Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Implementation and Evaluation of Brahms in the GNUnet Framework

Bachelorarbeit in Informatik

durchgeführt am
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
cand. inform.

Julius Bünger

August 2015

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Implementierung und Evaluierung von Brahms im GNUnet Framework

Bachelorarbeit in Informatik

durchgeführt am
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
cand. inform.

Julius Bünger

Aufgabensteller:	Prof. Dr.-Ing. Georg Carle
Betreuer:	M.Sc. Bartolomiej Polot
	M.Sc. Sree Harsha Totakura
Tag der Abgabe:	14. August 2015

Abstract

This thesis describes the implementation and its evaluation of Brahms in GNUnet. Brahms is an algorithm that was designed to solve the task of sampling random peers in an unstructured, peer-to-peer network, as well as keeping the network connected, while only maintaining information about a small portion of the network at each peer and still being resilient to malicious behaviour and churn. After giving an overview over the topic of random peer sampling itself, we describe a concrete implementation in the GNUnet peer-to-peer framework. In the evaluation we aim to be comparable to other algorithms having the same goal.

Zusammenfassung

Diese Arbeit beschreibt die Implementierung und deren Evaluation von Brahms in GNUnet. Brahms ist ein Algorithmus, der desingt wurde, um die Aufgabe, zufällige Knoten in einem unstrukturierten, peer-to-peer Netzwerk auszuwählen, als auch das Netzwerk verbunden zu halten, während nur Informationen über einen kleinen Teil des Netzwerks von jedem Teilnehmer gespeichert werden und trotzdem unanfällig für bösartige Aktivitäten und Fehler zu sein. Nachdem wir einen Überblick über die Thematik selbst geben, bescheiben wir eine konkrete Implementierung im GNUnet peer-to-peer Framework. Mit der Evaluierung zielen wir darauf ab, vergleichbar zu anderen Algorithmen mit dem selben Ziel zu sein.

Contents

1 Introduction and Motivation 1

1.1 Goal . . . 1

1.2 Outline . . . 1

2 State of the Art and Background 3

2.1 Brahms . . . 3

2.1.1 How Brahms works . . . 3

2.1.1.1 Gossip . . . 4

2.1.1.2 Sampler . . . 4

2.2 Applications . . . 5

2.2.1 Onion routing (OR) . . . 6

2.2.1.1 TOR . . . 7

2.3 GNUnet . . . 7

2.3.1 NSE . . . 7

2.3.2 CADET . . . 7

2.4 Related Work . . . 7

3 Design and Implementation 11

3.1 Maintenance of peer IDs and Random Choice . . . 11

3.2 Min-wise independent Permutation . . . 11

3.3 Staying connected . . . 11

3.4 Periodical re-initialisation . . . 12

3.5 Limited Push . . . 12

3.6 Pull Reply Size . . . 12

3.7 Interaction with higher Layers . . . 12

3.7.1 ID stream . . . 13

3.7.2 Information leakage . . . 13

3.8 Adapting to the Network Size . . . 14

3.9 Round Interval . . . 14

3.10 Initial View . . . 14

4 Evaluation **15**
4.1 Brahms Specifics 15
4.1.1 Comparison with Brahms 16
4.1.1.1 PSP (Perfect Sample Probability) 16
4.1.1.2 Portion of faulty IDs in the View 16
4.1.1.3 Fraction of perfect Samples and faulty IDs in the Sampler . 16
4.1.1.4 Effectiveness of Attack — Isolation of Peers 17
4.1.2 Further Brahms-specific Observations 17
4.1.2.1 Distribution of Peer IDs 18
4.1.2.2 Frequency of Blocking (not updating) and Cause 18
4.2 Comparison of Samplers 19
4.3 Performance in common Measures 19
4.3.1 Local 20
4.3.2 Global 20
4.3.2.1 Degree Distribution 20
4.3.2.2 Path length 20
4.3.2.3 Clustering 21
4.3.3 Possible Biases 21
4.3.3.1 Temporal 21
4.3.3.2 Spatial 21
4.4 Discussion 21
4.4.1 Work Load 21
4.4.2 Information Leakage 22
4.4.3 Identification of Malicious Peers 22
4.4.4 Biased View 22
4.4.5 Message Loss 22

5 Conclusion **23**

Bibliography **25**

1. Introduction and Motivation

In this paper we describe the implementation of Brahms in GNUnet and evaluate that implementation.

Brahms was proposed in [BGK$^+$09] by Bortnikov et al. to solve the issue of sampling a random peer in unstructured peer-to-peer networks.

Sampling random peers is an important task for many applications in unstructured peer-to-peer networks. Common ones are information aggregation and dissemination, network management and in general algorithms that are based on gossip. As executing the algorithm includes connecting to random peers, an overlay network with useful properties (similar to those of a random graph) is created and maintained. A concrete application, we want to highlight, is onion routing, where it is of great importance to establish connections to random peers in the network.

Most approaches, including Brahms, are based on push-pull gossip with the focus on how the information about other peers is locally maintained and in what exact fashion push-pull gossip is realised. Other approaches exploit for example random walks.

Approaches in structured systems are able to exploit their underlying structure like distributed hash tables or central authorities that maintain knowledge about the whole network.

1.1 Goal

The goal of this thesis is to evaluate the Brahms implementation in GNUnet. We try to do this in a way that makes it possible to easily compare the results with other implementations of random peer sampling algorithms.

Also we want to extend the GNUnet framework with a peer sampling algorithm.

1.2 Outline

This thesis is split into three parts: Background, Implementation and Evaluation.

The first part provides an overview over previous and related research on this topic, as well as a brief summary of existing solutions and a short introduction to the GNUnet framework the implementation was implemented in. Plus, an overview over different applications is given.

This is followed by a part that deals with the concrete realisation of the protocol. Details that are not covered by the highly abstract level of the Brahms protocol are discussed and their realisation is presented.

The last part is concerned with the evaluation of the previously discussed implementation. It will present different measures, their exact results, comparison to other work and their meaning. This is accompanied by a discussion of questions that raised during the implementation of the Brahms protocol.

The whole thesis is rounded up with a conclusion that covers a summary of the important insights gained during the thesis.

2. State of the Art and Background

This chapter gives an overview over some topics that are important for this thesis. At first, the Brahms protocol itself will be summarized. We will present some applications that are based on Brahms. An overview over GNUnet is given alongside the explanation why it is used as underlying framework.

2.1 Brahms

The Brahms protocol was introduced by Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot and Alexander Shraer in [BGK+09]. This thesis is heavily based on that protocol, which is briefly presented in this section. Brahms addresses the issue of sampling random peers in an unstructured network. The authors claim that it is resilient against churn and malicious activity while only maintaining a small view on the network on each peer.

Their approach is two-layered: One layer that cares about the connectivity of peers and spreading information based on gossip and a second layer that cares about the randomness of peers while only keeping information about a small portion of the network, they call *sampler*. The former is realised through the previously mentioned push-pull gossip.

They consider two attack scenarios: The first one tries to maximise the representation of the malicious peers by flooding the correct nodes with the ids of the malicious peers combined with holding back information of correct nodes. In the second attack scenario the malicious peers try to separate the network. As separation is easiest when separating one node from the rest of the network the attackers cumulate their resources on that node. This attack is also called “targeted attack”.

2.1.1 How Brahms works

Brahms executes in rounds.

Executing Brahms involves the following data structures: A *sampler* that was presented by Bortnikov et al. alongside with Brahms itself which will be described below in section 2.1.1.2, a list of peer IDs that holds a small fraction of the network called *view*, a *push list* and a *pull list*.

The messages types exchanged by the peers are *pushes*, *pull requests* and *pull replies*. Pushes contain a proof of work to prevent massive push-flooding, pull requests do not have payload and pull replies consist of the peers that are replied. What peers are replied is covered in the next section.

The parameters used are l_1, which is the size of the view, l_2 which is the size of the sampler, α, β, δ (with $\alpha+\beta+\delta=0$) which influence the gossiping, as described in the next section.

2.1.1.1 Gossip

The authors of Brahms chose to exploit the advantages of push-pull gossip. The details of the realisation on Brahms are given below.

In each round a given peer A sends a parametrized amount of pushes and pull requests to peers in its view. It adds the peer IDs obtained through received pushes and pull replies to the push and pull list respectively. With the end of a round the push and pull lists are emptied.

Updating the View and Blocking

Before the push and pull lists are emptied at the end of each round, the view is updated as follows. αl_1 random IDs are taken out of the push list, βl_1 random IDs are taken out of the pull list and δl_1 IDs are requested from the sampler and on receipt also put into the sampler.

To prevent suffering from malicious actions, the view is only updated if pushes and pull replies were received and the number of pushes was smaller than $\alpha * l_1$. Not updating is also called *blocking.*

2.1.1.2 Sampler

The sampler is the data structure introduced as the part of Brahms that cares about the random selection of peers that have been observed through gossip.

It consists of a number of elements that hold one peer ID at a time like illustrated in 2.1. On the execution of each round all peer IDs in the push and pull lists are put into the sampler (into every sampler element).

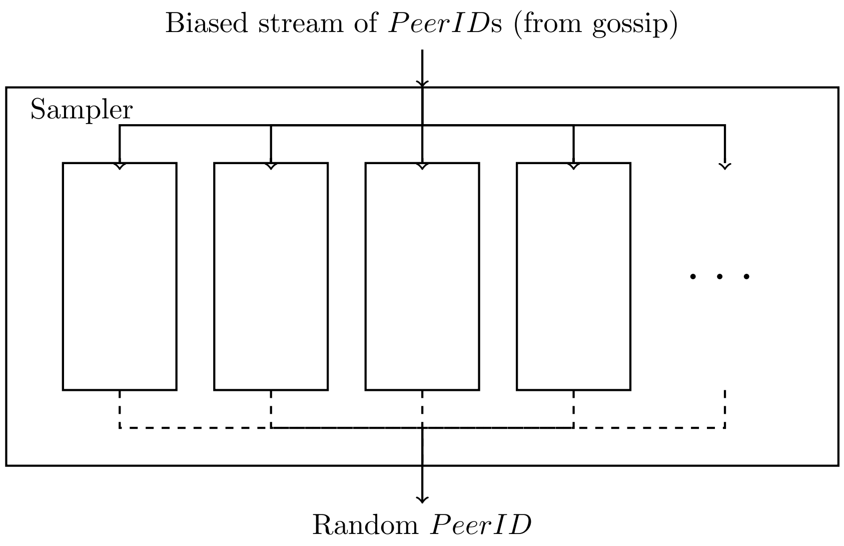


Figure 2.1: Sampler consisting of multiple sampler elements

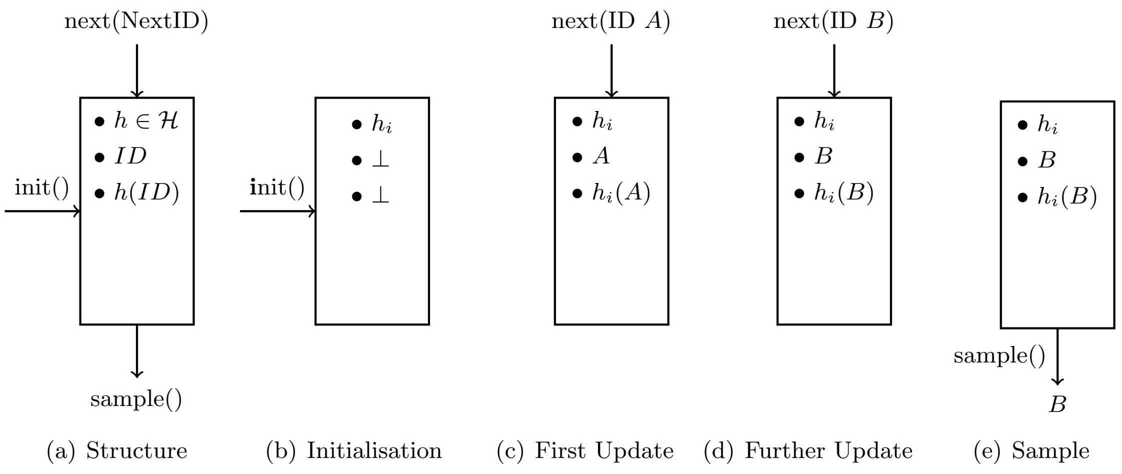


Figure 2.2: Sampler element

Min-wise independent permutations To understand how the sampler works, min-wise independent permutations [BCFM00] have to be understood first. They are a family of permutations $\mathcal{H}$ with input $[1 \cdots |U|]$ and the property that for an arbitrary set $X \in [1 \cdots |U|]$, $x \in X$ and a random $h \in \mathcal{H}$ the probability $Pr(min(h(X)) = h(x)) = \frac{1}{|X|}$. In words: For each element of a set (of peer IDs) the probability to have the minimum image under a randomly chosen permutation is the same.

Sampler element On creation these elements visualised in figure 2.2 are initialised with a random min-wise independent permutation (Figure 2.2(b)). In order to “choose” a random element from a number of elements *independently of their occurrence*, all of those elements (peer IDs in this case) have to be ‘put into’ an initialised element. Every time a peer ID is put into such an element, the previously initialised min-wise independent permutation is used to determine what ID will be stored. The image of Both peer IDs, the one already stored and the new input, under the permutation is computed. The ID with the smaller image is stored (Figure 2.2(d)). If the sampler element currently holds no ID the ID put into it is directly stored (Figure 2.2(c)). When requested, the stored ID is given back (Figure 2.2(e)).

The ID a sampler holds can be viewed as a random choice of all IDs that have been put into the sampler so far without giving respect to the frequency it has occurred.

The peer IDs held by the sampler elements are probed periodically and if the corresponding peer is off-line the respective sampler elements are re-initialised.

2.2 Applications

In the field of (unstructured) peer-to-peer networks, there are multiple applications that profit from random samples of the network.

First we are going to present some general concepts where random peer sampling is useful. In the end of this section we illustrate it’s usefulness in a practical example (onion routing).

Information dissemination

Efficiently disseminating information to peers in the network is covered in [Pit87, DGH$^+$88, EGKM04]. As actively spreading information in decentralised networks shows similar characteristics as epidemic spread, knowledge accumulated in this area of research is used.

Probabilistic multicast can be considered a more concrete application. Research considering this application is [BHO$^+$99, MK04].

As the similarity to epidemics might suggest, information is disseminated through the network by passing it to random peers.

Algorithms exploiting this mechanism can provide scalability, easiness to deploy, robustness, resilience to failure and can even in case of churn be highly reliable. Many of those features are achieved because no central entity – a possible single point of failure and bottleneck – is needed.

Information aggregation

The efficient aggregation of information in unstructured, dynamic peer-to-peer networks is a research topic that comprises the following tasks:

In general the goal is the development of mechanisms that enable a participant in such a network to gain global information like the network size, average load, average uptime, location and description of hotspots [JMB05], as well as the computation of aggregate functions like sums, averages, quantiles random samples etc. of the information stored at the nodes in the network [KDG03]. It also can be used to compute statistics as presented in [SRD$^+$09].

Other work on this topic also includes [BGMGM03, JKvS04, MJB04]. Kostoulas et al. are focussing on a single task like estimating the size of the given network ([KPG$^+$05]).

Those algorithms are commonly based on gossip. More precisely they often exchange information with peers randomly selected from the network.

Gossiping is also useful for those protocols as it provides scalability, efficiency, robustness and is adaptive while there is no need for a single point of failure.

Network Management

Random peer sampling is also an useful tool for improved and decentralised management of networks. A good structure of a network is essential for several aspects.

According to [OJ09], one application of random peer sampling is also the construction of overlay networks that approximate random graphs. This is useful to information dissemination with low-maintenance as the resulting graphs show the useful properties like being well connected and having a low diameter [KSSV00, KMG03]. A low diameter is a desired property for efficient routing and a well connected graph is less prone to partitioning.

2.2.1 Onion routing (OR)

Onion routing was introduced in [SGR97, RSG98]. It provides anonymous connections over a public network.

In order to communicate anonymously, a channel over multiple so-called “onion routers” is established. Through the application of cryptography, each hop on the created route does only learn its predecessor and successor. Nobody except the initiator should know who communicates with whom.

To minimise the threat of deanonymisation of an established channel through collaboration of onion routers, a new channel is chosen on a regular basis. The choice of onion routers is important in order to prevent (frequent) building of channels over collaborating nodes. Approaches exist that make use of further knowledge; either about the peers themselves or about the topology of the network. Without the knowledge about these routers, the best strategy would be to choose random ones.

2.2.1.1 TOR

TOR (The Onion Router) is an concrete implementation of onion routing aiming to anonymise traffic over the internet.

The peer selection of nodes to route over is realised in a centralised way. The membership information of all onion routers in the TOR network is maintained at a few (currently eight to ten) so-called "directory servers". A participant, in order to select peers to establish a channel over, has to retrieve the list of onion routers from those central servers and locally select random peers from that list. Due to the synchronisation protocol executed between the directory servers, it is enough to control 50% of those to be able to control the membership information.

The maintainers of TOR reported they expect that this attack vector is exploited [Pro].

2.3 GNUnet

GNUnet provides the following needed features and services: It is an overlay network that interconnects peers over different underlying protocols/connections [Wac15]. In order to address peers in this network and as part of the abstraction, it identifies peers by their unique public key.

Brahms assumes that peers are not able to create multiple (fake) IDs to rule out Sybil-attacks. In reality this is impossible to enforce, but GNUnet makes a massive creation of new peers expensive as creating a peer involves computing a proof of work.

2.3.1 NSE

Some of Brahms variables like the size of the sampler and the view are dependant on the size of the network. As it is an difficult issue to estimate the size of an unstructured peer-to-peer network, the network size estimation (NSE) service of GNUnet comes in very handy. It provides a reliable estimation of the size of the network that cannot be (easily) influenced by malicious peers.

2.3.2 CADET

Brahms assumes that the underlying connections are safe in the following way: Single peers are able to communicate with each other through a fully connected network with reliable links. Every peer is able to determine the source of every message and is not able to intercept messages between other peers.

CADET [PG14] is the part/service of GNUnet that provides communication with the other peers on the transport layer. It provides the desired characteristics. Every pair of nodes can communicate via bidirectional reliable links (knowing each other's ID).

Plus, CADET is one source of the initial view, as it provides the random peer sampling service with the information of peers that CADET is already connected to.

2.4 Related Work

In this section we are going to present what research has been done in this field already as well as work that is related. Many algorithms have been proposed to solve the issue of random peer sampling ([JKvS03, VGvS05, BvS08] and others). Most of them try to achieve better performance without considering the threat through Byzantine activity. Below we give an overview over more interesting or exemplary work.

Capturing accurate snapshots of the Gnutella network Stutzbach and Rejaie propose a peer sampling algorithm that is based on random walks in [SR05, SRS08]. This work is used for computing statistics of unstructured peer-to-peer networks as presented in [SRD$^+$06]. Unfortunately random walks can be easily exploited by attackers.

Choosing a random peer in Chord An approach that involves the use of a distributed hash table (DHT) is presented in [KLSY07] by King, Lewis, Saia and Young. Though they are able to proof that their method is capable of providing uniform samples, they are not secure against Byzantine behaviour.

Peer sampling with improved accuracy In this paper Ogston and Jarvis focus on identifying common sources of weaknesses in random peer sampling algorithms [OJ09]. They therefor analyse and compare two algorithms with their newly introduced algorithm that addresses those shortcomings. Their approach examines the representation of peer IDs at other peers in the network. Issues addressed are the temporal and spatial bias, the two algorithms suffer from. However, they do not consider the threat of malicious peers.

SPSS Jesi et al. presented different versions of an algorithm with the name ***S****ecure* ***P****eer* ***S****ampling* ***S****ervice*. They focus to solve the threat of malicious behaviour (namely two attack scenarios they introduce: The "Hub-attack"[JGGvS06] and the "Mosquito-attack"[JM09]) concentrating on the identification of malicious peers. Their solutions extend underlying peer sampling services (namely Newscast [JKvS03] and PSS [JGKvS04, JVG$^+$07]) that are not resilient to Byzantine behaviour. Their first approaches involved a (central) Certification Authority [JGKvS04, JGGvS06]. Later that approach was improved by proposing a completely decentralised version [JMvS07]. To achieve the goal of identifying malicious peers the authors present an interdisciplinary approach in [JMNvS09]: They estimate the *'prestige'* of a peer like structural leaders are identified in the field of Social Network Analysis. However, the latest version presented in [JMvS10] again involves a central CA. Although they present good performance achieved in simulations, no formal proof of the claimed features is given.

Uniform Node Sampling Service Robust against Collusions of Malicious Nodes Anceaume et al. let aside the gossiping and focus on the sampling of peer IDs itself. They propose a sampler that reads a possibly biased stream of elements (peer IDs in the case of peer sampling) and outputs an approximation of an uniform random stream of the same IDs [ABG10, ABS13]. They support their work with theoretical proofs as well as extensive simulations.

Gossip-based peer sampling in [JVG$^+$07] Jelasity et al. propose a framework to develop and evaluate a peer-sampling service. They also discuss the performance of their own service through extensive simulations. Their abstract model of a peer-sampling service focusses on two aspects: 1. a relatively small *partial view* and 2. a periodical refreshment thereof – like Brahms – using push-pull-gossip. Their analysis considers the local randomness a single peer observes as well as the global distribution that evolves during protocol execution. Over the whole paper they always consider the implications of the observed behaviour on the above protocols.

diehard The diehard battery of randomness tests was introduced by Marsaglia in [Mar]. It covers tests for the quality of randomness of (pseudo-) random number generators as well as streams of (pseudo-) random numbers. This is used to evaluate the implementation. In [MT02] the same author gives an insight in three test of randomness that are hard to pass.

Information about the whole network As stated by Jelasity et al. in [JVG$^+$07] a trivial approach would be to maintain membership information of the whole network at every single peer. Each peer could make a 'real' random selection locally without being faced with malicious behaviour. However, in practice this is not realistic for real-world applications as maintaining the membership information of a large and dynamic system incurs considerable synchronization costs.

3. Design and Implementation

This chapter presents the choices made to have a working implementation in the GNUnet framework.

3.1 Maintenance of peer IDs and Random Choice

The way peer IDs are stored locally is not only important for performance but may also have structural influence.

In the original design, the view may contain duplicates and whenever a random subset of a set is to be chosen, it also may contain duplicates. As we try to increase the number of *different* peer IDs we decided to deviate from the protocol here. With our implementation we aim at avoiding duplicates in every list (push/pull-lists and view). Avoiding duplicates is useful as every duplicate can be seen as a 'wasted slot of information'. There is no use in sending one peer ID multiple times in a pull reply or sending multiple pushes or pull requests in quick succession.

3.2 Min-wise independent Permutation

Pseudo-random functions as presented for example in [GGM86] are considered a good approximation of min-wise permutations, provided the input-range for those is large e.g. 2^{160}. We used a cryptographically secure hash function to implement those permutations. The initialisation is the step of choosing a key for the HMAC function at random. So applying the function on the peer identities gives random hashes that are deterministic per peer and initialisation.

3.3 Staying connected

The CADET service responsible for the underlying networking layer provides reliable channels and with that, cares about the liveliness of the connection. To be able to communicate with another peer a channel is established. The other peer will not close this channel unless it's going down. This way, when a channel goes down, we know that the peer is no longer reachable and do the necessary clean-up. Also, once the establishing of a channel to a peer ID succeeds, we learn that this peer is currently on-line and also that a peer to the corresponding ID exists.

3.4 Periodical re-initialisation

Brahms requires to probe other peers repeatedly to see whether they are still on-line. If they appear to be off-line, the sampler elements containing their ID are re-initialised.

In our implementation, connections towards all peers in the sampler are kept open. When such a connection is closed by the peer in the sampler all elements containing this ID are re-initialised.

3.5 Limited Push

The design of Brahms includes a limitation on the number of pushes one single peer is able to send in a specific period of time. This is called *limited push*. Bortnikov et al. propose to use some cryptographic mechanism like a proof of work or cryptocurrency.

As both solutions imply a high computational overhead neither was implemented. Instead the assumption was made that sending pushes is limited in bandwidth. For the evaluation of the implementation a limit of pushes is emulated via a hard limit of pushes that one malicious peer is able to send in one round.

Implementing a proof of work could be done in the following fashions or a combination of own peer ID, other peer's ID, time dependant (for example every hour).

A combination might be the computation of a proof of work every hour over the own id.

However, a proof of work is already used by the NSE service of GNUnet, so it would be an easy task implementing it.

3.6 Pull Reply Size

Bortnikov et al. point out that a peer that sent a pull request should only accept a single pull reply following that. This is useful to keep a malicious peer from sending arbitrary many (in the worst case infinitely) peer ids.

In our implementation the size of the pull reply is also limited to the maximum size a single message over CADET can take. But this might also limit a correct peer from sending its actual view and truncate it at a given size. To solve this one could split up the reply in multiple messages. But that would bring back the possibility of sending an arbitrary number of peer ids. Letting a malicious peer send infinitely can be prohibited via fixing the number of peers sent in the first message of the pull reply. This would fix the maximum number of peer IDs that can be set to the maximum representable with the representation of the number sent with the first message.

In essence it is not so important to fix the number of replies but to fix the number of replied IDs to a reasonable number. A viable approach would be to only accept at maximum 150% of the own view size.

3.7 Interaction with higher Layers

As the overall goal is to provide random samples to other applications as the random sampling of peers is no isolated topic but embedded in and thus interfering with other modules some extra issues have to be taken into account.

3.7.1 ID stream

Brahms achieves that a locally maintained, small set of peer IDs converges to a small random sample of the network. Once this sample converged it will be relatively static. The only changes occur when a peer is off-line, during the regular probing of peers in the sampler. The samplers containing the according ID is then re-initialised. Thus, the sampler will converge to a set of peers that is on-line for a long time.

To be useful for higher layers a sampling service should be able to provide random peer IDs on request. In the case of Brahms this could be done via replying a peer ID from the sample which is itself chosen at random.

In order to satisfy repeated requests this would not suffice as the peer IDs might be random but still chosen from a relatively small, rather statical set.

To be able to do so we implemented a modified version of the sampler proposed by Bortnikov et al. The sampler proposed in [ABS13] would probably solve this problem very well, but this has yet to be evaluated.

- To rule out that one sampler element is queried more than once we re-initialise each queried element.
- The peer ID a sampler currently holds represents a random choice of all IDs that have been put into it since the last (re-)initialisation. Therefore it is desirable to maximise the number of peers that were put into the sampler element since the last (re-)initialisation. When re-initialising after each query it has to be paid attention on how to choose the sampler element to be queried.

  If we choose one at random we might choose one that has recently be queried and re-initialised although there might be samplers that have not been queried and re-initialised for a long time. To solve this we query the sampler elements in a round robin-like fashion. That ensures that the element we are going to query is the one that 'saw' most peer IDs since its last (re-)initialisation.
- Querying the sampler element, which saw the maximum number of peer IDs, does not ensure that it saw 'enough' IDs. As quantifying how many peer IDs are 'enough' depends on a lot of factors we use the time as measure: No sampler element is requested more than once in the time of a single round. This is only a very basic limit that might be replaced by a stronger one. For example limiting the requests to sampler elements that 'have seen' more than a certain fraction of the estimate of the network size. A compromise between the randomness and waiting time for a response has to be made here.
- The latter might be a problem if we are querying lots of peer IDs fast. To be able to request many peer IDs quick enough we make an estimation on how many peers will be requested in the near feature and adapt the number of sampler elements accordingly.

3.7.2 Information leakage

Every time, Brahms responds to a pull request, it gives a possible adversary a bit of information about what peers have already been seen. It is an desirable property that an adversary cannot gain knowledge about a later choice from this information.

As a pull reply not only contains peer IDs that might have made their way into the view, ten percent definitely are drawn directly from the sampler. Over time it is possible to gain information on a big portion of the IDs contained in the sampler and it even might be possible to gain information about the initialisation of single sampler elements or, in the case of a weak number generator, about probable random choices being made in the future.

For this reason it is desirable to separate the sampling for client requests from the sampling used for the protocol itself.

3.8 Adapting to the Network Size

Bortnikov et al. suggest a size for view and sampler that depends on the network size. In order to use view and sampler with an adequate size, we make use of the network size estimation (NSE) GNUnet offers. Every time, NSE provides a new estimate we adapt the sampler size accordingly and compute the size the view will have after its next update.

3.9 Round Interval

Unlike the theoretical assumption made by Bortnikov et al. a real network does neither have a discrete global clock, nor are there zero processing times and the messages sent do not have latencies of "a single time unit". As the rounds do not execute in discrete time intervals, a time interval has to be chosen.

Processes in networks that are executed in rounds are known to synchronize with time. To actively avoid that all peers send their messages at the same time, we locally randomize the time until the next round is executed.

3.10 Initial View

Brahms assumes that a peer already knows some peers in the initialisation phase. GNUnet provides different sources for peer IDs:

CADET CADET maintains information about peers to which it is connected to as well as peers that are on the path to those. Its API provides a function to learn all those peer IDs.

Peerinfo This service keeps track of verified (validated) information about known peers in a persistent way. It receives information from other services which are responsible for validating the information through establishing connections towards them which involves a cryptographic hand-shake. It is possible to subscribe to this service in order to get continuous updates about additional peers.

Friends This might be of additional value as this API provides Brahms with information about peers that are with very high probability correct. It has to be paid attention nevertheless: Being friends doesn't keep those from being compromised.

4. Evaluation

This chapter is about measuring the performance of our implementation of Brahms. As the whole topic is about randomness, this is a difficult task. The ultimate goal is to make a statement about the quality of the randomness of the peers returned to the client. Also this evaluation aims to be well comparable with other implementations through the presented results. To achieve this, we are taking the following measures to achieve an thorough evaluation at different levels.

As the implementation of Brahms took longer than expected, we are not able to present the results for the following measures.

Brahms Specifics

First we are going to compare the performance of our implementation with the results of the simulation of Brahms presented by Bortnikov et al. in [BGK$^+$09]. This should yield insight in the difference between the more theoretical simulation to our emulation, as well as reveal possible deviations introduced through (minor) implementation differences. We therefore try to reproduce the numbers presented in tables in that paper.

Comparison of Samplers

To make a statement on the quality of the modified sampler we are going to compare it with the sampler presented by Anceaume et al. in [ABS13]. In order to do so, we use the same methods they used.

Performance in common measures

In order to have more implementation-independent measures of performance, we apply the measures Jelasity et al. proposed in [JVG$^+$07]. This is not only useful to compare our results with those of Jelasity et al. but also to be generally comparable to other (upcoming) implementations.

To make a qualified statement about the performance those evaluations are made in different scenarios (malicious, non-malicious).

4.1 Brahms Specifics

To compare our implementation of Brahms to the results of the simulation presented in [BGK$^+$09] we first try to reproduce the results presented in that paper. After that we have a look at some observations specific to the Brahms protocol.

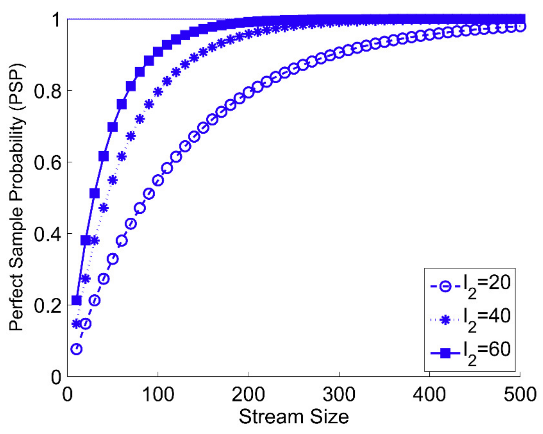


Figure 4.1: **Growth of the probability to observe at least one correct perfect sample (Perfect Sample Probability — PSP) with the stream size, for** 1000 **nodes,** $f = 0.2$**, and** $\rho = 0.4$**.** ([BGK+09] Figure 5)

4.1.1 Comparison with Brahms

To compare our observations with the ones from their simulation, we used the parameters, Bortnikov et al. showed to be the best for Brahms. Namely those are $\Theta(\sqrt[3]{n})$ for l_1 and l_2, $\alpha = \beta = 0.45$ and $\delta = 0.1$.

4.1.1.1 PSP (Perfect Sample Probability)

In the terminology of [BGK+09] a perfect sample or perfect ID is an ID that has the minimum image under the permutation function of a sampler element. Hence, a correct perfect sample is a non-malicious perfect sample. The *Perfect Sample Probability* denotes the probability that at least one correct perfect sample was observed. One such sample guarantees that we will not be disconnected from the rest of the benign peers (at least until the peer is on-line).

The growth of this probability dependent on the view size (l_2) over time is illustrated in figure 4.1.

4.1.1.2 Portion of faulty IDs in the View

It is also of interest how many faulty IDs are in the views of the peers at a given time. This indicates to how many malicious peers the average peer is connected. As can be seen in figure 4.2 the malicious peers manage to be overrepresented but are not able to totally poison the view of all peers even for large numbers of faulty pushes. The figure clearly shows the usefulness of the history updates.

Bortnikov et al. showed that the fixed point $0 < \hat{x} < 1$ is a fraction of malicious peers represented in benign views (called x) that x will eventually converge to.

4.1.1.3 Fraction of perfect Samples and faulty IDs in the Sampler

Even more important to know is the portion of perfect samples and faulty IDs in the sampler. This represents the final random sample of the whole network at each peer. The fraction of faulty IDs should converge to the actual fraction of malicious peers in the network, like 4.3 illustrates.

In the beginning, the fraction of malicious peer IDs shows a peak, as a high fraction of malicious peers is put into the sampler due to overrepresentation in the gossip. As a the

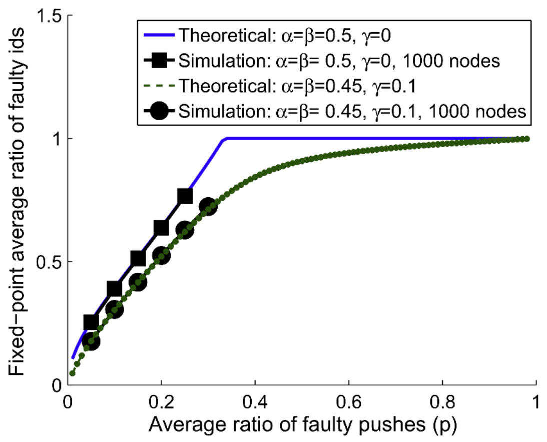
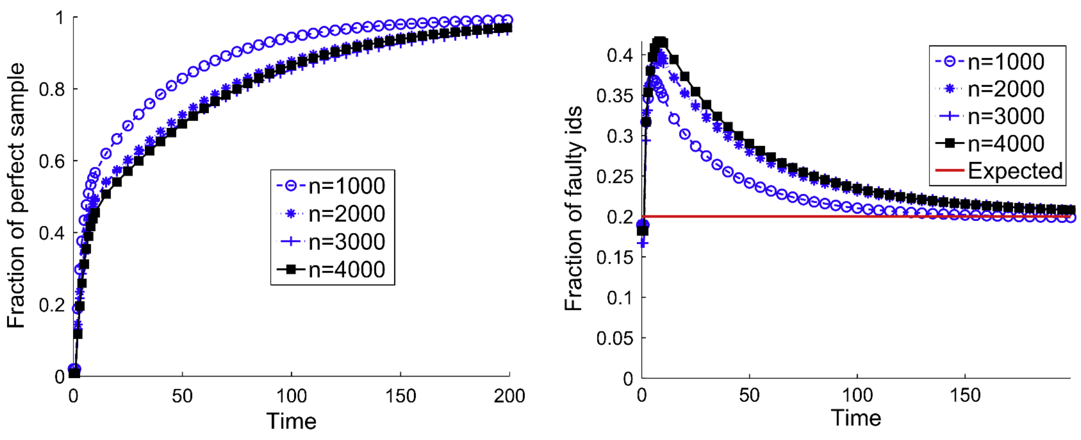


Figure 4.2: **Fixed point $\hat{x}$ of the system-wide fraction of faulty ids in local views, as a fraction of p, under a balanced attack.** ([BGK+09] Figure 7)



(a) Perfect samples vs time (b) Faulty ids vs. time

Figure 4.3: **Balanced attack: fraction of perfect samples (a) and faulty nodes (b) in $\mathcal{S}$, for $f = 0.2$, $n = 1000, \ldots, 4000$, and $l_2 = 2\sqrt[3]{n}$.** ([BGK+09] Figure 11)

peer ID that is hold by a sampler element only represents a random choice from all IDs observed so far and many of the benign peer IDs are not contained in the early stream, the malicious IDs are more likely to appear. The overrepresentation is lowered as more (uderrepresented) benign peer IDs are observed through gossip. As more (different) peers IDs are observed, the representation in the sampler approximates the real representation of peers. The representation of the malicious peers approximates their real representation.

4.1.1.4 Effectiveness of Attack — Isolation of Peers

Besides examining the amount to which peers are influenced under an attack, it is important to validate the claim and observation made by Bortnikov et al. that even a targeted peer is able to withsand an attack without being isolated from the rest of the (benign) peers.

4.1.2 Further Brahms-specific Observations

In this section we make some observations specific to Brahms that have not been made in [BGK+09].

4.1.2.1 Distribution of Peer IDs

Besides the (over-) representation of malicious peer IDs in the network, it is important that each peer ID is equally represented at peers over the whole network in malicious as well as in non-malicious settings. In order to analyse this we count the number of times every peer is represented at other peers. This idea is mainly inspired by Ogston and Jarvis [OJ09], who analysed the representation in the views of peers for different protocols. They did this analysis in order to show that some peer sampling algorithms face temoral and/or spatial bias. In Section 4.3.3 we consider the question whether that applies to our implementation of Brahms. In addition to the view we are also able to analyse the representation in the sampler.

In order to gain an overview over the whole emulation, the average number and standard deviation of occurrences in other views and samplers is analysed over time. To learn the amount different peers are (over/under-) represented over the whole run, the time peers are represented at other peers is summed up.

For non-malicious scenarios we expect to see a rather good distribution in the view as well as in the sampler. In contrast, for the Byzantine case we expect a notably increased representation of the malicious peers in the view but not in the sampler, as the analysis and simulation results presented by Bortnikov et al. suggest.

4.1.2.2 Frequency of Blocking (not updating) and Cause

In this section we consider the frequency of blocking peers and the distribution on the single causes of blocking. We also analyse the limitation of sending pushes in connection with blocking.

Bortnikov et al. did not evaluate blocking in their implementation as they assumed that either no blocking happens in their theoretic analysis or that the effect of blocking is marginal. However, we think evaluating blocking is needed as argued below.

We recall blocking: In the normal case a peer updates its view at the end of each round: It computes the new view as random selections from pushes, pull replies and the sampler. To reduce the impact of malicious behaviour and balance pushes and pulls, it blocks (does not update its view) if *a*) no pushes, *b*) no pull replies or *c*) too many pushes ($> \alpha l_1$) were received since the last update.

This has to be evaluated in the benign as well as in the malicious scenario. Blocking is undesirable as the peer does not provide updated information through pull replies in the following round and stays connected to the same peers. Thus it is important to know how often peers blocked for what cause in benign as well as malicious settings and use that knowledge to minimise blocking in the future. Analysing the frequency of blocking is needed to identify possible constraints on parameters (α,β,γ,l_1,l_2) and to quantify the (needs) limitation of sending pushes.

Benign setting

In the benign setting every node is expected to receive exactly as many pushes as it sends in one round — αl_1. We expect this to be true *in average.* As this perfect distribution of pushes on peers is highly unlikely and every peer that receives one message more than the expected amount blocks we expect to see a non-neglecting number of blocking peers. This is highly unwanted as blocking in the benign case does not prevent any malicious action. We propose to add a little tolerance factor on the amount of still acceptable received pushes depending on the frequency of the observed number of received pushes.

Malicious setting

If blocking is highly likely already in the benign case, the situation deteriorates for the case of adversaries that try to increase their representation by sending a higher amount of pushes. The ability for an attacker to do so strongly depends on the limitation on the sending of pushes. The simulations presented in [BGK$^+$09] assume the maximum amount an (attacking) peer is able to send is the number of pushes a normal peer would send in a round.

Two different scenarios have to be taken into account:

An attacker that tries to *force* blocking A possible attack scenario can exploit blocking in order to massively slow down or completely stop progress: This can be easily done by focussing the pushes on the largest possible subset of peers that is blockable with high probability. Every peer that is continuously blocked will (through churn) decrease its view until no peer is left in there. Even if a peer that was in the view previously comes on-line again is not put back into the view. As the view decreases, the number of peers the attacked peer sends pushes to is decreased and its representation decreases.

Once, a peer's view is empty (which can be checked with a pull request) it doesn't need to be attacked any more, as it it will never update afterwards. (As the view is empty, it will not send pull requests an thus will not receive replies which are needed in order to update the view.) The resources of an attacker can now be used to attack another peer until the whole network is blocked.

This attack is even possible under the strongest possible assumption on the limitation of pushes sent, that was made for the simulations: Even if the malicious peers are only able to send as many pushes in a round as every other peer, they are able to focus those on a (large) subset of peers in a way that keeps them continuously from updating.

An attacker that tries to *avoid* blocking In the case an attacker wants to increase its representation it wants to avoid blocking so the attacked peer can spread information of the attacker. The increase of the amount of pushes accepted for updating has to be weighted against the disadvantages in the malicious setting. If the disadvantages are negligible, there should be an increase, else it makes no sense to sacrifice vulnerability for better performance in benign scenarios. But this depends on the actual results.

4.2 Comparison of Samplers

This section compares our modified sampler with the sampler proposed in [ABS13].

We recall: Anceaume et al. proposed a sampler that aims at sampling elements (in the case of peer sampling those elements are the peer IDs) from a random stream.

They show theoretically and through extensive simulations the capabilities and limitations of their algorithm. The theoretical analysis of our modified sampler remains future work. In order to analyse their experiments they compared the in- and output stream using the *Kullback-Leibler (KL) divergence* which reliably measures the statistical difference between two streams of elements.

4.3 Performance in common Measures

In this section we are following the approach Jelasity et al. proposed in [JVG$^+$07]. This involves the examination of the randomness observed at a single peer as well as characteristics of the whole system.

In [OJ09] Ogston and Jarvis focus on possible biases in benign settings. We apply the same measures to evaluate the resistance against those.

4.3.1 Local

To evaluate the quality of randomness a single peer observes, Jelasity et al. use the same approach as used to evaluate the quality of random number generators.

As a random peer sampling service/protocol should provide each executing peer with independently random peer IDs from the whole network, testing the ID stream of a single peer like a (pseudo) random number generator should yield a structural weakness if there is one.

For the evaluation we planned to use the three difficult-to-pass tests Marsaglia and Tsang describe in [MT02].

4.3.2 Global

As the evaluation of the ID stream at a single peer might not reveal systematic problems that just don't affect the focused peer, Jelasity et al. also propose graph theoretic measures about the structural sanity of the network.

To apply the graph theoretic measures, the state of the network is described through a graph that is constructed as follows. Each peer is represented as a node and for each peer ID in the local view, a directed edge towards the respective node is added to the graph. Ideally, this graph would show the same properties as a random graph.

One measure used by Jelasity et al. is the *degree distribution.* In the used model, the *in-degree* of a peer is the number of peers that have this peer in their view. The out-degree is of minor interest as it is simply the view size (Bortnikov et al. suggest to use $\sqrt[3]{n}$) which should be the same for all peers as the network size estimation provided by gnunet-nse should be the same for all peers.

Other measures used to analyse the structure of the network are clustering (coefficient) and (average) path lengths. The clustering coefficient provides information about the degree of inter-connectivity between neighbours of a node. The shortest path length between two nodes is the minimal number of edges needed to traverse the graph from one to the other. The average path length of the graph is the averaged number of shortest path lengths between all nodes in the graph. This measure is an important indicator for the cost of information dissemination.

Analysing the graph theoretic measures, Jelasity et al. do this especially with the use in higher layers in mind.

4.3.2.1 Degree Distribution

The probably most important aspect degree distribution gives an insight in is the dynamic of the network. Possible dynamics include *chaotic*, *oscillating* and *converging.* Whereas convergence independent of the initial configuration is the desirable feature. The question of convergence is directly connected to the question to what the network will converge in the case of convergence. Like Jelasity et al. we sill also analyse the local dynamic.

4.3.2.2 Path length

This property is useful to make statements about the performance of the network in information dissemination scenarios. It is a lower bound on the cost of reaching a peer.

4.3.2.3 Clustering

According to Jelasity et al. it is important that the clustering coefficient does not grow too high as in the scenario of information dissemination this means a higher probability of redundant sending of information. Also a high clustering coefficient indicates the development of strongly interconnected clusters that are weakly interconnected. This rises the danger of partitioning.

4.3.3 Possible Biases

As some structural biases may not show up in the previous evaluations, we are going to explicitly test for those.

Ogston and Jarvis state in [OJ09] that peer sampling algorithms suffer from temporal as well as spatial dependencies between samples.

4.3.3.1 Temporal

More specifically, peers may be over- or under-represented at different points in time. This may have quite big implications on higher layers. The network might suffer from an overrepresented node that is not capable to handle a big amount of traffic that he receives due to this overrepresentation.

4.3.3.2 Spatial

For a really random choice it is important that each peer ID has the same probability to be chosen independently of its location. Spatial dependencies of the samples may lead to such a bias. Another disadvantage is the increased probability of network partitioning.

4.4 Discussion

In this section we are going to discuss some issues that have to be considered when using our implementation as well as general problems we noticed.

4.4.1 Work Load

In order to run the sampling service in a real-world network the load it creates for single peers has to be taken into account. The usefulness for other applications implies that this service is very likely to not be the only task run at the peers. Also single peers in the network might be small devices, not capable of executing a huge amount of operations.

There are several aspects that influence the computational complexity needed to execute this service: The most important is the *round interval.* It determines how often operations have to be executed. A short interval results in quick adaption to churn and fast spreading of information over the whole network. On the downside doing short rounds also means that connections are very often closed and opened and messages are sent at a higher frequency. The sending of messages plays an important role as it also means that pushes are sent at a higher rate which involve the computation of a *proof of work.*

Another mean to improve the effectiveness and number of messages sent are the *sizes* of the *view* and the *sampler.* The size of the view influences the behaviour of Brahms in a similar way as the modification of the round interval. Increasing the size of the sampler increases the perfect sample probability and the number of different elements at a relatively low cost.

In the end, the main limiting factors will be *a*) the actual need of fresh, random peers and *b*) the shape of real-world churn Brahms has to deal with. Those have yet to be evaluated.

4.4.2 Information Leakage

Another question that has not been considered in [BGK$^+$09] is how much sensitive information is leaked with a pull reply. Depending on the exact protocol, the information a peer provides restricts the possible random selection that peer is about to make. Plus, an attacker can easibly gain detailed information about the structure of the network and the success of his attack at selected peers. This way it receives much information that helps planning and fine-tuning a given attack (be it on selected targets or the whole network).

4.4.3 Identification of Malicious Peers

Brahms tries to prevent malicious behaviour without identifying the malicious peers. As Jesi et al. did quite some work on the identification in the context of random peer sampling it would be interesting if Brahms could be improved in that aspect.

Another idea is to use the additional information GNUnet provides to accomplish this task. As benign peers send their current view size that directly depends on their network size estimate a big deviation from the own estimate should raise suspect. Comparing the estimate with the one of the other peer could limit an attacker significantly in increasing the representation of malicious peers.

4.4.4 Biased View

Brahms tries to decrease the threat of a possible bias in the gossip through blocking. As Bortnikov et al. show in [BGK$^+$09], the views are still very biased. This is not desirable for many reasons. The view is an important building block that influences other parts in a significant way. When the malicious peers are overrepresented therein they gain quite some insight and influence on the rest of the network. It would be desirable to try to ‘filter’ the peer IDs obtained through gossip *before* storing them for later use. This could be done using the sampler presented by Anceaume et al. in [ABS13].

In particular this would render blocking and the *limited* push unnecessary as an attacker would gain nothing from being overrepresented in the gossip.

4.4.5 Message Loss

In the case of severe message loss, Brahms blocks updating too frequently. This makes the application in error-prone networks like mobile ad-hoc networks as well as in networks where and attacker is able to interrupt connections quite unattractive.

5. Conclusion

We presented the Brahms algorithm that is aimed to solve the problem of random peer sampling even in the presence of malicious behaviour. An overview over of related work was given. We described the problems we faced when implementing the algorithm in a “real-world” service.

Further, we show a number of different measures to evaluate the performance of Brahms alongside with work that presents results in the same measures. Some of those measures are specific to Brahms and others are general measures that can be applied to all algorithms trying to solve the task of sampling random peers. Unfortunately we were not able to present own results to those measures.

Finally we discussed some problems that have to be considered when implementing Brahms or another algorithm trying to solve the same task for real-world applications.

Many research has been done in the field of random peer sampling. Only some considered the threat through Byzantine behaviour. The authors of Brahms were the only ones to present a complete algorithm together with a thorough theoretical analysis and simulation. Yet no critical review or evaluation of real-world performance has been published.

Bibliography

[ABG10] Emmanuelle Anceaume, Yann Busnel, and Sébastien Gambs. Uniform and ergodic sampling in unstructured peer-to-peer systems with malicious nodes. In Chenyang Lu, Toshimitsu Masuzawa, and Mohamed Mosbah, editors, *Principles of Distributed Systems*, volume 6490 of *Lecture Notes in Computer Science*, pages 64–78. Springer Berlin Heidelberg, 2010.

[ABS13] Emmanuelle Anceaume, Yann Busnel, and B. Sericola. Uniform node sampling service robust against collusions of malicious nodes. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on*, pages 1–12, June 2013.

[BCFM00] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630 – 659, 2000.

[BGK$^+$09] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks Journal (COMNET), Special Issue on Gossiping in Distributed Systems*, April 2009.

[BGMGM03] Mayank Bawa, Hector Garcia-Molina, Aristides Gionis, and Rajeev Motwani. Estimating aggregates on a peer-to-peer network. Technical Report 2003-24, Stanford InfoLab, April 2003.

[BHO$^+$99] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, May 1999.

[BvS08] A. Bakker and M. van Steen. PuppetCast: A secure peer sampling protocol. In *Computer Network Defense, 2008. EC2ND 2008. European Conference on*, pages 3–10, Dec 2008.

[DGH$^+$88] Alan Demers, Dan Greene, Carl Houser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. *SIGOPS Oper. Syst. Rev.*, 22(1):8–32, January 1988.

[EGKM04] P.T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, May 2004.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, August 1986.

[JGGvS06] Gian Paolo Jesi, Daniela Gavidia, Chandana Gamage, and Maarten van Steen. A secure peer sampling service as a "hub attack" countermeasure. Technical report, Department of Computer Science University of Bologna, 2006.

[JGKvS04] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In Hans-Arno Jacobsen, editor, *Middleware 2004, ACM/IFIP/USENIX International Middleware Conference, Toronto, Canada, October 18-20, 2004, Proceedings*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer, 2004.

[JKvS03] Márk Jelasity, Wojtek Kowalczyk, and Maarten van Steen. Newscast computing. Technical Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, November 2003.

[JKvS04] Márk Jelasity, W. Kowalczyk, and M. van Steen. An approach to massively distributed aggregate computing on peer-to-peer networks. In *Parallel, Distributed and Network-Based Processing, 2004. Proceedings. 12th Euromicro Conference on*, pages 200–207, Feb 2004.

[JM09] Gian Paolo Jesi and Alberto Montresor. Secure peer sampling service: The mosquito attack. In *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE '09. 18th IEEE International Workshops on*, pages 134–139, June 2009.

[JMB05] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, August 2005.

[JMNvS09] Gian Paolo Jesi, Edoardo Mollona, Srijith K. Nair, and Maarten van Steen. Prestige-based peer sampling service: Interdisciplinary approach to secure gossip. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1209–1213, New York, NY, USA, 2009. ACM.

[JMvS07] Gian Paolo Jesi, David Hales Maarten, and van Steen. Identifying malicious peers before it's too late: A decentralized secure peer sampling service. In *Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on*, pages 237–246, July 2007.

[JMvS10] Gian Paolo Jesi, Alberto Montresor, and Maarten van Steen. Secure peer sampling. *Computer Networks*, 54(12):2086 – 2098, 2010. P2P Technologies for Emerging Wide-Area Collaborative Services and Applications.

[JVG$^+$07] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM Trans. Comput. Syst.*, 25(3), August 2007.

[KDG03] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 482–491, Oct 2003.

[KLSY07] Valerie King, Scott Lewis, Jared Saia, and Maxwell Young. Choosing a random peer in chord. *Algorithmica*, 49(2):147–169, 2007.

[KMG03] A.-M. Kermarrec, L. Massoulie, and A.J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *Parallel and Distributed Systems, IEEE Transactions on*, 14(3):248–258, March 2003.

[KPG$^+$05] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and Al Demers. Decentralized schemes for size estimation in large and dynamic groups. In *Network Computing and Applications, Fourth IEEE International Symposium on*, pages 41–48, July 2005.

[KSSV00] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 565–574, 2000.

[Mar] George Marsaglia. The marsaglia random number cdrom including the diehard battery of tests of randomness. http://www.stat.fsu.edu/pub/diehard.

[MJB04] A. Montresor, Márk Jelasity, and O. Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *Dependable Systems and Networks, 2004 International Conference on*, pages 19–28, June 2004.

[MK04] R. Melamed and I. Keidar. Araneola: a scalable reliable multicast system for dynamic environments. In *Network Computing and Applications, 2004. (NCA 2004). Proceedings. Third IEEE International Symposium on*, pages 5–14, Aug 2004.

[MT02] George Marsaglia and Wai Wan Tsang. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3):1–8, 2002.

[OJ09] Elth Ogston and Stephen A. Jarvis. Peer sampling with improved accuracy. *Peer-to-Peer Networking and Applications*, 2(1):24–36, 2009.

[PG14] Bartlomiej Polot and Christian Grothoff. Cadet: Confidential ad-hoc decentralized end-to-end transport. In *Med-Hoc-Net 2014*, 2014 2014.

[Pit87] Boris Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

[Pro] The Tor Project. Possible upcoming attempts to disable the tor network. https://blog.torproject.org/blog/possible-upcoming-attempts-disable-tor-network.

[RSG98] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. Anonymous connections and onion routing. *Selected Areas in Communications, IEEE Journal on*, 16(4):482–494, May 1998.

[SGR97] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *In IEEE Symposium on Security and Privacy*, pages 44–54, 1997.

[SR05] Daniel Stutzbach and Reza Rejaie. Capturing accurate snapshots of the gnutella network. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2825–2830 vol. 4, March 2005.

[SRD$^+$06] Daniel Stutzbach, Reza Rejaie, Nick Duffield, Subhabrata Sen, and Walter Willinger. On unbiased sampling for unstructured peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, pages 27–40, New York, NY, USA, 2006. ACM.

[SRD$^+$09] D. Stutzbach, R. Rejaie, N. Duffield, S. Sen, and W. Willinger. On unbiased sampling for unstructured peer-to-peer networks. *Networking, IEEE/ACM Transactions on*, 17(2):377–390, April 2009.

[SRS08] Daniel Stutzbach, Reza Rejaie, and Subhabrata Sen. Characterizing unstructured overlay topologies in modern p2p file-sharing systems. *IEEE/ACM Trans. Netw.*, 16(2):267–280, April 2008.

[VGvS05] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

[Wac15] Matthias Wachs. A secure and resilient communication infrastructure for decentralized networking applications. Phd, Technische Universität München, München, 02/2015 2015.