the course of this paper we presented the Umbrella architecture, a novel protocol based on a distributed hash table that supports key publishing and retrieval on top of an overlay network for content distribution. We have analysed our system and through both theoretical and simulation means proved its correctness and efficacy. Its main novelty lies in its fixed-size routing table sustained by each node, which is able to provide efficient routing in $O(\log N)$ steps even when more than half of the system’s population suddenly fails.

REFERENCES