SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems

Kalman Graffi, Aleksandra Kovacevic, Song Xiao, Ralf Steinmetz
Technische Universität Darmstadt, Multimedia Communications Lab KOM
Merckstraße 25, 64283 Darmstadt, Germany. Email: {graffi,sandra}@kom.tu-darmstadt.de

Abstract

In order to ease the development and maintenance of more complex P2P applications, which combine multiple P2P functionality (e.g. streaming and dependable storage), we suggest to extend structured P2P systems with a dedicated information management layer. This layer is meant to generate statistics on the whole P2P system and to enable capacity-based peer search, which helps the individual functionality layers in the P2P application to find suitable peers for layer-specific role assignment. We present in this paper SkyEye.KOM, an information management layer applicable on DHTs, which fulfills these desired functionality. SkyEye.KOM builds an over-overlay, which is scalable by leveraging the underlying DHT, easy to deploy as simple add-on to existing DHTs and efficient as it needs O(log N) hops per query and to place peer-specific information network wide accessible. Evaluation shows that SkyEye.KOM has a good query performance and that the costs for maintaining the over-overlay are very low.

1 Introduction

The field of peer-to-peer (P2P) research is broadening in recent years, ranging from classical overlays and content distribution, to multimedia streaming, dependable storage with replication management, distributed computation and many other functional layers. With the growth of application areas for the P2P paradigm, more and more mature solutions are presented (e.g. BitTorrent [3] instead of Napster). However, current P2P applications often focus only on a few or single functionality, e.g. Skype [9] searches and connects users in an unstructured overlay, file sharing applications mostly enable to lookup file providers, Zattoo [13] offers media streaming but no user interaction.

P2P applications combining various functionality on one P2P host are still rare. Imagine an application in which you can search (unstructured overlay) or lookup (structured overlay) specific content, which you can download (content distribution) or is directly streamed (P2P streaming). After consuming the content, you may add a comment to the specific content according to your role (security), that is then replicated (replication) and synchronized (versioning) according to specific criteria.

In this paper we present SkyEye.KOM, an information management (IM) over-overlay applicable as a further layer (IML) on any DHT. SkyEye.KOM generates statistics on the whole P2P network and provides the functionality of capacity-based peer search, finding a set of peers with desired capacities. We believe, that these are two challenges that need to be addressed in order to enable complex multifunctional P2P applications.

System Statistics: Current P2P applications do not reveal statistics on the status of the network. We argue that metrics on overlay performance, network topology and peer load can be measured by the P2P application itself using a dedicated information management layer (IML). Statistics presenting e.g. the average traffic load per peer, the standard deviation and even confidence intervals may reveal limitations in the protocol and support P2P application designers to improve the mechanisms applied.

With an IML providing statistics on the P2P network in real-time, interested parties (e.g. P2P application providers) can calculate their costs more precisely, developers can detect limitations in their protocols faster and the system itself could apply self-optimizing mechanisms.

Capacity-based Peer Search: Combining various functionality in one single P2P application, states challenging requirements on the efficiency of each functionality layer. As example imagine a P2P application for decentralized simulations offering efficient job dispatching, remote computation and replicated storage of the results. The first functionality requires peers with high bandwidth capacities, the second peers with high CPU and memory capacities and the third peers with large storage space and long expected online times.

Instead of having each functionality layer individually looking for appropriate peers fulfilling the desired requirements, one dedicated information management layer should provide them with the peer IDs of suitable peers. A functionality layer may ask e.g. for the contact information of 5

peers which have at least 200KB/s upload capacity on average, have been online for 5 hours, and have at least 10Mb available storage space.

An IML providing the functionality of capacity-based peer search enables the building of complex P2P applications in which specific tasks are assigned to capable peers. The load of information gathering is taken from the various P2P functionality layers, so that focus shifts from how to obtain the information to how to use the information.

In Section 2 we present the assumptions and goals for building an IML. Our solution, SkyEye.KOM, is presented in Section 3 in detail. We described the evaluation setup and results in Section 4. In Section 5 we discuss related work, and conclude our work in Section 6.

2 Assumptions and Goals

In this section, we summarize the key aspects of the problem statement for building an information management layer (IML) for structured P2P systems. Goal of the IML is to provide statistics on the P2P network and enable queries for peers regarding their capacities.

We state following assumptions for the IML. A P2P host maintains a DHT which provides the functionality *route(key, msg, nextHop)*. This function is also defined by Dabek et al. in [4]. The function enables a node in the IM over-overlay (IMOO) to send a message to a node which is responsible for a specific key in the DHT (which may represent a role in the P2P network). We further assume that the DHT layer provides information on the keys a peer is responsible for in the DHT. A peer should know in a DHT, whether it is responsible for a specific key or not. This paper does not discuss security issues, we assume protocol-compliant behavior of the peers.

Building an IML states various requirements on the quality of the solution. The architecture should fulfill the following non-functional requirements. The IM over-overlay should scale, both in regard of the number of peers and the number of peer attributes. A robust IMOO should apply mechanisms to overcome peer failure and churn. The load for maintaining the IMOO should as much as possible be balanced on all peers participating in the P2P network. The heterogeneity of peer capabilities (CPU, memory, bandwidth ...) should be taken into account, by allowing each peer to specify a maximum load to tolerate. With this, stronger peers can contribute more and weaker peers are not overloaded. The traffic and computational overhead of the IMOO has to be small, as it is meant as extension to future's complex P2P applications, not as main application.

3 SkyEye.KOM - Our Approach

In this section we present SkyEye.KOM, our approach for an IML, that gathers information from the peers in the P2P network and provides statistics on the system and the functionality of capacity-based peer search.

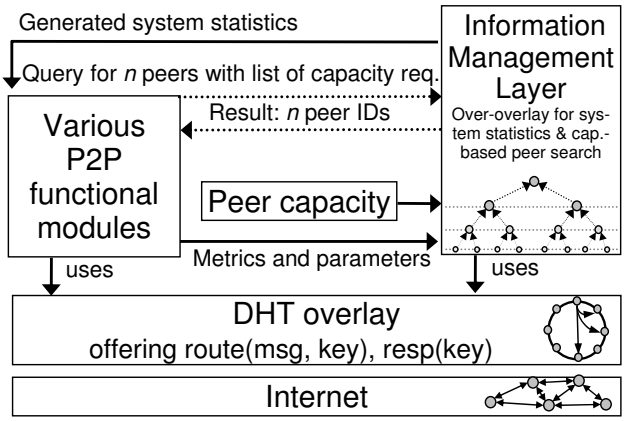


Figure 1. SkyEye.KOM as Over-overlay

SkyEye.KOM implements the IML as an over-overlay using the route and lookup functionality of the underlying DHT, as depicted in Figure 1. The clear interface makes it independent from the specific DHT used. The ID space is recursively partitioned in ID intervals called *Domains*. For each Domain, a characteristic ID is calculated using a deterministic function that maps the ID interval to a single ID in it, called *Domain Key*. The peer responsible for the Domain, called *Coordinator* of the Domain, is identified by being responsible for the Domain Key in the DHT. The recursively partitioned Domains and with this the corresponding Coordinators build a b-tree. Peers identify their position in the tree based on their ID, and send periodically information messages to the Coordinator one level above them in the tree. These messages, called updates, contain both information on the individual peer capacity and aggregatable statistics information. Coordinators periodically pass the aggregated statistics and the list of peer capacities one level higher in the tree in a push-based manner. Having this core-tree for information gathering, peers can send queries regarding a set of peers with specific capacities to their Coordinators, which forward the query up the tree, until one Coordinator has information on the required set of peers. Any Coordinator can be asked for network statistics on the peers in his Domain. In order to relieve the load, Coordinators may choose more capable *Support Peers* from their Domain and dispatch all update and query load to them. Having some Coordinators dispatching their demanding duties to Support Peers results in an easy to maintain support-tree with peers capable to fulfill the requested task of information management.

3.1 Architecture of SkyEye.KOM

SkyEye.KOM is an overlay on top of an underlying Distributed Hash Table (DHT) using the routing and ID-mapping functionality of the DHT.

Let $p \in S_{ID}$ be a peer ID and S_{ID} the ID space, then there exists a subset $S_p \subseteq S_{ID}$ so that peer p is responsible

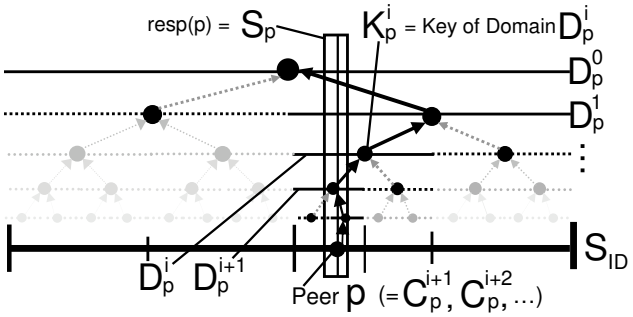


Figure 2. Definitions used in SkyEye.KOM

for all (object) IDs/keys in that set. Following counts

$$\forall p, q \in S_{ID} \ : \ p \neq q \ \rightarrow S_p \cap S_q = \{\} \tag{1}$$

We define the responsibility function $resp$ as follows:

$$resp: \ S_{ID} \rightarrow S_{ID} : o \rightarrow p \ with \ o \in S_p \tag{2}$$

Please note, that by using the core functionality of a DHT we build an over-overlay, which is applicable on any DHT that offers these to functions. Figure 2 depicts the definitions we introduce in this section.

In order to aggregate the information of individual peers, we establish a tree structure in the over-overlay. The tree is built by recursively segmenting the ID space S_{ID} in intervals (which we call *Domains*) and assigning a responsible peer to each Domain, which we call *Coordinator* of the Domain. The depth of the tree is O(log N). Each level of the tree aims at storing the information on all peers in the ID space, but with increasing tree depth the information is shared on more peers. A Coordinator is in charge to maintain the information of all the peers, whose IDs are in its Domain. However, by setting specific thresholds on the load capacity, the peers are not overloaded.

We define a *Domain* as continuous interval D_*^l in the ID space S_{ID}. Domains at the same level l in the tree do not overlap. Let $p \in S_{ID}$ be a peer and D_p^l be a sequence of Domains containing p with l as level counter. Then following counts

$$\forall p \in S_{ID} \ : \ D_p^0 = S_{ID} \tag{3}$$

$$\forall l \in \mathbb{N} \, \forall p \in S_{ID} \ : \ p \in D_p^l \tag{4}$$

$$\forall l \in \mathbb{N} \, \forall p \in S_{ID} \ : \ D_p^{l+1} \subseteq D_p^l \tag{5}$$

$$\exists k \in \mathbb{N} \, \forall l \in \mathbb{N} \, with \, l \geq k \, \forall p \in S_{ID} \ : \ D_p^{l+1} = D_p^l \tag{6}$$

Each Domain is maintained by a Coordinator, the Coordinators of the various Domains establish the tree by sending each other information updates. Two approaches exist how to choose the Coordinator of a Domain: using stateless allocation to a peer responsible for a specific ID or dynamic assignment based on peer capacities.

In our solution we combine the best of both solutions. We build a core-tree using a deterministic function which provides the ID of the Coordinator responsible for a peer ID. The function comes with no maintenance-overhead as any peer can locally calculate which other peer it has to contact. Coordinators in the core-tree can pick supporting peers from the Domain they are responsible for and dispatch load to these *Support Peers*. With time, a second support-tree with more capable peers is established which carries the load dispatched from weak peers in the core-tree.

For the following, we resume to describe the principles for establishing the core-tree and describe details on load balancing and using heterogeneity of peers in Section 3.3.

In the core-tree, each peer p identifies, using the deterministic function, its Coordinator(s) for the Domains D_p^l, that contain the peer's ID. A Domain of level l (e.g. $[i_a, i_b]$) is partitioned in b (Sub-)Domains of level $l+1$ (e.g. $[i_a, i_1], [i_1 + 1, i_2], [i_2 + 1, i_3], ..., [i_{b-1} + 1, i_b]$), with this the Domains build a b-tree structure. We map Domains to peers, that become then Coordinators of the Domain, using the responsibility function $resp$. A specific ID in the Domain, called *Domain Key*, determines the Coordinator by the responsibility function. We use a mapping function K to map Domains to their Keys, K has to fulfill Eq. 7 and 8.

Let K be the function mapping a Domain (subset of S_{ID}) to an ID in S_{ID}, let $\wp(S_{ID})$ be the power set (set of all subsets) of S_{ID}, and let K_p^l be the key of the Domain D_p^l, then following holds

$$K: \ \wp(S_{ID}) \rightarrow S_{ID} : \tag{7}$$

$$\forall p \in S_{ID} \, \forall i \in \mathbb{N} \ : \ K_p^i \in D_p^i \tag{8}$$

For the ease of representation we use a simple function for K.

$$K_p^l := min(D_p^l) + \frac{max(D_p^l) - min(D_p^l)}{2} \tag{9}$$

Now we can define the Coordinators C_p^l of a specific Domain D_p^l containing a peer p for all levels l in the core-tree. Let D_p^l be a Domain, then its Coordinator C_p^l is a peer and defined as

$$C_p^l \ = \ q \in S_{ID}, \ with \ q = resp(K_p^l) \tag{10}$$

This means, the Coordinator of the Domain D_p^l (which is in the l^{th} level, and contains the ID p) is defined as the peer which is responsible for the key K_p^l in the over-overlay and the mapped ID space of the underlying DHT. Please note, that we use the index p in C_p^l only to identify the Domain D_p^l which the Coordinator C_p^l is responsible for. Coordinators of a height larger than 1 are only related to Domains, not to individual peers.

Every peer p in the network identifies a single Coordinator C_p to which it periodically send its peer-specific information called *update*. Each peer p in the network may be

a Coordinator of a Domain D_p^l and receive updates, these updates are then periodically sent in the network to the Coordinator C_p^{l-1} one level higher.

Now we discuss how a peer identifies its Coordinator and how a Coordinator identifies its level and the Coordinator one level higher. A peer p may be Coordinator for several Domains on different levels in the tree. This comes from the fact, that DHT responsibility area $resp(p)$ is an interval, which may contain some of the Keys K_p^i of the Domains D_p^l the peer p is in. In order to identify its Coordinator C_p to which p has to send its individual peer information, peer p calculates the Keys K_p^* of the Domains D_p^* it is in l_{max} levels deep, with

$$l_{max} \; = \; max(i \in \mathbb{N} \, with \, K_p^i \notin S_p) \qquad (11)$$

The Coordinator C_p of peer p is then $C_p = C_p^{l_{max}}$. Here we use the assumption, that a peer can determine, whether it is responsible for an ID in S_{ID} or not. This Coordinator C_p is then the first owner of a Domain Key, that lies not in the responsibility range of peer p. For all Domains D_p^* below level l peer p is Coordinator, thus no messages have to be sent on lower levels.

A peer q receiving updates is a Coordinator of some Domain, it periodically propagates the received updates up the tree. In order to identify the Coordinator one level higher, peer q calculates the Domain Keys K_q^* it is responsible for (using the function $resp$). The Coordinator one level higher is then $C_q^{l_{min}-1}$ with

$$l_{min} \; = \; min(i \in \mathbb{N} \, with \, K_p^i \in S_p) \qquad (12)$$

The peer q identifies the level of its largest Domain and with this the Coordinator one level higher.

To join SkyEye.KOM, peers send a regular update to their Coordinator. No specific join or keep-alive maintenance is required as we rely on the route functionality of the underlying DHT. The failing of a Coordinator C_*^l is detected by peers or Coordinators C_*^{l+1} of lower levels, which fail in sending update messages to the Coordinator. As soon as a peer p identifies that the Coordinator C_p^l failed, it starts a lookup for the peer now being responsible for the Domain Key $res(K_p^l)$. The identified peer is then the new Coordinator. Although the information is lost, it is refreshed in the next update interval, when the Coordinators of lower levels send their updates to the new Coordinator. No further maintenance is needed if a failure occurs.

3.2 Functions Provided by SkyEye.KOM

Having described the structure of the tree, we focus in this Subsection on how the tree is used to disseminate peer information and to state queries regarding the network statistics and individual peer capabilities.

Peers and Coordinators send update messages periodically one level higher in the tree. Update messages consist of an aggregatable information part for monitoring purposes and a non-aggregatable part for capacity-based peer queries.

To enable capacity-based peer search, peers decide on the capabilities they offer to the network, e.g. CPU capacity, upload and download bandwidth, main memory, storage space or their network or geographical position. These attributes are used as keys and create with the corresponding value a peer specific key-value pair. The non-aggregatable information part in the update is the set of these pairs linked to the peer ID. Further, a time to live (TTL) counter is added, which is decreased in each level in the tree, this allows outdated information to expire.

SkyEye.KOM provides the function to resolve queries of the type: Give me n peers fulfilling a set of requirements on the known peer attributes (e.g. asking for a minimum storage space, a maximum load,...). Queries contain a field identifying the requester, defining the number of requested peers and a list for requirements on peer attributes and how they are connected: AND, OR, $\leq$, $\geq$. Peers address their queries to their responsible Coordinators. The Coordinator checks locally whether it has information about n peers fulfilling the desired requirements. Then it either replies with n peers fulfilling the criteria or it redirects the query one level higher in the tree. If no Coordinator in the tree can respond to the query, the root of the tree responds with a list of peers fulfilling the criteria (less than n). Please note, that peers do not have to agree on a set of valid attributes. Still, complex queries considering and connecting various attributes are possible.

For monitoring purposes, aggregatable peer-specific information is sent by each peer in its periodic updates. All peers have to agree on a set of metrics that can be aggregated and are valuable to collect statistics on. Aggregation enables for some metrics (e.g. CPU load) to calculate the system wide average value, standard deviation and confidence intervals.

To give examples, with SkyEye.KOM one could obtain statistics on the number of peers in the overlay, their average online-time and the churn rate. By aggregating observations of individual peers, one could further measure statistics on the number of hops per lookup, the hit rate and the overlay per underlay hop penalty, which describes the underlay awareness of the overlay. Statistics on the topology can be obtained by aggregating node degrees and peer-specific maximum hop counts. Statistics on the load in the network is valuable for many functionality layers in a P2P application. Load can be described as resource provision (CPU, memory, storage space, bandwidth) for the network, both absolute and weighted with the individual peer capabilities in a time interval or at all. This load information can be measured for various functionality layers, like the replicating storage layer, multi- or broadcast enabling layers and even for SkyEye.KOM itself.

Peers place the metrics, e.g. the number of incoming messages in the last minute, in the aggregatable information part of the update message and send their update to their Coordinator at peer-specific time intervals. Periodically, the Coordinator aggregates the received information (e.g. calculates the domain wide average on the number of incoming messages) and sends the compressed information to its Coordinator one level higher. At the root of the tree, the monitoring information is complete and can be used. On lower levels, statistics on subsets of the tree are available.

In order to retrieve these statistics, peers send a request to their Coordinator at an arbitrary level and receive the statistics. The size of the result message is not related to the level of the Coordinator asked, as aggregated information keeps its size.

3.3 Load Balancing in the Tree

The tree structure described in Subsection 3.1 fulfills already the functional requirements stated for an IM over-overlay. In this Subsection we solve the two main limitations resulting from the tree structure: First, inefficiently long update and query paths resulting from a deep tree in which the Coordinators are mainly underloaded. And second, overloaded Coordinators in the tree, that do not have the capabilities to fulfill their Coordinator task.

In order to address both the underloaded and overloaded peers, we introduce three thresholds T_{Min}, T_{Max} and $T_{Support}$. Each Coordinator should be responsible for at least T_{Min} and at most T_{Max} peers. Being responsible for more than T_{Max} peers requires to keep up to many connections, whereas being responsible for less then T_{Min} peers leads to a high number of levels in the tree. The parameter T_{Min} is a system-wide parameter regulating the height of the tree, whereas T_{Max} is a peer specific parameter denoting the maximum load of the individual peer.

In order to decrease the height of the tree, Coordinators check upon receiving an update, whether the number of peers they know to be responsible for is between T_{Min} and T_{Max}. If a Coordinator C_p^l receives an update from peer p, and C_p^l is responsible for less than T_{Min} peers, then C_p^l advises peer p to send its next N updates to C_p^{l-1}. Coordinator C_p^{l-1} may advise peer p to send its updates to C_p^{l-2} and so on. However, Coordinators dispatching peers hold state on the number of redirected peers in order to know, when to stop dispatching. Addressing of updates and queries is not strict, beginning at a deeper part of the tree only disburdens peers at higher levels of the tree, that are responsible for more information. If updates or queries are addressed "too" deep or high in the tree, the information is anyways included and queries are resolved.

In order to decrease the load on Coordinators we introduce Support Peers. Deterministically chosen Domain Keys may put weak peers into charge of being responsible for Domains. The Coordinator of a Domain may decide that itself is incapable to carry the whole load that is required. Coordinators have to store the information of the peers they are responsible for, process information updates and react on queries.

Coordinators limit their information cache size to T_{Max}, storing only capacity information on the strongest T_{Max} peers. Queries ask in general for a small number of peers fulfilling specific criteria. In the case that the number of incoming peer updates exceeds $T_{Support}$, the Coordinator picks supporting peers in order to store all information on the peers in its Domain.

Support Peers may be chosen based on their capabilities by the Coordinator from the set of monitored peers in order to dispatch load to them.

Each Coordinator appoints the best m Support Candidates (SC) in its own Domain (sorted in descending quality according to some metric): SC_1 to SC_m. Support Peers for its own Domain are chosen from the peers $SC_{\frac{m}{b}+1}$ to SC_m. The information about the best $\frac{m}{b}$ Support Candidates (SC_1 to $SC_{\frac{m}{b}}$) is passed one level higher, so that in this larger Domain more valuable candidates are available.

Once an overloaded Coordinator picks a Support Peer, it announces to the peers it is responsible for its Support Peer(s) in a reactive manner. The peers, address then for a given time period their updates and queries to the Support Peers. If the Support Peer is overloaded as well, another Support Peer is chosen and the load is shared among the Support Peers. All Support Peers responsible for peers of the Coordinator synchronize their information periodically in order to keep themselves up to date with the information. Only one peer, either the Coordinator or one of its Support Peers sends information updates one level higher in the tree, this peer processes the aggregatable information provided by the other peers.

If the Support Peer is overloaded as well, another Support Peer is chosen and the load is shared among the Support Peers. Once the Support Peer reports to have less load than T_{Max}, the Coordinator takes over the load. Then the Support Peer is released and it dispatches all remaining updates and queries to the Coordinator. Having the thresholds T_{Max} and $T_{Support}$ prevents restless responsibility swapping at one threshold.

4 Evaluation

For the evaluation of our solution we simulated SkyEye.KOM and measured relevant metrics with focus on the tree characteristics, the query performance and the costs for maintaining the over-overlay.

We simulated 1000, 5000 and 10000 peers in the event-based P2P simulator PeerfactSim.KOM [7] which implements an underlay based on global network positioning [8], various P2P functionality and a set of DHTs. We used an

abstracted DHT component, which enables us to see the behavior of SkyEye.KOM independent of any specific DHT. The abstracted DHT dispatches the lookup messages between the peers considering transmission delays and emulating the DHT functionality.

Using this DHT, we implemented SkyEye.KOM as P2P application in which peers build the SkyEye.KOM tree by performing lookups to their identified Coordinators and periodically send them updates. The peers have 3 capacity attributes, out of which one contains a random but fixed number, and two values that change randomly in different intervals. These values represent the peer's capacity like free storage space, CPU usage and available memory. One Query is performed by each peer in every update interval, which are for all peers equally large. Peers state queries for a set of five peers whose first capacity value is larger than a random number, which characterizes the query complexity ranging from 1 to 15. This random number is normally distributed, 1 is the easiest query. We evaluated the tree characteristics, the query performance and the costs under this scenario.

4.1 Results

As metrics for the tree characteristics we have chosen the tree depth, which shows how scalable and balanced the tree is. This has an implication on the freshness of the data. With the start of the simulation, peers begin to join as depicted in Figure 3(a). The depth of the tree increases logarithmically in all three network sizes and stabilizes, even under churn. For the next evaluation steps, we omitted the graphs with 1000 peers due to clarity and the similarity of the results.

The logarithmic scale of the tree has direct effect on the freshness of the information stored in the tree. Information is inducted by the peers at the lowest possible position in the tree and then propagated with every update interval towards the root. Figure 3(b) shows the age of the information which is propagated by the Coordinators. The figure shows that the height of a Coordinator and the age of the information are nearly similar. With the tree based approach, SkyEye.KOM is capable to announce the capabilities of each peer in $O(logN)$ update steps.

As the information in SkyEye.KOM gets older, results may contain the IDs of peers that failed or left the network. Simulations showed that the average ratio of online peers in the result set is near 0.985 and independent of the level of the Coordinators (no figure). Due to the short update paths, SkyEye.KOM is able to provide a near real-time view on the peers in the system.

In this tree, the Coordinators have a limited buffer size (T_{max}) for maintaining peer information. Figure 3(c) shows us the ratio of the available peers a Coordinator monitors and which ratio of peers it ignores, a ratio related to T_{max}. With T_{max} the individual Coordinator load is limited but their knowledge on their Domain is incomplete, which may lead to increased overhead, as queries are forwarded higher in the tree, although the Domain offers suitable results. This has an effect on the hop count for queries and is a tradeoff which is worth to investigate.

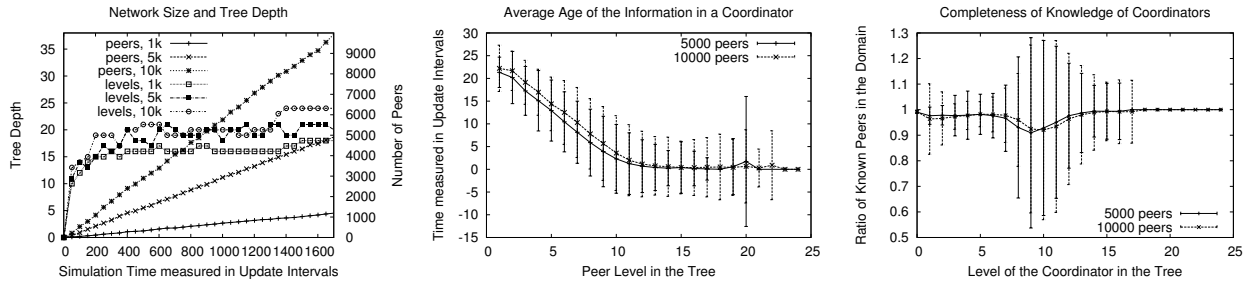
In order to measure the query performance, we observe the number of hops needed to find a suitable answer for the queries. Figure 4(a) shows that the number of hops a query was forwarded up in the tree, until an appropriate set of matching peers were found, ranges from 0 to 4 hops in average. Although there are fluctuations in the hop count, regarding the level of the query initiator, the results show that queries are answered after a few hops. The delay for answering queries is very low, as queries use information paths between the Coordinators, which have already established direct connections to each other.

Our next focus is on where in the tree queries are resolved and how this position is related to the complexity of the query. As peers may be Coordinators on various levels in the tree, we measure the position of resolving a query as the difference of the level of query injection and the hop count. Please note, that by this we can only estimate the position in the tree. The average depth of query resolving in relation to the query complexity is depicted in Figure 4(b). More complex queries traverse higher in the tree and load for easier queries is balanced in the tree. However, the resolving load is at about level 6. This results from the fact, that peers inject their queries at the highest point in the tree they are responsible for. This effect can also be seen in Figure 4(c). The average injection level for queries is around level 8 (as most of the peers are) and with 2-3 query hops most of the queries are resolved around level 6. By adjusting the level on which a query is injected in the tree, we can optimize the tradeoff between fast results and lower load on higher peers.

An overview on the traffic overhead per update interval is depicted in Figure 4(c). It shows that the number of update messages per peer is between 1 and 3, which is very small in comparison to common routing tables. The update messages are used to disseminate the information and to maintain the tree. Most of the messages in the over-overlay are resulting from the queries, which are mostly injected around level 8. However, queries and their results are small in size, so that the message overhead per update interval is low. By increasing the period of updating at the cost of freshness, Coordinators can adjust the traffic overhead they are willing to contribute.

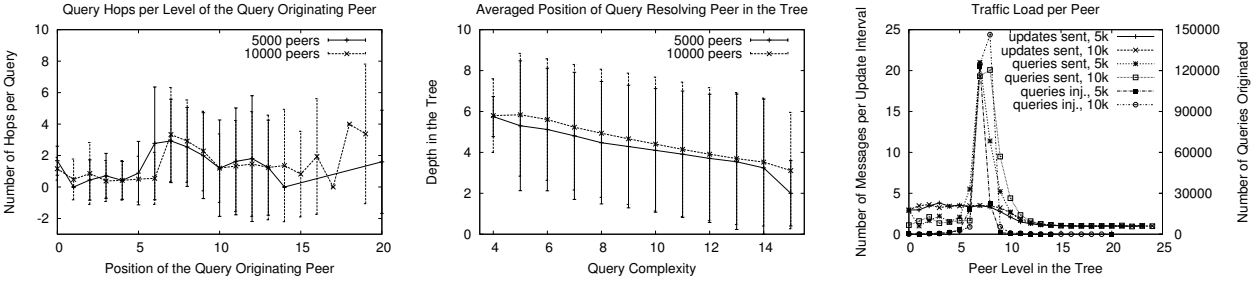
4.2 Conclusion

The cost for each peer using SkyEye.KOM is limited to the exchange of a few update messages per update interval. This low overhead is reached by using the underlying DHT functionality and omitting tree maintenance. This design decision leads to scalability, robustness of the tree, a up-to-date information on the peers and the network state. Churn



(a) Number of Peers and Tree Depth (b) Information Freshness (c) Coverage Scope of the Coordinators

Figure 3. Evaluation Results Corresponding to the Tree Structure



(a) Hops per Query (b) Position of the Answering Peer (c) Traffic Overhead for SkyEye.KOM

Figure 4. Evaluation Results Regarding the Query Performance and Costs

has no crucial effect on the tree, as a new Coordinator can instantly be identified. Each peer is able to define a personal load maximum, which supports the heterogeneity of the peers. All peers are contributing according to their individual capabilities. SkyEye.KOM provides capacity-based peer search and monitoring capabilities in a light-weight, easy to apply manner for structured P2P overlays.

5 Related Work

Various papers have addressed peer and system information management for P2P networks.

DASIS [1] is a module extending the routing table of the used overlay to store additional routing specific information, no further IM data structure is proposed. It strongly depends on the details of the used overlay and can only be used for small portions of information.

T-MAN [6] is a proactive gossip-based overlay topology management system, in which each peer exchanges periodically its knowledge with neighbors. Information spreads only slow in the system and is hard to update. In SkyEye.KOM, information is propagated in a structured manner, enabling the refreshing of information in O(log N) update intervals.

P2P-Diet [5] extends hybrid unstructured P2P overlays with the functionality of ad-hoc and continuous search for specific objects (and peers). P2P-Diet provides network monitoring and capacity-based peer search causing significant overhead by broadcasting information updates and extensive maintaining operations. Maintenance costs for the tree in SkyEye.KOM are low, as we use a deterministic function to identify the nodes in the tree.

Astrolabe [10] has been published in 2003 as a distributed (structured) IM system, many concepts can be adapted to the P2P scenario. In Astrolabe, nodes join several so-called zones, which are corresponding to the nodes' hierarchical host name. Creating a topology according to the hierarchical zones results in an inefficient tree of depth O($log\ |IDspace|$) containing various empty zones, the tree in SkyEye.KOM is $O(log\ |N|)$ deep.

Willow [11] extends the idea of Astrolabe [10] to a DHT overlay integrating various functionality of P2P layers. The solution is more efficient, but not overlay independent.

SOMO [14] is a metadata overlay for the resource management in P2P DHTs. SOMO builds a tree top down on the peers in the ID space, identifying nodes in the tree using a stateless function. In SOMO the information is pulled up towards the root, aggregated and pushed back. This requires peers responsible for a region to periodically look for unattached peers in their region. SkyEye.KOM follows a push-based approach saving probing costs. However, SOMO does not provide mechanisms for load-balancing and is limited due to its pull-based approach.

CONE [2] builds a tree, using the natural order of the peer IDs, peers with higher IDs are parent nodes of peers with smaller IDs. The tree is used to aggregate peer information in a reactive manner, though the overhead generated through updates is significant. In SkyEye.KOM updates are transmitted proactively, in peer-specific intervals, leaving time for messages to arrive and to be processed in a group. Further, SkyEye.KOM allows besides information

aggregation, capacity-based search for peers.

SDIMS [12] allows information aggregation and attribute-based search for peers as well. SDIMS builds in contrast to SkyEye.KOM for each peer attribute its own tree, which distributes the load. Although SDIMS can optimized for traffic efficient updating and querying of single attributes, the split of the aggregation tree also cut off the relationship between the attributes. A complex query consisting of the retrieval of multiple attribute values requires multiple steps, which causes more time and message overhead than in a solution with a combined aggregation path like in SkyEye.KOM.

The presented solutions try to optimize for either overlay-independency, enabling complex queries, being load balanced, providing a wide range of functionality and yet be easy and simple to use. However, the presented solutions fail in one or several design goals. With SkyEye.KOM, we address all of these requirements.

6 Conclusion

In this paper we discussed the motivation for building an information management layer (IML) for structured P2P systems, which provides statistics on the whole P2P system and helps the individual functionality layers (e.g. the DHT storage layer) on a P2P host to find suitable peers for a layer-specific role assignment (e.g. storing replicas).

Knowing mean values, standard deviations and confidence intervals on critical system metrics (e.g. traffic load on peers, number of hops per lookup) in distributed P2P applications is a desired functionality. Statistics help developers of P2P applications to improve their mechanism and companies offering P2P-based applications to calculate their costs for supporting servers more precisely. Further, it enables self-optimizing mechanism to be implemented in the P2P application, by setting system parameters in dependency to the system statistics.

Capacity-based peer search enables queries for e.g. 7 peers, offering at least 500MB storage space and 200KB/s upload bandwidth. This functionality of the IML disburdens other functionality layers in a P2P application from the load of finding appropriate peers for a layer-specific task.

We defined the functional and non-functional goals for an IML and presented SkyEye.KOM, an over-overlay applicable on DHTs which implements the desired functionality.

SkyEye.KOM is an IM over-overlay applicable on DHTs, building a tree with structured information flows. Having a core-tree for proactively performed information gathering, complex capacity-based peer queries considering multiple peer attributes can be stated. SkyEye.KOM further provides advanced statistics on the P2P network, which enables interested parties to analyze the status of the network. As Coordinators in the core-tree can dispatch load to Support Peers and set a peer-specific maximum load to tolerate, load balancing is addressed and the heterogeneity of the peers is used.

We evaluated SkyEye.KOM in simulations regarding the establishment of the tree structure, the query performance and the overhead. The evaluation shows the good query performance of SkyEye.KOM and that due to the deterministic Coordinator assignment no tree maintenance is needed in the over-overlay, even under churn.

Our solution is scalable by leveraging the underlying DHT, easy to deploy as simple add-on to existing DHTs, efficient with O(log N) hops per query and update and it comes with very low maintenance costs due to the deterministic function assigning the peer position in the tree.

We believe that an IML, like SkyEye.KOM, has the potential to become a valuable component in future's modular multi-functional P2P applications.

References

[1] K. Albrecht, R. Arnold, M. Gähwiler, and R. Wattenhofer. Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave. In *Proc. of IEEE P2P '04*, pages 227–234. IEEE Computer Society, 2004.

[2] R. Bhagwan, G. Varghese, and G. Voelker. CONE: Augmenting DHTs to Support Distributed Resource Discovery. Technical Report CS2003-0755, University of California, San Diego, 2003.

[3] BitTorrent. http://www.bittorrent.com.

[4] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proc. of IPTPS '03*, 2003.

[5] S. Idreos, M. Koubarakis, and C. Tryfonopoulos. P2P-DIET: An Extensible P2P Service that Unifies Ad-Hoc and Continuous Querying in Super-Peer Networks. In *Proc. of ACM SIGMOD '04*, pages 933–934. ACM Press, 2004.

[6] M. Jelasity and O. Babaoglu. T-Man: Gossip-based Overlay Topology Management. In *Proc. of ESOA'05*, 2005.

[7] A. Kovacevic, S. Kaune, P. Mukherjee, N. Liebau, and R. Steinmetz. Benchmarking Platform for Peer-to-Peer Systems. *it - Information Technology*, 46(3), 2007.

[8] E. Ng and H. Zhang. Predicting Internet Network Distance with Coordiantes-based Approaches. In *Proc. of INFOCOM'02*, 2002.

[9] Skype. http://www.skype.com, 2004.

[10] 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):164–206, 2003.

[11] R. van Renesse and A. Bozdog. Willow: DHT, Aggregation, and Publish/Subscribe in one Protocol. In *Proc. of IPTPS '04*, pages 173–183. Springer, 2004.

[12] P. Yalagandula and M. Dahlin. Research Challenges for a Scalable Distributed Information Management System. Technical Report CS-TR-04-48, The University of Texas at Austin, Department of Computer Sciences, 2004.

[13] Zatoo - TV to Go. http://www.zatoo.com/, 2007.

[14] Z. Zhang, S. Shi, and J. Zhu. SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT. In *Proc. of IPTPS '03*, volume 2735. Springer, 2003.