Pervasive and Mobile Computing 40 (2017) 593–610

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

SkyEye: A tree-based peer-to-peer monitoring approach

Kalman Graffi *, Andreas Disterhöft

University of Düsseldorf, Universitätstrasse 1, Düsseldorf, Germany

ARTICLE INFO

Article history:
Available online 13 July 2017

Keywords:
p2p
Monitoring
Tree-based overlay
Distributed aggregation

ABSTRACT

Peer-to-Peer overlays, used by applications like distributed online social networks, need the ability to adjust their system parameters and thus the system performance to provide a certain level of quality, i.e. to ensure maximum response times. For the control of p2p network, a reliable information basis about the system's state is essential, i.e. what the current average response time is. A precise and rapidly adapting monitoring mechanism can provide the desired knowledge. In this paper we propose SkyEye, a tree-based monitoring approach which operates on top of an existing structured p2p overlay. It provides through a lightweight, error-prone protocol a continuous monitoring of all peers in the network. It gathers and disseminates sophisticated statistics on a wide range of metrics from all peers, to all peers. Evaluation shows a superior precision and adapting rate compared to an implementation of the gossip-based push-sum algorithm in scenarios with churn and without churn.

© 2017 Elsevier B.V. All rights reserved.

1. Motivation

Peer-to-Peer (p2p) networks are well-studied decentralized structures which operate in a self-organized fashion, but also without any global overview. However, it is very useful to keep an eye on metrics or key performance indicators of a running p2p system, e.g. in order to identify weaknesses, bugs and misleading trends. The basic prerequisite of adapting a dynamic p2p network is a precise and efficient monitoring mechanism for collecting and disseminating global information emergence. System-wide monitoring is trivial in a centralized structure, as the central server can raise statistics but it has scaling issues. However, decentralized monitoring in a p2p network is a challenge as each participant's statistics should be gathered, aggregated and disseminated in an efficient way. With this global information, various optimizations in the network can be initiated.

In the scope of this paper we present, discuss and evaluate the decentralized monitoring of such system-specific information. Our solution, SkyEye, is a mechanism that addresses and fulfills these requirements. The key feature of SkyEye is the aggregation and dissemination of information within a tree structure which is built on top of a distributed hash table (DHT).

The goal of the monitoring of system-specific information is to retrieve an exhaustive statistical view on a wide set of metrics on all peers in the network and to disseminate it to all peers in the network. The set of metrics is an extendable list, which is common to all peers in the network and contains metrics which are based on local measurements of all peers, such as the bandwidth consumption or observed lookup delays or a peer. The statistical view on the metrics, i.e. average, minimum, maximum and so on, is taken over the measurements of all peers in the network, thus leading to a global view on the system statistics. The goal of the gathering of this global view is to disseminate the global view to all peers in the network and thus let them know about the status of the system.

* Corresponding author.
E-mail address: graffi@hhu.de (K. Graffi).

http://dx.doi.org/10.1016/j.pmcj.2017.07.003
1574-1192/© 2017 Elsevier B.V. All rights reserved.

Nomenclature

p2p	peer-to-peer
kbr	key based routing
SI	sinus interval
RPE	rounds per epoch
DHT	distributed hash table
UI	update interval
bF	branching factor

In order to formalize the desired functionality, we describe suitable interfaces. Let $m_i(p)$ be a locally measured metric (i) in a peer p, such as the hop count of all lookups the peer performed. And let further $m_i(P)$ be the corresponding global metric (i) for the set of all peers P, such as the hop count of all lookups of all peers. Let $M(p)$ be the list of all metrics of peer p in a local view and $M(P)$ the list of all metrics of all peers in a global view. Then following services are provided by the mechanism for monitoring of system-specific information:

- (*Aim* **1**): $M(P) \leftarrow$ getStatistics() - retrieves the global view on all metrics
- (*Aim* **2**): addLocalStatistics($M(p)$) - add local observations to global view.

To provide these services, the mechanism states following dependencies:

- (*Req*. **1**): $M_i(p) \leftarrow$ getLocalMetric(i) - local measurement of metrics
- (*Req*. **2**): Structured p2p overlay with kbr-functionality as described in [1].

The main goal of the monitoring function is to obtain a global view on the status of the system (*Aim* **1**), based on local measurements of all peers (*Req*. **1**). For that, the peers are able to communicate their status (*Aim* **2**) while using a kbr-compliant structured p2p overlay (*Req*. **2**).

Our solution, SkyEye, uses an overlay on top of a structured p2p overlay which implements the key based routing (kbr) interface [1]. Thus, our solution is independent of a certain p2p overlay implementing the kbr-interface and is more generally applicable. Particularly, we assume two dedicated functions by the structured p2p overlay:

- *Boolean isMyKey(Key K)* allows to check whether the local peer is responsible of a given key k and can be derived from the *range operation* using in the kbr-interface. Note that this decision can be implemented by using local information only, thus this check is quick and cheap.
- *void route(Key K, Message M, Nodehandle hint)* is specified in the kbr-interface which can be subdivided into the following Function 3 and 4.
  - *Nodehandle getNodehandle(Key K, Nodehandle hint)* gives the contact information of the peer responsible for a given key.
  - *void send(Message M, Nodehandle p) allows to send a message to a dedicated peer, of which the nodehandle is already known.*

Having the Function 2 provided by the structured p2p overlay is sufficient for SkyEye. Current structured p2p overlays either already fulfill these requirements or can be easily extended to do so.

Our main contributions in this paper are the following:

- Presentation of the topology and protocol of our distributed, tree-based, proactive, push-based, lightweight, monitoring overlay called SkyEye which can run on-top of kbr-compliant p2p overlays
- Extensive parameter study of our SkyEye implementation
- Comparison with unstructured monitoring approach Push-Sum [2].

This section covers the requirements for our monitoring approach and the rest of this paper is organized as follows. First, in Section 2 we analyze and categorize existing monitoring approaches by their design decisions and expose ours. It is followed by the definition of the core topology of SkyEye in Section 3. The protocols of SkyEye is described in Section 4. Section 5 continues with the evaluation of our approach. We conclude our work and give an overview on future work in Section 6.

2. Related work and design decisions

Having discussed the requirements of a monitoring solution, in this section, we classify related work by their design decisions and outline the differences to our solution SkyEye. Having a view on the global statistics of a network allows to

consider the information on the quality and dynamics of the network for optimized decisions. Monitoring information is used in [3] to foresee low availability of replicating nodes and nodes with relevant duties and thus to counteract by selecting further nodes for replication. Through information on the priority of the messages, differentiated services can be provided through adaptive strategies to forward messages in p2p overlays, as presented in [4]. Optimized routing in p2p-based social networks based on monitoring information including social interaction patterns is discussed in [5]. Self-stabilization is a property, that allows to converge from any given connected topology to a desired topology, such as Chord as described in [6]. This approach can benefit from global monitoring statistics, as the self-stabilization process can be accelerated. Finally, monitoring information giving insights on the quality and weaknesses of the network, can be used to implement a control-loop for p2p systems, as suggested in [7,8]. Here, the monitoring information is obtained through a distributed approach, analyzed and parameter changes are decided which are then communicated and executed throughout the network. By this, the network is capable to identify and resolve its own weaknesses and e.g. reach and hold a specific goal with regard to the lookup hop count or lookup delay. Thus, several improvements become possible through a monitoring approach.

Centralized vs. Distributed.

Network and distributed system management has been investigated thoroughly in literature in previous decades. For large-scale networks with no commercial provider, like p2p systems, centralized approaches such as the simple network management protocol (SNMP) [9] or network/transport layer-focused approaches [10] are not suitable. A monitoring solution in a p2p system must be distributed as well and hosted by all participating peers. We pursue a distributed approach using a p2p system, distribute the monitoring load and allow for fully distributed solutions. In the field of distributed monitoring several approaches exist like T-MAN [11], push-sum [2], CONE [12] and SOMO [13].

Integrated vs. New Layer.

Next, overlay-integrated and overlay-independent solutions are to be distinguished. While the first approach reuses the overlay infrastructure resulting in lower maintenance costs, it is limited to only one overlay. Additionally, extensions for the same overlay from different authors may be incompatible. Monitoring approaches that are integrated in existing overlays are DASIS [14] or Willow [15] which are integrated in overlays with prefix-based routing table, P2P-Diet [16] which is part of a hybrid unstructured p2p overlay, HilbertChord [17] being part of a Chord variant. Overlay-independent solutions, such as SkyEye, allow for a wider use.

Monitoring Scope.

For the monitoring of the system status, one could either monitor the whole p2p system or focus on a (small) representative subset of the peers and interpret their status. In literature, various solutions exist to providing access to a representative, unbiased sample set in p2p networks. In works, such as [18], approaches to select a random, unbiased set of nodes are discussed. Several approaches discuss, how to derive an unbiased sample of peers, for example with random walks [19], newscast [20], gossiping [21] and roaming agents [22]. The interpolated view of the samples is rarely sufficiently, also it is unable to capture valid minimum and maximum values. We aimed with SkyEye at a global view which contains all potential statistics of all peers.

Monitoring Topology.

The purpose of the monitoring topology is to connect all peers and allow for a coordinated information flow. A random topology is used by T-MAN [11] and push-sum [2], where each peer exchanges periodically its knowledge with neighbors. Information spreads slowly and is hard to update.

SkyEye uses a tree-based approach which allows to efficiently aggregate and disseminate information up respectively down the tree. The tree height defines the freshness of the aggregated statistics, which can be obtained without any redundant information transfer, thus highly optimized. Also the costs are bound by the fixed node degree of the nodes. Regardless of the position in the tree, each node encounters the same load. Lastly, one may note that trees are vulnerable to churn. However, our solution provides mechanisms which dampen this effect while retaining the benefits of a tree structure.

Reactive vs. Proactive.

A monitoring solution can operate reactively, i.e. by gathering monitoring information upon (rare) query, or proactively, i.e. continuously gathering monitoring information to have answer available for frequent queries. Reactive approaches, such as CONE [12] or the solutions proposed in [23], thus generate tremendous overhead upon retrieving of the monitoring results. Also pull-based approaches, such as SOMO [13], in which the nodes have to be found first, induce high initial costs. In SkyEye updates are pushed proactively up the tree, in peer-specific intervals, leaving time for messages to arrive and to be processed in a group. The overhead for preparing the information proactively for all peer is less than the overhead for alternative reactive query resolutions.

Energy-efficiency and communication channels.

In wireless sensor networks [24], a specialization of p2p networks, the energy efficiency is relevant as these nodes depend on an installed battery. Hence, operation systems like TinyOS [25] are proposed to optimize the energy-efficiency of such nodes. Furthermore, while operating in wireless networks, cognitive radio networks [26,27] can expand and optimize the use of limited radio channels. In this paper, we are focusing on general p2p networks, where typically neither power limitations nor limitations on communication channels exist. Though, in SkyEye we minimize the bandwidth consumption on participating nodes.

3. SkyEye: Establishing the core monitoring topology

Next, we describe the idea, model and creation of the topology of SkyEye on top of a kbr-compliant [1] overlay. In Section 4, we describe then how this SkyEye topology is used to gather and disseminate the monitoring information.

The main idea for setting up the SkyEye topology is that each node computes based on its overlay ID and its responsibility range its position in the monitoring tree. The (inner) nodes in the monitoring tree are responsible for specific IDs, so called Domain Keys, which can be computed. The nodes follow a specific algorithm to check their responsibility for these Domain Keys, once they find themselves responsible, they have identified their position in the tree and the ID of their parent node in the tree, they have to communicate to. In the following we derive the algorithm for the nodes to identify their position in the tree.

Let P be the set of all peers and S_{OID} be the identifier space of the overlay. The overlay ID space contains both peer identifiers (ID) and object identifiers (ID,keys). All peers have a peer ID ($p \in P$), so that $P \subseteq S_{OID}$, the set of all object identifiers is equal to S_{OID}. In a DHT, object identifiers are linked to peer identifiers. We denote the function $owner$ assigning object IDs to peer IDs. For every object ID i, a corresponding peer is assigned.

$$\forall i \in S_{OID} : \ !\exists p \in P \ with \ owner(i) = p. \tag{1}$$

A peer is always responsible for a convex set of object identifiers, called the responsibility range or *scope* of the peer.

$$\begin{aligned} &scope(p) : P \rightarrow \wp(S_{OID}) \\ &scope(p) := owner^{-1}(p); \ p \in P \ ; scope(p) \subseteq S_{OID} \\ &\forall p, q \in P : \ p \neq q \Rightarrow scope(p) \cap scope(q) = \{\}. \end{aligned} \tag{2}$$

The structured overlay provides the function *Boolean isMyKey(key K)* that tells whether the local peer p is responsible for a given key K or not. For $p \in P$ and $K \in S_{OID}$, following counts:

$$\texttt{Booleanp.isMyKey(keyK)} = \begin{cases} true & if \quad K \in scope(p) \\ false & if \quad K \notin scope(p). \end{cases} \tag{3}$$

For the consistency of the formulas, we also add:

$$p.isMyKey(key\ K) \Leftrightarrow K \in scope(p) \Leftrightarrow owner(K) = p. \tag{4}$$

The second assumed function is *void route(key K, message M, nodehandle hint)*. It is used to route a message M to a peer p with $scope(p) = K$. The peer *hint* should be used as next hop in routing, if set. Having introduced the nomenclature used in the description of the p2p overlay, next, we introduce SkyEye's topology on top of the kbr-compatible p2p overlay.

3.1. A unified ID space

SkyEye should be usable on all structured p2p overlays fulfilling the assumptions specified in Section 1 . SkyEye reuses the routing and ID-mapping functionality of the underlying structured p2p overlay. In order to have an ID space in SkyEye independent of the overlay, we propose a unified ID space $S_{ID} \subseteq \mathbb{R}$ that ranges from 0 to 1. It can be parameterized in its granularity by defining minimal steps, e.g. $2^{ID_{max}}$ of a given structured overlay with ID_{max} the maximum ID of the overlay. Any ID used in a current overlays can be mapped to this interval, for example by dividing the overlay ID by the size of the overlay ID space. The benefit of floating-point numbers is that the granularity may be varied while keeping the distribution of the IDs in the ID space.

Let S_{OID} be the overlay ID space and *IDmap* the function mapping from S_{OID} to S_{ID}. The mapping function should retain convexity of identifier subsets and be linear in relation to the responsibility function:

$$\begin{aligned} &\forall i \in S_{OID} : \ IDmap(owner(i)) = owner(IDmap(i)) \\ &\forall p \in P : \ IDmap(scope(p)) = scope(IDmap(p)). \end{aligned} \tag{5}$$

This requirement on the mapping function allows easy mapping of the responsibility areas of a peer in both ID spaces. In the unified ID space S_{ID} of SkyEye we reuse, by remapping of the IDs, the routing functionality of the underlying structured p2p overlay and the responsibility function *isMyKey*.

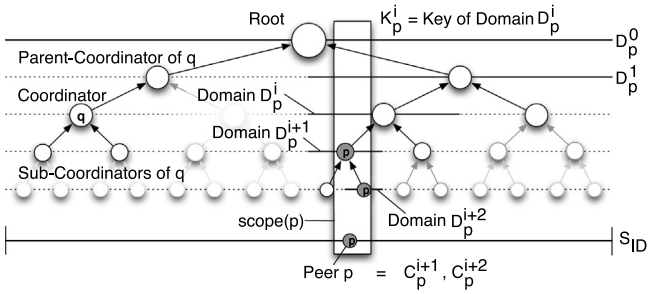


Fig. 1. Definitions used in SkyEye.

3.2. Definition of domains

The monitoring tree of SkyEye is built by recursively segmenting the ID space S_{ID} in intervals, called *Domains*, and assigning a responsible peer to each Domain, called *Coordinator* of the Domain. The depth of the tree is $O(logN)$. Each level of the tree stores in union the information of all peers in the ID space, but with increasing tree depth the information is shared with more peers. A Coordinator is in charge to maintain the information of all the peers, whose IDs are in its Domain. An overview on the used terminology, is given in Fig. 1. In the following introduction of the terms, we refer to this figure.

We define a *Domain* as a continuous interval in the ID space S_{ID}. Domains at the same level l in the tree do not overlap, they form a partition. Let $p \in S_{ID}$ be a peer, then the sequence of Domains containing peer p are labeled D_p^l with l as level counter. The root of the tree is at level 0, thus the following counts:

The Domain of the root covers the whole ID space, thus all information retrieved from this Domain results in a global view.

$$\forall p \in S_{ID} \ : \ D_p^0 = S_{ID}. \tag{6}$$

A peer ID p, for which a Domain is calculated, is always part of the Domain.

$$\forall l \in \mathbb{N}, \ \forall p \in S_{ID} \ : \ p \in D_p^l. \tag{7}$$

The same counts also for the peer for all levels in the tree.

$$\forall l \in \mathbb{N}, \ \forall p \in S_{ID} \ : \ D_p^{l+1} \subseteq D_p^l. \tag{8}$$

Thus, the term D_p^l identifies for a peer p Domains in which the peer is part of at every level l. The recursion ends in two cases. First, with a Domain being identical to the predecessor Domain, typically by having only the ID p in the ID range:

$$\exists k \in \mathbb{N}, \ \forall l \in \mathbb{N} \ with \ l \geq k \ \forall p \in S_{ID} \ : \ D_p^{l+1} = D_p^l. \tag{9}$$

In practice this case does not occur, as the second termination criteria is mostly active. The calculation of the Domains of a peer stops at the point when the scope of the peer covers the whole Domain:

$$\forall p \in S_{ID}, \ \exists k \in \mathbb{N}, \ \forall l \in \mathbb{N} \ with \ l \geq k \ : \ D_p^l \subseteq scope(p). \tag{10}$$

Each Domain is maintained by a Coordinator, the Coordinators of the various Domains, eventually, establish the tree by sending each other information updates. In the core tree, each peer p identifies, using the deterministic function, the Coordinators for the Domains D_p^l that contain the peer's ID p. A Domain of level l (e.g. $[i_a, i_b]$) is partitioned in β (Sub-)Domains of level $l+1$ (e.g. $[i_a, i_1]$, $[i_1+1, i_2]$, $[i_2+1, i_3]$, $\ldots$, $[i_{\beta-1}+1, i_b]$). By this process, the Domains and not the peers build a b-tree structure. The parameter β is the branching factor of the tree. We map Domains to peers, that become then Coordinators of the Domain, using the responsibility function $owner$. A specific ID in the Domain, called *Domain Key*, determines the corresponding Coordinator by the responsibility function. We use a mapping function K to map Domains to their Keys, K has to fulfill Eq. (11).

Let K be the function mapping a Domain (subset of S_{ID}) to an ID (Domain Key) 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:

$$\begin{aligned} &K \ : \ \wp(S_{ID}) \rightarrow S_{ID} \\ &\forall p \in S_{ID}, \ \forall l \in \mathbb{N} \ : \ K_p^l := K(D_p^l) \in D_p^l. \end{aligned} \tag{11}$$

In order to compute the Domain Key K_p^l at level l, a peer p first calculates the ID range of the Domain D_p^l by using the following two formulas. Here, β denotes the branching factor of the SkyEye tree and ID_p the ID of the peer p:

$$min(D_p^l) = \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l} \tag{12}$$

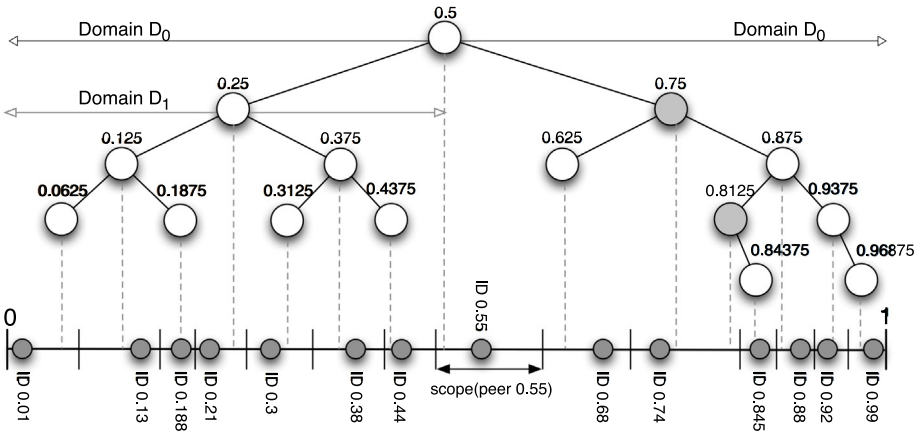


Fig. 2. SkyEye tree topology and data flow.

$$max(D_p^l) = \frac{\lfloor ID_p \cdot \beta^l \rfloor + 1}{\beta^l}. \tag{13}$$

Afterwards, the Domain Key K_p^l can be calculated. For the ease of representation, we present a simple function for K, taking K_p^l as the middle value of a Domain D_p^l. We calculate the IDs K_p^l, for which a peer p tests its responsibility, for every level l:

$$\begin{aligned} K_p^l &:= min(D_p^l) + \frac{max(D_p^l) - min(D_p^l)}{2} \\ &:= \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l} + \frac{\frac{\lfloor ID_p \cdot \beta^l \rfloor + 1}{\beta^l} - \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l}}{2} = \frac{\frac{\lfloor ID_p \cdot \beta^l \rfloor + 1}{\beta^l} + \frac{\lfloor ID_p \cdot \beta^l \rfloor}{\beta^l}}{2}. \end{aligned} \tag{14}$$

Now, we can define for every level l in the tree the Coordinator C_p^l of a specific Domain D_p^l. Let D_p^l be a Domain, then its Coordinator C_p^l is a peer $p \in P$ and defined as

$$C_p^l := owner(K_p^l). \tag{15}$$

This means, the Coordinator of the Domain D_p^l, which is in the lth level and contains the ID p, is defined as the peer which is responsible for the key K_p^l in the unified ID space and the mapped ID space of the underlying DHT. We use the index p in C_p^l only to identify the Domain D_p^l for which the Coordinator C_p^l is responsible.

3.3. Definition of simple, parent and same-level coordinators

Every peer p in the network identifies a single (Parent-)Coordinator C_p to which it periodically sends its monitoring information called *update*. Each peer p in the network is a Coordinator of a Domain D_p^l and may receive updates. The received updates are aggregated with the local measurements, and then periodically sent in the network to the Parent-Coordinator C_p^{l-1} one level higher.

Now, we discuss how a peer identifies its position in the tree and its parent node in the tree, which is its Parent-Coordinator. We refer to Fig. 2. In order to identify its Parent-Coordinator C_p to which p has to send its monitoring information, peer p calculates the Keys K_p^* of the Domains D_p^* it is in. For instance peer $0.88 \in S_{OID}$ calculates the Domain Keys including 0.5, 0.75 and 0.875. It checks whether it is responsible for one of these Domain Keys, this is the case for peer 0.88 and Domain Key 0.875. A peer p may be Coordinator for several Domains on different levels in the tree. We observe this effect at peer 0.74, which is responsible for the Domain Keys 0.75 and 0.8125. This comes from the fact that DHT responsibility area obtained with $scope(p)$ is an interval, which may contain some of the Keys K_p^* of the Domains D_p^* the peer p is in. Here, we use the postulated DHT-function $isMyKey(keyK)$, so that a peer can determine, whether it is responsible for an ID in S_{ID} or not. The position of a peer in the tree is defined as the first, thus largest, Domain it is responsible for. Thus, the peer with the ID 0.74 is Coordinator of the Domains corresponding to the Domain Key 0.75 and 0.8125, but finds it position in the tree in the upper Domain (i.e. corresponding to the Domain Key 0.75).

The peer p identifies the level of its largest Domain and with that the Parent-Coordinator C_p one level higher, which is its parent node in the tree. The largest Domain a peer is responsible for is at level $level_p^{min}$, is calculated as follows:

$$level_p^{min} := min(i \in \mathbb{N} \; with \; K_p^i \in scope(p)). \tag{16}$$

The Parent-Coordinator C_p of peer p is then

$$C_p := C_p^{level_p^{min}-1} = owner(K_p^{level_p^{min}-1}). \tag{17}$$

This Parent-Coordinator C_p is responsible for a Domain, which is one step larger and has a Domain Key that lies not in the scope of peer p. To this Parent-Coordinator, peer p sends periodical update messages containing monitoring information. In the example, peer 0.74 sends update messages to the peer responsible for the Domain Key 0.5 and acts as a Coordinator for the Domains corresponding to the Domain Keys 0.75 and 0.8125, thus as Parent-Coordinator of the corresponding child nodes.

Additionally, peers may contact their Same-Level Coordinators to request their monitoring data. For this let, p be a given peer, which is a Coordinator for level i, thus responsible for the unified ID K_p^i. Its Same-Level Coordinators C_j^i with $C_J^i := \{C_j^i \mid j \in [1; \beta^i - 1], i > 0, \beta > 1\}$ are calculated as follows:

$$\begin{aligned} C_j^{i'} &= owner(K_p^i + j * (max(D_p^i) - min(D_p^i))),\ \forall j \in J \\ C_j^i &= \begin{cases} C_j^{i'}, & \text{if } C_j^{i'} \in [0; 1] \\ C_j^{i'} - 1, & \text{otherwise.} \end{cases} \end{aligned} \tag{18}$$

3.4. Joining and leaving the tree topology

To join the SkyEye tree topology, peers calculate their positions in the tree using Formulas (14) and (15) as well as their Parent-Coordinator. Afterwards, they request data from their Same-Level Coordinators, which can be calculated by using Formulas (18), in order to get some meaningful data for their first update message. Whenever a node receives answers on their requests or a certain time has elapsed, it aggregates its current data and sends the first update message to its Parent-Coordinator which is replied with an acknowledgment message. These update messages are used to inform the Parent-Coordinators about their *Sub-Coordinators*, i.e the child nodes in the tree. As the Parent-Coordinator and Sub-Coordinators are known to a peer through this simple step, the whole topology is connected. We rely on the assumption that the underlying DHT works properly. The maintenance of the tree topology is done with periodically sending update messages. No further steps are needed, even in the case of failing Parent-Coordinators or Sub-Coordinators.

The failing of a Parent-Coordinator C_*^l is detected by its Sub-Coordinators, which are Coordinators C_*^{l+1} of lower levels. They detect a missing acknowledgment message. As soon as a peer p identifies that the Parent-Coordinator C_p^l failed, no further actions need to be done. The peer uses the second postulated function `void route(key K, message M, nodehandle hint)` with the parameters $K = K_p^l$, M being the update message and optionally the previously retrieved ID of the new Coordinator as a *hint*. Eventually, the link to the Parent-Coordinator is updated. Although the information of the failed Coordinator is lost, the new Coordinator is updated when the Sub-Coordinators send their updates to their new Parent-Coordinator.

In the case of the failure of a Sub-Coordinator, nothing else needs to be done, as the new Sub-Coordinator determines its new Parent-Coordinator during the calculation of its position in the tree. No further maintenance is needed. We assume that peers fail and do not leave gracefully and that (strong) churn exist. Having only few message types in SkyEye for establishing and maintaining the tree topology, we avoid a stateful protocol which might lead to failures, deadlocks and instability. With the established SkyEye topology, we address the monitoring protocols in it next.

4. SkyEye: Information aggregation and monitoring protocols

Monitoring statistics give insights into the quality characteristics of a running p2p system. System-specific characteristics are described in a set of metrics, which are monitored and measured permanently on each peer. We use the term *metric* both for the classification and type of a measured value (e.g. hop count) and also for the actual value (e.g. 5). In this section, we describe the aggregation of metrics and the protocols and mechanisms used to gather, aggregate and disseminate system-specific monitoring information.

4.1. Aggregation of the metrics

This subsection describes the statistical presentation of the monitoring information and the diverse aggregation functions, which are used to aggregate metrics in SkyEye. An aggregation function is a function that returns a single value from a set of input values. Let *Values* be a continuous set of values (e.g the set of `double`), then an aggregation function f is a specific function (e.g. sum) that matches a non-empty set of values to a single value: $f : \wp(Values)/\{\} \rightarrow Values$

A typical aggregation function is the sum, minimum and cardinality of a set. In order to have usable aggregation functions for the monitoring purpose, the aggregation functions must be commutative and associative. With these properties, the order of the parameters in the aggregation function as well as the aggregation of aggregated results do not influence the final aggregation result. Table 1 depicts the list of statistical information which are retrieved from the measurements of the individual peers. The first four aggregation functions (*min*, *max*, *sum*, *count*) compute directly a single numeric value

Table 1
Formulas of the aggregation functions.

Aggregation Functions
$count(a_1, \ldots, a_n) = n$
$\min(a_1, \ldots, a_n) = \min(a_1, \min(a_2, \ldots \min(a_{n-1}, a_n) \ldots))$
$\max(a_1, \ldots, a_n) = \max(a_1, \max(a_2, \ldots \max(a_{n-1}, a_n) \ldots))$
$sum(a_1, \ldots, a_n) = \sum_{i=1}^{n} a_i$
$sum_squares(a_1, \ldots, a_n) = \sum_{i=1}^{n} a_i^2$
$mean(a_1, \ldots, a_n) = \frac{sum(a_1, \ldots, a_n)}{count(a_1, \ldots, a_n)}$
$variance(a_1 \ldots a_n) = \frac{sum_squares(a_1, \ldots, a_n)}{count(a_1, \ldots, a_n)} - \left(\frac{sum(a_1, \ldots, a_n)}{count(a_1, \ldots, a_n)}\right)^2$
$standard_deviation(a_1, \ldots, a_n) = \sqrt{variance(a_1, \ldots, a_n)}$

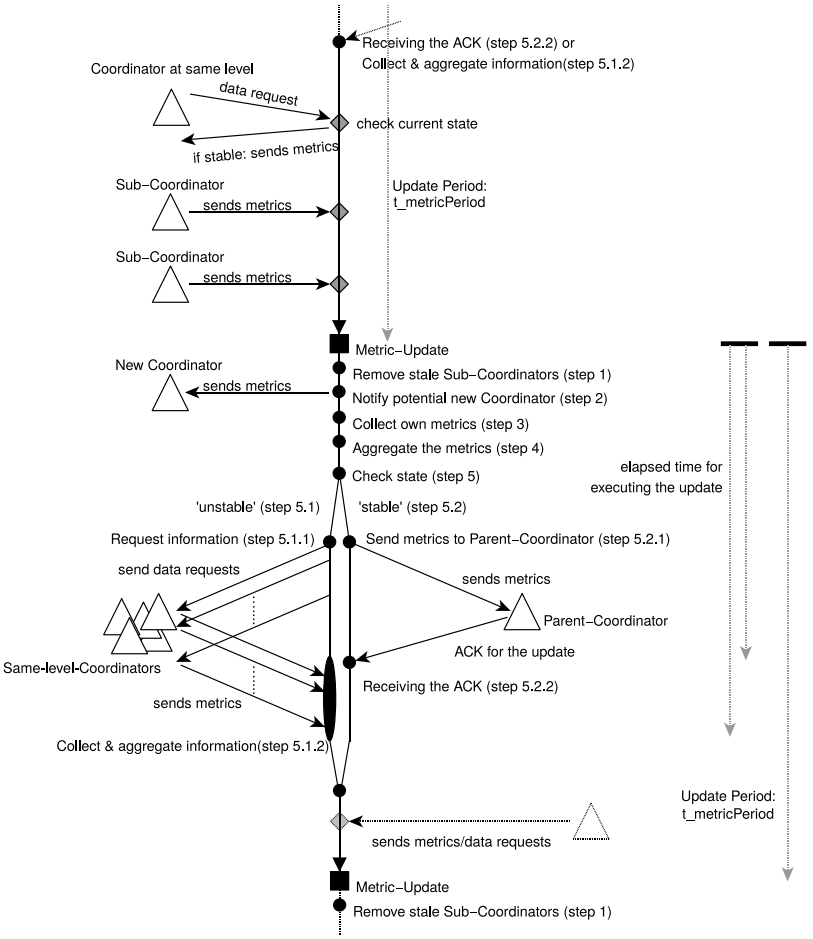


Fig. 3. Protocol for gathering system-specific information in the SkyEye tree.

based on several numeric input values. The mean, variance and standard deviation cannot be calculated directly, they are combined from other aggregation functions, one of them being the sum of squares. These aggregation functions fulfill are commutative and associative. In order to collect the information of all peers in one statistical representation, information from lower levels in the tree have to be aggregated and sent upwards in the tree. The design of the corresponding protocol for gathering a global view on system-specific information is described in the following.

4.2. Protocol for monitoring the system-specific statistics

In this subsection, we describe the protocol of SkyEye for gathering and disseminating the systems statistics in structured p2p systems. The protocol is depicted in Fig. 3. Its components and steps, stated in parenthesis, are explained throughout the next subsections.

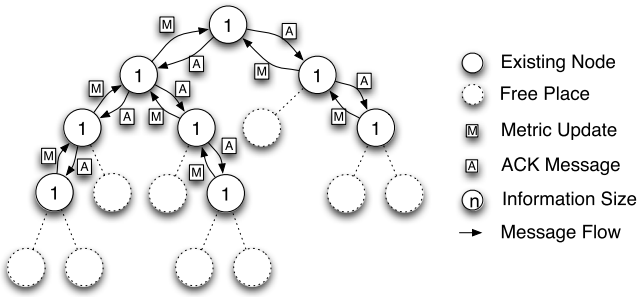


Fig. 4. Gathering and disseminating system statistics.

4.2.1. Gathering system statistics for a global view

In order to gather the system statistic on all peers in the network coordinately, all individual peer observations have to be aggregated. As a first step, each peer p uses its sensor component to derive its individual local measurement regarding all metrics (see step 3). Here, we use the assumed dependency (*Req*. **1**) for the provision of these metrics. The flow of the information is depicted in Fig. 4.

All peers are Coordinators for a Domain with possibly further peers in it. If no Sub-Coordinators exist in that Domain (i.e. the corresponding peer is the only peer in the Domain), it periodically sends its update information to its Parent-Coordinator. This update information consists only of the single measurement of the single peer in the Domain. With this, it implements the function (*Aim* **2**) for injecting the local measurements of single peers. The update intervals, $t_{metricPeriod}$ or *UI*, of the peers do not have to be synchronized. However, a similar value is recommended. The leaf nodes in the monitoring tree topology have no further tasks in the process of gathering monitoring information.

If a Coordinator has Sub-Coordinators in its Domain, it receives periodical update messages from them, termed *metric updates*. This is visualized in the upper part of Fig. 3 These metric information from the Sub-Domains are aggregated at the Coordinator and also aggregated with the local measurements of the Coordinator, implementing function (*Aim* **2**). Before aggregation, the Coordinator inspects the time stamp of every Sub-Coordinator entry. If the period between the update and the actual time exceeds a given threshold called $t_{oldMetrics}$, the entry is removed. This prevents the aggregation of old metrics from Sub-Coordinators, which may already have a new Coordinator or just went offline. The value for $t_{oldMetrics}$ is expressed as a multiple of the metric update interval $t_{metricPeriod}$ (alias *UI*). This is the first step in each metric-update iteration (see step 1). Previous simulations, which are not presented in the scope of this paper, show that this should range between 1.5 and 2 for no-churn and between 3 and 4 for churn scenarios, which is a good trade-off between freshness and availability of the monitoring data. Thus, the Coordinator of a Domain aggregates all fresh information of lower Domains and creates a metric update according to all aggregation functions (sum, minimum, maximum, count, mean, standard deviation), that describes the system status corresponding to its own Domain. This process is mentioned as step 4.

Depending on the current state of the node (see step 5), it either communicate with same-level coordinators or continues with the normal data flow. For the unstable state, see Section 4.2.3. Here, we assume that the state is stable. In the begin of the next update interval, the Coordinator checks, whether it is still in charge for its DomainKey, and if is the case, it sends the metric update to the Coordinator one level higher (step 5.2.1), and there, the same procedure is applied. This is done until the root is reached. At the root of the tree, the monitoring information regarding the system status is complete and can be used for the dissemination of this information downwards the tree. On lower levels, statistics on subsets of the tree are available.

4.2.2. Dissemination of the global view to all peers

For the dissemination of information from the top to the bottom of the tree, i.e. to all peers, a pushed-based and proactive approach is used in SkyEye. The dissemination of the information implements the function (*Aim* **1**) as stated in the section on functional requirements for the monitoring mechanism.

To reach all the peers, the established monitoring tree topology is used. Whenever a Coordinator or the root receives and processes a metric update from lower levels of the tree, it acknowledges this update to inform the Sub-Coordinator about the validity of its Parent-Coordinator and to provide the Sub-Coordinator with a fresh view on the system status. This step is mentioned as 5.2.2. The acknowledgment message contains the global system statistics received from one level higher in the tree. Thus, starting from the root, all lower Coordinators in the tree receive the acknowledgment message with the global system statistics and propagate it further down. Eventually, every peer receives the global statistics on the p2p system, with the degree of freshness depending on their position in the tree. By this, even new peers in the tree are directly informed about the p2p system's status within their first sent data.

4.2.3. Tackling churn

In the following we refer to churn as the event of peer departure or arrival in p2p networks. When churn occurs and peers leave the system, and thus the tree, the tree structure could be temporarily disrupted. Therefore we introduce two states for a node: stable and unstable. In the stable state, nodes follow the rules for the gathering and dissemination specified in Sections 4.2.1 and 4.2.2. In the unstable state (see step 5.1), which lasts for $t_{unstableState} = [1.5; 2] * t_{metricPeriod}$ at maximum after a change, nodes try to not pollute the network with false monitoring information and implement therefore two mechanisms.

Firstly, nodes taking over a Coordinator position do not participate immediately after the position change in the tree, instead, they wait for potential information pushed by other nodes. The timing for this wait time is important, as the corresponding Parent-Coordinator does not get any fresh data in the meantime and even deletes outdated information after a certain time interval. In Fig. 3, the according processes are step 5.1 and the incoming requests from Coordinators at the same level are visualized at the top of the figure. Nodes in an unstable state ask their same-level Coordinators for their estimation, see steps 5.1.1 and 5.1.2, which can be integrated into the aggregating message. In order to cap the maximum bandwidth consumption for this mechanism, Coordinators, which are on levels $\leq MaxGossipLevel$, randomly ask up to *MaxNoOfMsgs* of their neighboring Domains. Thus aggregated information for this single update interval is obtained and the node switches to a stable state.

Secondly, whenever a node changes its position due to a responsibility shift it may provide its information to the node which is now responsible for this position in the tree. This is mentioned as step 2 in Fig. 3. Thus the newly joined node can continue with the information basis of the former coordinator and starts in a stable state. Both steps are only relevant for one update interval, as within this interval fresh information from the node's Sub-Coordinators come in that is used in the following rounds.

5. Evaluation

In this section we evaluate the structured SkyEye, which communicates using a newly established tree topology on top of a DHT, with Push-Sum, a gossip-based approach by Blasa et al. [2]. We outline, compare and discuss similarities and differences caused by their structure in a parameter impact study in terms of its *accuracy*, *adaption* and *costs* in scenarios with and without churn.

5.1. Evaluation overview

To evaluate the quality of SkyEye, we simulated it in the PeerfactSim.KOM [28,29] simulator. The current global estimate is reported to the analyzer once every 15 s by a certain reporting node as a reference value. The node reporting the estimate is the current root node of the tree while using SkyEye or a fixed predefined node in Push-Sum. For emulating an application, we implemented one exemplary measurement using a *sinus* function as it provides a dynamic function which easily allows to measure the precision of the monitoring approaches. Each node calculates the value for the sinus function and applies it for its local measurement using the following formula using *SI* as sinus interval: $\frac{(time \bmod SI)2\pi}{SI}$. As further reference metrics we use the node count and the bandwidth consumption, which is also used to depict the costs.

The precision of the measured data and the adaption rate for the measured values to match the expected ones are important figures to rate the quality of the monitoring application. We measure both figures by comparing the measured data against the expected data for a discrete simulation time. The costs of monitoring can be measured by analyzing the traffic overhead produced by the application and network layer, separately. The following figures about the costs are measured in byte per minute for the whole network. To prevent runaway values from altering a result, ten different seeds were used for each simulation, producing different and independent results for each scenario.

In all scenarios the same actions were conducted, stating a join phase within the first 60 min, a 30 min stabilization window and the 90 min workload phase. The join process for all nodes is equally distributed over the complete duration of the join phase. The stabilization phase ensures the stability and correct setup of the p2p overlay. At the beginning of the workload phase, all nodes start to distribute their data to their intended receiver, equally distributed over the interval of one minute, to ensure a timely offset. Additionally the reporting node starts to report its global view to the analyzer at the same time. Within the scenario configurations, we modified the node count, running all scenarios with 5000 nodes.

The following sections cover the Scenarios A.1 to E.2 of Table 2. First, in Scenario A we perform a parameter study for the structured monitoring using SkyEye. We present several parameter variations, analyze the behavior and extract the best combinations depending on the adaption rate, precision and costs. Scenario B extends the evaluation of SkyEye by analyzing the impact of different overlays using the parameter set from Scenario A. In order to compare the SkyEye results, we show in Scenario C a summary of a parameter study for unstructured monitoring using the Push-Sum algorithm [2] operating on top of a Gnutella 0.6 overlay [30]. With optimal parameters obtained from Scenarios A and C we compare both approaches with and without churn in the last scenarios. In Scenario D we perform the comparison between the unstructured Push-Sum and structured SkyEye approach. Lastly in Scenario E, we analyze the influence of churn on SkyEye and the Push-Sum approach, each using the best working parameter set obtained from Scenarios A and C, respectively. Scenarios A to D have churn disabled, thus our mechanism to tackle churn, presented in Section 4.2.3, is only enabled in Scenario E.

Table 2
Setups with *UI*: update interval, β: branching factor, *RPE*: rounds per epoch.

General Settings	
Simulat. details	*PeerfactSim.KOM*, 10 seeds per setup. 5000 nodes simulated
Network model	Delay based on global network coordinates, no packet loss
Actions	0–60 min join overlay, from 90 min on start monitoring, at 180 min simulation ends. If churn is enabled, it will start at minute 90.
(A) Parameter Impact Studies for Structured Monitoring	
A.1 *UI*	Variation *UI*: 15 s / 30 s / 60s. Fixed: *Pastry*, $\beta = 4$, $SI = 10$ m
A.2 β	Variation β: 2 / 4 / 8. Fixed: *Pastry*, $UI = 30$ s, $SI = 10$ m
(B) Overlay Comparison for Structured Monitoring	
B Overlays	Variation: *Chord*, *Pastry*. Fixed: $\beta = 4$, $UI = 30$ s, $SI = 10$ min
(C) Parameter Impact Studies for Push-Sum	
C.1 *UI*	Variation *UI*: 5 s / 10 s / 20 s. Fixed: $RPE = 20$, $SI = 45$ m, *Gnutella 0.6*
C.2 *RPE*	Variation *RPE*: 10 / 20 / 30. Fixed: $UI = 10$ s, $SI = 45$ m, *Gnutella 0.6*
(D) Comparison of Structured and Unstructured Monitoring	
D Comparison	Variation: Structured (SkyEye) and Unstructured Monitoring (Push-Sum) Fixed: *Pastry*, $\beta = 4$, $UI = 15$ s, $SI = 45$ min, $RPE = 20$
(E) Enabling Churn for Structured and Unstructured Monitoring	
E.1 SkyEye Churn	Variation: Churn enabled/disabled Fixed: SkyEye, *Chord*, $\beta = 4$, $UI = 30$ s, $SI = 10$ m, $MaxGossipLevel = 7$, $MaxNoOfMsgs = 32$, $t_{unstableState} = 2 * UI$, $t_{oldMetrics} = 4 * UI$
E.2 Push-Sum Churn	Variation: Churn enabled/disabled Fixed: Push-Sum, *Chord*, $UI = 10$ s, $SI = 45$ m, $RPE = 20$

5.2. Scenario A - Parameter impact studies for skyeye

In this section, we evaluate the parameter impact studies, which we performed for SkyEye. Scenario A.1 describes and explains the outcome of changing the update interval, while Scenario A.2 shows the differences between various branching factors for building the monitoring tree.

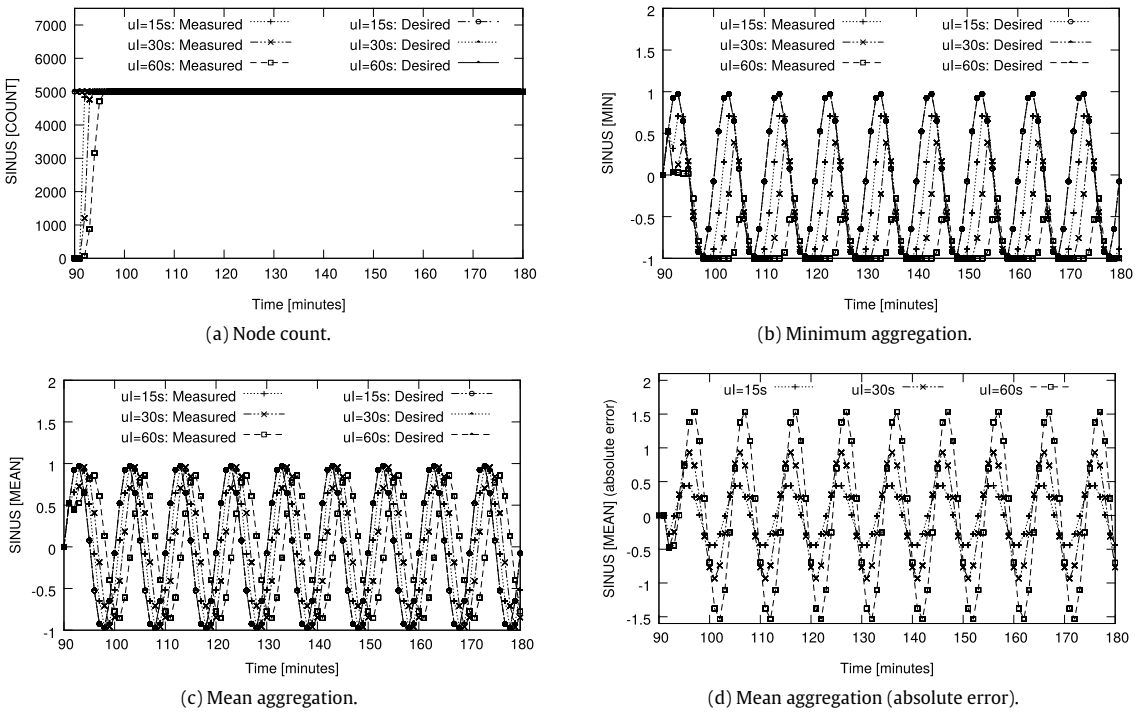
Scenario A.1 - Update Interval Variations.

Fig. 5 shows the results for the update interval variations on top of the *Pastry* overlay [31]. We first analyze the monitored node count during the workload phase in Fig. 5(a). Due to the lack of churn, the root node of the monitoring tree never resets its global estimation, so the measured node count stays at the correct value once the aggregated data from all nodes reaches the top of the tree. Although the difference in precision of the update intervals is minimal, the figure shows a slightly quicker adaption to the correct value for shorter intervals.

The delay within the adaption for different values of *UI* can be further observed in Fig. 5(b), showing the monitored minimum for the sinus function, here all nodes are synchronized to the global sinus function as a reference value. We observe the peak maximum value (the point at which the actual data value falls below the observed value again) for $UI = 30$ s reaching $+0.4$, meaning a tolerable delay. However, the peak point for $UI = 60$ s is at -0.3, showing a big gap in the adaption to a change in the reference data. Changing the update interval from $UI = 30$ s to $UI = 15$ s only results in a marginal improvement.

Fig. 5(c) shows the mean value observed by SkyEye for each update interval. As seen before by analyzing the minimum value, an update interval of $UI = 60$ s has considerably lower peak points compared to the actual value expected, hence the precision is far worse compared to a lower update interval. Maximum and minimum absolute error for $UI = 60$ s reach from $+1.5$ to -1.5 (as seen in Fig. 5(d)). Halving the update interval to $UI = 30$ s leads to a better precision (peak points are near to 1 and -1 respectively) and a better adaption rate (absolute error reaching from $+0.95$ to -0.95). Halving the update interval to $UI = 15$ s hardly changes the precision, as the absolute error reaches from $+0.45$ to -0.45.

Regarding the costs of SkyEye, we consider the amount of bytes sent by the monitoring approach during the workload phase, compared to the overall bytes sent on the network layer. For an update interval of $UI = 15$ s, the sum of overall data sent per minute stays at $1.032 * 10^7$ bytes, i.e. 2 kb/min per node, while the sum of bytes sent by SkyEye only is at $5.12 * 10^6$ bytes per minute, i.e. 1 kb/min per node. Increasing the update interval to $UI = 30$ s leads to a decrease in the sum of overall bytes sent through the net layer to $5.16 * 10^6$ byte/min (1032 byte/min per node) and reduces the amount of bytes sent through the monitoring layer to $2.56 * 10^6$ byte/min (512 byte/min per node). Further increasing the update interval to $UI = 60$ s again halves the sum of overall bytes sent to $2.58 * 10^6$ byte/min (516 byte/min per node) overall and



(a) Node count.

(b) Minimum aggregation.

(c) Mean aggregation.

(d) Mean aggregation (absolute error).

Fig. 5. Scenario A.1 - Update interval variations for SkyEye.

through the monitoring layer to $1.28 * 10^6$ byte/min (256 byte/min per node), further reducing the peer load. Considering the loss of precision and the much slower adaption rate, an update interval of $UI = 30$ s seems to be the best compromise between an acceptable peer load on the one hand and a high precision and quick adaption rate on the other.

Scenario A.2 - Branching Factor Variations.

We evaluate different branching factors β for SkyEye in Fig. 6, observing similar results to those in Scenario A.1.

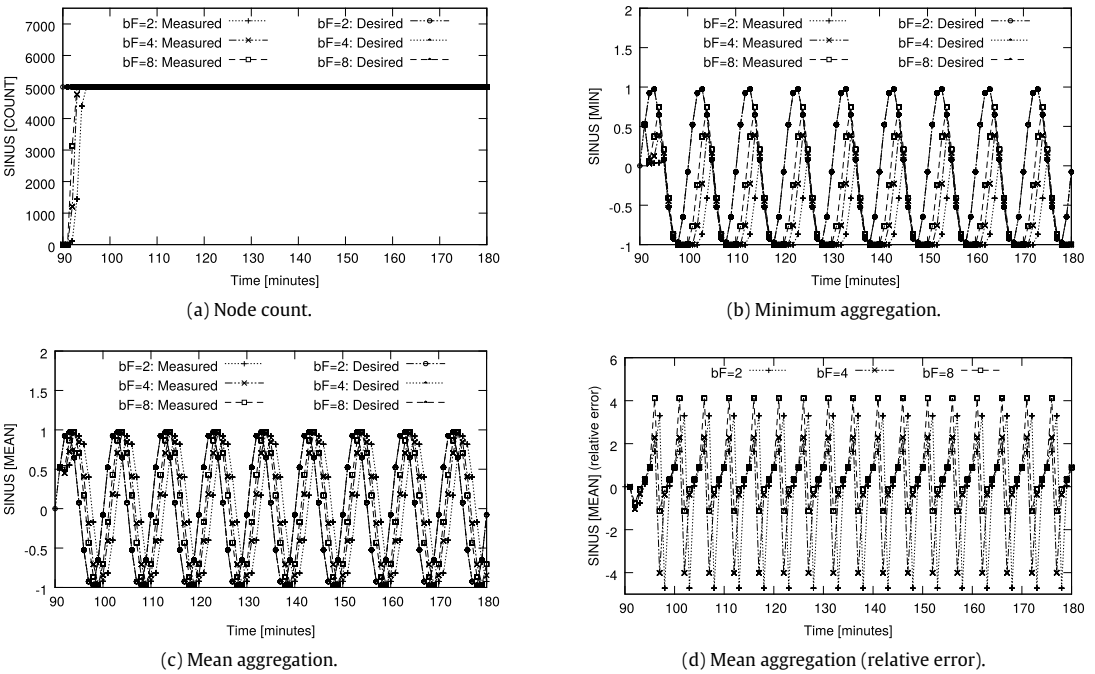
Looking at the monitored node count in Fig. 6(a), we see the same consistency in the amount of nodes observed by the monitoring application. Analogous to the previous Scenario A.1, we encounter a slightly higher adaption rate for $\beta = 4$ and $\beta = 8$ compared to $\beta = 2$ in the beginning of the workload phase.

We further observe a difference in the adaption rate for $\beta = 2$ in comparison to the higher branching factors in Fig. 6(b). The peak maximum value for all branching factors never reaches the true top of the sinus function, however $\beta = 8$ and $\beta = 4$ reach a value of 0.75 respectively 0.4, while $\beta = 2$ achieves a maximum peak of 0.1. The gap between the adaption rates can be explained by the increased depth of the monitoring tree for a lower branching factor and the resulting need for several data pushes to reach the root of the tree.

By taking a look at the mean value for the sinus function in Fig. 6(c), we observe $\beta = 8$ and $\beta = 4$ reaching both maximum and minimum peak values of the desired value, while the peak-to-peak amplitude of $\beta = 2$ is a bit shorter (by roughly 0.1). Evaluating error information (see Fig. 6(d)) confirms a considerable lower precision for $\beta = 2$ with an relative error ranging from -4.4 to 3.2. Increasing the branching factor to $\beta = 4$ also increases the adaption rate (relative error ranging from -4 to 2.1). Increasing the branching factor to $\beta = 8$ only slightly improves the adaption rate ranging from -1 to 4. However, increasing the tree height by decreasing β leads to more aggregation and update pushes required for nodes at lower levels.

Keeping the update interval at $UI = 30$ s and only changing the branching factor to different values, leaves the costs of the monitoring application unchanged as each node pushes an update message to its parent node and receives the current global estimate, known to the parent node, as a response. Thus, the sum of bytes sent through the monitoring layer remains the same for all branching factors. The traffic for monitoring purposes overall in the network 2560 kb/min, i.e. 512 byte/min per node. The amount of bytes sent through the network layer overall is also almost equal throughout all branching factors. For $\beta = 2$, $\beta = 4$ and $\beta = 8$ we have 5160 kb/min overall network load (1032 byte/min per node). Slight variations in the range of around 1 kb/min for all nodes together are due to different seed runs for a single parameter simulation.

Although the evaluation results show $\beta = 8$ is a slightly better choice over $\beta = 4$, we chose the latter as default value for upcoming simulations. Assuming a balanced tree $\frac{\beta-1}{\beta}$ of all nodes are leaf nodes. The greater β, the higher the share of leaf nodes is. With $\beta = 8$ the download bandwidth of around 87.5% of all nodes is not used as they are leaf nodes. Thus the bandwidth share is more decent with a lower branching factor. Furthermore, a lower branching factor is also desirable for



(a) Node count.

(b) Minimum aggregation.

(c) Mean aggregation.

(d) Mean aggregation (relative error).

Fig. 6. Scenario A.2 - Branching factor variations for SkyEye.

scenarios with churn, where the exchange on the same domain level comes into play. As an exchange of data between leaf nodes does not provide extra information we aim on a lower branching factor.

5.3. Scenario B - Overlay comparison for SkyEye

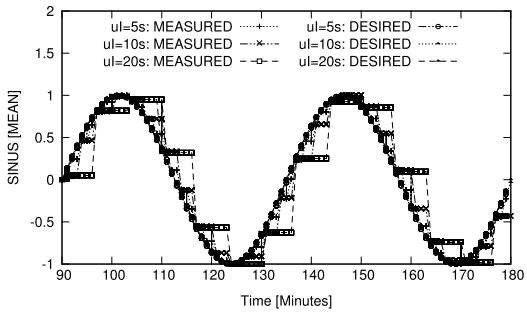
After evaluating the results for different update intervals and the branching factors, we chose $UI = 30$ s and $\beta = 4$ as optimized parameters for SkyEye and present a comparison between Chord [32] and Pastry [31]. In our simulations, we observed that the monitoring performance was the same for SkyEye operated on Chord as well as on Pastry. A difference is found in the costs. An average hop count for lookups takes 4.6 hops in Chord (Standard deviation: -3.20 and +6.25) and nearly 1 (1.01 - 1.03) in Pastry. Also Chord comes with $131.47 * 10^6$ byte/min overall (26,294 byte/min per node), while Pastry suffices with $5.16 * 10^6$ byte/min overall (1032 byte/min per node). We conclude higher costs and a worse adaption rate for the Chord overlay, which is most probably due to the fact, that it proactively fixes the routing tables instead of performing reactively. Thanks to a bigger routing table and the absence of churn, lookups in Pastry are answered in nearly one hop after a short reconstruction phase. The higher amount of hop counts in Chord results in the higher costs.

5.4. Scenario C: Parameter impact studies for push-sum

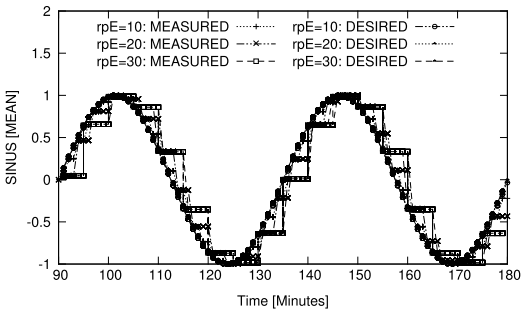
In this section we briefly assess a parameter study and the performance of an unstructured monitoring approach implementing the gossip-based algorithm Push-Sum [2]. For this, Scenarios C.1 and C.2 condense the parameter study of the update interval *UI* and epoch length *RPE*, respectively.

Scenario C.1 - Update Interval Variations.
Fig. 7(a) shows the results for Push-Sum with $UI = \{5\text{ s}, 10\text{ s}, 20\text{ s}\}$ and $RPE = 20$ fixed. The characteristic approximation per round of the Push-Sum algorithm can be extracted from Fig. 7(a). In summary, a reduced *UI* leads to higher adaption rate but induces higher costs due to more traffic. For $UI = \{5\text{ s}, 10\text{ s}, 20\text{ s}\}$ we have in overall traffic $30.4 * 10^6$ byte/min, $15.2 * 10^6$ byte/min and $7.6 * 10^6$ byte/min with per node costs of 6,080 byte/min, 3,040 byte/min and 1,520 byte/min, respectively. Therefore it is a tradeoff between adaption rate and costs, as halving *UI* leads to more than doubled costs. We will use $UI = 10$ s.

Scenario C.2 - Rounds Per Epoch Variations.
Fixed *UI* to 10 s, here, we evaluate the impact of rounds per epoch with $RPE = 10, 20, 30$. Fig. 7(b) shows the following habit: the more rounds per epoch the higher the precision of one measured value and the worse the adaption rate as the time to approximate a certain state of the system is extended. For $RPE = 10$ the correct value is almost never reached and $RPE = 30$ just keeps the correct value for a longer time thus we choose $RPE = 20$ along with $UI = 10$ s for a good tradeoff.

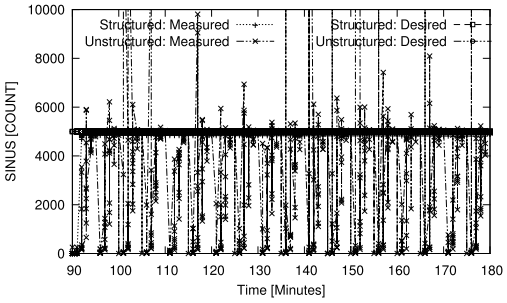


(a) Scenario C.1 - Update interval variations for push-sum.

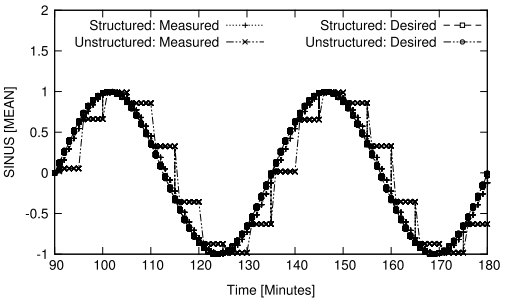


(b) Scenario C.2 - Rounds per epoch variations for push-sum.

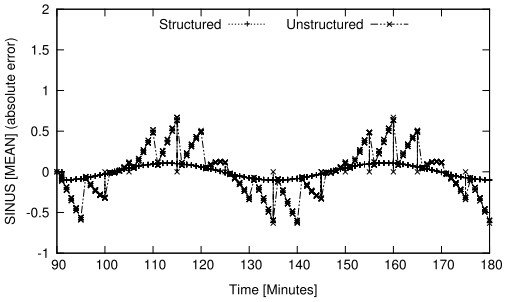
Fig. 7. Results of scenario C.1 and C.2.



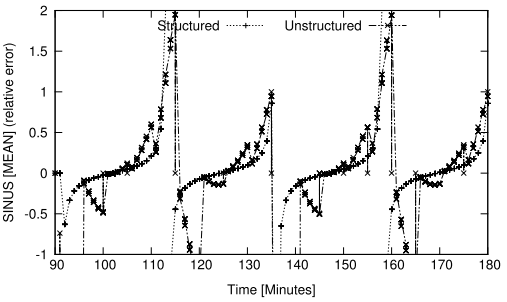
(a) Node count.



(b) Mean aggregation.



(c) Mean aggregation (absolute error).



(d) Mean aggregation (relative error).

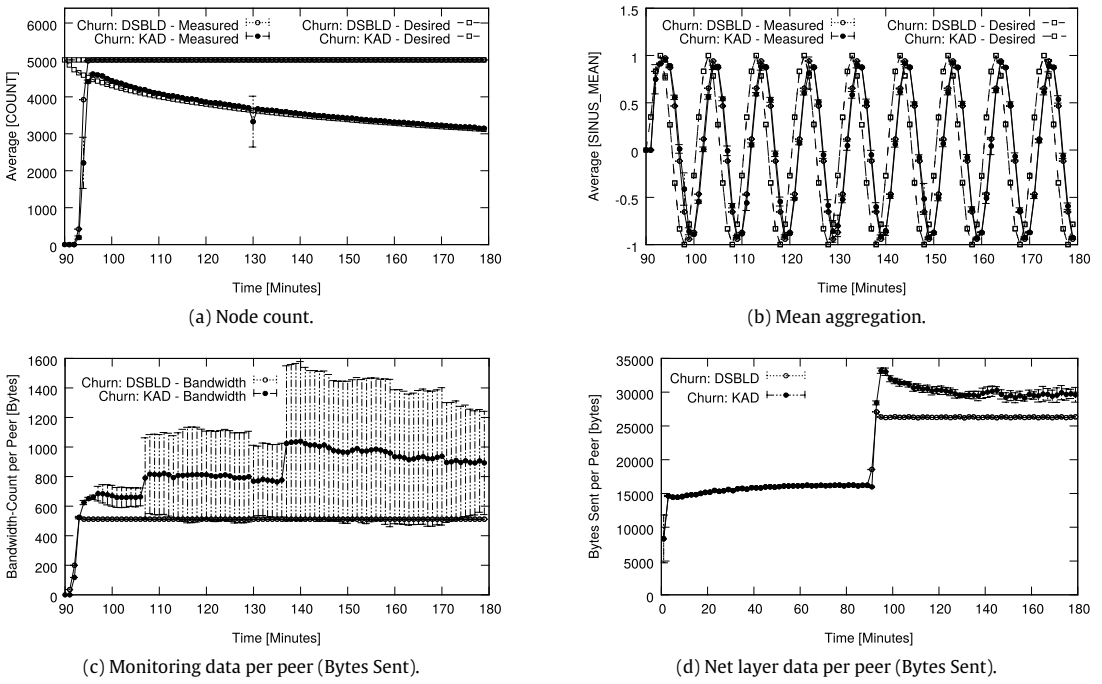
Fig. 8. Scenario D - Comparing SkyEye and push-sum.

5.5. Scenario D - Comparison of SkyEye and push-sum

In this section, we compare both SkyEye and Push-Sum on top of Pastry. We verify our conclusions from both parameter impact studies of SkyEye and Push-Sum, by comparing both approaches on the Pastry overlay. We ran all simulations with optimized settings, choosing $\beta = 4$ and $RPE = 20$ with $SI = 10$ m, to improve visibility of the epoch behavior. For UI we decided to apply $UI = 15$ s instead of $UI = 30$ s for SkyEye and $UI = 10$ s for unstructured in favor for the latter. Having a common parameter setting, we are able to compare both approaches on a similar level.

Looking at the node count in Fig. 8(a), we observe SkyEye continuously displaying the correct value after approximately two minutes, while the Push-Sum is trying to even out the correct value during each epoch. Since we choose $UI = 15$ s as an compromise, our observations are within the expected scope of the results we got in Fig. 7(a) for $UI = 10$ s and $UI = 20$ s.

Analyzing the mean value for the sinus function in Fig. 8(b), we spot the measured fluctuations due to the use of epochs in Push-Sum, compared to the smooth measurement curve of SkyEye. Evaluating the absolute error in Fig. 8(c), we observe the peak error reaching from -0.7 to $+0.7$ for Push-Sum, while SkyEye stays within very thin margins around zero with a maximum peak at ± 0.1. The relative error (see Fig. 8(d)) demonstrates a consistent behavior for SkyEye with peak errors around the zero value of the curve, and converging to zero near the maximum amplitude of the sinus function. However, within Push-Sum, the relative error oscillates wildly.



(a) Node count.

(b) Mean aggregation.

(c) Monitoring data per peer (Bytes Sent).

(d) Net layer data per peer (Bytes Sent).

Fig. 9. Scenario E.1 - SkyEye on chord with churn.

Finally, we take a look at the bandwidth costs. SkyEye induces monitoring traffic of $5.12 * 10^6$ byte/min (1 kb/min per node), resulting in an overall traffic of $10.32 * 10^6$ byte/min (2 kb/min per node) which contains DHT stabilization traffic. Push-Sum induces monitoring traffic of $10.2 * 10^6$ byte/min (2 kb/min per node), resulting in an overall traffic of $10.2 * 10^6$ byte/min (2 kb/min per node), as the unstructured network does not require further stabilization traffic. We observe almost twice the amount of monitoring traffic sent by Push-Sum compared to the bandwidth costs of SkyEye. This is due to larger messages in the unstructured approach as we additionally need components like the weight and neighbor queue. Nonetheless, the comparison of the overall traffic reveals almost the same bandwidth consumption on the network layer. We conclude that SkyEye outperforms the unstructured in terms of precision and adaption rate with an almost equal bandwidth consumption.

5.6. Scenario E - Impact study for enabling churn

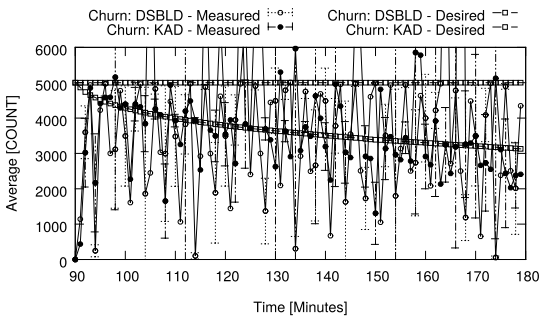
After having identified the suitable parameter settings for SkyEye and Push-Sum, next we evaluate their behavior under churn. For churn, we used the KAD churn model based on real KAD overlay measurements by M. Steiner et al. [33]. In SkyEye, we enable our mechanism to tackle churn as described in Section 4.2.3. Instead of using the superior Pastry overlay in terms of performance as mentioned in Section 5.3, we ran a Chord overlay in this scenario. Tests showed that Pastry performs poorly under churn without any workload that passively fixes the routing tables. Chord fixes proactively its routing table and is thus more stable under churn.

Scenario E.1 - Enabling churn for SkyEye.

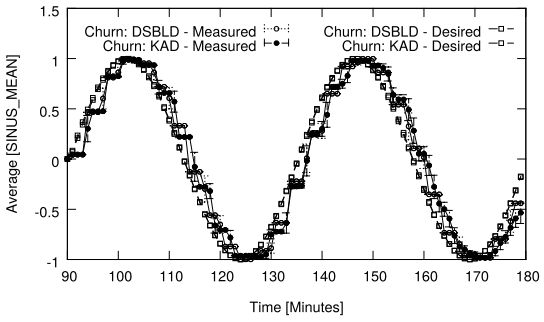
In SkyEye, same-level contacts are used to obtain a first estimation for a Coordinator on a new position. For the relevant parameters *MaxGossipLevel* and *MaxNoOfMsgs* in this context, we found in simulations the values 7 and 32 suitable. As the tree is not perfectly balanced, levels deeper than 10 might exist, thus the limit of 7 is valid.

Fig. 9 shows the results of the comparison between enabled and disabled churn on top of a Chord overlay for SkyEye. Since this scenario heavily depends on the churn events during a run and its influence on the tree structure, we demonstrate the average values with a confidence level of 95%. Additionally, each inner figure contains the no-churn data, so a comparison is easily made.

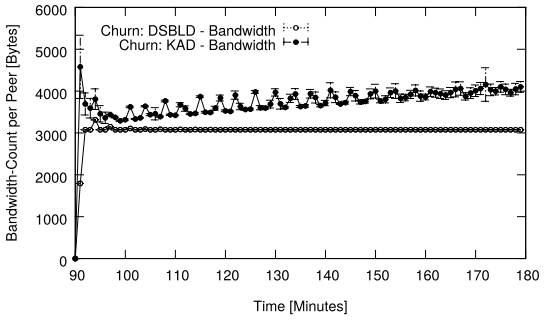
Fig. 9(a) shows a fast convergence of the global view and a high measurement precision. The measured peer count is slightly higher than the desired, in numbers, maximum 4% at minute 98 and afterwards around 1%. The dropout at minute 130 can be explained by a leaving coordinator positioned at a lower level in the tree, i.e. near the root node. This leads to a reconstruction of the tree and loss of monitoring information on position change. The mechanisms described in Section 4.2.3 ensure no reset data, i.e. only the local view, will be pushed upwards the tree by exchange monitoring views



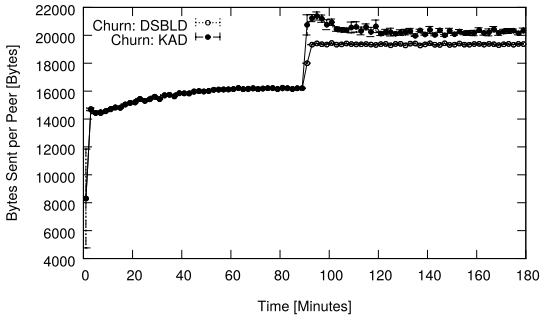
(a) Node count.



(b) Mean aggregation.



(c) Monitoring data per peer (Bytes Sent).



(d) Net layer data per peer (Bytes Sent).

Fig. 10. Scenario E.2 - Push-Sum on chord with churn.

with Coordinators at the same level. As other nodes at the same level may also face the same lack of information no data is forwarded to the parent. This leads to a starving parent which erases monitoring data after a certain amount of time ($4 * UI$). Without gossiping with same-level Coordinators, dropouts are more often and dramatic.

The precision under churn is still very good (cf. Fig. 9(b)). At the beginning of the monitoring phase, the mean aggregation reveals a slightly greater phase shift in the direction of the x-axis than the no-churn scenario. Starting at minute 100, the churn and no-churn phases are almost equal. Besides, the peak-to-peak value lost <0.1 and the confidence intervals are fairly small. Unlike the minimum aggregation where the confidence intervals are relatively large and the peak reaches only a value of -0.05. This is as information is kept back longer instead of sending probably wrong information. Thus, on a certain level, we favor precision over freshness when it comes to churn.

The cost for monitoring only, shown in Fig. 9(c), reveal a large increase in traffic throughout the whole workload phase. The maximum increase is at minute 140 with 103% which is still fairly low with $1034 \frac{\text{byte}}{\text{minute}} = 17 \frac{\text{byte}}{\text{second}}$. This is due to gossiping with same-level Coordinators in the presence of churn. We encounter large confidence intervals as the traffic for gossip messages highly depends on the nodes affected by churn and their position in the tree. Fig. 9(d) shows the increase in the total traffic which mainly originates from Chord's stabilization tasks. The increase is maximum 26% at minute 95 and average at minute 147 with 12%. Thus, enabling churn results in a slight loss in precision, worse adaptation rate and 12%-26% higher bandwidth consumption.

Scenario E.2 - Enabling churn for push-sum.

Push-Sum, as shown in Fig. 10, is less affected by churn, due to the absence of structure and the need to maintain a structure. Both precision and adaption rate are only marginally affected. The node count in Fig. 10(a) shows the same fluctuations of the measured value regardless of the participation of churn. However, enabling churn produces a slightly higher amount of overshooting values, which might be caused by nodes that go offline while keeping relevant information, such as a crucial weight value for the computation of the node count. For the rest of the epoch, the information remains lost. The results of the mean aggregation in Fig. 10(b) confirm the observations we also explained in Section 5.4. Both simulations produce very similar results regarding precision and adaption rate.

Lastly, Fig. 10(c) presents the traffic created by Push-Sum. Until the end of the simulation we have an accumulated increase of 33% compared to the no-churn scenario. This is due to an increase in denied gossip requests while waiting for a never coming answer from an offline node and, as a result, further gossip request messages. The traffic also increases on the network layer by 10% at maximum in minute 95 and on average by 4% in minute 153. As in the scenario with SkyEye, the increase is owed by the stabilize tasks of Chord. But while SkyEye requires for each aggregation step a lookup operation, the

unstructured approach only gossips within the neighborhood. This is reflected by a smaller increase of the network layer bandwidth consumption.

Summarizing, our implementation of the Push-Sum algorithm is hardly affected by churn, thus the precision and adaption rate remains the same (low level) as mentioned in Section 5.4. However, the number of gossip messages sent increases due to an increase in deny and gossip request messages.

6. Conclusions and future work

SkyEye implements a monitoring layer for structured p2p systems providing the API for key-based routing. It creates an additional (unified) ID space, which is recursively partitioned in ID intervals called *Domains*. For each Domain, a characteristic ID is calculated using a deterministic function that maps the Domain to a single ID in it, called *Domain Key*. The peer responsible for the Domain, termed *Coordinator* of the Domain, is identified by being responsible for the Domain Key in the DHT. The recursively partitioned Domains and the corresponding Coordinators build a tree structure. Peers identify their position in the tree based on their ID and periodically send information messages to the Coordinator one level above them in the tree. These messages, termed updates, contain aggregatable statistics information. Coordinators periodically pass the aggregated statistics one level higher in the tree in a push-based manner, if a Coordinator is in a stable state, i.e. it gathered enough information from its children. Otherwise, if a Coordinator is in an unstable state, it communicate with its same-level Coordinators in order to get a first estimation. The aggregated statistics received by a Coordinator describe the statistics of the peers in the corresponding Domain. messages are used to disseminate the global statistical view retrieved from higher levels in the tree as well as maintaining the tree topology. In conclusion, a tree structure is built on top of a DHT to gather and disseminate global system statistics.

In the evaluation we identified good parameter settings for our approach SkyEye with the unstructured monitoring approach Push-Sum [2] through simulations and provides a comparison of both approaches without and under churn.

With the identified optimal parameters set and no churn, SkyEye proves to be highly precise, stable and with almost no fluctuation. The precision of the unstructured monitoring approach Push-Sum is far worse due to the use of epochs, which rely on distributing the same measurement over a rather long duration. Since SkyEye completely relies on an intact tree infrastructure, churn might lead to a decrease in the precision. SkyEye uses a protocol for same-level Coordinators to compensate information loss through node position changes in the tree. We recognized some losses in the adaptation rate and slightly more bandwidth consumption. On the other side, Push-Sum is affected by churn as well, mainly in counting nodes, since information is dropped and the requesting node is locked for a short time period by a node leaving the network. Finally we can outline that SkyEye fulfills all requirements stated in Section 1 and performs very precisely and cost efficiently in scenarios with and without churn.

For future work we plan to investigate the influence of malicious nodes and reconstruct possible data loss. Also, we want to monitor node capacities and make them searchable. Monitoring data in such systems, e.g. by creating data statistic rankings, may prove useful to provide a certain level of quality of service, thus improving the p2p experience.

Acknowledgments

This work was supported by the DFG Grant GR4498 1-1 (OverlayMeter).

References

[1] F. Dabek, B.Y. Zhao, P. Druschel, J. Kubiatowicz, I. Stoica, Towards a Common API for Structured Peer-to-Peer Overlays, in: IPTPS'03: Proc. of the Int. Workshop on Peer-to-Peer Systems, in: LNCS, 2735, Springer, 2003, pp. 33–44.
[2] F. Blasa, S. Cafiero, G. Fortino, G.D. Fatta, Symmetric Push-Sum Protocol for Decentralised Aggregation, in: AP2PS'11: The Third Int. Conf. on Advances in P2P Systems, 2011.
[3] K. Graffi, D. Stingl, C. Gross, H. Nguyen, A. Kovacevic, R. Steinmetz, Towards a P2P Cloud: Reliable Resource Reservations in Unreliable P2P Systems, in: Proc. of the IEEE Int. Conf. on Parallel and Distributed Systems, ICPADS 2010, 2010, pp. 27–34.
[4] K. Graffi, K. Pussep, S. Kaune, A. Kovacevic, N. Liebau, R. Steinmetz, Overlay Bandwidth Management: Scheduling and Active Queue Management of Overlay Flows, in: Proc. of the IEEE Conf. on Local Computer Networks, LCN, 2007, pp. 334–342.
[5] B. Guidi, T. Amft, A.D. Salve, K. Graffi, L. Ricci, DiDuSoNet: A P2P Architecture for distributed Dunbar-based Social Networks, Peer-to-Peer Netw. Appl. 9 (6) (2016) 1177–1194.
[6] M. Benter, M. Divband, S. Kniesburges, A. Koutsopoulos, K. Graffi, Ca-Re-Chord: A Churn Resistant Self-Stabilizing Chord Overlay Network, in: Proc. of the Conf. on Networked Systems, NetSys 2013, 2013, pp. 27–34.
[7] K. Graffi, D. Stingl, J. Rueckert, A. Kovacevic, R. Steinmetz, Monitoring and Management of Structured Peer-to-Peer Systems, in: Proc of the Int. Conf. on Peer-To-Peer Computing, P2P 2009, 2009, pp. 311–320.
[8] K. Graffi, Monitoring and Management of Peer-to-Peer Systems (Ph.D. thesis), Technische Universität Darmstadt, 2010. http://tuprints.ulb.tu-darmstadt.de/2248/.
[9] J. Case, M. Fedor, M. Schoffstall, J. Davin, RFC 1157: Simple Network Management Protocol, SNMP, 1990, http://ietf.org/rfc/rfc1157.txt.
[10] G. von Bochmann, A. Hafid, Some principles for quality of service management, Distrib. Syst. Eng. 4 (1) (1997) 16–27.
[11] M. Jelasity, A. Montresor, Ö. Babaoglu, T-Man: Gossip-based fast overlay topology construction, Comput. Netw. 53 (13) (2009) 2321–2339.
[12] R. Bhagwan, G. Varghese, G. Voelker, CONE: Augmenting DHTs to Support Distributed Resource Discovery, Technical Report CS2003-0755, University of California, San Diego, 2003.
[13] Z. Zhang, S. Shi, J. Zhu, SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT, in: IPTPS'03: Proc. of the Int. Workshop on Peer-To-Peer Systems, in: LNCS, vol. 2735, Springer, 2003.

[14] K. Albrecht, R. Arnold, M. Gähwiler, R. Wattenhofer, Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave, in: IEEE P2P'04: Proc. of the Int. Conf. on Peer-To-Peer Computing, IEEE, 2004.
[15] R. van Renesse, A. Bozdog, Willow: DHT, Aggregation, and Publish/Subscribe in one Protocol, in: IPTPS'04: Proc. of the Int. Workshop on Peer-To-Peer Systems, in: LNCS, vol. 3279, Springer, 2004, pp. 173–183.
[16] S. Idreos, M. Koubarakis, C. Tryfonopoulos, P2P-DIET: An Extensible P2P Service that Unifies Ad-Hoc and Continuous Querying in Super-Peer Networks, in: ACM SIGMOD'04: Proc. of the Int. Conf. on Management of Data, ACM, 2004, pp. 933–934.
[17] D. Shen, Y. Shao, T. Nie, Y. Kou, Z. Wang, G. Yu, HilbertChord: A P2P Framework for Service Resources Management, in: Springer GPC'08: Proc. of the Int. Conf. on Advances in Grid and Pervasive Computing, 2008, pp. 331–342.
[18] C.P. Hall, A. Carzaniga, Uniform Sampling for Directed P2P Networks, in: Springer Euro-Par'09: Proc. of Int. Euro-Par Conf., 2009, pp. 511–522.
[19] L. Massoulié, E.L. Merrer, A.-M. Kermarrec, A.J. Ganesh, Peer Counting and Sampling in Overlay Networks: Random Walk Methods, in: ACM PODC'06: Proc. of the Annual ACM Symposium on Principles of Distributed Computing, 2006, pp. 123–132.
[20] N. Tölgyesi, M. Jelasity, Adaptive peer sampling with newscast, in: LNCS Euro-Par'09: Proc. of the Int. Euro-Par Conf., 2009, pp. 523–534.
[21] D. Kempe, A. Dobra, J. Gehrke, Gossip-Based Computation of Aggregate Information, in: IEEE FOCS'03: Proc. of the Symposium on Foundations of Computer Science, 2003, pp. 482–491.
[22] W. Dan, Z. Rongjuan, A Layered Resource Management Model in P2P System, in: IEEE PDCAT'05: Proc. of the Int. Conf. on Parallel and Distributed Computing, Applications and Technologies, 2005, pp. 569–572.
[23] J. Li, D.-Y. Lim, A Robust Aggregation Tree on Distributed Hash Tables, in: Proc. of the 2004 Student Oxygen Workshop, 2004.
[24] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, IEEE Commun. Mag. 40 (8) (2002) 102–114.
[25] M. Amjad, M. Sharif, M.K. Afzal, S.W. Kim, TinyOS-new trends, comparative views, and supported sensing applications: A review, IEEE Sens. J. 16 (9) (2016) 2865–2889.
[26] N. Devroye, M. Vu, V. Tarokh, Cognitive radio networks, IEEE Signal Process. Mag. 25 (6) (2008).
[27] F. Akhtar, M.H. Rehmani, M. Reisslein, White space: Definitional perspectives and their role in exploiting spectrum opportunities, Telecommun. Policy 40 (4) (2016) 319–331.
[28] A. Kovacevic, S. Kaune, H. Heckel, A. Mink, K. Graffi, O. Heckmann, R. Steinmetz, PeerfactSim.KOM - A Simulator for Large-Scale Peer-to-Peer Networks, Tech. Rep. Tr-2006-06, TU Darmstadt, 2006.
[29] K. Graffi, PeerfactSim.KOM: A P2P System Simulator - Experiences and Lessons Learned, in: IEEE P2P'11: Proc. of the Int. Conf. on Peer-To-Peer Computing, 2011.
[30] T. Klingberg, R. Manfredi, Gnutella 0.6, 2002, http://rfc-gnutella.sourceforge.net/src/rfc-06-draft.html.
[31] A.I.T. Rowstron, P. Druschel, Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, in: IFIP/ACM Middleware'01: Proc. of the Int. Conf. on Distributed Systems Platforms, 2001.
[32] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, in: SIGCOMM'01: Proc. of the Int. Conf. on Applications, Technologies, Architectures, and Protocols for Comp. Comm., 2001, pp. 149–160.
[33] M. Steiner, T. En-Najjary, E.W. Biersack, Analyzing peer behavior in KAD, Institut Eurecom, France, 2007. Tech.