Minicamp: Middleware for Incomplete Participation in Structured Peer-to-Peer Monitoring Protocols

Andreas Disterhöft
Technology of Social Networks, University of Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf, Germany
Email: disterhoeft@cs.uni-duesseldorf.de

Kalman Graffi
Technology of Social Networks, University of Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf, Germany
Email: graffi@hhu.de

***Abstract*—In order to provide a certain level of quality to the users of a peer-to-peer network, the network's status needs to be monitored in the first place. In this paper, we tackle the problem of monitoring peer-to-peer networks with structured monitoring solutions with the constraint of incomplete participation of the nodes in the monitoring process. We propose a middleware overlay which operates between the common p2p overlay aiming at routing and resolving lookups and a monitoring overlay aiming to obtain and disseminate statistics on the network. Our solution covers an assignment, maintenance and probing scheme to build up a common global view for all nodes in the common p2p overlay. Evaluation shows a high precision for the assignment and probing scheme in the presence of only a fraction of the nodes in the network contributing to the monitoring.**

I. Introduction

Peer-to-peer networks provide various functions in a fully decentralized manner. In order to maintain a quality level of these functions, information on the quality and cost of the network needs to be monitored, analyzed and used for improvements, as suggested in [10]. In the last decade, different monitoring solutions for peer-to-peer networks have been proposed. Structured approached, such as CONE [5], SDIMS [17], SOMO [18] and SkyEye.KOM [11], establish a further monitoring topology on top of the p2p overlay to have efficient tree structures for obtaining and disseminating monitoring information from and to the peers. To the best knowledge of the authors, none of the existing monitoring approaches address the problem of incomplete participation by the nodes. If nodes in the overlay do not all participate in monitoring solutions, corresponding monitoring structures, e.g. trees, may get disturbed or even collapse totally. This leads to a devastating loss of the monitoring system's precision.

We label nodes which participate in the monitoring as active nodes, and those which do not as passive nodes. In this paper we extend our work on Minicamp [7] and present our middleware overlay in detail and provide an analysis of its performance and costs. In particular, we address the case of incomplete participation in the monitoring and present a solution that provides precise monitoring results, even if only a small share of the nodes are active. We assume that a large fraction of the nodes might refuse or be incapable to participate in monitoring, due to malicious intent or due to running a protocol version, which does not consider participation in the monitoring overlay. This might be the case, when the monitoring feature is released with a new version of the p2p software and only a subset of the nodes update to this new feature and become so active nodes. E.g. for Java [2] we see that only half of all users updated to the most recent version (Java 8), which has been released three years ago. The same goes for updates of Windows [1] versions.

We propose Minicamp, a *m*iddleware for *in*complete part*ici*p*a*tion in *m*onitoring *p*rocotols, with following aims:

A1) Provide a functionality to support the monitoring of active nodes within the p2p overlay
A2) Monitor both active and passive nodes in order to provide a full global view on all active nodes and a limited global view on passive nodes
A3) Offer the possibility to reuse current monitoring solutions

Due to the existence of good structured monitoring solutions, these should be used for the monitoring of the active nodes. We propose the middleware Minicamp to allow the additional monitoring of the passive nodes, which are incapable to participate in the monitoring approach. Minicamp is an overlay structure on top of the common p2p overlay, which organizes how active nodes probe and monitor passive nodes.

In a nutshell, we present the approach of Minicamp in Sec. II as follows. First (see Sec. II-A), the active nodes must find each other in the p2p overlay full of active and passive nodes in order to create a middleware overlay consisting only of active nodes. After the creation of the Minicamp overlay, in Sec. II-B, a bijective projection between the p2p and Minicamp overlay defines the responsibility of an active node in the ID space. See in Sec. II-C that each active node then continuously scans for passive nodes in his responsibility range. Then, in Sec. II-D, active nodes continuously probe the identified passive nodes to obtain useful information for the system-wide monitoring. Lastly in Sec. II-E, a structured monitoring overlay is applied atop Minicamp, i.e. an over-over-overlay, to gather and disseminate the compiled monitoring information from the active nodes. By this, the statistics on active and passive peers, which are probed by their responsible active nodes, are obtained. Sec. III offers an in-depth evaluation of the proposed solution. We show that Minicamp supports monitoring in networks with incomplete participation and even

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

with only 1% active nodes the precision is very high. We conclude in Sec. IV.

II. Our Solution: Minicamp

Minicamp, our *m*iddleware for *in*complete part*ic*ip*a*tion in structured p2p *m*onitoring *p*rocotols, allows active nodes to find each other, gather in a middleware overlay in which a clear assignment is made, which active node has to scan which ID ranges to find and probe passive nodes in it. Using a conventional structured monitoring protocol, such as SkyEye.KOM [11], on top of the Minicamp overlay allows to obtain and provide global statistics both on all active and passive nodes. Minicamp states following requirements:

R1) The p2p overlay should implement the KBR-Interface [6].
R2) The p2p overlay uses an one-dimensional ID space or can be translated to such, e.g. use space-filling curves [14].
R3) Node and object IDs must be located within one ID space
R4) Convex node responsibility range: if node A with ID id_A answers on a lookup of the ID id_{lookup} then the node A is at least responsible for all IDs between id_A and id_{lookup}

A. Active Nodes Discovery / Join Middleware Overlay

First, all active nodes have to find each other and join/create the Minicamp overlay. Therefore each active node runs a discovery process. We propose *KBR-supported* and *DHT-supported* discovery processes.

1) KBR-supported Discovery: The KBR-supported discovery is an 'optimistic probabilistic message sending' approach, which uses the Requirement R1). Here, each active node periodically sends special messages to its neighbors and random distant IDs. If an active node answers, this node is used as bootstrap to join the Minicamp overlay. Due to the probabilistic approach one may not find any other active nodes, especially, if only a small fraction of nodes is active. Then, a node is meant to be the first active node and creates a new Minicamp overlay. Consequently, more than one Minicamp overlay might exist, thus isolated overlays are possible. To mitigate this phenomenon, the periodically obtained active nodes answering on the special messages, are looked up in the Minicamp overlay. If another parallel overlay is discovered, an overlay merge will be initiated. For this purpose an overlay merger proposed by Amft [3] can be used.

2) DHT-supported Discovery: This approach uses a logical linked list containing contact data of active nodes. Each active node appends its contact information to the list, that is implemented in the overlay's DHT. Its head is located at a well-defined position, e.g. $id = hash('activeNodeList')$. Random access for the i^{th} entry is implemented by concatenating the outcome of the hash function, i.e. $hash^i('activeNodeList')$. Its tail is identified by null, i.e. an empty entry. Entries are removed by marking them as deleted. This is achieved by using soft-state entries which are marked if its owner do not update it periodically. For the list's maintenance, successfully joined nodes append their contact data to the list or override a marked entry. The lookup speed can be improved by implementing an exponential and binary-like search for either a marked entry or the list's tail. Race conditions may occur for the appending process when two active nodes write at the same position. Therefore, we propose to initiate lookups after a certain time to check whether previously saved contact data has been overwritten. If it happened, the affected node restarts the operation. The advantage over the KBR-supported approach is that the list does not need to be complete and correct as one single alive bootstrap node is enough for bootstrapping. Due to race condition handling, fragmented or isolated overlays are not possible, even while creating the Minicamp overlay.

B. Responsibility Assignment

In order to partition the ID space among the active nodes, i.e. the responsibility assignment, we necessitate Requirements R2) and R3) and propose a bijective projection from the p2p overlay to our Minicamp overlay. Here, we discuss the first step on how to assign ID space ranges to active nodes.

Let $\mathbb{S}_{all}$ be the set of all nodes in the p2p overlay. Let $\mathbb{S}_{active} \subseteq \mathbb{S}_{all}$ and $\mathbb{S}_{passive} \subseteq \mathbb{S}_{all}$ be the subsets of all active and passive nodes, respectively, with $\mathbb{S}_{passive} \cap \mathbb{S}_{active} = \emptyset$ and $\mathbb{S}_{passive} \cup \mathbb{S}_{active} = \mathbb{S}_{all}$. Furthermore, let $\tau_{p2p} \subset \mathbb{N}_0$ and $\tau_{mw} \subset \mathbb{N}_0$ be the ID spaces of the p2p and Minicamp overlay, respectively. The responsibility ranges for the p2p and Minicamp overlay are defined as following:

$$\phi_{p2p} := \mathbb{S}_{all} \rightarrow \tau_{p2p}^2 : n \mapsto [l_n\,;\,u_n] \quad (1)$$

$$\phi_{mw} := \mathbb{S}_{active} \rightarrow \tau_{mw}^2 : n \mapsto [l_n\,;\,u_n] \quad (2)$$

The ID range is denoted by an interval $[l;u]$, where l is the lower and u the upper bound. In general, active nodes need to map their responsible ID range in the Minicamp overlay to the p2p overlay's ID space.

$$\phi_{proj} := \tau_{mw} \rightarrow \tau_{p2p} : i \mapsto \frac{i * \tau_{p2p_{max}}}{\tau_{mw_{max}}} \quad (3)$$

$$\phi_{projRange} := \tau_{mw}^2 \rightarrow \tau_{p2p}^2 : [l;u] \mapsto [\phi_{proj}(l);\phi_{proj}(u)] \quad (4)$$

$$\phi_{scanRange} := \mathbb{S}_{active} \rightarrow \tau_{p2p}^2 : n \mapsto \phi_{projRange}(\phi_{mw}(n)) \quad (5)$$

This is done by using (3). For the inverse function of (3) one simply need to swap $\tau_{p2p_{max}}$ and $\tau_{mw_{max}}$. Having the mapping of ranges between both overlays (4), we define the mapping for the scan range, each active node has to scan passive nodes for. This mapping is shown in (5), where each active node's responsibility range of the Minicamp overlay is projected on the p2p overlay. By doing so, the p2p overlay's ID space is completely partitioned between these active nodes. On churn events in the Minicamp overlay, its stabilization methods come into play, which lead to responsibility changes and directly affect the scan range for the active nodes in (5). After successfully calculated the scanning range for an active node, in the next subsection we refer to the scanning process.

C. Scanning for passive nodes

The next step is the scan process which starts immediately after an active node has joined the Minicamp overlay (see Section II-A). Here each active node aims to find all passive

Algorithm 1 Building Block for Scanning and Maintenance

```
1: local variables
2:     respRanges, responsibility range data structure
3:     unscanned, queue holding unscanned ranges
4: end local variables
Input: Maintenance step delay delay
5: procedure N.START(delay)
6:     unscanned ← INITQUEUE()
7:     respRanges ← INITRESPONSIBILITYRANGEPRIORITYQUEUE()
8:     unscanned.ADDTAIL(φ_SCANRANGE(n))        ▷ For φ_scanRange see (5), Step 1
9:     n.INITIALSCAN()
10:    while true do                                                ▷ Step 3
11:        n.MAINTENANCESTEP()
12:        n.SLEEP(delay)
13:    end while
14: end procedure
15: procedure N.INITIALSCAN()
16:    while ([l; u] ← unscanned.REMOVEHEAD()) ≠ null do            ▷ Step 2
17:        n.SCANONCE([l; u])                                       ▷ call specific impl.
18:    end while
19: end procedure
20: procedure N.MAINTENANCESTEP()
21:    if ([l; u] ← unscanned.REMOVEHEAD()) ≠ null then  ▷ blocks available
22:        n.SCANONCE([l; u])
23:    else                          ▷ No unscanned blocks. Refresh oldest block
24:        n.FIXOLDEST(respRanges.GETOLDESTENTRY())   ▷ call specific impl.
25:    end if
26: end procedure
```

peers in its responsibility range defined in Sec. II-B. From the global point of view, the desired set we obtain is:

$$\forall\, \alpha \in \mathbb{S}_{active}\,,\, \Pi_\alpha \;=\; \Big\{\, \rho \;\Big|\quad \rho \in \mathbb{S}_{passive} \;\wedge\; id_\rho \in \phi_{scanRange}(\alpha) \,\Big\} \quad (6)$$

The general idea of all approaches is the following: Each active node $\alpha \in \mathbb{S}_{active}$ scans its responsible range $\phi_{scanRange}(\alpha)$ for passive nodes by starting lookups and react on their outcome. For the following, we presuppose that Requirement R4) is met, as this is the only given information in the generic scan process (see Section II-C1). Also, we can exclude other nodes lying between the lookup ID and passive node ID. Our aim is to efficiently scan the mentioned range by keeping i) the number of lookups and ii) the maintaining costs minimal for keeping (6) up-to-date. The lookup operation of the common p2p overlay, returning exactly one node responsible for a certain ID, is the only function that can be used to scan the range. Thus, we need $\mathcal{O}(|\Pi_\alpha|)$ lookups.

Algorithm 1 presents the common building block for our proposed scanning and maintaining algorithms. Two important data structures are initialized in line 5: The `unscanned` queue holds unscanned ranges and the `respRanges` holds our current scanned responsibility ranges with the node and its range. In Step 1 (line 8) the scan range is calculated and the initial charting is started (line 15). Here, unscanned blocks are removed from the queue till it is empty (Step 2). In each iteration the implementation of `scanOnce` (line 17) is called. After the initial chartering is done, Step 3 starts in line 10 and performs one `maintenanceStep` every `delay` seconds. In each `maintenanceStep` (line 20), an unscanned block is processed, if available. Otherwise the oldest block is extracted from `respRanges` and the specific implementation for the method `fixOldest` is called (line 24).

In the following, we propose three approaches implementing this building block. If information about the responsibility management of the common p2p overlay is not available,

Algorithm 2 Generic Scanning and Maintenance

```
Input: Interval with lower and upper bounds [l; u]
1: procedure N.SCANONCE([l; u])
2:     m ← (l+u)/2
3:     r_m ← n.NODELOOKUP(m)
4:     respRanges.ADDRANGEFORNODE(r_m, [MIN(m,r_m.ID),MAX(m,r_m.ID)])
5:     unscanned.ADDTAILIFNOTEMPTY([l;MIN(m,r_m.ID)+1])
6:     unscanned.ADDTAILIFNOTEMPTY([MAX(m,r_m.ID)+1;u])
7: end procedure
Input: Oldest passive node's ID and its range oNode, [l; u]
8: procedure N.FIXOLDEST(oNode, [l; u])
9:     lNode ← n.NODELOOKUP(l)
10:    uNode ← n.NODELOOKUP(u)
11:    if lNode.ID = oNode.ID = uNode.ID then                ▷ no new nodes
12:        respRanges.REFRESHTIMESTAMP(oNode)
13:    end if
14:    if lNode.ID ≠ oNode.ID then          ▷ new passive node on lower side
15:        alive ← n.PING(oNode)
16:        respRanges.ADDRANGEFORNODE(lNode,
       [MIN(lNode.ID,l);MAX(lNode.ID,l)])
17:        if alive = true then                          ▷ known passive node is alive
18:            respRanges.SETLOWERBOUNDFORNODE(oNode,oNode.ID)
19:            unscanned.ADDTAILIFNOTEMPTY(
           [MAX(lNode.ID,l)+1;oNode.ID−1])
20:        else
21:            respRanges.REMOVEENTRY(oNode)
22:            unscanned.ADDTAILIFNOTEMPTY([MAX(lNode.ID,l)+1;u])
23:        end if
24:        if lNode.ID < l then      ▷ tricky case: check lower neighboring range
25:            respRanges.REMOVEENTRIESBETWEEN(lNode.ID,l)
26:            nNode,[l_l; u_l] ← respRanges.GETENTRYFORID(lNode.ID − 1)
27:            alive ← n.PING(nNode)
28:            if alive = true then
29:                respRanges.SETUPPERBOUNDFORNODE(nNode,nNode.ID)
30:                unscanned.ADDTAILIFNOTEMPTY([nNode.ID + 1; lNode.ID −
               1])
31:            else
32:                respRanges.REMOVEENTRY(nNode)
33:                unscanned.ADDTAILIFNOTEMPTY([l_l; lNode.ID − 1])
34:            end if
35:        end if
36:    end if
37:    if uNode.ID != oNode.ID then
38:        ...              ▷ new passive node on upper side: Analogous to above case
39:    end if
40: end procedure
```

a *generic* divide and conquer approach is loaded. Otherwise, optimized algorithms for either *unilateral* overlays like Chord [16] or *bilateral* overlays like Pastry [15] are used.

1) Scanning: Generic Approach: Algorithm 2 shows the method implementations of the generic approach desired by the common building block. The main idea of this generic solution is a divide and conquer chartering and maintaining. In each step of the chartering phase (line 1), the ID in the middle of the range is looked up. For the responding node a new entry is created or an existing entry's range is adapted, respectively. Two unscanned ranges may result after this step, which are added to the unscanned queue. The chartering phase can be parallelized in order to speed up this initial phase.

During the maintenance phase, starting from line 8, the oldest chartered block is fixed. Here, exactly two lookups are performed, for the lower and upper bound. If no new nodes have been found (line 11), the block's timestamp is refreshed and this iteration is done. Otherwise, without the loss of generality, a new passive node has been spotted on the lower side (line 14). The upper side is handled accordingly (line 37). The range of this node is created respectively adapted and a ping to the fixing block's owner is performed. A ping is an operation which sends one single message to a peer in one hop. If the peer is alive, see line 17, the lower bound

of this node is deleted and added to the unscanned ranges as there might be more new nodes. The scan for this range is performed right in the next maintenance step. Otherwise, if the peer is offline, see line 20, the entry of this node is removed and the remaining range is added to the unscanned queue. Additionally, lines 24 to 35 handle the case that a node is responding from a neighboring range. Here, all blocks in the range of the new node's ID and the lower bound's ID are deleted, if and only if, the block's owner is located inside the range. Furthermore, the block located on the lower side of the new node is retrieved and its owner is pinged in order to handle the situation similar to the above mentioned procedure.

For the complexity calculation we distinguish between the chartering and maintenance phase. Let N be the number of nodes, i.e. $|\mathbb{S}_{all}|$, and $X := \tau_{p2p}$ the ID space of the common p2p overlay. In the average and worst case all peers are equally distributed throughout the whole ID space X. During the chartering case, we find in N lookups the equally distributed peers and the remaining space is divided into $N+1$ sub spaces, where the divide and conquer algorithm has to perform on ranges of size $\frac{X}{N+1}$. This results into $N + (N+1) * (log_2(\frac{X}{(N+1)}))$ lookups. As $X \gg N$, there are $\mathcal{O}(N * log(X))$ lookups.

During the maintenance phase, exactly two lookups and two pings at maximum are performed. Whenever we encounter a churn event, a small part of the whole scan range has to be rescanned, which is of the same complexity as seen above but with $N = \frac{|\mathbb{S}_{all}|}{|\mathbb{S}_{active}|}$ and $X = \frac{X}{|\mathbb{S}_{active}|}$.

2) Scanning: Unilateral Overlays: Unilateral overlays are overlays with a ring topology where the responsibility range of each node starts with its node ID and goes till the node ID + 1 of its predecessor or node ID - 1 of its successor, respectively. This mapping is clarified by (7).

$$\phi_{unilat} := \mathbb{S}_{all} \rightarrow \tau^2_{p2p} \; : n \mapsto [id_n; id_{n_{successor}} - 1] \vee \\ n \mapsto [id_{n_{predecessor}} + 1; id_n] \quad (7)$$

Without loss of generality, we assume the second case, which is analogous to Chord [16]. Our proposed approach is shown in Algorithm 3. The idea for the initial chartering is to start lookups for the ID of a known passive node lower bound - 1 in order to get its direct predecessor. This is shown in line 1, where for each step exactly one lookup for the lower bound is started, its corresponding node is saved in our `respRanges` datastructure and the range of the ID of the answering node + 1 with the upper bound is added to the `unscanned` datastructure, which will be picked up in the following scan iteration. Unlike the generic case, this Algorithm can not be parallelized as each subsequent call is dependent on the previous iteration. Nevertheless, we only need $\frac{|\mathbb{S}_{all}|}{|\mathbb{S}_{active}|}$ lookups per active node, i.e. $\mathcal{O}(N)$, as we are able to find one new passive node with each lookup.

For the maintenance (line 6) the lower bound of the oldest entry's range is looked up first. If the answering node is the known node (line 9) the entry is refreshed. Otherwise, if a new node answers, it is relevant whether the new node is between

Algorithm 3 Unilateral Scanning and Maintenance

```
Input: Interval with lower and upper bounds [l; u]
 1: procedure N.SCANONCE([l; u])
 2:     r_l ← n.NODELOOKUP(l)
 3:     respRanges.ADDRANGEFORNODE(r_l, [l,r_l.ID])
 4:     unscanned.ADDTAILIFNOTEMPTY([r_l + 1;u])
 5: end procedure
Input: Oldest passive node's ID and its range oNode, [l; u]
 6: procedure N.FIXOLDEST(oNode, [l; u])
 7:     nNode ← n.NODELOOKUP(l)
 8:     isNewNode ← respRanges.ISNEWNODE(nNode)
 9:     respRanges.ADDRANGEFORNODE(nNode,[l; nNode.ID])
10:     if nNode.ID < oNode.ID then
11:         alive ← n.PING(oNode)
12:         if alive = true then                    ▷ known passive node is alive
13:             respRanges.SETLOWERBOUNDFORNODEWITHREFRESH(oNode,
    nNode.ID+1)
14:         else
15:             respRanges.REMOVEENTRY(oNode)
16:             unscanned.ADDTAIL([nNode.ID + 1; oNode.ID])
17:         end if
18:     else
19:         respRanges.REMOVEENTRIESBETWEEN(l,nNode.ID)
20:         if isNewNode = true then
21:             rNode ← respRanges.GETNODEFORID(nNode.ID+1)
22:             respRanges.SETLOWERBOUNDFORNODEWITHOUTREFRESH(
    rNode, nNode.ID+1)
23:         end if
24:     end if
25: end procedure
```

the entry's lower bound and the old node or its ID is greater than the old one's ID. In the former case (lines 10 to 17), the old node may be still online, so a ping is performed in line 11. If the older node is still online, its lower bound is set accordingly and the entry is refreshed, otherwise, its entry is removed and the remaining range is added to `unscanned`. The latter case, where the new node's ID is greater than the old known node's ID, indicates that the old known node and potentially one or more of its successors are offline (lines 18 to 24). In this case, all entries between the lower bound and the new node must be removed and, if the new node is not a known node, its successor's lower bound has to be adapted. The successor is extracted from our mapping, thus, we do not refresh the entry of the successor, as we do not fully checked the status of this entry. Per maintenance step we need exactly one lookup and one ping at maximum. As each active node maintains $\frac{|\mathbb{S}_{all}|}{|\mathbb{S}_{active}|}$ entries in average, for a complete maintenance pass $\mathcal{O}(N)$ lookups are needed, which results in a total complexity of $\mathcal{O}(N * \log(N))$.

3) Scanning: Bilateral Overlays: Bilateral overlays are overlays with a ring topology, where the responsibility range covers all IDs closest to this node, i.e. reaches from half way to the predecessor till half way to the successor of the node. See (8) for the definition of this mapping.

$$\phi_{bilat} := \quad \mathbb{S}_{all} \rightarrow \tau^2_{p2p} \; : \\ n \mapsto \left[\lceil \tfrac{id_n + id_{n_{predecessor}}}{2} \rceil; \lceil \tfrac{id_n + id_{n_{successor}}}{2} \rceil - 1\right] \quad (8)$$

Generally, the width of the lower and upper hand-side range is different. A prominent example for this mapping is Pastry [15]. We give the idea of this algorithm, whose main difference to the generic approach is that the algorithm can stop its search for more nodes between two already mapped nodes much earlier. Thus, if the node responsible for the upper bound ID is answering, as of (8), no other nodes can lie between the lower

and upper ID, thus the ranges for the lower and upper entry can be adjusted. For the complexity of the chartering phase we need exactly $N*(N-1)$ lookups. Each maintenance step takes exactly 2 lookups, thus the complexity equals the charterings.

D. Probe Passive Nodes for Monitoring Data

In this step, we use of the Minicamp overlay, assuming (6) is given with a certain precision. More precisely, each active node $\alpha \in \mathbb{S}_{active}$ probes the passive nodes in its responsibility range Π_α in order to obtain the monitoring information provided by these nodes. Probing means to extract monitoring information by indirect requests, as the passive nodes do not want or cannot support the actual monitoring protocol. The probed information is stored locally in a transient storage space. The main objective is to obtain the following function: Let δ_{γ_t} be the local monitoring view of node γ at time t. See (9) for the set we aim to approximate, where ϵ should be minimal, thus the data should be as fresh as possible. Such probing tasks can be implemented using our interfaces.

$$\forall\, \alpha \in \mathbb{S}_{active}\,,\, \Delta_{\alpha_t} = \left\{ \delta_{\gamma_{t+\epsilon}} \mid \gamma \in \Pi_\alpha \ \wedge\ \epsilon > 0 \right\} \quad (9)$$

E. Apply Monitoring Overlay

In the final step, the active nodes use existing monitoring approaches such as SkyEye.KOM [11], to aggregate and disseminate the monitoring information. For the input of the monitoring component, active nodes include both, their local measurements and the transient data base containing measurements from the passive nodes they probe in the ID space they are responsible for. In total information of all nodes in the p2p overlay are covered.

III. Evaluation

Our main interest for the evaluation of Minicamp are:

- Performance of the Minicamp overlay's topology. Including the analysis of overlapping areas, distribution throughout the active nodes, correctness respectively precision of the obtained view by the Minicamp overlay vs. the global view described by (6).
- Precision of the monitoring data including probed information of passive nodes compared to a 100% precision probing mechanism and compared to the reference precision where all nodes are active.
- Costs: Traffic used for maintaining and assigning passive nodes to active nodes and the probing mechanism. The delay (i.e. freshness) of the monitoring data using the middleware and monitoring component is compared to the freshness in a network with only active nodes.

A. Methodology

We implemented our solution, described in Section II, in the peer-to-peer simulator PeerfactSim.KOM [8] and ran the structured monitoring overlay SkyEye.KOM [11] on top.

The simulation of probing mechanisms (see Section II-D) is leaned on real world bandwidth probing approaches like mentioned by Goebel et al. [9]. The authors send packet trains through the Internet and measure the bandwidth of the receiver, i.e. a passive node, by interpreting the delay and the packet train's payload. Further, Hooghiemstra et al. [13] propose a Weibull distribution for the Internet delay on investigated routes across the network. We simulate the probing error, of e.g. the passive node's bandwidth, by taking a value from the Weibull distribution [13], move it towards the exact value by an offset of the desired mean minus the distribution's mean, and scale it to the target's interval bounds, if applicable.

TABLE I
Parameters chosen for our simulations

Parameter	Value(s)
Number of nodes	5000
Number of seeds	≥ 10
Churn	Active (from minute 160), Inactive
Simulation time	210 minutes
Common / Minicamp overlay	Chord / Chord
Active node share	1%, 10%, 50%
Mapping: Lookup delay / Max. pending lookups	10 seconds / 10
Real Data Provider (TailDrop / ScalingPolicy)	Lookup Hop Count (0% / No)
Deterministic Data Provider (TailDrop / ScalingPolicy)	Various (Sinus, Dirac, ZicZac) (0%, 1E-06%, 1% / No, Reroll)

The share of active node in the network are motivated by the distribution of software version shares starting with the release of a new version and its progress to the most popular version. Statistics [1] and [2] show the version share of Windows and Java. Windows 7 and Java 7/8 have a share of roughly 50% and WinXP and Java 6 have a share of around 10%. Additionally to these shares we added a share of 1% which is a synthetic scenario and stress test for our mechanism and represents a fresh release of a version.

In the scope of this paper we present simulations with Chord [16] operating on both overlays, the common and middleware. Furthermore, we focus on the optimized search for the Chord protocol mentioned in Section II-C2.

A detailed list of parameters and their variations we are focusing on is provided by Table I. Each scenario is simulated at least ten times with 5000 nodes. Active nodes in the Minicamp overlay decide all 10 seconds which lookups have to be started to map or maintain the area they are responsible for. The monitoring overlay's data provider is either real or deterministic data. In the latter, nodes model a representative function for their dynamic parameters, which solely depends on time, e.g. the sinus function. Due to the fact that the aforementioned Weibull distribution and possible values of real data live in the interval of $[0; \infty]$, no tail drop and scaling policies have been applied here. For deterministic data, we simulated scenarios with various tail drops, which lead to a bounded interval of the heavy-tailed Weibull distribution, and scaling policies to fit the outcome of the bounded distribution on a bounded target interval, e.g. $[-1; 1]$ for sinus, are applied. The Reroll policy ignores values outside of the target interval and forces a reroll. In each scenario, we compare the passive node probing mechanism with 100% precision data obtained from passive nodes and the desired 100% participation of both, active and passive nodes, in the monitoring component, which equals a scenario without any middleware overlay. Evaluation

results are presented in Figures 1– 5 and discussed below.

B. Middleware Overlay: Performance

For the evaluation of Minicamp's performance, we focus on scenarios with and without churn and take a closer look on the influence of the active node share.

The assignment of passive to active nodes and the overlay's ID space share are shown in Fig. 1a and 1b. The distribution highly depends on the used hash function, thus the projection (4), which is more like a normal than a desirable equal distribution. In the 1% scenario, each of the active nodes is responsible for at least one passive node. For the 10% and 50% scenario this share degrades - around 90% respectively only 50% of the active nodes maintain at least one passive node. The influence of churn is minimal.

Each active node scans its area for passive nodes. Fig. 1d shows the correctness of node mapping with churn disabled which is interchangeable with the progress of the initial scan. Scenarios with a share of 10% and 50% of active nodes can complete this initial scan after 5 and 10 minutes while the 1% scenario takes more than one hour to initially map the whole ID space. This is mainly due to the lookup search interval, which is 10 seconds, and an uneven distribution of the load. The active node with the highest load scanned an area containing 445 passive nodes (cf. Fig. 1a), which needs 75 minutes. Fig. 1c shows the initial and maintenance scans under churn. Scenarios with 10% and 50% active nodes can hold the area mapping up with an error less than 5% while the 1% scenario can continuously improve its scan to a nearly 90% mapping of the area.

The correctness of the node mapping versus the global view described by (6) is shown in Fig. 1d and 1e. Both present the precision of the found nodes and the false positive in the already mapped area. Without churn, we can deduce a fast and precise node mapping in the scanned area. The scenario under churn (cf. Fig. 1e) reveals the impact of changing responsibilities due to active node churn and raising false positives and decreasing precision due to passive node churn. The former results in new unscanned areas, thus potential new passive nodes. Fig. 1f reveals the number of losing passive nodes due to mismapped areas is low (less than 5%). The decrease of the precision comes from (re-)joining passive nodes and the increase of the false positives from passive nodes going offline which is noticed when this passive node's area is fixed. Nonetheless the precision of all active node share scenarios is at least 90%, whereas the 10% active node share performs best (98%). The maximum false positive rate is 5%.

A *stability* value is calculated by active nodes to estimate the churn rate in their area and adjust the number of lookups to scan or fix their area. Fig. 2 shows the course of this parameter with churn enabled. It reveals a raising value at the begin while initially scanning the area and an overall lower value for smaller active node shares due to higher area share.

C. Monitoring: Precision

To evaluate the monitoring performance, we refer to the worst case. The worst case is the scenario with churn enabled and 1% node participation, as these need to probe the remaining 99%. We consider as relevant metrics for the monitoring performance the freshness and the precision. For this, we compare the *Default*, *Full* and *Partly* scenarios. Default is a scenario where all nodes are active. Full considers incomplete participation and runs the Minicamp overlay with a full precision, optimal probing. Partly is like Full but applies our probing methodology (cf. Section III-A). We monitor both real data (lookup hop count) and deterministic data (sinus).

Fig. 3a shows the relative error count in the number of data sets included in the aggregated global view of the monitoring system. The difference between the Default, Full and Partly graph reflects the actual number of involved nodes, thus, active nodes plus their scanned passive nodes. In contrast to Fig. 1e these numbers include the active nodes and false positives.

Our used monitoring approach SkyEye.KOM aggregates data up a tree and forwards the aggregated data down the tree. This data contains aggregatable metrics like the minimum, maximum and mean of the data sets. The absolute error of the minimum and maximum in the aggregated data set to the desired global view is an indicator for the freshness of data.

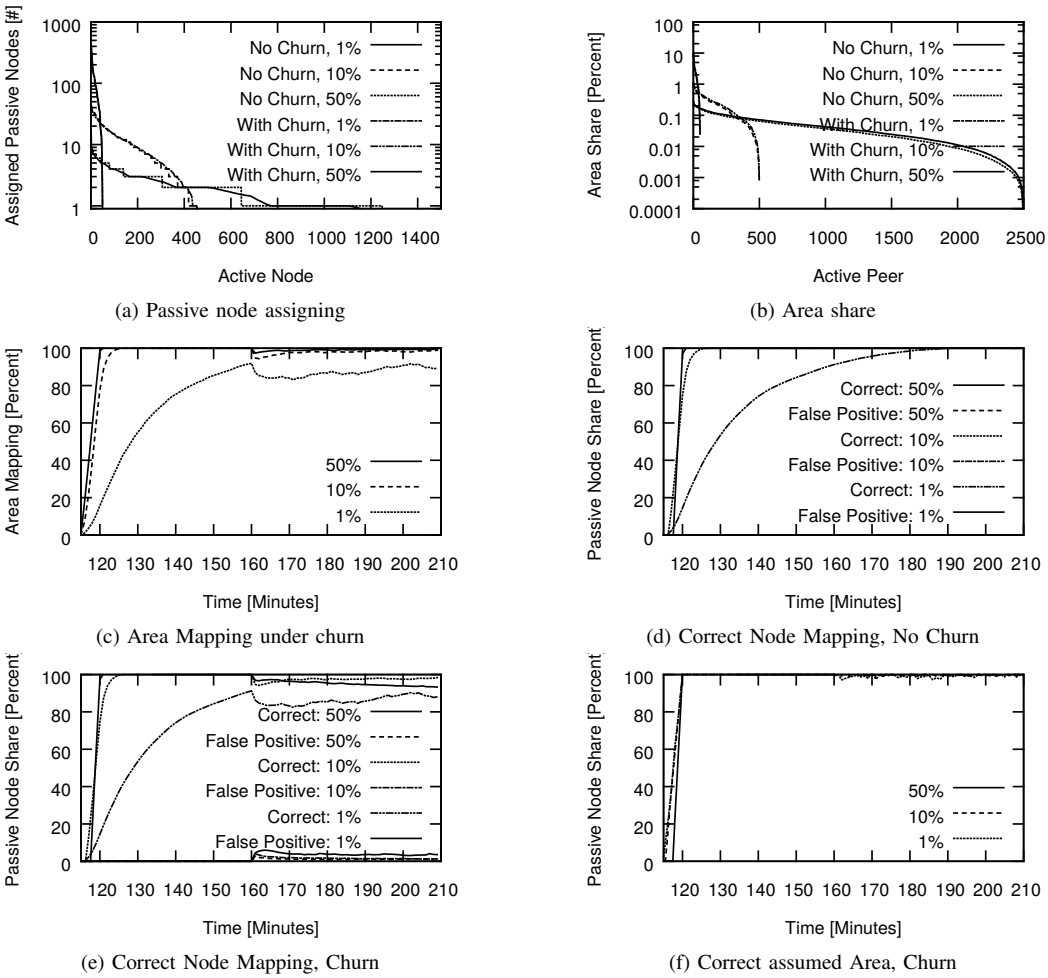
For real and deterministic data without tail drop and no policy for the target interval (cf. Fig. 3b-3c) we get a lot of overestimations due to no target interval bounds. This critically influences the maximum metric. With deterministic data and applied Reroll policy (cf. Fig. 3d), Full performs better than Default and Partly slightly worse than Default.

Next, the mean of aggregated data is the most interesting of the monitoring data in terms of precision. For real data (cf. Fig. 3e) all three monitoring scenarios perform almost equally. Our probing simulation is more shaky than the other scenarios as very high maximum values and low minimum values influence the mean marginally. For the deterministic data with no tail drop (cf. Fig. 3f- 3g), Full and Partly perform almost exactly the same, while Default being worse due to the aforementioned increased delay caused by the tree's increased depth. Just like the real data, when zoomed in, the mean error of Partly is shaking around the Full graph. With the Reroll policy without tail drop (cf. Fig. 3g), we encounter higher errors than Full but still below Default. Also, the sinus curve is deformed, thus, ancillary to the shift due to the delay when aggregating data, the data suffer from an additional error. When applying tail drop of 1% of the data (cf. Fig. 3h), this effect shrinks and gets closer to the Full graph.

Although, the monitoring suffers from the Minicamp overlay's precision, monitoring data from real data like the lookup hop count and deterministic data is able to cope with the precision of a scenario where all nodes (active and passive) participate in the monitoring mechanism.

D. Costs: Bandwidth

Fig. 4a and 4b show the share of the outgoing bandwidth grouped by their message type for 1% and 50% active node shares and churn enabled. Probe requests and responses are sent by our probing mechanism, monitoring messages are sent by the monitoring system SkyEye.KOM and aggregated



(a) Passive node assigning

(b) Area share

(c) Area Mapping under churn

(d) Correct Node Mapping, No Churn

(e) Correct Node Mapping, Churn

(f) Correct assumed Area, Churn

Fig. 1. Evaluation Outcome for Middleware Performance

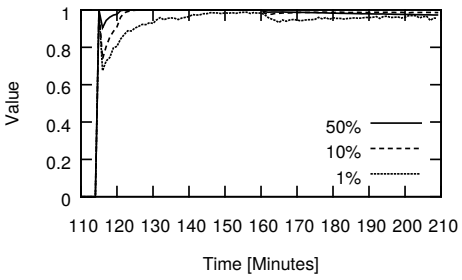


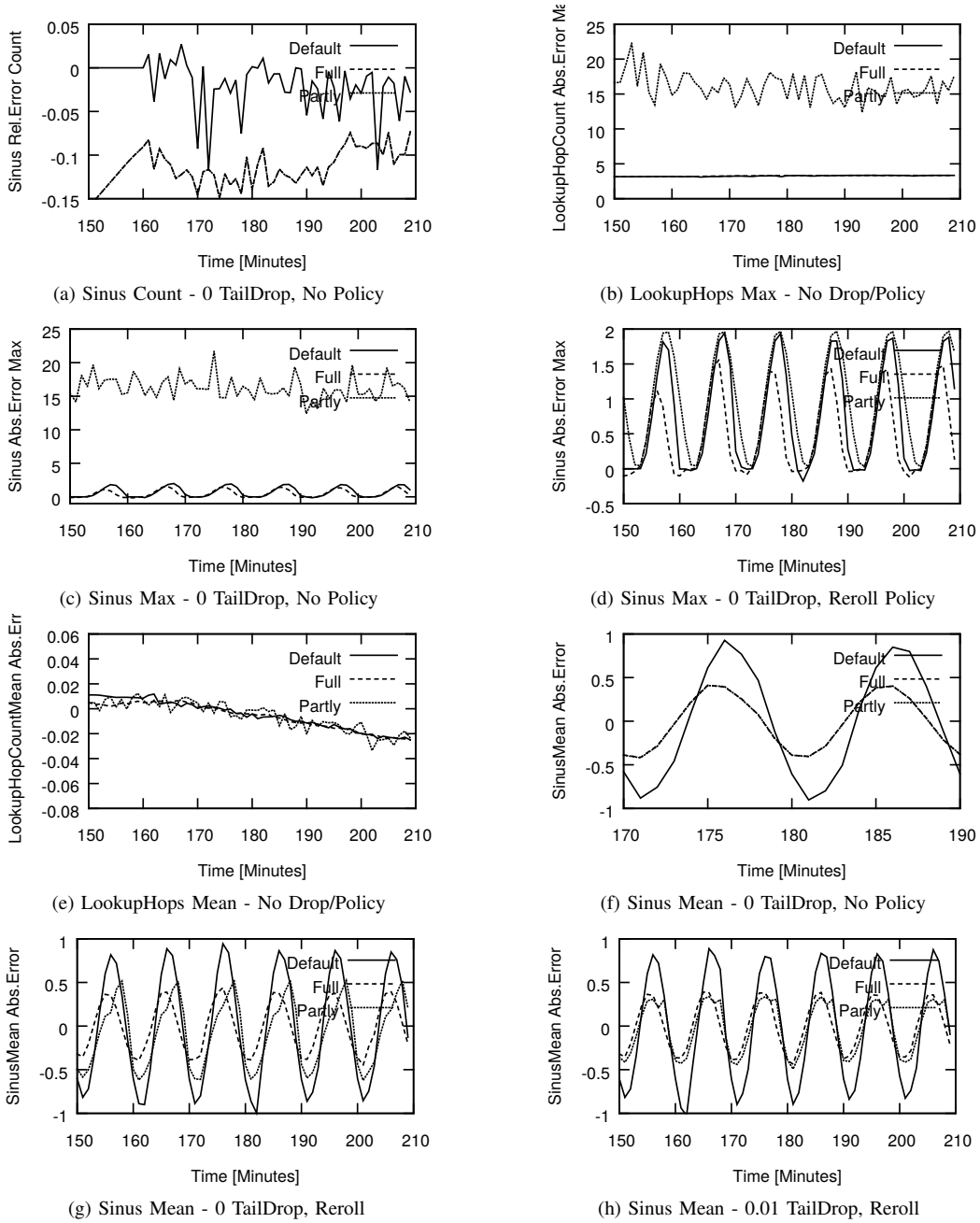
Fig. 2. Middleware Performance: Stability Parameter - Under Churn

summarizes all arising message types, including p2p overlay messages. Notable, is the unfair proportion of sent probe requests (active nodes) and their appropriate response (passive nodes). Thus, in a scenario with a small share of active nodes, active nodes with a low bandwidth may get overloaded.

The number of lookups performed per active node is presented by Fig. 5 in a scenario with churn enabled. The maximum amount of pending lookups per node is min{max reasonable searches, max parallel searches, stability factor * (estimated) number of nodes in the assumed area}. The maximum number of parallel search is set to 10 and the node estimation is based on the own share of the whole ID space and the total number of nodes in the overlay, which is provided by the monitoring. Thus, a fully utilized active node performs 10 lookups every 10 seconds, so 60 in one minute. In a complete stable network with all active nodes having at least one passive node assigned, each active node initiates exactly 6 lookups a minute (once all 10 seconds). When churn is disabled, the stable state is always reached just after the initial mapping phase has finished. In scenarios with churn enabled only the two higher active node shares (10% and 50%) reach the stable state, whereas the lowest active node share does not reach the stable state but is still utilized by 16%-40% till the end of the simulation. This is due to the maintenance actively fixing holes in the responsible area caused by churning passive nodes.

E. Costs: Delay

The age, i.e. freshness, of data provided by the monitoring approach is crucial. Here, we investigate the age of the monitoring data when the Default scenario (all nodes are active) and the Full/Partly scenario (different probing approaches for



(a) Sinus Count - 0 TailDrop, No Policy

(b) LookupHops Max - No Drop/Policy

(c) Sinus Max - 0 TailDrop, No Policy

(d) Sinus Max - 0 TailDrop, Reroll Policy

(e) LookupHops Mean - No Drop/Policy

(f) Sinus Mean - 0 TailDrop, No Policy

(g) Sinus Mean - 0 TailDrop, Reroll

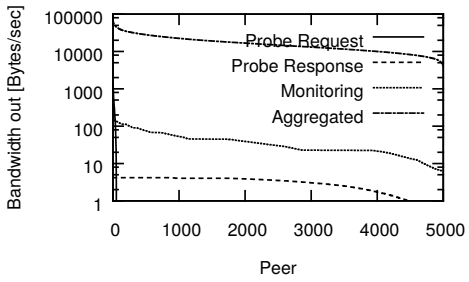
(h) Sinus Mean - 0.01 TailDrop, Reroll

Fig. 3. Evaluation Outcome for Monitoring Performance

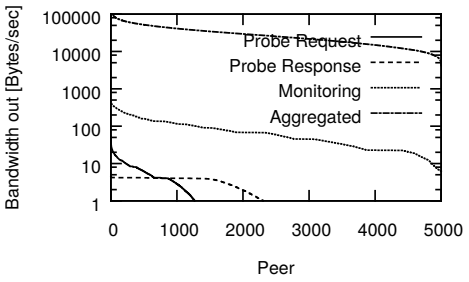
TABLE II
COSTS: MONITORING DATA DELAY (AGGREGATED AND GLOBAL VIEWS)

Scenario	Aggregated View (All / Only Active)			Global View (All / Only Active)	
	Max [sec.]	Min [sec.]	Mean [sec.]	Max [sec.]	Mean [sec.]
1%	228.1 / 112.5	0.5 / 9.9	82.8 / 36.0	272.0 / 131.0	108.8 / 64.0
10%	256.9 / 174.7	4.4 / 0.9	84.7 / 65.6	245.0 / 196.0	106.0 / 79.0
50%	244.1 / 236.6	0.4 / 3.2	84.7 / 73.9	299.0 / 214.0	108.0 / 101.0

the passive nodes) is applied. The only difference between the scenarios with and without churn is a small jitter of the data. In Table II we present the outcomes of scenarios without churn for the different active node shares. The aggregated view is the global view in the root of the monitoring tree. The global view is the global view in the leaf nodes of the monitoring tree, which is propagated from the root to all leaf nodes. The influence of the amount of active nodes, i.e. the size of the Minicamp overlay, is as follows. The mean age of the monitoring data (both up and down the tree) decreases with decreasing number of nodes as the tree's height is shrinking. This leads to a fresher view by sacrificing the natural load distribution in such p2p networks. Thus, the smaller the active node share, the fresher the monitoring data and the higher the per active node load (cf. Section III-D).



(a) Bandwidth - under Churn, 1% share



(b) Bandwidth - under Churn, 50% share

Fig. 4. Evaluation Outcome for Costs

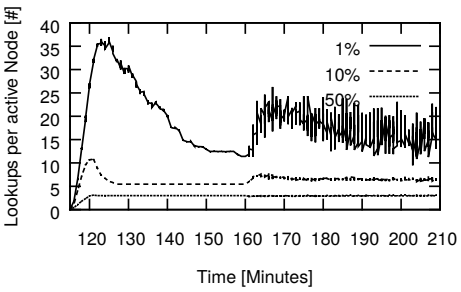


Fig. 5. Lookups for Mapping - Under Churn

IV. Conclusion and Future Work

In this paper, we extend our work on Minicamp [7] and tackle the problem of creating a global monitoring view in p2p networks with incomplete participation of the peers in the structured monitoring protocols. Our contribution, Minicamp, is a middleware overlay consisting only of active peers which operates between the common p2p overlay, such as Chord [16] or a variant [4], and a monitoring overlay, such as SkyEye.KOM [11]. We propose an assignment, maintenance and probing scheme to build up a common global view. Evaluation shows a scaling distribution and high precision ($> 90\%$) with low false positives ($< 5\%$) for our assignment scheme. The precision of probing and creating global view via monitoring is close to the optimum. Costs and the costs distribution vary depending on the active node share. For future work we aim to reduce costs for participating nodes and adapt to node's heterogeneity in scenarios with low participating share, such as in p2p-based social networks [12].

Acknowledgment

We would like to thank Franz Kary for his great work. Computational support and infrastructure was provided by the Centre for Information and Media Technology (ZIM) at the University of Düsseldorf (Germany).

References

[1] "NetMarketShare: Operating Systems' Share," https://www.netmarketshare.com/, accessed: 2017-12-23.

[2] "Plumbr Java Version Share," https://plumbr.eu/blog/java/java-version-and-vendor-data-analyzed-2016-edition, accessed: 2017-12-23.

[3] T. Amft, "Design, Implementation and Evaluation of Merging Mechanisms for large-scale and dynamically partitioned Networks," Master's thesis, Department of Computer Science, Heinrich Heine University Düsseldorf, Apr. 2014.

[4] M. Benter, M. Divband, S. Kniesburges, A. Koutsopoulos, and K. Graffi, "Ca-Re-Chord: A Churn Resistant Self-stabilizing Chord Overlay Network," in *Proc. of the Int. Conf. on Networked Systems (NetSys)*, 2013.

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

[6] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz, and 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*, ser. Lecture Notes in Computer Science, vol. 2735. Springer, 2003.

[7] A. Disterhöft and K. Graffi, "Minicamp: Prototype for Partial Participation in Structured Peer-to-Peer Monitoring Protocols," in *LCN '17: Proc. of the Int. Conf. on Local Computer Networks*, 2017, pp. 203–206.

[8] M. Feldotto and K. Graffi, "Comparative Evaluation of Peer-to-Peer Systems Using PeerfactSim.KOM," in *IEEE HPCS'13: Proc. of the Int. Conf. on High Performance Computing and Simulation*, 2013.

[9] N. Goebel, T. Krauthoff, K. Graffi, and M. Mauve, "Moving measurements: Measuring network characteristics of mobile cellular networks on the move," *Computer Communications*, vol. 97, pp. 81–95, 2017.

[10] K. Graffi, "Monitoring and management of peer-to-peer systems," Ph.D. dissertation, Technische Universität Darmstadt, 2010. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/2248/

[11] K. Graffi, A. Kovacevic, S. Xiao, and R. Steinmetz, "SkyEye.KOM: An Information Management Over-Overlay for Getting the Oracle View on Structured P2P Systems," in *Proc. of the Int. Conf. on Parallel and Distributed Systems*. IEEE, 2008.

[12] B. Guidi, T. Amft, A. D. Salve, K. Graffi, and L. Ricci, "DiDuSoNet: A P2P Architecture for Distributed Dunbar-based Social Networks," *Peer-to-Peer Networking and Applications*, vol. 9, no. 6, pp. 1177–1194, 2016.

[13] G. Hooghiemstra and P. V. Mieghem, "Delay Distributions on Fixed Internet Paths," Delft University of Technology", Technical Report 20 011 020, 2001.

[14] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Company New York, 1966.

[15] A. I. T. Rowstron and 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. Conference on Distributed Systems Platforms*, 2001.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *SIGCOMM '01: Proc. of the Int. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2001, pp. 149–160.

[17] P. Yalagandula and M. Dahlin, "Research Challenges for a Scalable Distributed Information Management System," The Univ. of Texas at Austin, Dept. of Computer Sciences, Tech. Rep. CS-TR-04-48, 2004.

[18] Z. Zhang, S. Shi, and 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*, ser. Lecture Notes in Computer Science, vol. 2735. Springer, 2003, pp. 170–182.