Dominik Stingl, Christian Groß, and Karsten Saller: Decentralized Monitoring in Peer-to-Peer Systems. In: Wolfgang Effelsberg, Ralf Steinmetz, Thorsten Strufe, vol. LNCS, no. 7847, chap. 6, p. 81-113, Springer, June 2013. ISBN 978-3-642-38672-5.

Chapter 6
Decentralized Monitoring in Peer-to-Peer Systems

Dominik Stingl, Christian Groß, Karsten Saller

From the early days, with the design of peer-to-peer overlays or with the decentralized storage and retrieval of content [2], researchers began to investigate how to control and manage these peer-to-peer systems. One important step towards the control and management of them is the assessment of the system's performance. For this purpose, *monitoring* constitutes an inevitable and necessary element, because it provides the required data basis comprising information about the system and its participating peers. Given this information, the peers themselves or an overlay operator are able to adapt and improve the system according to changing parameters and conditions. Examples for the utilization of monitored data are manifold: (i) Bubblestorm [32] or Viceroy [22] use the monitored number of peers to influence the overlay construction, (ii) DASIS [1] improves a peer's join process based on monitored data, and (iii) InfoEye [19] even monitors the access frequency of monitored data to reduce latency and cost for the provisioning of such information.

Due to the versatile applicability, a multitude of *decentralized monitoring mechanisms* for peer-to-peer systems have been developed to provide meaningful statistics about the system and its participating peers. Each of these approaches satisfies different requirements with a varying performance. They can range from heuristic snapshots at low cost to detailed views of the system at higher cost, assuming static or highly dynamic peer-to-peer systems. Out of this set of mechanisms, the decision for the selection of an appropriate monitoring mechanism is a problem. Due to the varying requirements as well as achieved performance and resulting cost, a fair comparison between different solutions is hard to achieve, if not impossible.

To overcome the lack of comparability between existing approaches, we present a benchmarking methodology for decentralized monitoring mechanisms in peer-to-peer systems, using our knowledge from previous work [28]. The methodology is designed to enable comparable evaluation studies, which can serve as a reference for future approaches. It covers (i) the identification of relevant non-functional requirements, (ii) the selection of a set of respective workloads, and (iii) the definition of appropriate metrics, including where they should be measured.

To implement this specific benchmarking methodology, we rely on the general benchmarking model and methodology, presented in Chapter 2 and 3. Section 6.1 sketches the functional requirements, from which an interface definition is derived. The relevant non-functional requirements are detailed in Section 6.2. Afterwards, the synthetic workloads are described in Section 6.3, while Section 6.4 details the required metrics. Examples of existing implementations are given in Section 6.5. Finally, we present the application of our benchmarking methodology in Section 6.6.

6.1 Interface Definition

Each implementation of a decentralized monitoring mechanism must implement a predefined interface to apply the different workloads and to capture relevant metrics. Unfortunately, there is no de facto standard for an interface to access the provided functionality of a decentralized monitoring mechanism. Even the scope of the functionality this class of systems should offer has not been defined. To design the required interface in the context of our benchmarking methodology, we first analyze the common functionality and then provide the interface.

6.1.1 Functional Description

Similar to monitoring in ordinary networks, the goal of a decentralized monitoring mechanism in peer-to-peer systems is to collect information about the system and the participating peers to reveal insights about the system's state and characteristics of its peers. In turn, the participating peers are able to use this information to adapt or optimize their behaviour. In contrast to centralized approaches [26, 4, 33], where one or a set of dedicated entities is responsible for collecting the monitored data and distributing the resulting information, decentralized monitoring mechanisms try to integrate the participating peers into these procedures [34, 16, 36, 13]. Thus, besides periodically capturing the local data, peers are also responsible for the collection and distribution procedures.

Depending on the monitoring approach, only a fraction of peers can be involved into the previously described monitoring procedures. These approaches rely on a technique which is characterized as sampling [17]. Only elected peers estimate the current state of the system based on the collected data from a subset or sample of all peers. To determine an adequate and representative sample of peers, probabilistic algorithms such as random walks are applied [24]. Besides this class of monitoring mechanisms, other approaches exist that try to include all peers into the monitoring procedures. The generated results ideally reflect the measurements of each single peer in the peer-to-peer system. The whole chapter as well as the resulting benchmarks are designed for the second class of monitoring mechanisms.

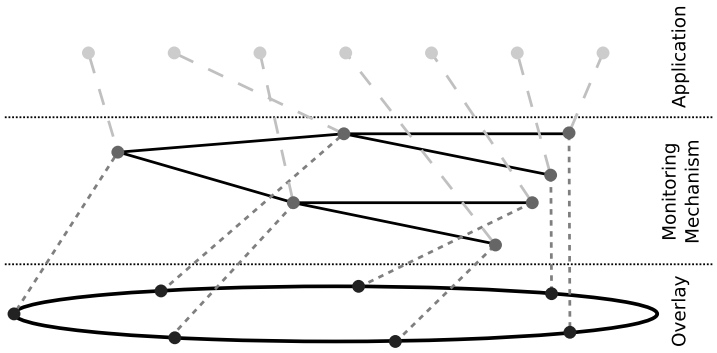


Fig. 6.1: Example of a peer-to-peer system with an integrated tree-based monitoring mechanism

For the integration of a decentralized monitoring mechanism into a peer-to-peer system, the monitoring mechanism establishes its own topology on top of the peer-to-peer overlay and below a possible application, as shown in Figure 6.1. This monitoring topology with its corresponding functionality can either be implemented as a separate layer or directly be combined with an overlay. In this new system architecture, each peer obtains a local monitor and is responsible to measure specific data. The measured data is collected over the established topology and stored at one or several peers. Afterwards, each peer can access the resulting information.

The specific data which must be monitored and collected is represented by a set of *attributes*. This set might either be static, or the approach might allow to add additional or remove unnecessary attributes [34]. Examples range from attributes of the underlay (e.g., the transmitted traffic at the network layer [23]), over overlay-specific attributes [10] to application-related attributes [34]. Since monitoring data is only exchanged in addition to application- and peer-to-peer-related data, this data should not become the dominant factor for the traffic in the network but only consume as little bandwidth as possible. To limit and compress the size of the monitored data, especially in systems with a large number of users, the utilization of aggregation functions for the data is an appropriate and frequently applied method. Typically, the set of common aggregation functions of decentralized monitoring mechanisms comprises minimum, maximum, sum, average, or standard deviation [20, 3]. So, instead of sending all the monitored values of an attribute, these values are compressed by using those aggregation functions. The aggregate of an attribute, which is calculated by the decentralized monitoring mechanism and includes the values of all participating peers, is called the *global view* of that attribute. The *global state* of the system then consists of the global views of all the attributes.

The following three sections detail (i) the underlying topology, (ii) the data collection procedure, and (iii) the resulting dissemination procedure.

6.1.1.1 Monitoring Topology

Trees [36, 6, 10] and *meshes* [16, 13, 5] constitute the prevailing topologies, which are used by a decentralized monitoring mechanism to collect the monitored data and to disseminate the generated results. As outlined by Makhloufi et al. [21], the selection of a topology for a decentralized monitoring mechanism constitutes the main decision criterion, which influences the data collection and result dissemination procedures.

For the creation of a tree, several approaches rely on the underlying peer-to-peer overlay [36, 6, 10], such as a distributed hash table, to create the topology, whereas other approaches [8] simply create a spanning tree on the overlay. The collected data is propagated over the topology from the leaves over the inner nodes towards the root. Depending on the chosen procedure for result dissemination, this topology can also be used to distribute the calculated results.

On top of a mesh topology, a peer does not have a predefined peer or a set of peers to communicate with. Instead, one or several neighbors are randomly chosen to exchange the monitored data with. The neighbors can either be provided by the peer-to-peer overlay by relying on the routing table or by an additional service, such as a peer sampling service [14]. The random selection of neighbors for the information exchange results in a *gossip-based* type of communication [16, 13, 5]. This epidemic communication paradigm is often used as a synonym for mesh-based monitoring mechanisms.

6.1.1.2 Data Collection

The data collection procedure specifies how the monitored data is collected. For this procedure, we describe the different techniques to propagate the data .

In terms of propagation, decentralized monitoring mechanisms can actively transmit the monitored and collected data, which results in a *push*-based propagation. The transmission of data as an answer to a request is denoted as a *pull*-based propagation. Tree-based approaches can choose between push- [10] and pull-based [36, 23] propagation. During the collection procedure, a child sends its monitored data to its parent. The parent processes the received data of all children as well as its own measurements. Afterwards, the parent sends the data to its parent. Relying on this procedure, the monitored data might always be propagated to and stored at the root [18, 10], or collected and stored at inner nodes [36, 37]. A parent aggregates the received information using a generic aggregation function [36], such as $V_{i,type,name} = f_{type}(V^0_{i-1,type,name}, V^1_{i-1,type,name}, \ldots, V^{k-1}_{i-1,type,name})$, where i represents the level of the parent in the tree, $type$ and $name$ denote type and name of the attribute and determine the corresponding aggregation function f_{type}, and V^m identifies the mth child of the parent.

Gossip-based approaches mainly push the data [16, 31], while a fraction of approaches [13, 5] reacts with an answer on the push message, which results in *push-pull*-based data collection. For the correct mode of operation, gossip-based

approaches, which apply aggregation to compress the size of the data, divide the time into *cycles*. In terms of push-based approaches, each peer sends its current value v and the corresponding weight w of an attribute to k peers ($k > 0$) during one cycle. Before the transmission, v and w are updated as follows: $v = v/(k+1)$ and $w = w/(k+1)$. At the end of a cycle, a peer sums all received vs and ws including its own, which serve for the next cycle as v and w. In terms of push-pull-based approaches, each peer i sends its current value v_i only to another peer j during a cycle. The receiving peer j answers with its current value v_j. Afterwards, both peers aggregate v_i and v_j to v as follows: $v = (v_i + v_j)/2$. Both, the push- and push-pull-based approach, stop if the accuracy of the current estimate for an attribute exceeds a given threshold [31] or if a predefined number of cycles has elapsed [13]. This period of time is denoted as an *epoch*. At the end of an epoch, each peer has an estimate of the global view of an attribute. In terms of push-based approaches, the estimate is calculated as v/w, whereas for the push-pull-based approach, v already represents the estimate.

To start the push- or pull-based data collection procedure, existing approaches either use a *periodic* or *event-based* execution. For the first case, a given time interval specifies the time between two consecutive executions [5, 10]. Based on the description of the data collection procedure, it becomes apparent that gossip-based approaches mainly rely on a periodic execution. For the event-based execution, the data is only collected and forwarded if a certain event occurs at a peer. Typical examples for this event comprise (i) the measurement of an attribute value that significantly deviates from the previous measurement [23, 12] or (ii) a query for the global view of one or several attributes [36].

6.1.1.3 Result Dissemination

The result dissemination procedure defines how the generated results are disseminated among the peers. Existing procedures range from a *proactive* to a *reactive* result dissemination.

With the proactive dissemination the collected data is transmitted to all or only a subset of the interested peers in the overlay. To reach the subset or all peers in the overlay, the sending peer either has a dedicated list of recipients or, when disseminating the results to all peers, it relies on the established topology [36, 10]. With the reactive dissemination, the collected data is only sent to the requesting peers.

As a consequence of the push- or push-pull-based data collection, the proactive dissemination is implicitly integrated in gossip-based monitoring. In contrast, tree-based monitoring approaches allow to choose between different result dissemination strategies: (i) the root either proactively disseminates the results down the tree [10], (ii) reacts on a request of a peer [25], or (iii) allows to define how the results are disseminated [36].

6.1.2 Deduction of an Interface

Based on the description of the functional requirements above, we narrow down the functionality of a decentralized monitoring mechanism to a set of essential methods and design the respective interface to access those methods. Although some approaches allow to add additional and remove unnecessary attributes at runtime, we assume that the number of attributes during a benchmark is fixed. In addition, we limit the utilized aggregation functions to the five classical functions mentioned in Section 6.1.1.

Each participating peer of the overlay locally measures the predefined set of attributes and stores them. The decentralized monitoring mechanism collects this data according to the underlying topology and utilized data collection procedure. Subsequently, each peer is able to retrieve the global view of the collected attributes.

The common functionality of a decentralized monitoring mechanism can be summarized with the following interface:

- `setLocalValue(String name, double value, long time)` persists a locally measured value of an attribute for later collection. The parameter `time` specifies when the current value was locally measured.
- `getGlobalViewOfAttributes()` returns the global view of all monitored attributes.

Every monitoring approach, applying our decentralized benchmark, must provide the mentioned functionality and implement the specified interface in order to be comparable.

6.2 Non-Functional Requirements

Based on the general considerations about quality aspects for peer-to-peer systems in Section 3.3, we now define the relevant non-functional requirements for decentralized monitoring mechanisms. The resulting set is divided into the two categories *workload independent* and *workload dependent* quality aspects.

Workload-independent Quality Aspects

- **Performance.** In general, the performance of a peer-to-peer system consists of the quality aspects *responsiveness*, *throughput*, and *validity*, as described in Section 3.3. Out of these three categories, validity and responsiveness are of major importance for decentralized monitoring in peer-to-peer systems. Validity characterizes the quality of the provided results and is represented through *accuracy* and *staleness*. Responsiveness characterizes how fast these results are provided to a requesting peer.

  - **Validity** is a central aspect for monitoring, because it characterizes the quality of the provided results, which, in turn, can be divided into accuracy and staleness. In the context of decentralized monitoring mechanisms, these results are represented by the generated global views of the monitored attributes. Accuracy describes how precise the results are and if the monitoring mechanism is able to correctly capture and present the system's state. Staleness addresses the age of the provided results. In contrast to responsiveness, which only considers the time to retrieve the global view, staleness also includes and considers the age of values, which are used to calculate the results. Thus, it represents the time span from capturing the first value of an attribute until the point in time when a peer obtains the global view.
  - **Responsiveness** covers the aspect how fast a requesting peer is served with the current global view. The responsiveness of a monitoring mechanism heavily depends on the applied strategy to collect data and disseminate results (cf. Section 6.1.1.2 and 6.1.1.3). A monitoring mechanism can be very responsive and immediately deliver the results, because the request is locally answered by the requesting peer. In contrast, the responsiveness of a monitoring mechanism might decrease if the request for the current global view triggers the distributed collection process for the set of attributes.

- **Cost.** We consider only the communication cost, produced by the monitoring mechanism to calculate and distribute the global view of the attributes.
- **Fairness.** To evaluate the fairness of a system, performance and cost with their related metrics serve as the basis for the calculation. In this chapter, we define fairness as the uniform distribution of either performance or cost between the peers while not taking their available resources into account. With respect to cost, a fair monitoring mechanism should evenly distribute the communication overhead among the peers to avoid overloaded peers. Dealing with the performance, a fair system should offer the same access to the provided services and avoid starving peers. For decentralized monitoring mechanisms, we investigate how the performance in terms of staleness and accuracy as well as the cost differ.

Workload-dependent Quality Aspects

- **Scalability.** In the context of decentralized monitoring mechanisms, scalability is divided into *horizontal* and *vertical* scalability, as detailed in Section 3.3.3. Horizontal scalability addresses the increase of peers in the system. On the one hand, this increases the number of peers which must be monitored, and on the other hand, the number of requests for the monitored results.
  Vertical scalability varies the applied load on the system. In order to increase the load on a decentralized monitoring mechanism, there are two workload factors that can be varied: (i) by varying the number of monitored attributes, the resulting amount of data is increased; (ii) by varying the number of requests for the global

view of aggregates, the frequency to collect and disseminate the data might be increased or decreased.

- **Robustness** characterizes how a peer-to-peer system handles unpredictable external events or severe failures. For decentralized monitoring mechanisms, we consider massive fluctuations of peers, induced by massive crashes or massive arrivals of peers.
- **Stability** characterizes the ability of a decentralized monitoring mechanism to deal with the random behavior of the autonomous participating peers in the peer-to-peer system. While the autonomy of peers covers (i) the application-specific load induced by the consumption of a service or an application as well as (ii) the uncontrollable arrival and departure of peers, we only consider the random behavior and the autonomy of participating peers in terms of different churn levels. The application-specific load, which results from different request rates, is covered by the scalability above.

6.3 Workload

In the following, the synthetic workload models are presented, which are used to benchmark a decentralized monitoring mechanism. The different workloads can be classified according to the three workload-dependent quality aspects *scalability*, *robustness*, and *stability*, as previously defined. Besides, the synthetic workload model also comprises a model to assess the validity of a monitoring mechanism, as described at the end of this section.

The workloads rely on a set of workload factors, which are summarized in Table 6.1. The table presents the default values to which the factors are set if they are not varied. The parameter *number of peers* determines how many active peers are simulated within a scenario. The *mean peer session length* specifies how long a peer is online, participating in the peer-to-peer system. To model a scenario without churn, the corresponding value must be set to *infinite*, because no peer leaves the network. The number of monitored attributes outlines how many attributes must be monitored, collected, and disseminated. Finally, the *request rate* defines how often a peer requests the global view of the monitored attributes. At the end of this section, after the description of the different workloads, Table 6.2 summarizes the workload setup including the assignment of values to the corresponding workload factors. The values with an overline mark the default values for that workload factor.

To obtain valid and comparable results, the correct procedure to apply a workload on a decentralized monitoring mechanism must be defined so that each experiment is conducted in the same way. The required information comprises the answers to the two questions (i) *when* to start the workload and (ii) *how* long it should be applied. Based on the definition of the different phases of a workload (cf. Section 3.4.1), the underlying experiment lifecycle for the application of a workload on a decentralized monitoring mechanism is divided into three phases. As depicted in Figure 6.2, the *bootstrap* phase lasts $60min$ so that all peers can join. Afterwards, the whole system

Workload factor	Unit	Default value
Number of peers	peers	10,000
Mean peer session length	min	Infinite
Number of monitored attributes	attributes	10
Request rate	requests/min	0.1

Table 6.1: Identified workload parameters and their default values

levels out and becomes stationary during a *silent* phase of 20*min*. Finally, the *testing* phase is set to 180*min*; it is used to apply the workload as well as to measure the metrics. During each testing phase, only one workload factor will be varied. In terms of the workloads that rely on offline variation[1], we periodically measure the produced data of the simulation with an interval of a minute during the testing phase. In terms of the online variation, we measure the produced data during the first 30 minutes of the testing phase (80*min* to 110*min*), apply the workload, and measure the produced data during the last 30 minutes (230*min* to 260*min*) to capture the differences before and after the application of the workload.

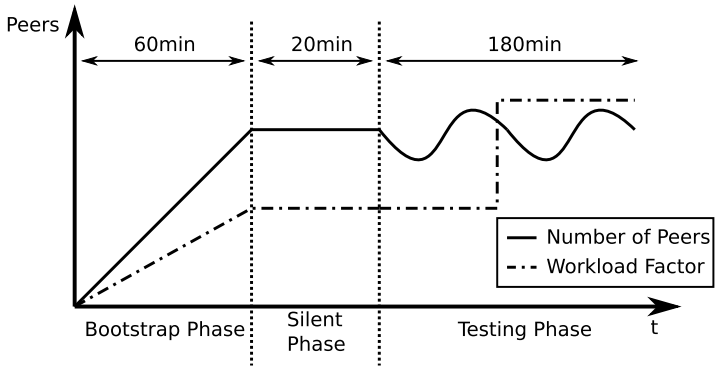


Fig. 6.2: Experiment lifecycle for the application of a workload on a decentralized monitoring mechanism

Baseline Workload

The baseline workload models idealized conditions, comprising a network with a reliable transmission of data and without churn. The workload provides insights into

[1] Offline variation and online variation will be explained during the following description of the different workloads.

the behavior of the monitoring mechanism under such conditions. Every workload factor is set to its default value, as listed in Table 6.1, and remains constant.

The baseline workload states a reference for the remaining workloads: (i) in the first place, it can be used to compare how performance and cost of a monitoring mechanism deviate from this reference under different workloads; (ii) it serves as reference to assess how the workload-independent quality aspects of a particular monitoring mechanism are met for other workloads.

Scalability Workloads

Starting from the baseline workload as described above, we use horizontal as well as vertical scaling to benchmark the scalability capabilities of the monitoring mechanism.

The *horizontal scalability* workload consists of several separate runs. During each run the number of participating peers in the system is multiplied by an order of magnitude for each run. In the following, we denote this variation of the workload factor during several runs as *offline variation*. The workload investigates how a monitoring mechanism handles a growing number of peers and thus an expanding peer-to-peer system. It focuses on the communication cost and evaluates if they change due to the higher amount of peers in the system. Besides, it investigates the impact of an expanding system on the responsiveness of the monitoring mechanism as well as on the accuracy and staleness of the provided results.

In contrast to that, vertical scaling stresses the system in terms of an increasing load. The *vertical scalability I* workload examines how the monitoring mechanism scales under an increasing number of monitored attributes, while the number of peers remains constant. Similar as for the horizontal scalability workload, this workload scheme consists of several separate runs. During each run the amount of monitored attributes is multiplied by an order of magnitude. Due to the increased load in the peer-to-peer system, the workload helps to identify if a higher load is equally distributed or if the monitoring mechanism, although decentralized, might reveal bottlenecks. Furthermore, it evaluates the effect of an increased load on the staleness and validity of the retrieved results due to a higher amount of monitored attributes.

The *vertical scalability II* workload increases the number of requests for the global view of attributes. For the variation of requests, a *Poisson process* is used to model the time between two consecutive requests of a peer. The intensity of the Poisson process is configured by the workload factor request rate and influences the number of requests per peer. Similar to the previous description of the vertical scalability workload, this workload consists of several runs during which the workload factor request rate is multiplied by an order of magnitude per run. Based on this workload, it can be assessed if a monitoring mechanism balances the requests equally or if only a fraction of peers or even a single peer might be in charge to answer the requests. Especially in systems with a pull-based data collection or a reactive result dissemination procedure, the request rate can heavily influence the

performance and cost, because each request might trigger the collection and dissemination procedure again.

Stability Workloads

To investigate the stability of a decentralized monitoring mechanism, a *churn* workload is applied, which relies on an exponential churn model. The exponential churn model is configured by the workload factor mean peer session length and defines the mean of the underlying exponential distribution. The churn workload evaluates the stability of a monitoring mechanism based on different churn levels. The workload consists of several runs, and during each run the corresponding workload factor is halved.

We measure how the performance is affected by the reorganization of the monitoring mechanism and the whole peer-to-peer system with different frequencies of arriving and leaving peers. Depending on the considered monitoring mechanism, the churn workload can have an immense influence, especially on validity. Due to a constant reorganization of the monitoring topology, the monitored data might be incorrectly collected and/or disseminated. Furthermore, a short session time of peers might end up in wrong monitoring results because measured attributes of transient peers quickly become stale or might not be included in the global view at all.

Robustness Workloads

For robustness, we investigate the system behavior under two different workloads defined by the *massive join* and *massive leave* workloads. During the massive join workload, we assume that the number of peers doubles in the system, whereas for the massive leave, we assume that 50% of peers ungracefully crash. In the following, we denote this variation of the workload factor during one run as *online variation*. Both workloads stress the monitoring mechanism and the peer-to-peer system, because they must deal with a sudden change in the system status as well as in the number of peers. For the massive leave workload, we differentiate between a collapse of the monitoring mechanism due to the breakdown of the peer-to-peer system or due to the inability of the monitoring mechanism to reorganize itself.

To quantify the robustness of a monitoring mechanism, we examine validity, performance, and cost with their dedicated metrics. We consider a system to be robust if these metrics reach predefined levels after a crash or a massive join.

Validity

To determine the validity of a monitoring mechanism, the participating peers perform their tasks and monitor a set of attributes, while the previously presented workloads are applied. Using the captured attributes, the monitoring mechanism calcu-

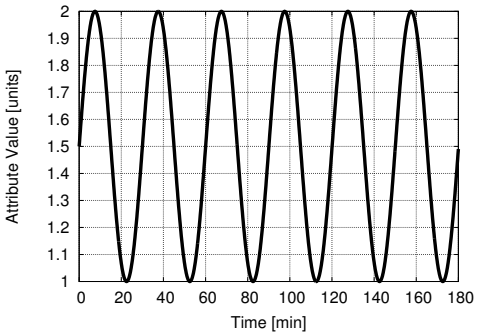


Fig. 6.3: Sine reference signal with a period of 30 min

lates the global view for each attribute, which is subsequently (proactively or reactively) disseminated to the participating peers and compared to the *correct global view*. The validity of a monitoring mechanism under different workloads is thus made obvious.

The calculation of the correct global view, which represents the current status of a system at a certain point in time, heavily depends on the applied evaluation environment. During a simulation it is possible to generate an exact snapshot of the simulated system, which represents the correct global view and serves for a comparison. In contrast, taking a snapshot of a system in a testbed requires additional steps. Each peer either sends its locally measured values to a central entity, which generates the snapshot of the system, or the peers locally store the data, while snapshots are generated after an experiment. The accuracy of the snapshot, which represents the correct global view of the system and serves for a comparison, heavily depends on the current synchronization in the testbed.

To assess the validity of a decentralized monitoring mechanism based on monitored attributes, the *peer count* states a commonly used and acknowledged attribute. It is duplicate-sensitive, and the monitored number of peers directly indicates if all the considered peers are included. Besides the peer count, we rely on a *reference signal* as our second monitored attribute. A reference signal is generated by a *value generator*, which provides each peer with a value, depending on the implemented function of the generator. The reason for this decision results from the fact that we can specify the complexity of the reference signal by defining, e.g., nearly constant or highly varying functions. Moreover, the generated values neither depend on the surrounding peer-to-peer system nor on the current workload scenario, thus, they are not biased. For our benchmark, Figure 6.3 depicts the implemented sine function, as proposed by Graffi [9]. The reasons for the selection of a sine function are twofold: Through steep slopes, it can be observed how fast the considered monitoring mechanisms capture the increasing or decreasing values. In addition, the periodicity of the signal enables to detect if the calculated global view of that attribute deviates

Workload	Workload factor	Variation	Type of variation
Baseline	None	–	–
Horizontal scalability	Peers	100, 1,000, 10,000	Offline variation
Vertical scalability I	Attributes	10, 100	Offline variation
Vertical scalability II	Request rate [requests/min]	0.1, 1, 10	Offline variation
Stability	Mean peer session length [min]	60, 30, 15	Offline variation
Massive join	Peers	10,000 simultaneously joining peers	Online variation
Massive leave	Peers	5,000 simultaneously leaving peers	Online variation

Table 6.2: Workload setup

over a longer period of time or is influenced by occurring events (e.g., massive joins or crashes).

6.4 Metrics

For the benchmark, the following metrics are introduced to quantify how well the non-functional requirements of a decentralized monitoring mechanisms are met. To avoid that the identified metrics are captured at different places in the system and lead to incomparable results, they must be measured at each peer. Based on these per-peer metrics, the global metrics for the whole system can be derived (cf. Section 3.5). Table 6.3 lists the utilized symbols.

Symbol	Description
T	The set of time samples
$P(t)$	The set of online peers at time $t \in T$
$A(t)$	The set of attributes being monitored at time t
$X_m(a,t,p)$	The *measured* global aggregate X of an attribute $a \in A(t)$ at time $t \in T$ available at a peer $p \in P(t)$
$X_c(a,t,p)$	The *correct* global aggregate X of an attribute $a \in A(t)$ at time $t \in T$ at a peer $p \in P(t)$, which is obtained via global knowledge
$\tau_{\min}(X(a,t,p))$	The time of the oldest value of an attribute being included into an aggregate
$\tau_{\max}(X(a,t,p))$	The time of the most recent value of an attribute being included into an aggregate
$\Delta t_{\text{agg}}(X(a,t,p))$	The time span that contains all included values for a global aggregate and is calculated as $\Delta t_{\text{agg}}(X(a,t,p)) = \tau_{\max} - \tau_{\min}$
$\Delta t_{\text{prop}}(X(a,t,p))$	The propagation time for a global aggregate from a data sink to a peer, which is responsible to disseminate the data to other peers, e.g., as in SDIMS [36]. This time is $= 0$ for mechanisms that disseminate results in a proactive manner.

Table 6.3: List of mathematical symbols as defined in [28].

Per-peer metrics

The following metrics are used to quantify the non-functional requirements for performance and cost.

Responsiveness Metric

- To quantify responsiveness, the lookup time for a request of the global view of the monitored attributes is used. In this context, $t_{\text{req}}(X_m(a,t,p))$ represents the time in seconds from the transmission of the request to its answer. For mechanisms that disseminate results in a reactive manner, this time will be ≥ 0. For systems with proactive result dissemination, the request may result in a lookup in a peer's local storage, thus leading to $t_{\text{req}} = 0$.

Cost Metric

- Considering the metric for this quality aspect, we rely on the total traffic, as described in Section 3.5:

$$c_d(p,t) = c_{d_{up}}(p,t) + c_{d_{down}}(p,t)$$

This traffic summarizes the traffic of the whole peer-to-peer system including the peer-to-peer overlay and the monitoring mechanism. Measuring the overall traffic reveals the *indirect* traffic of a monitoring mechanism, which arises, e.g., if the monitoring mechanism initiates a peer lookup that is resolved by the peer-to-peer overlay.

Validity Metric

- With $\varepsilon_X(a,t,p)$, the monitoring error for an aggregate X of an attribute $a \in A(t)$ at peer $p \in P(t)$ at time $t \in T$ is specified. As mentioned for the validity in the previous section, the error is calculated based on the *measured* global aggregate $X_m(a,t,p)$ and the *correct* global aggregate $X_c(a,t,p)$. In the area of decentralized monitoring mechanisms, there exist several approaches to calculate the error. In their scenario, Kostoulas et al. [17] rely on the two metrics *root mean square error* (RMSE) and *standard deviation of error* to quantify the accuracy of their monitoring approach. While the RMSE assesses the distance in terms of error between the measured and correct values, the standard deviation of error outlines how this distance varies. Besides these two metrics for the total error, there are several common approaches to calculate a relative error metric. While Considine et al. [7] propose to use $\frac{|X_m(a,t,p)-X_c(a,t,p)|}{X_c(a,t,p)}$, we rely on

$$\varepsilon_X(a,t,p) = \frac{X_m(a,t,p) - X_c(a,t,p)}{X_c(a,t,p)}.$$

A relative error metric facilitates the comparison of different results, because the total number of peers within simulated scenarios needs not be equal. Moreover, our proposed calculation of the error enables to investigate the resulting error in

more detail. As the calculation is not based on the absolute value of the difference between the measured and correct global view, the obtained relative error indicates if the considered monitoring mechanism under- or overestimated the correct global view.

- $t_{\text{stale}}(X_m(a,t,p))$ denotes the staleness or age of an aggregate in seconds, observed at peer $p \in P(t)$. The staleness comprises (i) the time Δt_{agg} to aggregate the data, (ii) the time $\Delta t_{\text{prop}}(X_m(a,t,p))$ to disseminate the data to another peer, as well as (iii) the lookup time $t_{\text{req}}(X_m(a,t,p))$, resulting in the following calculation:

$$t_{\text{stale}}(X_m(a,t,p)) = \Delta t_{\text{agg}} + \Delta t_{\text{prop}} + t_{\text{req}}$$

Global metrics

For the definition of global metrics, which are calculated from the per-peer metrics, we rely on the definition for the aggregation of metrics, detailed in Section 3.5:

- The average of a metric $\overline{x}(t)$ over the set of peers at time $t \in T$
- The average of a metric $\widetilde{x}(p)$ over the set of time samples per peer $p \in P$
- The total average of a metric $\widehat{x}$.

In terms of the fairness, which can be calculated for the performance and the cost for the set of peers, we rely on Jain's fairness index.

6.5 Example Implementations

In the area of decentralized monitoring mechanisms, different approaches have been developed that are suitable for a wide application range with varying requirements. The developed approaches range from dedicated solutions for peer-to-peer systems, grids or wireless sensor networks to solutions for large-scale distributed systems in general. Besides the dedicated class of approaches for peer-to-peer systems, such as DASIS [1], Willow [35], or SkyNet.KOM [10], most of the remaining approaches rely on the basic peer-to-peer concepts. Astrolabe [34] communicates over a predefined tree topology using an epidemic communication protocol, which is similar to Gnutella[2]. Other monitoring mechanisms, such as SDIMS [36] and PRISM [11], rely on the routing functionality of Pastry [27] to build their trees which are used to exchange the monitored data.

[2] The Annotated Gnutella Protocol Specification v0.4 `http://rfcgnutellasourceforge.net/developer/stable/index.html`.

6.6 Benchmarking Results

This section presents the application of the distributed benchmark described above and discusses the obtained results. Before going into detail, we describe the chosen monitoring mechanisms on which the benchmark is executed.

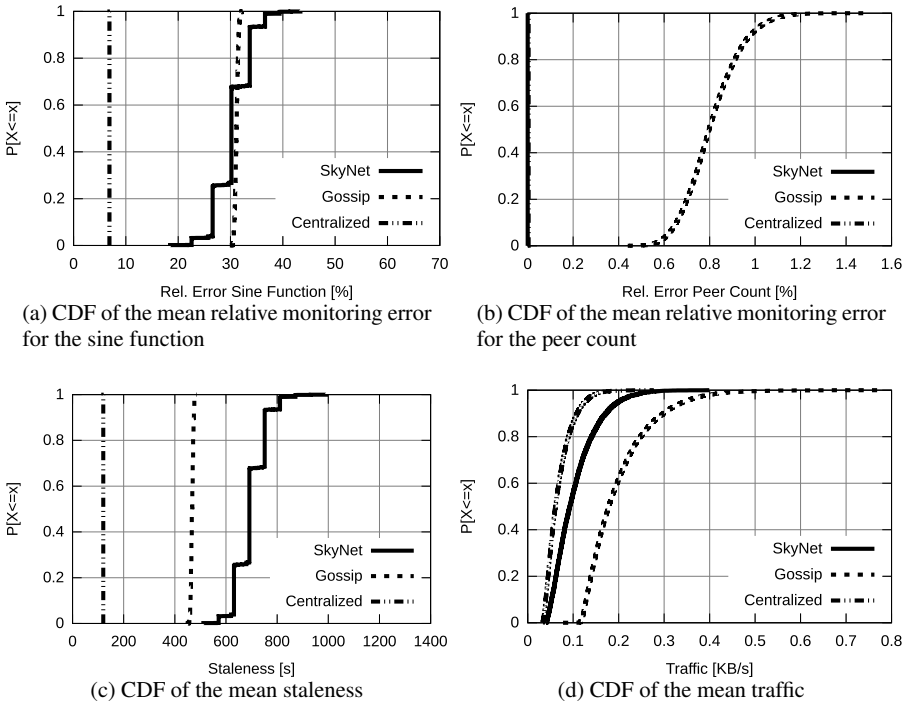
6.6.1 Simulation Setup

To apply the benchmark on the selected monitoring mechanisms, we rely on simulations and use the peer-to-peer simulation framework PeerfactSim.KOM [29]. Each of the selected monitoring mechanisms is set up on top of a Chord overlay [30], because at least one of the chosen monitoring mechanisms requires a DHT, as detailed below. Since the design and behavior of a decentralized monitoring mechanism mainly depends on the selected topology, as outlined in Section 6.1.1, three mechanisms with different topologies are selected.

SkyNet [10] is a tree-based monitoring mechanism, which relies on a DHT to build its tree topology. Over the tree, each peer periodically pushes the locally measured attributes to the root, which in turn proactively disseminates the calculated results down the tree. Thus, SkyNet uses a push-based data collection, while the results are proactively disseminated. For the periodic data collection and result dissemination, we use the proposed values by Graffi et al. and set both update intervals to 60s. The branching factor of the tree is set to 4.

The approach from Jelasity et al. [13], which we denote as Gossip in the following, is a mesh-based monitoring mechanism, which uses gossiping to communicate. It does not depend on any specific overlay as long as each peer can randomly choose one of its neighbors to exchange information with. During a cycle, each peer pushes its data to the selected neighbor, which processes the data and answers with the available information in turn. Given this communication pattern, the considered approach uses push-pull-based data collection with an implicit proactive result dissemination, because each peer is provided with the global view of monitored attributes at the end of an epoch. To configure the required parameters of the approach, we use the proposed values by Jelasity et al. and set the cycle length to 10s, while the number of cycles per epoch is set to 30.

Besides the two decentralized approaches, we have implemented a centralized monitoring mechanism as a reference. The approach relies on a separate server, which is in charge of collecting the measured data and distributing the aggregated results. The centralized approach is set up on top of the overlay. Each participating peer periodically pushes its locally measured data to the server. In turn, the server proactively disseminates the computed global view to all peers in the system. Similar to the tree-based approach, the centralized solution implements a push-based data collection, while the results are proactively disseminated. To configure the approach, the update intervals for both the periodic data collection and result dissemination are set to to 60s. As our previous evaluation has shown [28], the obtained



(a) CDF of the mean relative monitoring error for the sine function

(b) CDF of the mean relative monitoring error for the peer count

(c) CDF of the mean staleness

(d) CDF of the mean traffic

Fig. 6.4: Per peer results for performance and cost, measured for the baseline workload

results of the centralized approach represent an optimal solution, which serves as a reference. Therefore, we mainly detail the results of the two decentralized approaches and refer to the centralized solution where appropriate.

During the following evaluation, the vertical scalability II workload is not used, because a variation of the request rate does not influence the considered monitoring mechanisms, which implement push-based data collection and proactive result dissemination. Thus, a monitoring request for the global view is directly resolved by a lookup in a peer's local storage.

6.6.2 Baseline Workload

Figure 6.4 shows the results for the baseline workload in terms of performance and cost. Relying on the definition of the global metrics, as defined in Section 3.5, the respective cumulative distribution function (CDF) displays the distribution of the global metric $\widetilde{x}(p) = \frac{1}{|S|}\sum_{s\in S} x(p,s)$, which represents the average of a metric x at a

specific peer p over the set of sample timestamps s. Starting with the performance in terms of validity, Figure 6.4a and 6.4b outline that SkyNet outperforms Gossip, given an optimal network without churn or message loss. In terms of monitoring the constant number of peers in the system, SkyNet even catches up with the centralized solution, whereas Gossip exhibits a slight mean relative error of 0.8% compared to the correct global view, which is obtained by a snapshot of the simulator. With respect to monitoring a dynamic attribute with varying values, Figure 6.4a outlines the effect of a tree topology as well as flat topology on the accuracy of results. While the relative error does not considerably differ for Gossip, the different levels of the tree topology result in an increased relative error per level.

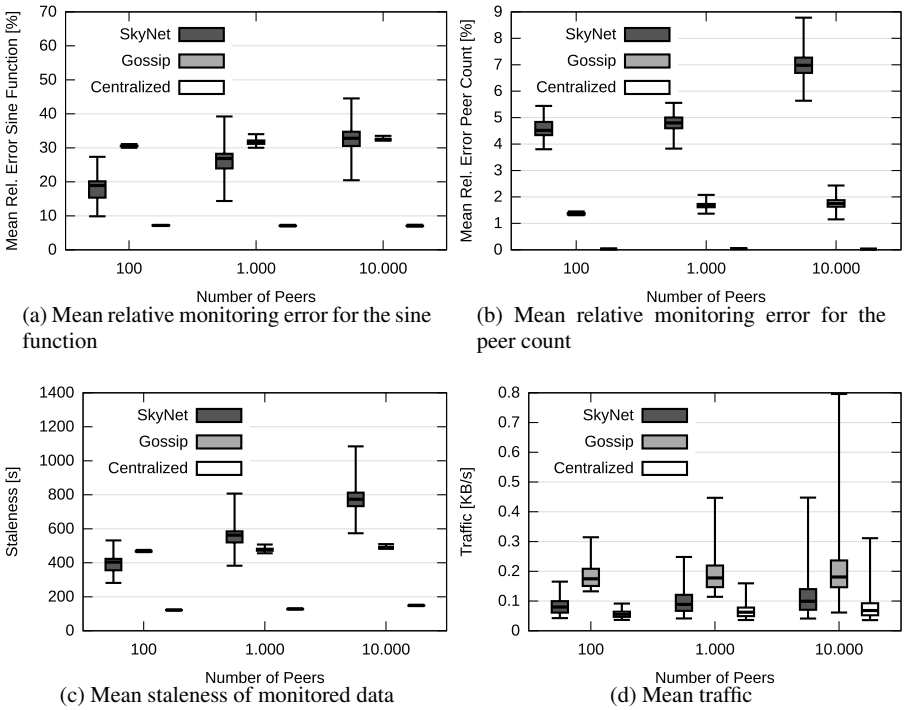
The impact of the two different topologies on the performance of a decentralized monitoring mechanism becomes apparent as well when looking at Figure 6.4c, which displays the staleness of results. For Gossip, the peers are nearly simultaneously provided with the results, whereas the staleness increases per level in the tree.

In terms of cost, Gossip generates the highest amount of traffic on average. In contrast to SkyNet or the centralized approach, the increased traffic results from a shorter update interval to distribute the data. Although SkyNet and the centralized approach have the same update intervals to transmit data, the resulting traffic of SkyNet is higher than that of the centralized approach. The reason for the increased traffic results from the fact that a peer of SkyNet must communicate with its parent and four children on average, whereas the communication of a peer in the centralized approach is limited to pushing the data to one peer (the server) and requesting the results.

6.6.3 Horizontal Scalability Workload

Figure 6.5 displays the results for accuracy, staleness, and cost as box plots. Similar to the baseline workload, the box plots outline the distribution of the global metric $\widetilde{x}(p)$. The whiskers are set to the 2.5 and 97.5 percentile, covering 95% of all values, whereas the box represents the values between the first and third quartile. The line inside the box is the median.

In contrast to the baseline workload with idealized network conditions, the horizontal scalability workload is applied on a peer-to-peer system with an Internet-like message loss [15]. Figure 6.5b shows the direct impact of message loss on the relative peer count error, because the error increases for both decentralized monitoring mechanisms. The higher impact of the message loss on SkyNet results from the fact that a loss of a message next to the root leads to a loss of data collected over many peers, or that the same peers are not provided with the global view of the system. Since Gossip operates on a flat topology, the impact of a lost message between any pairs of peers is the same. Besides the introduced message loss, both Figure 6.5a and 6.5b show the influence of the number of peers on the relative monitoring error. While the relative error in terms of the peer count does not differ for the centralized



(a) Mean relative monitoring error for the sine function

(b) Mean relative monitoring error for the peer count

(c) Mean staleness of monitored data

(d) Mean traffic

Fig. 6.5: Per peer results for performance and cost, measured for the horizontal scalability workload

approach (0.03% for 100 and 10,000 peers, respectively) or slightly increases for Gossip (1.37% for 100 and 1.75% for 10,000 peers), a larger peer-to-peer system leads to an increasing relative error in SkyNet. The higher number of peers leads to an increased number of levels of the tree, which becomes apparent by the increasing relative error for both attributes. In contrast, Gossip provides the results always after a certain amount of time, which depends on the configured length of the cycle and the epoch, thus leading to the constant behavior in the presence of a growing number of peers. On the one hand, this static behavior is beneficial, because the size of a peer-to-peer system does not influence the accuracy of the system. On the other hand, the relative error is unnecessarily high in smaller systems (cf. Figure 6.5a), and for larger systems the current configuration of the cycle and epoch length might not suffice to collect the data from all peers. Figure 6.5c confirms the observed trends for SkyNet and Gossip. While the staleness of results in SkyNet increases for a larger peer-to-peer system, it remains constant for Gossip, which leads to the outcome that the size of the system does not influence staleness, but also that the dissemination of results could be accelerated in smaller systems.

Taking a look at the induced traffic, Figure 6.5d displays the interesting fact that for each monitoring mechanism the resulting traffic does not change for a given subset of peers, whereas the remaining peers must carry the increased load. For every mechanism, the median of each box plot shows that the traffic slightly increases for 50% of the peers, which even partially holds for 75% of peers, as outlined by the constant upper end of the box. Based on this observation, the remaining 25% of the peers must carry the logarithmically increasing load, as indicated by the growing upper whisker. The fairness index for the traffic (cf. Table 6.4) confirms this uneven distribution, because the index decreases for each monitoring mechanism between a peer-to-peer system with 100 and 10,000 peers. A reason for the uneven load balancing results from the traffic of the overlay itself. Even the centralized approach exhibits this unfair behavior, where the resulting traffic to collect and disseminate data is independent on the number of peers in the system, because each peer only exchanges information with the server.

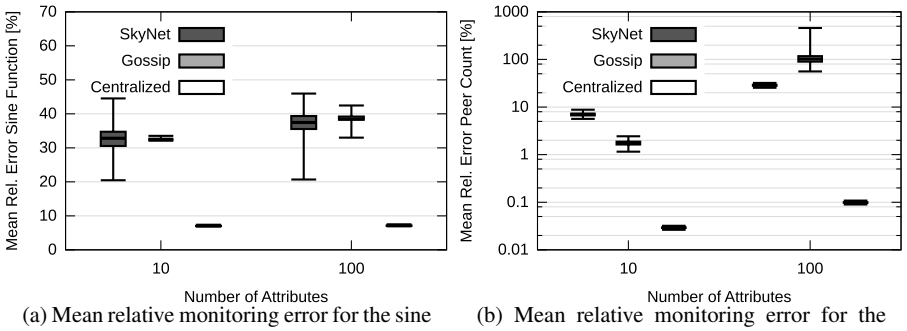
It can be concluded that the considered decentralized monitoring mechanisms put additional load on the peer-to-peer system, while the additionally load does not change the logarithmic increase of traffic as a function of a growing system, which has already been shown in Section 4.6.3.

6.6.4 Vertical Scalability Workload

Considering the results of the vertical scalability workload, which comprises the variation of load caused by a varying number of attributes, Figure 6.6c displays an increasing staleness of the results for a higher number of monitored attributes. This observation leads to the conclusion that an increasing traffic decelerates the data exchange in every system and results in stale data. Starting with the centralized approach, even during one hop, the higher number of attributes increases the staleness of results (mean staleness of 148.5s for 10 attributes; mean staleness of 156.6s for 100 attributes). For the considered decentralized monitoring mechanisms, the impact of a decelerated data exchange even multiplies: the data must be exchanged over several hops, which leads to an increased staleness of the results for both decentralized approaches, and it even disturbs the underlying synchronization of Gossip.

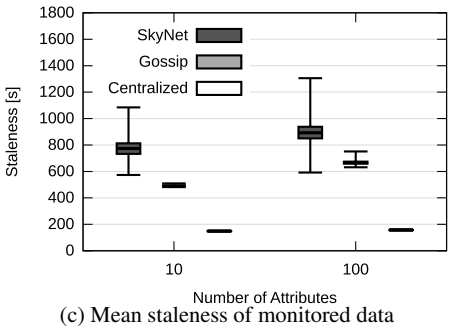
The degrading influence of the increased load is also reflected by the decreasing accuracy, as depicted in Figure 6.6a and 6.6b. Whereas the relative error of the sine function does not yet display the heavy impact of traffic on the monitoring results, the relative peer count error exhibits this influence. As shown in Figure 6.6b, the growing load results in a considerable loss of information, which especially influences the underlying calculation of the peer count for Gossip.

Figure 6.6d displays the resulting traffic for each monitoring mechanism under the varying load. In contrast to the horizontal scalability workload, where only a fraction of peers had to deal with a higher load, the number of attributes influences the resulting traffic for each peer, as shown by the box plots. The fairness index proves this statement for the traffic, because the reduction is not as strong as during

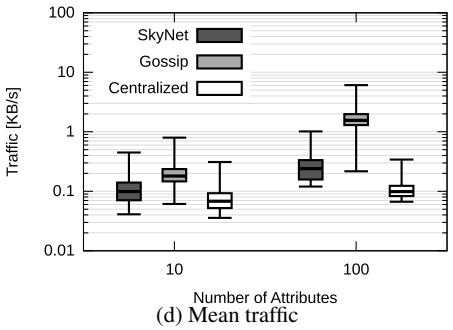


(a) Mean relative monitoring error for the sine function

(b) Mean relative monitoring error for the peer count



(c) Mean staleness of monitored data



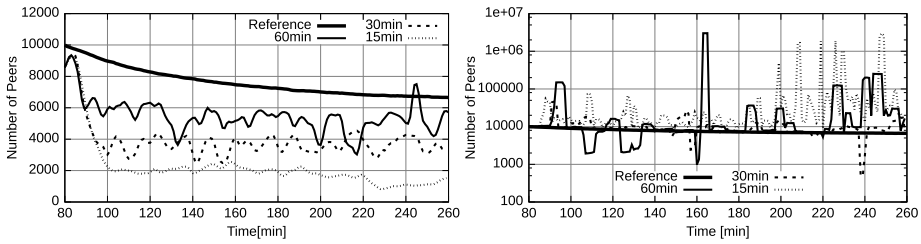
(d) Mean traffic

Fig. 6.6: Per peer results for performance and cost, measured for the vertical scalability workload

the horizontal scalability workload. Instead, the increasing load even balances the uneven distribution of traffic of the overlay, because the transmission of attributes becomes the predominant factor.

6.6.5 Stability Workload

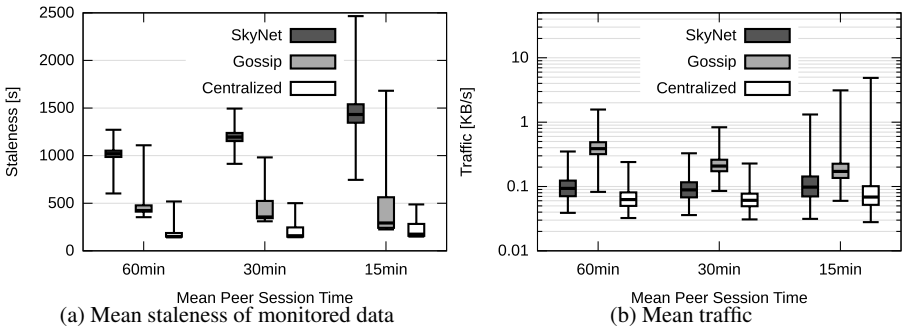
For the following workloads, we focus on the relative peer count error and omit the results for the sine function. In contrast to the previous benchmarks, the number of active peers varies over time during the remaining workloads, thus this monitored attribute enables to evaluate the accuracy based on an attribute with changing values. In terms of accuracy, Figure 6.7 does not display the mean relative peer count error per peer. Instead, we rely on the global metric $\bar{x}(t) = \frac{1}{|P(t)|} \sum_{p \in P(t)} x(p,t)$, which is the average of a metric x at a specific point in time t over the set of peers P(t). Figure 6.7 does not contain the results of the centralized approach, because the varying mean peer session length does not have an impact on the accuracy of the central-



(a) Average of the disseminated results for the monitored number of peers within SkyNet

(b) Average of the disseminated results for the monitored number of peers within Gossip

Fig. 6.7: Average of the disseminated monitoring results for the stability workload



(a) Mean staleness of monitored data

(b) Mean traffic

Fig. 6.8: Per peer results for staleness and traffic, measured for the stability workload

ized approach, which correctly monitors the current number of peers in the system. Figure 6.7a displays the average monitored number of peers per minute of SkyNet and outlines that it suffers from a decreasing mean peer session length. SkyNet is not able to construct a monitoring tree, which incorporates all peers, because the overlay suffers from the decreasing mean peer session length as well, thus it is not capable to provide the required lookup functionality. As a result, SkyNet constantly underestimates the current number of peers in the system. Figure 6.7b depicts contrary results for Gossip, which considerably overestimates the current number of peers in the system. In contrast to SkyNet, the mesh-based approach does not suffer from the missing lookup functionality of the overlay but from the short mean peer session lengths. Due to the high fluctuation, the arriving peers falsify the calculation of the peer count, which leads to the mentioned overestimation. The overestimation worsens, because the short session times even disturb the synchronization mechanism, resulting in the unusual distribution of staleness (cf. Figure 6.8a). In terms of SkyNet, the results for staleness confirm that the topology construction for the measurement tree is not successful and creates degenerated trees, which increase the age of monitored values.

Figure 6.8b depicts the traffic and outlines that a varying mean peer session length does not change the tendency between the different monitoring mechanisms in terms of traffic. Gossip still causes the highest traffic followed by SkyNet and then the centralized approach. But similar to the horizontal scalability workload, it can be observed that, dependent on a decreasing mean peer session length, the average load increases, and it is carried by a fraction of the peers. Based on the results for the centralized approach, it becomes obvious that the increasing traffic results from the overlay and its ongoing attempts to rebuild its structure, which can be observed for both decentralized monitoring mechanisms as well. The uneven distribution of this increased traffic can be confirmed for each monitoring mechanism when looking at the decreasing fairness index of the decreasing mean peer session length (cf. Table 6.4).

6.6.6 Robustness Workload

6.6.6.1 Massive Leave Workload

Similar to the stability workload, Figure 6.7 displays the averaged monitored number of peers over time for the three monitoring mechanisms. It can be observed that the centralized approach is able to handle the sudden departure of 50% of peers, because this event does not influence the direct data exchange between the remaining peers and the server. SkyNet considerably suffers from the massive leave of peers, which becomes apparent by the considerable drop of the averaged monitored number of peers. Since the underlying overlay is not capable of recovering from this massive crash, providing the required lookup functionality again, SkyNet cannot recreate its topology. As a result, it is not capable of capturing the corrected number of peers in the system, which becomes apparent by the fluctuating results that considerably underestimate the number of peers. Gossip handles the sudden crash better and provides more accurate results. Due to the fact that the approach does not rely on a specific functionality of the underlying overlay, it is able to recover from the crash to a certain degree. The available neighbors of a peer in the broken overlay are nearly sufficient to create meaningful results that reflect the current state of the system. But as shown in Figure 6.10a, which displays the staleness before and after the crash, it can be observed that in Gossip a fraction of peers exhibits synchronization problems. These problems become apparent by the varying staleness, as indicated by the outlying whiskers in the plot, which leads to stale results for a fraction of the peers. In contrast, the very low age of the results after the crash indicates that SkyNet creates several very small trees, incorporating only a handful of peers, leading to the low staleness results.

In terms of traffic, Figure 6.10b outlines that a massive peer crash does not change the tendency between the different monitoring mechanisms regarding the resulting traffic. Moreover, it can be observed, similar to the stability workload, that only a fraction of peers must deal with an increased traffic, resulting from the recovery

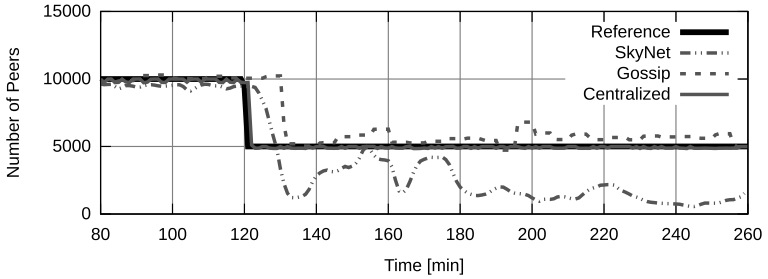
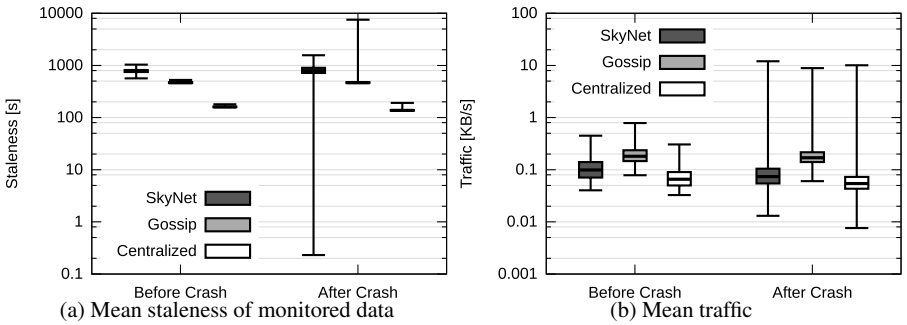


Fig. 6.9: Average of the disseminated results for the monitored number of peers for the massive leave workload



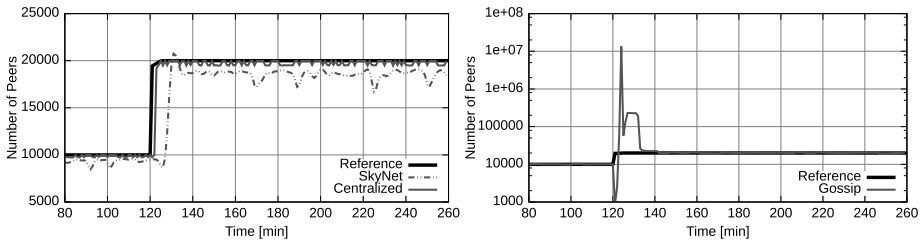
(a) Mean staleness of monitored data (b) Mean traffic

Fig. 6.10: Per peer results for staleness and traffic, measured for the massive leave workload

attempts of the overlay. The corresponding decreasing fairness index confirms the uneven distribution.

6.6.6.2 Massive Join Workload

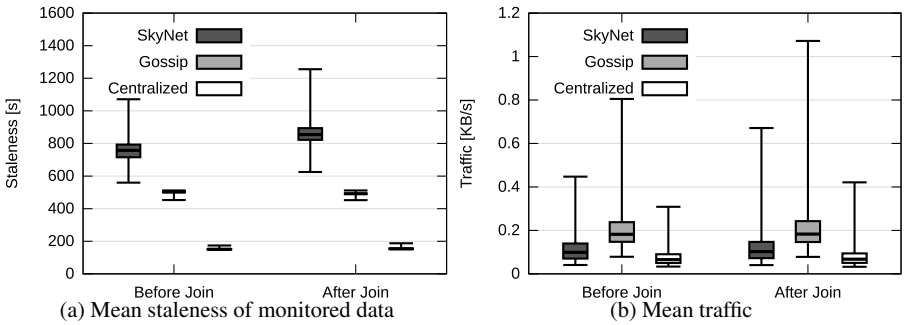
For the discussion of the results for the monitored number of peers, we split the results into two plots to adequately evaluate them. Figure 6.11a displays the mean monitored number of peers over time for SkyNet and the centralized approach, wheres Figure 6.11b depicts this metric for Gossip. The reason for the separation becomes apparent when looking at the results for Gossip, because right after the number of peers in the system doubles, the monitored number of peers is heavily overestimated. The overestimation results from the recovery of the underlying overlay as well as from the resynchronization of the newly arrived peers. After this fluctuation, the monitored number of peers levels out to the correct number of peers in the system. The results for the mean monitored number of peers that were measured during the measurement phase of 30 minutes before (mean relative peer count



(a) Average of the disseminated results for the monitored number of peers within SkyNet and the centralized approach

(b) Average of the disseminated results for the monitored number of peers within Gossip

Fig. 6.11: Average of the disseminated results for the massive join workload



(a) Mean staleness of monitored data

(b) Mean traffic

Fig. 6.12: Per peer results for performance and cost over time, measured for the massive join workload.

error of 1.7%) and after the massive join (mean relative peer count error of 1.95%) confirm the displayed results in Figure 6.11b. In terms of SkyNet and the centralized approach (cf. Figure 6.11a), it can be observed that the centralized approach immediately catches up with the current numbers of peers in the system. Due to the fact that the underlying overlay is capable to provide its lookup functionality even after the massive arrival of peers, SkyNet is able to span its tree over the new peers and to adequately capture the current state of the system. Although Figure 6.11a displays larger fluctuations after the arrival of the new peers, the results for the mean relative peer count error that were measured during the measurement phase of 30 minutes before (mean relative peer count error of 7.19%) and after (mean relative peer count error of 6.83%) the application of the workload indicate that Skynet is robust enough to handle the newly arriving peers well.

In terms of staleness and traffic, Figure 6.12a and 6.12b display the usual behavior for each monitoring mechanism in the presence of an increasing peer-to-peer system as long as the monitoring mechanism is correctly operating. For SkyNet the staleness of the results increases, because the height of the tree grows to integrate

Fairness of traffic	Baseline	100 Peers	10,000 Peers	10 Attributes	100 Attributes	Mean Peer Session Length 60min	Mean Peer Session Length 15min	Before Massive Leave	After Massive Leave	Before Massive Join	After Massive Join
SkyNet	0.819	0.906	0.81	0.81	0.788	0.859	0.643	0.81	0.125	0.81	0.796
Gossip	0.882	0.954	0.88	0.88	0.9	0.894	0.78	0.878	0.391	0.878	0.867
Centralized	0.851	0.953	0.848	0.848	0.917	0.889	0.438	0.839	0.089	0.839	0.822
Fairness of mean relative peer count error											
SkyNet	1.0	0.993	0.996	0.996	0.999	0.994	0.994	0.993	0.992	0.966	0.984
Gossip	0.976	0.999	0.989	0.989	0.945	0.002	0.005	0.939	0.678	0.937	0.941
Centralized	1.0	0.992	1.0	1.0	0.999	0.978	0.978	0.996	0.999	0.998	0.997

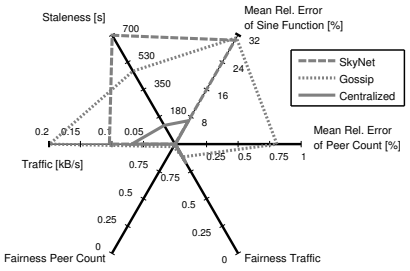
Table 6.4: Jain's fairness index for the distribution of traffic and the mean relative peer count error.

the arriving peers, while the staleness of results for Gossip and for the centralized approach remains constant, as already observed and discussed for the horizontal scalability workload (cf. Section 6.6.3). After the massive join, the resulting traffic remains nearly constant for a fraction of 50% and even 75% of the peers, whereas the remaining fraction of peers must handle the increasing traffic. This behavior of each decentralized monitoring mechanism is inline with the observed results for the traffic during the horizontal scalability workload, including also the decreasing fairness index for the traffic. Due to the fact that the number of peers in the system is not increased by an order of magnitude but only doubled, the observed impact regarding the increasing traffic and the decreasing fairness index is not that high.

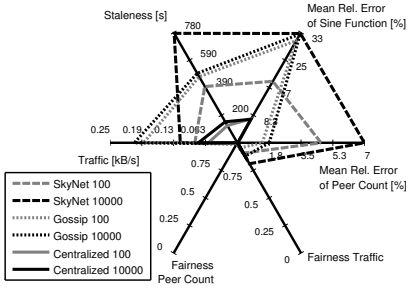
6.6.7 Evaluation Summary

After the detailed discussion of the results, the general observations and conclusions are summarized. Figure 6.13 condenses the obtained results for each applied workload, using Kiviat charts. Each plot displays the previously discussed six metrics on a separate axis. The orientation and dimension of each axis is chosen so that results next to the origin of an axis reflect a good outcome, whereas results at the end of an axis indicate a bad outcome. The result for each metric is presented as $\widehat{x} = \frac{1}{|S||P|} \sum_{s \in S} \sum_{p \in P} x(p,s)$, which represents the total average of a metric x over the set of sample timestamps S and the set of all peers P.

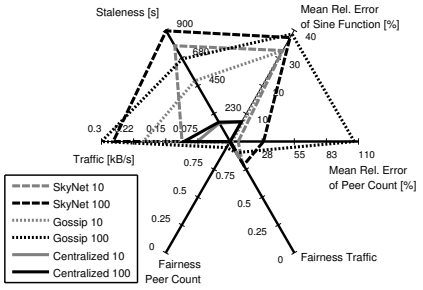
During the *baseline workload*, which assumes a perfect network, SkyNet provides perfect results in terms of the monitored number of peers. In terms of dynamic attributes, the accuracy of SkyNet degrades and reaches the mean relative



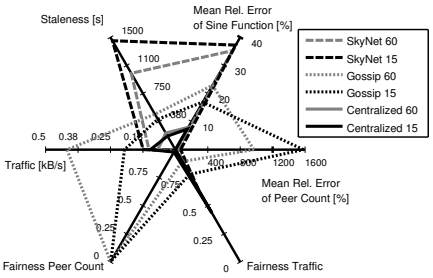
(a) Baseline workload



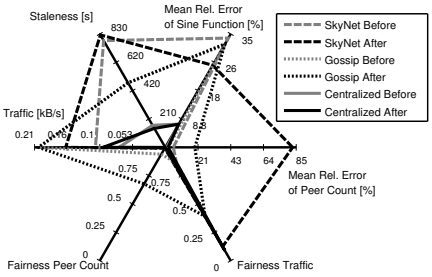
(b) Horizontal scalability workload



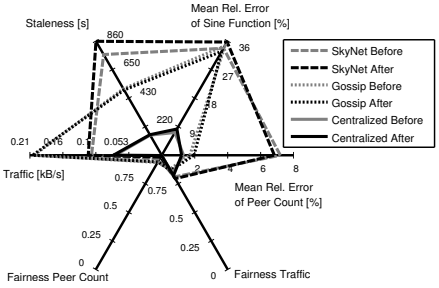
(c) Vertical scalability workload



(d) Stability workload



(e) Massive leave workload



(f) Massive join workload

Fig. 6.13: Overview of the benchmarking results for the six workloads

error of Gossip. The increased error of SkyNet results from the hierarchical topology, because each level of the tree increases the relative error. The disadvantage of a hierarchical compared to a flat topology becomes also apparent when considering the results for staleness. The flat topology leads to fresher results, whereas SkyNet suffers from the hierarchical topology in terms of staleness. Considering the cost, Gossip induces the highest traffic.

The *horizontal scalability workload* reveals the impact of a growing peer-to-peer system on SkyNet, as displayed in Figure 6.13b. Due to a growing tree, which increases the time to collect and deliver the results, SkyNet suffers from an increased staleness and monitoring error. In contrast, the flat hierarchy as well as the temporal synchronization of Gossip lead to nearly constant results in terms of staleness and the relative monitoring error. Based on this observation, it can be concluded that the size of the system has a low impact on the performance of Gossip, which is on the one hand beneficial, because accuracy and timeliness do not depend on the size of the system. On the other hand, it cannot be adapted to smaller or larger systems, which might lead to incorrect or unnecessary old or imprecise results. In terms of traffic, the results outline that the considered decentralized monitoring mechanisms cause additional traffic but that the overall peer-to-peer system still scales logarithmically with the number of participating peers. A closer look at the presented results reveals that the increasing load is unevenly distributed, because only a fraction of the peers must carry the additional load.

In contrast to the horizontal scalability workload, the *vertical scalability workload* has an impact on each monitoring mechanism, including the centralized approach as well. The performance of each monitoring mechanism degrades in terms of accuracy and leads to stale results, as depicted in Figure 6.13c. Even the underlying synchronization of Gossip is disrupted by the increased traffic, with the result that the staleness increases. The heavy influence becomes apparent as well when looking at the accuracy in terms of the relative peer count error, where even the centralized approach degrades, while Gossip supersedes SkyNet. In contrast to the horizontal scalability workload, the effect on the traffic is clearly perceptible, while the load is carried by all peers.

The *stability workload* reveals that both decentralized monitoring mechanisms are not able to deal with short mean peer session lengths in a peer-to-peer system, as outlined in Figure 6.13d. SkyNet constantly underestimates the current number of participating peers, because the tree cannot incorporate each participant of a peer-to-peer system due to the temporarily unavailable lookup functionality of the overlay. In contrast, Gossip heavily overestimates the current number of peers. The corresponding aggregation function, which calculates the peer count, suffers from the high frequency of arriving and departing peers. In terms of staleness, SkyNet and Gossip suffer from the dynamic behavior of the peers, which leads to an increased staleness of the results for both decentralized approaches. Concerning the traffic, Figure 6.13d displays for each mechanism that a decreasing mean peer session length increases the traffic in the system.

Based on the results of the *massive leave workload*, it can be concluded that SkyNet cannot cope with a crash of a peer-to-peer system at all. Though the reduced mean relative error of the sine function after the massive crash claims that the monitoring mechanism is working properly (cf. Figure 6.13e), the drastically increasing relative peer count error refutes this assumption. According to the mean relative peer count error, Gossip is able to handle a subset of simultaneously leaving peers but only to a certain degree, because the corresponding results in terms of staleness confirm that a fraction of peers deals with temporal synchronization issues.

After the massive crash, the traffic in each of the considered systems increases due to the ongoing attempts to rebuild the overlay. Similar to the stability or horizontal scalability workload, the increased load is carried by a fraction of peers, which is reflected in the decreasing fairness index.

Finally, the *massive join workload* shows that each monitoring mechanism handles a high number of simultaneously joining peers. The obtained results for the mean relative peer count error and error of the sine function (cf. Figure 6.13f) reveal that especially each decentralized monitoring mechanism provides a comparable accuracy even after the number of peers in the system has increased. The obtained results for the staleness confirm the proper functioning of each monitoring mechanism: SkyNet exhibits an increasing staleness due to the growing tree, whereas Gossip nearly remains constant, since its synchronization is properly working. The impact on the traffic and its distribution among the peers resembles the obtained results of the horizontal scalability workload, but does not clearly reflect this behavior (cf. Figure 6.13f), because the number of peers in the system is only doubled and not increased by an order of magnitude.

6.7 Conclusion

In this chapter, we have presented an extended benchmarking methodology for decentralized monitoring in peer-to-peer systems, based on our previous work [28]. The methodology is targeted at a unified evaluation of the considered mechanisms to enable and facilitate a fair and reusable comparison between existing and future approaches in this area. At the beginning, we defined the relevant non-functional requirements, which must be taken into consideration when evaluating a decentralized monitoring mechanism. Based on the requirements, we designed a set of workloads to address and evaluate the identified requirements. The design of the workload comprised the identification of corresponding workload factors and the description how these factors must be changed during a workload. Finally, we completed our methodology with the identification of appropriate metrics in order to quantify to which extend the non-functional requirements have been fulfilled. Besides the identification, we standardized where the metrics must be measured in order to avoid that different measurement points yield to deviating results.

In addition to the description of the methodology, we applied the workloads on two decentralized and one centralized monitoring mechanism to demonstrate the practical applicability. We showed how the considered monitoring mechanisms react on the different workloads, comprising the identification of disadvantages or advantages of a certain mechanism during the applied workloads. Given the results, conclusions can be drawn, which decentralized monitoring mechanism, or at least, which underlying concepts of the considered mechanism are suited for certain scenarios. The results outline as well that the decentralized monitoring mechanisms depend on the behavior of the selected overlay. While SkyNet heavily relies on the lookup functionality, Gossip only requires the provided neighborhood of the over-

lay, which leads, for instance, to a better performance for the massive leave workload. If SkyNet and Gossip are set up on top of another overlay, which provides the same functionality as Chord, but behaves differently for the applied workloads, the benchmarks will lead to different results for SkyNet and Gossip. Thus, the overlay must always be taken into account, when judging the benchmark results for the monitoring mechanisms.

References

[1] Keno Albrecht et al. “Aggregating Information in Peer-to-Peer Systems for Improved Join and Leave”. In: *Proceedings of the 4th International Conference on Peer-to-Peer Computing*. 2004, pp. 227–234.

[2] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. “A Survey of Peer-to-Peer Content Distribution Technologies”. In: *ACM Computing Surveys* 36.4 (2004), pp. 335–371.

[3] Mayank Bawa et al. *Estimating Aggregates on a Peer-to-Peer Network*. Tech. Rep. 2003-24. Stanford InfoLab, 2003.

[4] Jerome Boulon et al. “Chukwa, a Large-Scale Monitoring System”. In: *Proceedings of Cloud Computing and its Applications*. 2008, pp. 1–15.

[5] Ruud van de Bovenkamp, Fernando Kuipers, and Piet Van Mieghem. “Gossip-Based Counting in Dynamic Networks”. In: *Proceedings of the 11th International Conferences on Networking*. 2012, pp. 404–417.

[6] Justin Cappos and John H. Hartman. “San Fermín: Aggregating Large Data Sets Using a Binomial Swap Forest”. In: *Proceedings of the 5th Symposium on Networked Systems Design and Implementation*. 2008, pp. 147–160.

[7] Jeffrey Considine et al. “Approximate Aggregation Techniques for Sensor Databases”. In: *Proceedings of the 20th International Conference on Data Engineering*. 2004, pp. 449–460.

[8] Mads Dam and Rolf Stadler. “A Generic Protocol for Network State Aggregation”. In: *Radiovetenskap och Kommunikation RVK*. 2005.

[9] Kalman Graffi. “Monitoring and Management of Peer-to-Peer Systems”. PhD thesis. Technische Universtiät Darmstadt, 2010.

[10] Kalman Graffi et al. “Monitoring and Management of Structured Peer-to-Peer Systems”. In: *Proceedings of the 9th International Conference on Peer-to-Peer Computing*. 2009, pp. 311–320.

[11] Navendu Jain et al. *PRISM: PRecision Integrated Scalable Monitoring*. Tech. Rep. TR-06-22. Department of Computer Sciences, University of Texas at Austin, 2006.

[12] Navendu Jain et al. “STAR: Self-Tuning Aggregation for Scalable Monitoring”. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. 2007.

[13] Mark Jelasity, Alberto Montresor, and Ozalp Babaoglu. “Gossip-Based Aggregation in Large Dynamic Networks”. In: *ACM Transactions on Computer Systems* 23.3 (2005).

[14] Márk Jelasity et al. “The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations”. In: *Proceedings of the 5th International Conference on Middleware*. 2004.

[15] Sebastian Kaune et al. “Modelling the Internet delay space based on Geographical Locations”. In: *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 2009, pp. 301–310.

[16] David Kempe, Alin Dobra, and Johannes Gehrke. “Gossip-based computation of aggregate information”. In: *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*. 2003, pp. 482–491.

[17] Dionysios Kostoulas et al. “Active and Passive Techniques for Group Size Estimation in Large-Scale and Dynamic Distributed Systems”. In: *Journal of Systems and Software* 80.10 (2007), pp. 1639–1658.

[18] Ji Li, Karen Sollins, and Dah-Yoh Lim. “Implementing Aggregation and Broadcast over Distributed Hash Tables”. In: *Computer Communications* 35.1 (2005), pp. 81–92.

[19] Jin Liang, Xiaohui Gu, and Klara Nahrstedt. “Self-Configuring Information Management for Large-Scale Service Overlays”. In: *Proceedings of the 26th International Conference on Computer Communications*. 2007, pp. 472–480.

[20] Samuel Madden et al. “Tag: A Tiny AGgregation Service for Ad-hoc Sensor Networks”. In: *ACM SIGOPS Operating Systems Review*. Vol. 36. 2002, pp. 131–146.

[21] Rafik Makhloufi et al. “Decentralized Aggregation Protocols in Peer-to-Peer Networks : A Survey”. In: *Modelling Autonomic Communications Environments*. Vol. 5844. Springer, 2009, pp. 111–116.

[22] Dahlia Malkhi, Moni Naor, and David Ratajczak. “Viceroy: A Scalable and Dynamic Emulation of the Butterfly”. In: *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing*. 2002, pp. 183–192.

[23] Matthew L. Massie, Brent N. Chun, and David E. Culler. “The Ganglia Distributed Monitoring System: Design, Implementation, and Experience”. In: *Parallel Computing* 30.7 (2004), pp. 817–840.

[24] Laurent Massoulié et al. “Peer Counting and Sampling in Overlay Networks: Random Walk Methods”. In: *Proceedings of the 25th Annual Symposium on Principles of Distributed Computing*. 2006, pp. 123–132.

[25] James Newell and Indranil Gupta. *Storia: Time-Indexed Information Monitoring for Large-scale P2P Networks*. Tech. Rep. Departmant of Computer Science, University of Illinois Urbana-Chapaign, 2006.

[26] KyoungSoo Park and Vivek S. Pai. “CoMon: A Mostly-Scalable Monitoring System for PlanetLab”. In: *ACM SIGOPS Operating Systems Review* 40.1 (2006), pp. 65–74.

[27] Antony Rowstron and Peter Druschel. “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems”. In: *Proceedings of the International Conference on Distributed Systems Platforms*. Vol. 2218. Springer, 2001, pp. 329–350.

[28] Dominik Stingl et al. “Benchmarking Decentralized Monitoring Mechanisms in Peer-to-Peer Systems”. In: *Proceedings of the 3rd Joint WOSP/SIPEW International Conference on Performance Engineering*. 2012, pp. 193–204.

[29] Dominik Stingl et al. “PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems”. In: *Proceedings of the International Conference on High Performance Computing & Simulation*. 2011, pp. 577–584.

[30] Ion Stoica et al. “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”. In: *Proceedings of the Conference on Applications, Tech-*

nologies, Architectures, and Protocols for Computer Communications. 2001, pp. 149–160.

[31] Wesley W. Terpstra, Christof Leng, and Alejandro P. Buchmann. "Brief Announcement: Practical Summation via Gossip". In: *Proceedings of the 26th Annual Symposium on Principles of Distributed Computing*. 2007, pp. 390–391.

[32] Wesley W. Terpstra et al. "Bubblestorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search". In: *ACM SIGCOMM Computer Communication Review* 37.4 (2007), pp. 49–60.

[33] Ashish Thusoo et al. "Data Warehousing and Analytics Infrastructure at Facebook". In: *Proceedings of the 2010 SIGMOD International Conference on Management of Data*. 2010, pp. 1013–1020.

[34] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining". In: *ACM Transactions on Computer Systems* 21.2 (2003), pp. 164–206.

[35] Robbert Van Renesse and Adrian Bozdog. "Willow: DHT, Aggregation, and Publish/Subscribe in one Protocol". In: *Peer-to-Peer Systems III*. Vol. 3279. Springer, 2004, pp. 173–183.

[36] Praveen Yalagandula and Mike Dahlin. "A Scalable Distributed Information Management System". In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 379–390.

[37] Praveen Yalagandula and Mike Dahlin. "Shruti: A Self-Tuning Hierarchical Aggregation System". In: *International Conference on Self-Adaptive and Self-Organizing Systems*. 2007.