# DECA: A Hierarchical Framework for DECentralized Aggregation in DHTs

Marc S. Artigas[1], Pedro García[1], Antonio F. Skarmeta[2]

[1] Universitat Rovira i Virgili,
Tarragona, Spain
{marc.sanchez, pedro.garcia}@urv.cat

[2] Universidad de Murcia,
Murcia, Spain
skarmeta@dif.um.es

**Abstract.** As Structured Peer-to-Peer (P2P) Networks become popular, there is an emerging need to monitor continuously the huge number of participants in a robust and scalable manner. To this end, aggregation has emerged as a basis for the self-management of these networks. However, the structured P2P networks lack today of efficient mechanisms for the decentralized computation of these aggregates. In this paper, we propose a hierarchical theoretic model based on Cayley Graphs, which overcomes the requisite to accommodate growth without impacting the efficiency of distributed applications. Also, the paper presents an aggregation protocol that fuses the fault-resilience of gossip algorithms with the scalability of trees. In particular, simulation results show that this algorithm is capable to cope with the distributed and unreliable nature of P2P networks.

**Keywords:** Structured P2P Networks, DHT, Aggregation, Hierarchical model, Gossip, Cayley Graphs.

## 1 Introduction

In recent years, Structured Peer-to-Peer (P2P) Networks, which offer an efficient, scalable, resilient and self-organizing substrate for building distributed applications have become widely popular, including Chord[1], CAN[2], Pastry[3] and Tapestry[4]. As these networks grow in popularity, there is an emerging need to collect a variety of statistical information about resources and/or peers to allow primarily individuals and then administrators to perform global control actions without explicit coordination. However, the P2P principles themselves pose a challenge for developing large scale management applications. Particularly, it is apparent that their decentralized nature should not be violated for any reason. Precisely, it is this need of decentralization what do P2P management systems be markedly different from traditional ones. In this line, we believe that the main challenge of P2P paradigm concerning management lies in *developing decentralized architectures capable to support up-to-date techniques without disregarding the symmetric functionality of peers*. To this end, a first natural step consists of enabling AGGREGATION, i.e. the computation of global network parameters through the use of functions MIN, MAX, SUM, COUNT or AVG, which is an efficient technique for provisioning nodes with valuable information.

A research area that can benefit substantially from AGGREGATION is autonomic computing: monitoring is an essential feature of autonomic systems and entails to log statistics about autonomic elements which serve as the basis for self-adaptation; self-healing, self-protection etc. Note that the essence of autonomic computing systems is self-management, the intent of which is to free system administrators from the details of system operation and maintenance.

The focus of this paper is on a decentralized model for computing AGGREGATION functions in Distributed Hash Tables, DHTs. Briefly, a DHT is a decentralized object-location mechanism for P2P systems that manages the distribution of content among a dynamic set of nodes by using a consistent mapping of keys to nodes. The DHT abstraction provides the same functionality as a hash table — associating key-value pairs with physical network nodes rather than hash buckets. They provide the *put*(key, value) and *get*(key) functions to allow DHT members to efficiently store and retrieve stored resources by *name* without using centralized servers. In practice, we consider that a nice AGGREGATION scheme for DHTs must fulfill the following:

i. *Accuracy*: let us consider a request to compute an aggregate function F. Then, "*accuracy*" refers to the maximum allowed error ($\zeta$) for the returned estimation $\hat{F}$. We define accuracy as $(1-\hat{F}/F_{true})$, where $F_{true}$ denotes the true value for function F. Then, if the answer lies in the range $[(1-\zeta)F_{true}, (1+\zeta)F_{true}]$, we say that the aggregation scheme ensures *practical validity*.

ii. *Scalability*: popular P2P networks maintain a large number of participants throughout the time. Consequently, a management protocol must scale to networks of very large size, that is, the load and the traffic generated by message exchanges must be small and evenly distributed.

iii. *Robustness*: the participants of a DHT are expected to be very dynamic. This means that our protocol must adapt gracefully to changes in the overlay, including node and link failures.

In this paper, we propose DECA: a hierarchical management framework capable to augment ordinary DHTs with robust, accurate and scalable AGGREGATION facilities.

Significant contributions distinguish DECA from previous work. These are:

− *Hierarchical P2P-theoretic model* that effortlessly exploits the in-built recursive decomposition of ordinary DHTs. The main reason is that hierarchies are ideal for accommodating growth and isolating faults. To this end, we introduce a powerful tool based on Cayley graphs which converts an overlay — like Chord, Pastry, CAN ... — in a collection of clusters mimicking a hierarchical organization. Hereinafter, we refer to this as function ΄H. Although in [5] we devise a pragmatic technique for constructing hierarchical DHTs, however, ΄H extents are broader. It endeavors to uncover the hidden hierarchical structure of typical DHTs. Also, we assume that there exists an effective mapping tool that groups topologically close nodes into the same cluster. It is our position to consider that topology concerns are of paramount importance for a practical P2P AGGREGATION service. To the date, we believe that DECA is the first attempt to build an effective management infrastructure built on top of a hierarchical DHT.

− *Efficient* AGGREGATION *protocol* that blends together the *scalability* of *trees* with the *resilience* of *epidemic algorithms*. To compute a global parameter, a system must provide redundancy to ensure practical validity. Node failures must not lead to severely hampering management operation. Hence, a gossip protocol is a better alternative than a single tree to compute function F over the weights of all nodes in the system.

The paper is organized as follows. Section 2 reviews related work. Section 3 describes formally our hierarchical model and AGGREGATION algorithm. Section 4 provides a discussion about function ́H. Section 5 provides Whirl, the Chord instance of our framework. Section 6 describes our simulation results. Finally, Section 7 draws some conclusions.

## 2 Related Work

Although there exists a large body of literature in the network management area, we believe that DECA is the first attempt to build a P2P distributed AGGREGATION [6, 7] mechanism based on hierarchical distributed hash tables, DHTs.

To the best of our knowledge, there are only a few proposals that try to solve the node aggregation problem in DHTs. In [8, 9, 10], a tree structure is constructed and maintained to propagate aggregates to the root. Except SOMO[10], fault-tolerance is achieved in these approaches by a mechanism that reconstructs the tree after node addition, removal or failure. Ji Li et al. in [8] demonstrated analytically that even with the presence of a refreshing algorithm that corrects failures; tree infrastructures cannot ensure practical validity. Besides, tree-based aggregation schemes share another important difficulty. Since link and node congestion increase as the distance to the root decreases, the scalability of the whole system becomes tightly coupled not only to the nodes capacities, but also to the actual tree organization of participants.

Willow[11] and DASIS[12] built a logical binary tree on top of a P2P system to provide participants with aggregation facilities. Whereas Willow is a general purpose aggregation service, DASIS employs aggregates to load balance the underlying peer-to-peer graph. In contrast, DECA is a multilayered hierarchical model which offers an efficient dissemination infrastructure; it organizes participants in a hierarchy of self-contained clusters which are, in practice, locality-aware overlays.

GAP[13] is characterized by the construction and maintenance of a BFS tree on top of the network and consequently, it suffers from the same aforementioned drawbacks.

The most similar abstraction in spirit to ours is Astrolabe[14]. In short, Astrolabe is a distributed management service designed to monitor and report continuously the state of a collection of dynamically changing resources to users. To do it, it organizes the resources into a hierarchy of domains, called *zones*, and associates attributes with each zone. A *zone name*, which is unique and describes its position in the hierarchy, is given to each zone to globally identify it. Similar to us, Astrolabe uses aggregation and a gossip protocol for quickly spreading changes throughout the system. However, it presents three important shortcomings. It is not self-organized since the hierarchy is implicitly defined when the administrators name the zones. It is semi-decentralized as each zone elects a set of hosts, called *representatives*, to gossip on behalf of the zone and it is not fault-tolerant. With only one representative per zone, Astrolabe is highly sensitive to host crashes.

## 3 A Hierarchical Management Framework

To meet the scalability demands for distributed AGGREGATION, we propose a hybrid technique that blends together the scalability of trees with the resilience of epidemic algorithms.

### 3.1 Hierarchical Model

We are considering a dynamic P2P overlay graph $G(V(t), E(t))$, where $V(t)$ is the set of vertices at time $t$, and $E(t) \subseteq V(t) \times V(t)$ is the set of edges that may change over time. Each node $x$ has an associated value $w_x(t)$ at time $t$. It represents the value that is being subjected to the AGGREGATION function F.

Because G is a DHT, it has a finite $m$-bit identifier space of $2^m$ elements denoted as I. Besides, G is also a hierarchical substrate recursively built up from a *proper nesting* of clusters. By a proper nesting, we mean that for any pair of clusters in G, the two clusters are either disjoint, or one is a proper subset of the other. Technically, the latter means that our architecture defines a partial-order tree; nodes are organized into clusters, clusters are organized into superclusters, superclusters are organized into hyperclusters etc. As a result of this partial ordering, the set I is at tier-0, the highest tier, whilst in any subsequent tier-$k$ provided $k > 0$, the potential number of identifiers for each cluster falls off. This occurs because each cluster $C \in G$ is a proper subset of I, that is, $C \subset I$. In other words, there are not duplicate identifiers since $I \supseteq G$ and $C \in G$. To conclude, let $L$ denote the number of tiers.

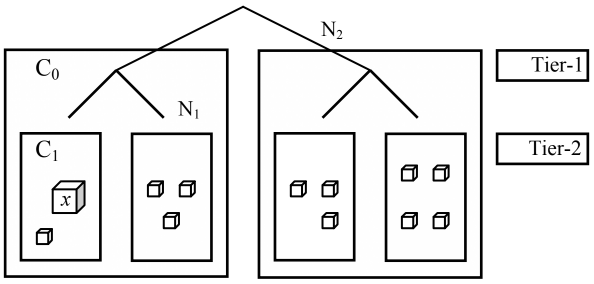


**Fig. 1**. A DECA hierarchy

**Node status**. Each node $x$ (that is a peer) is contained in a sequence of telescoping clusters: $C_{l-1}(x) \subset C_{l-2}(x) \subset \ldots \subset C_0(x)$, for some $l \in \{0, \ldots, L-1\}$ and where $C_{l-1}(x)$ denotes the $x$'s leaf cluster and $l-1$ its tier depth. For each $C \in G$, we say that C is a leaf cluster if and only if C is not a union of a finite number of other sets in G. In other words, any node is part of a raising sequence of larger groups up to the root. To retain the routing capabilities of the flat design, each node $x$ requests for some routing information at each tier-$k$, for $k > 0$. Particularly, it is ΄H responsibility to provide an efficient hierarchical substrate equivalent in degree and diameter to the flat design. Later in section 4, we discuss in more detail the ΄H involvements.

Let $N_i(x)$ be the set of tier-$i$ subtrees not including node $x$. Let $N$ be the collection of sets $N_i(x)$ for all $i \in \{0, 1, \ldots, L - 1\}$. Then, for each subtree $S_j(x) \in \{N_i(x) \mid \forall N_i(x) \in N\}$ node $x$ knows at least one node, namely, $x'$ in $S_j(x)$. Such a node $x'$ is said to be the $x$'s delegate node in $S_j(x)$. In fact, a delegate node $x'$ can be viewed as a kind of bridge since it lets a pair of nodes exchange its weights $w(t)$ at a given time $t$. Because function ´H strips G into smaller graphs of the same "*family*", each node $x$ has at least one delegate in each height-$k$ subtree, $k > 0$. The latter is advantageous in that nodes joining or leaving require only local changes in the network. Besides, it isolates faults; it enables effective bandwidth utilization, adaptation to the underlying network and a scalable network management. We note that no global information about the structure of the hierarchy is necessary; it suffices for each node to know its current position in the hierarchy and the list of ancestor groups up to the root. To ease the membership management, we assume that each cluster has a unique group identifier.

When a node $x$ joins the system, $x$ asks an arbitrary existing node, say $y$, to determine the closest node to $x$ — using a topological aware mapping. Denote this closest node by $z$. Node $x$ then initializes its routing table with $z$'s routing table. Let *x's* routing table be defined as follows. Let $D$(x) be the set of $x$'s delegate nodes. Let $L(x)$ be the $x$'s routing table for its leaf cluster. Then, the $x$'s routing table consists of $D(x) + L(x)$ nodes. For robustness to node failures, a "diversity" property should be maintained across the routing tables of nodes in the system. To accomplish this goal, it suffices for each node $x$ to apply the maintenance rules of the flat design along the sequence of telescoping clusters: $C_{l-1}(x) \subset C_{l-2}(x) \subset \ldots \subset C_0(x)$. Therefore, maintenance of the hierarchical network is relatively simple and almost identical to the flat overlay.

- **Network Awareness**. As mentioned above, one of the most challenging questions facing DHTs is whether they can compute aggregates for real-time management. To do it, a common technique consists of integrating the so-called "*proximity*" into the target DHT. In DECA, this means that nodes that are topologically close are organized into clusters; topologically-close clusters are gathered into *superclusters* etc. Because the cluster *nestings* come directly up from the underlying topology, DECA can assume an asynchronous distributed model i.e., a known upper bounds on message transmission delays, and clock drift rates. Hence, we can integrate all the above bounds in a known universal maximum delay $\Delta$ between any pair of nodes. A message sent a time $t$ will be received by the destination node within time $t$ to time $t + \Delta$. Also, we can assume that the communication time between two nodes within its leaf cluster is smaller than $\Delta$. This method may introduce errors but exact synchronization is not necessary, so gossiping fully suffices.

### 3.2 Node Aggregation

In this section, we address the fundamental question regarding DECA: "*What is the actual protocol used to calculate the global function F over data residing at the nodes of a DHT*?" We answer this question proposing a hybrid protocol that combines a gossip-based dissemination mechanism with a tree-based propagation structure that reflects the hierarchical organization of groups. First, a node "*gossips*", within its leaf cluster, about the individual weights it knows about. Then, it computes an estimate of F in a bottom-up fashion using the estimates it receives from *delegate nodes*. As G is expected to materialize into a *proper nesting* of clusters, our algorithm requires about $O(h)$ phases to compute F, where $h$ is the height of the tree.

**Protocol overview.** The major aim of DECA is to cope with the inaccuracies found in aggregates and provoked by the transient nature of P2P networks, where a significant fraction of nodes become inactive within a short period of time. In such settings, a pure tree-based approach tends to be rather unpractical since a single node failure can cause the root to miss the information of the whole subtree below the fault. In order to achieve more fault-tolerance, we need more redundancy in messages sent. Thus, gossip protocols, which are simple and fault-tolerant — though, at the cost of a higher number of messages —, constitute an attractive alternative. Fortunately, we believe that our scheme far affords the expenses of carrying out such form of massive data dissemination. Recall that clusters are expected to be small and topologic-centric. So scalability is achieved through the distribution of AGGREGATION across the clusters.

Regarding the flow of information, we distinguish between *push*, *pull* and *push-pull gossip protocols*. Assume a node $x$ calls node $y$. Then, we have:

i. In *push* gossip, the rumor is *pushed* if $x$ tells $y$ the rumor.
ii. In *pull* gossip, the rumor is *pulled* if $x$ requests $y$ for the rumor.
iii. In *push-pull* gossip, the rumor is both *pulled* and *pushed*.

These protocols are reliable in a probabilistic sense. Karp et al. [15] show that if a *push-pull* gossiping is run for $O(\ln n)$ rounds, then, *w.h.p.*, all nodes have the rumor, and in addition, the total number of messages sent is $O(n \ln \ln n)$.

In order to bound the communication cost, we assume a *push* gossip algorithm that can be described as follows. Each node $x$ receives rumors for $O(\log n)$ *epochs*, where an epoch is a fixed time interval of length $\Delta$ (network delay for clusters is $O(\Delta)$). In each epoch, each node $x$ gossips to $\delta$ other nodes, randomly chosen from the subset of nodes $x$ knows about. We will refer to this subset as $x$'s local view $\Gamma_x$. Views may be partial and inconsistent but large enough to ensure a fast convergence.

In *push* gossip, nodes gossip at a constant rate in each round, and therefore, the number of messages sent is $O(n \log n)$. However, we can reduce such overhead by tailoring the choice of targets to the underlying graph topology G, but a more general strategy applicable to any P2P topology is desirable.

To conclude this section, we describe how a node $x$ obtains the estimate for level $i$, for $(i > 0)$. We call this method GET_ESTIMATE($i$, F). Let $S_i(x)$ be the set of all height-$i$ subtrees not including $x$. Let $\psi_{i,j}(x)$ be the estimate corresponding to subtree-$j$ at height-$i$ not including $x$. Then, procedure GET_ESTIMATE($i$, F) consists of two steps. First, it obtains the estimates $\psi_{i,j}(x)$ from all $x$'s sibling subtrees $S \in S_i(x)$ through the corresponding delegate nodes. Finally, it applies function F over these estimates.

After these preliminaries, we are ready to describe how our algorithm manages to compute F.

**Aggregation algorithm**. DECA algorithm starts "*simultaneously*" at each node. The algorithm consists of 3 phases:

1. *Phase* 1: this phase lasts $O(\log\ max\{|C_k|\}_{k \geq 0})$ epochs, where $|C_k|$ denotes the cardinality of leaf cluster $k$. In order to estimate $O(\log\ max\{|C_k|\}_{k \geq 0})$, it suffices for nodes to use function MAX over the cardinalities of leaf clusters. In this phase, each node $x$ "*gossips*" about the individual weights stored across its leaf

cluster, which of course include $x$'s local weight. To do it, node $x$ *i*) *periodically* (once in every epoch) selects a random subset of nodes from its local view $\Gamma_x$ and *ii*) sends them an arbitrary weight chosen uniformly at random along with the identifier of the node that keeps it. In turn, $x$ discovers the weights of other nodes of its own cluster the first time it receives them via a *push* message. After $O(\log max\{|C_k|\}_{k \geq 0})$ epochs, $x$ applies the function F to all weights it has collected in its cluster, and bumps itself to phase 2.

2. *Phase i* ($1 < i \leq h + 1$): in phase $i$, each node $x$ invokes GET_ESTIMATE($i$, F) to obtain the estimate for tier-$i$. Then, $x$ bumps itself to phase $i + 1$. Note that any node has not available the estimate for its height-$i$ subtree until phase $i$ terminates.
3. *Final phase*: when a node x finds itself in phase $i = h + 1$, it has an estimate of the global function F evaluated over the entire DHT. Then, the protocol finishes at $x$.

**Time Complexity.** The number of phases for the algorithm is $h + 1$, where $h$ is the height of the hierarchy. Besides, let $\lambda$ be the set of leaf clusters. Since phase 1 lasts $O(\log c)$ rounds, where $c = max\{|C|: C \in \lambda\}$, the time complexity for the algorithm is $O(\log c + h)$.

**Message Complexity.** The communication cost is $O(bn \log n + bn)$, where $b$ is the size in bits of the weights and $n$ the number of nodes in the network.

## 4 The Hierarchical Function ΄H

In general, the problem of finding a suitable ΄H is complex. In [16], Ganesan et. al. provide a framework to transform a variety of DHTs into their hierarchical versions. As a part of our ongoing research, we use instead Cayley graphs, a common technique in design of interconnection networks, to devise the exact ΄H for a given input overlay. Briefly, Cayley graphs are extremely helpful in the analysis of static topologies with regards to quality measures such as diameter or degree — our hierarchical DHTs must preserve the same degree and routing performance as the flat designs. In particular, many Cayley graphs, such as hypercubes, are hierarchical. Further, if an algebraic theoretic model that can elucidate the hierarchical structure of a graph exits, then to procure it a proper ΄H is quite simple: *upon an overlay is proven to be a Cayley graph, it suffices to use the standard definition of hierarchical Cayley graph to approximate it.* Although many Cayley graphs are hierarchical, not all of them admit a recursive decomposition into smaller graphs of the same family. More specifically, we are only interested in those DHTs that are recursively built up by adding isomorphic copies of smaller *Cayley* graphs. A hierarchical DHT is not just a graph, but rather a family of graphs $G_0$, $G_1$, $G_2$ etc. defined for any of the potential static sizes.

DEFINITION 1. Let $\Gamma(V, \circ)$ be a finite group, 1 its neutral element, and let $S \subseteq V - \{1\}$ be closed under inversion (i.e. $s^{-1} \in S$ for all $s \in S$). The Cayley graph $G(\Gamma, S) = (V, E)$ of $(V, \circ)$ is the graph on $V$ where $x$, $y$ are adjacent if and only if $x \circ y^{-1} \in S$. In other words, there is an edge $(x, y)$ if and only if there exists a generator $s \in S$ such that $x \circ s = y$.

Cayley graphs include a large number of families of graphs, like hypercubes, star and pancake graphs [17] etc. The next definition constitutes the core of our scheme to build up hierarchies from the ground:

DEFINITION 2. Let $< \{s_1,s_2,\ldots,s_i\} >_\Gamma$ be the subgroup of $\Gamma(V, \circ)$ generated by the set $\{s_1,s_2,\ldots,s_i\} \subseteq S$ i.e. the smallest subgroup of $\Gamma$ which contains $\{s_1,s_2,\ldots,s_i\}$. Let $\Gamma(V, \circ)$ be a finite group and $S \subseteq V - \{1\}$ such that $S = S^{-1}$. Then, the Cayley graph $G(\Gamma,S)$ is said to be hierarchical if there exits an ordering $\{s_1, \ldots, s_k\}$ of the generators of G such that the subgroups $< \{s_1,s_2,\ldots,s_i\} >_\Gamma$ are all distinct.

The above definition yields a surprising outcome. If we order the generators such that each $s_{i+1}$ is outside the subgroup generated by the subgroup $<\{s_1, s_2,...,s_i\}>_\Gamma$, then we can obtain an accurate approximation of ´H. Specifically, each $s_{i+1}$ incorporates the additional edges required to interconnect the exact number of copies of graph $G_i$ to produce the next family graph $G_{i+1}$. Therefore, Cayley graphs give us a useful hint to the question of how to get a suitable method to obtain hierarchical substrates from flat topologies. Although Cayley graphs are very helpful, they only give a static solution to the problem of hierarchical construction. In fact, adopting a concrete instance of ´H involves a further analysis to determine if ´H properly fits in a dynamic environment where the nodes join or leave independently. However, this topic is beyond the scope of this work.

To conclude, we want to underlie that in our case study, the hierarchical version of Chord we obtain via ´H preserves the logarithmic bound $O(\log N)$, both in degree and number of routing hops.

## 5 Whirl: The Hierarchical Version of Chord

In this section, we concisely present the hierarchical version for Chord. First, we give the Cayley graph definition for Chord. Second, we discuss the specific function ´H which manages to map Chord to *Whirl*, our hierarchical version of Chord. Since a detailed description of Whirl was provided in [5], we omit the irrelevant details from network management viewpoint.

Let $\Gamma$ be the cyclic group $(\mathbb{Z}_{2^m}, +)$ of $2^m$ elements with generators $2^i$ for $i=0\ldots m-1$. The Cayley Graph $G(\Gamma, \{\pm 2^i: i\in\{0,\ldots,m-1\}\})$ is the Chord graph with diameter $m/2$ and degree $2m$.

The next theorem claims that Chord is a hierarchical Cayley graph. It also sets the order for the generators $\pm 2^i$, for $i=0\ldots m-1$, which reveals the in-built ´H for Chord:

THEOREM 1. Let $\Gamma$ be the cyclic group $(\mathbb{Z}_{2^m}, +)$ of $2^m$ elements. Let $G(\Gamma, \{\pm 2^i: i\in \{0,\ldots,m-1\}\})$ be the Cayley graph of Chord. Then, Chord is a hierarchical overlay if and only if its generators are ordered as follows: $\pm 2^{m-1}, \pm 2^{m-2}, \ldots, \pm 1$.

PROOF. Omitted due to space constraints.

The above definition theorizes that a Chord graph of $2^m$ elements can be viewed as two interleaved copies of a Chord graph of $2^{m-1}$ nodes with the additional connections linking adjacent vertices. For example, consider the cyclic group $(\mathbb{Z}_8, +)$ for a Chord graph of 8 vertices $\{0, 1, ..., 7\}$ and degree 6. The ordered set of generators for the *Cayley* graph is $\{\pm 4, \pm 2, \pm 1\}$. Then, the Chord graph of 8 vertices can be obtained by applying the generator $\pm 1$ to two copies of 4 vertices, one with vertex set $\{0, 2, 4, 6\}$, the other with vertex set {1, 3, 5, 7} and both with generators {±4, ±2}. It is easy to notice that the vertex sets of both copies are left cosets of group ($\mathbb{Z}_8$, +). Concretely, this leads to a more general implication. It signals that the union of two copies of the graph $G_i$ to produce graph $G_{i+1}$ entails to establish only one additional link per node. Technically, the latter is the reason why Whirl is optimal regarding both degree and diameter. For further details, refer to [5].

Bearing in mind the above facts, function ʹH for Chord is two-fold: (*i*) *it retains the standard Chord's rule for link creation* and (*ii*) *performs a simple operation on node identifiers*. The reason to maintain Chord's rule is for enabling "*hierarchical*" lookups in Whirl. In general, a hierarchical lookup works as follows. Suppose a node *q* looks for an item *k*. First, *q* tries to find *k* within its leaf cluster to take advantage of network proximity. If *q* finds *k* the search stops. Otherwise, the query reaches the closest predecessor *p* of *k* at this tier. Then, node *p* switches to the next higher cluster and continues routing on that cluster. By repeating the latter, item *k* finally is found.

Besides, ʹH divides the node identifiers in two parts. A PREFIX of $m - p$ bits and a SUFFIX of $p$ bits provided $0 \leq p \leq m$, where $m$ is the length of node identifiers within Chord circular identifier space $[0, 2^m)$. The SUFFIX determines the cluster of a node (*cluster* ID) whereas the PREFIX, drawn uniformly at random, specifies the identity of a node inside the actual cluster (*node* ID). Then, the application of the generator ±1 to any pair of neighboring clusters with an *m*-bit identifier space, produces a larger cluster with a ($m$+1)-bit identifier space and "*ring*" links between adjacent vertices. Also, the generator ±1 installs the delegate nodes at that tier. As a result, merging a pair of clusters of the same level is achieved easily by *i*) *contracting the SUFFIXes* and *ii*) *extending the PREFIXes* of nodes one bit, respectively. The last result has a broader scope: it lets nodes reuse the links of "lower" clusters for routing at "higher" tiers. As simulation results demonstrate in next section, our theoretic model is capable to elucidate a hierarchical *degree-optimal* version of Chord.

## 6 Simulation Results

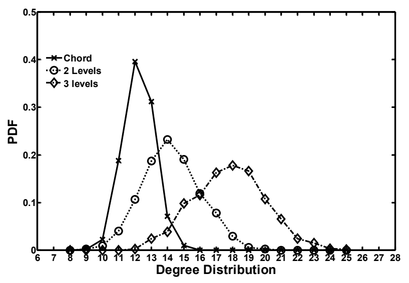
We now present a battery of tests to evaluate the functionality and performance of our framework through simulation. Because the underlying substrate is a Chord overlay, maintenance costs are of the same order as Chord. The main hypotheses we want to evaluate are:

1. *The basics of Whirl*: the degree distribution and average number of routing hops to demonstrate the quality of function ʹH.
2. *The proximity*: the capacity that offers a hierarchical DHT for adaptation to the underlying network in comparison to its flat design.
3. *The performance of DECA*: it includes the convergence time for a tailored *push* gossip algorithm, applicable to Chord; the communication cost in terms of number of messages sent and the accuracy of our solution.

All of the experiments for this section use a hierarchy with a fan-out of 2 at each internal node. The number of tiers in the hierarchy varies from 1, plain Chord, to 5.

The number of nodes oscillates from 1K to 4K, and all the nodes choose a random 14-bit identifier. Nodes are evenly distributed across the leaf groups.

**΄H Test.** Our first set of experiments evaluates function ΄H abilities. In particular, the *distribution of the number of links* and *the average number of routing hops* are provided. Regarding the first metric, figure 2.a) plots the distribution of number of links for a 1K-node network in a 1-level (Chord), 2-level and 3-level hierarchy. We observe that for Chord this distribution is peaked around the average of 12 links/node. As the number of tiers in the hierarchy increases, the mean shifts to the right whilst the distribution "*flattens out*" to the left of this value. Although the number of links is not exactly the $\log|V|$, we see that the average number of links is $\log|V| + c$, i.e. $O(\log|V|)$, where $c$ is a small constant that depends on the number of tiers. On the other hand, figure 2.b) depicts the average number of hops required to route between two nodes as function of the network size. We see that the number of routing hops is extremely close to $\log|V|$ irrespective of the number of tiers. To evaluate the above metric, we inserted 1K items into the system. Later, ten nodes were requested to retrieve all items. For each item, a "*hierarchical lookup*" was issued and the number of hops it spent, accounted. In conclusion, DECA through function ΄H is capable to produce hierarchical versions of flat DHTs that preserve the same degree and number of routing hops.



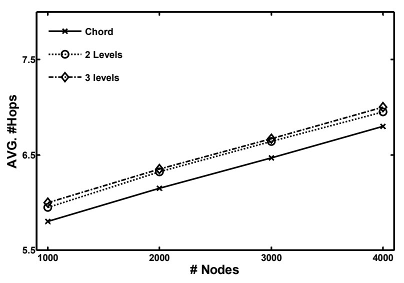


**Fig 2. a)** Number of links distribution **b**) Average Number of Routing Hops

**Proximity Test.** In this test, we evaluate DECA routing performance in terms of network delay. To make a fair comparison, both the flat Chord and its hierarchical version have the same number of nodes, denoted $|V|$. In order to measure DECA adaptation to the physical network, we use GT-ITM[18] topology generator to produce a 100-node highly connected *backbone*. For each node in the *backbone*, we attach a number of graphs representing *stub* domains. The latency weights are: 10ms for *backbone* edges, 100ms for *backbone-stub* edges and 5ms for *stub-stub* links. To construct the desired $|V|$-node network, we attach each node to a *stub* through a 1ms-latency edge. Figure 3 depicts the relative performance of an arbitrary 3-level, 4-level Whirl graph versus a plain Chord without proximity. Clearly, it illustrates that Whirl widely outperforms Chord.

**Performance Test.** This test evaluates DECA performance in terms of *convergence time*, *number of messages* sent and degree of *accuracy*. To this end, we implement a *push* gossip protocol over Chord partially based on the foundations of *lbpcast* [19]. Each node $x$ maintains a partial view $\Gamma_x$ of the system that varies as node $x$ discovers other nodes via *push* messages. Also, we employ COUNT function for our analysis.

Basically, this test aims to count the total number of nodes in the overlay. For all the tests, the probability of a node crash (without recovery) in every epoch and in every phase *i* (*i* >0) is *pf* = 0.001.

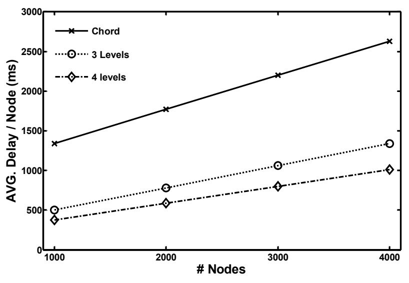


**Fig 3**. Average delay between nodes

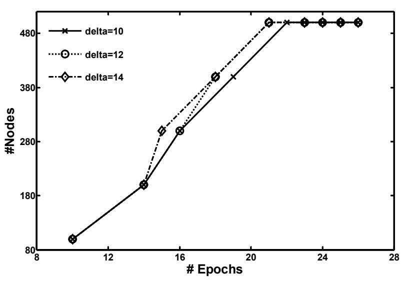
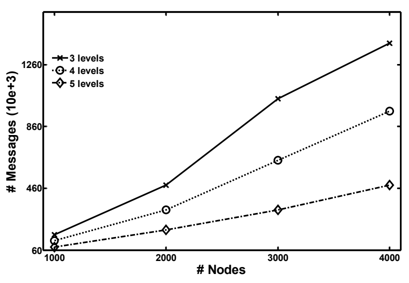


**Fig 4**. Convergence Time.

Figure 4 shows the relationship between the "*gossip*" rate δ and the number of rounds that it takes to aggregate an event in a leaf cluster. The figure shows that increasing δ decreases the number of necessary *epochs* to aggregate a value, but conveys also the fact that the gain is not proportional. Figure 5.a) illustrates the cost in number of messages that DECA spends. Note that leaf clusters are expected to be several orders of magnitude smaller than a flat DHT. Then, it is easy to see that as the average depth of the hierarchy increases, the overhead sharply reduces. Finally, figure 5.b) presents the fraction of nodes whose estimation of COUNT function is outside the range [0.9|*V*|, 1.1|*V*|]. The high degree of *accuracy* comes from: *i*) the high degree of reliability provided by the *gossip* protocol and *ii*) from the small quantity of delegate nodes required to aggregate. The advantage of using delegates is that the maintenance algorithm of the flat design carries out their refreshing.



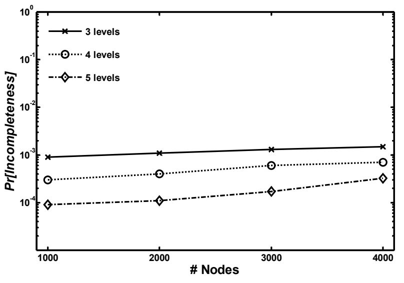


**Fig 5. a)** Number of messages sent (10e+3) **b)** Incompleteness probability

## 7 Conclusions

In this paper, we have expressed the need for a scalable AGGREGATION framework for DHTs as a basis for wide-area management architectures. We have argued the reason why traditional approaches for solving this problem do not scale in large groups, and do not perform well over fault-prone networks. To cope with this, we have sought for new theories that fit neatly into Peer-to-Peer paradigm foundations. In this line, we have introduced a hierarchical abstraction that manages to extend AGGREGATION to P2P wide-area networks. Also, a hierarchical methodology based on Cayley Graphs, which produces hierarchical systems from the usual *flat* DHTs, has been unveiled. Finally, an AGGREGATION algorithm which is hybrid, since it combines the fault-resilience of gossip algorithms with the scalability of trees, has been carefully devised to achieve the scalability/fault-tolerance requirements of large-scale P2P networking.

## References

1. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet application". In ACM SIGCOMM, 2001.
2. S. Ratsanamy, P. Francis, J. M. Hellerstein, and S. Shenker. A scalable content-addressable network. In ACM SIGCOMM, 2001
3. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In IFIP/ACM Middleware, 2001.
4. B. Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. IEEE Journal on Selected Areas in Communication, 22, 2004.
5. M.S. Artigas, P. García, J. Pujol, A. F. Skarmeta. Cyclone: A novel design schema for Hierarchical DHTs. In Proc. 5th Conf. On P2P Computing, 2005.
6. S. Madden, M. J. Franklin , J.M. Hellerstein, W.Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. SIGOPS Operating System Review 36 (2002).
7. M. Bawa, H. García-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. In *Manuscript*, 2003.
8. Ji Li, Karen Sollins, Dah-Yoh Lim. Implementing aggregation broadcast over distributed hash tables. In ACM SIGCOMM, 2005.
9. G.Doyen, E.Nataf, and O.Festor. A hierarchical architecture for a distributed management of P2P networks and services. In Proc. of DSOM'05, 2005.
10. Z. Zhang, S.-M. Shi, and J. Zhu, SOMO: Self-Organized Metadata Overlay for resource management in p2p DHT. In Proc. IPTPS'03, Feb. 2003.
11. R. Van Renesse, and A. Bozdog. Willow: DHT, Aggregation, Publish/Subscribe in One Protocol. In Proc. IPTPS'04, Feb. 2004.
12. K. Albrecht, R. Arnold, and R. Wattenhofer. Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave. In Proc. 4th Conf. On P2P Computing, 2004.
13. M. Dam and R. Stadler. A Generic Protocol for Network State Aggregation. In Proc. Radiovetenskap och Kommunikation (RVK), 2005.
14. R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst., 21(2), 2003.
15. R. Karp, C. Schindelauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In Proc. 41st IEEE Symp. on Foundations of Computer Science, 2000.
16. P. Ganesan, G. Krishna, H. García-Molina. Canon in G major: Designing DHTs with hierarchical structure. In Proc. ICDCS, 2004.
17. Akers, S.B., Krishnamurity, B.: A group-theoretic model for symmetric interconnection networks. IEEE Trans. Comput. Vol. 38, 1989.
18. E.Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In Proc. INFOCOM96, 1996.
19. P. Eugster, R. Guerraoui, S.B. Handurukande, A.-M Kemarrec, P.Koutnetsov. Lightweight probabilistic broadcast. In Proc. Conf. Dependable Systems and Networks (DSN), 2001.