Bootstrapping Skynet: Calibration and Autonomic Self-Control of Structured Peer-to-Peer Networks

Timo Klerx*, Kalman Graffi†
*Department of Computer Science, University of Paderborn, Germany
†Technology of Social Networks Group, University of Düsseldorf, Germany
Email: timo.klerx@upb.de, graffi@cs.uni-duesseldorf.de

***Abstract*—Peer-to-peer systems scale to millions of nodes and provide routing and storage functions with best effort quality. In order to provide a guaranteed quality of the overlay functions, even under strong dynamics in the network with regard to peer capacities, online participation and usage patterns, we propose to calibrate the peer-to-peer overlay and to autonomously learn which qualities can be reached. For that, we simulate the peer-to-peer overlay systematically under a wide range of parameter configurations and use neural networks to learn the effects of the configurations on the quality metrics. Thus, by choosing a specific quality setting by the overlay operator, the network can tune itself to the learned parameter configurations that lead to the desired quality. Evaluation shows that the presented self-calibration succeeds in learning the configuration-quality interdependencies and that peer-to-peer systems can learn and adapt their behavior according to desired quality goals.**

I. Introduction

Peer-to-peer (p2p) systems have gained a lot of attention in the last decade, supporting large-scale networks for various applications [1]. Typically, p2p overlays, such as Chord or Kademlia, are used to connect all nodes in the p2p system. Most of these overlays are exhaustively configurable, ranging from routing table sizes, timeouts or, if enhanced, also in the strategies of how to choose contacts. While applying overlays for a specific use case, these configurations are fixed and typically not changed ever again. Thus, a small network starting with an initially optimal configuration keeps using this configuration even if the network size (e.g. scaling to millions of users), the usage type (e.g. from file sharing to a notification overlay) or the node capacities change. As a result, the provided quality of the overlay remains only best effort and the configuration is rarely ideal.

In this paper, we present an approach to keep the configuration of an overlay network optimal: We use neural networks [15] to learn the impact of the configuration on the resulting overlay performance. Thus, if a desired overlay performance is chosen, the network could tune itself to optimal configurations, thus implementing a distributed control loop. The Monitor-Plan-Analyze-Execute (MAPE) [2] cycle for autonomous systems describes the steps for such a distributed control loop. In the monitor phase, statistics about the network are computed. These statistics are analyzed in the analyze phase and handed over to the plan phase where a plan is constructed on how to transform the state of the system into the desired one. In the execute phase, the plan or parts of it are executed and the cycle restarts. In p2p systems the quality of the overlay can be monitored in terms of quality metrics using tree-based statistic aggregation protocols (e.g. [4]) or gossip-based solutions (e.g. [3]). An adaptation of the behavior can be reached through reconfiguration. For the execution phase, i.e. the overlay reconfiguration, modified tree-structures (e.g. [5]), broadcasting or local adaptation are applicable.

The main challenge that remains is in the analyze and plan phase: How to pick the overlay configuration based on the observed system quality in order to reach desired quality goals? The focus of this paper is to address this challenge and to provide an approach for the calibration of the p2p overlay, i.e. the systematic analysis of the effects of a wide set of configurations for the overlay. We review related work in Sec. II followed by an overview of our approach (Sec. III): We systematically simulate Chord [6] in Sec. IV under various configurations. Based on the collected data, we use neural networks to learn and predict the behavior of the overlay in Sec. V. Evaluation in Sec. VI shows that the configurations are well learned and can be reproduced. In Sec. VII, we conclude that using our approach for overlay calibration, p2p overlays are enabled to learn their own behavior and could potentially choose optimal configurations.

II. Related Work

Several aspects of the MAPE cycle have been discussed in literature. The MAPE cycle has been introduced in [2] by IBM. There, it is stated that the only remaining chance for overcoming the burden of complexity will be self-managing autonomic systems. For the monitoring of p2p systems, aggregation protocols [7] have been proposed. These protocols take local measurements of individual nodes and create statistics for the whole p2p system. The main classes of aggregation protocols are gossiping and tree-based approaches. Tree-based monitoring approaches create additional overlays on top of the structured p2p overlay to gather, aggregate and disseminate monitoring information through these. Examples for this approach are SDIMS [8], SkyEye.KOM [4] or SOMO [9]. While trees are efficient and create statistics on the network in $O(\log N)$ for a single measurements with N nodes in the network, the tree-based approaches require structured p2p overlays to operate. Furthermore, they are affected by churn.

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901)

978-1-4799-0521-8/13/$31.00 ©2013 IEEE

Gossip-based approaches can be used in any (connected) graph, i.e. they do not state further requirements on the overlay type. Through simple message exchanges with neighboring nodes, they allow to exchange local estimations on the statistics and converge to the global values with low variance. [10] describes the fundamental basis of the anti-entropy calculation. Gossip-based approaches are [3], [11] or [12].

While monitoring only resembles the capturing of the current network status, the goal-oriented adaptation of the p2p systems is in our focus. For p2p-based video streaming [13] and p2p-based VoIP [14], quality control has been discussed. While the adaptation of streaming settings on a single node directly affects the observed streaming quality on that node, in overlays a large number of nodes needs to adapt their configuration so that the quality, e.g. of lookups, is changed.

P2P overlays from the early days, such as Chord or Kademlia, are all self-organizing, but they do not adapt their configuration to changed environmental parameters. In [5] we use the tree-structure of SkyEye.KOM to gather statistics on the p2p overlay, analyze them in the root of the tree and push new configurations for the p2p overlay in order to control the hop count in the network. While we presented in [5] that the MAPE cycle can be closed, the analyze and plan phase used manual rules. For a fully autonomous system, the analyze and plan phase also need to be autonomic in the p2p network.

In this paper, we focus on the interpretation of monitored overlay behavior through the learning of parameter-metric interdependencies. We are able to identify configurations from the learned behavior of the overlay in simulations with various configurations when a desired quality goal is set.

III. Overview on our Calibration Approach

In order to introduce our learning and calibration approach, we briefly introduce neural networks. We use these networks to learn the effects of configurations on quality metrics. Artificial neural networks are mostly used for supervised machine learning, i.e. they need labeled data consisting of input and output data. With this labeled data it is possible to train a neural network such that it can abstract the data and predict unknown outputs if only the inputs are given (cf. [16], [17]).

Our goal is to construct neural network regressors via machine learning which can calculate a value for each overlay configuration parameter given a preferred p2p system state, described in terms of quality metrics, and the current state of the overlay network. To do so, we create a large training set through simulations. In these simulations we systematically vary several configuration parameters in order to capture a broad range of possible quality states for learning.

For our system model, we consider overlay parameters O (changeable by single nodes, see Table I) and environmental parameters E (unmodifiable by single nodes, see Table II) which affect the performance of the overlay, described in metrics M (see Table III). For the proof-of-concept, we use Chord in the evaluations, which was modified to enable the change of parameters at runtime. Any overlay providing this ability, which can be easily integrated into overlays like Pastry or Kademlia, and using a monitoring component, can be calibrated with our approach.

The simulated data that we create for our learning basis consists of a lot of data points of a function $f : O \times E \rightarrow M$ which maps overlay and environment parameters (O, E) onto metrics (M). We then rearrange the data to a function $\hat{f} : E \times M \rightarrow O$. Thus, the function $\hat{f}$ computes values for the overlay configuration parameters, in the case that the current values of the environment parameters are handed in and desired quality metrics are defined. Using a distributed monitoring approach and the function $\hat{f}$ allow to implement the MAPE cycle for p2p overlays. To approximate $\hat{f}$ we use Feed-Forward Neural Networks, one neural network for each overlay parameter. We found that using one neural network for predicting each overlay parameter separately results in lower error rates than using one neural network for predicting all parameters at once.

IV. Data Generation

In this section, we present how we create valid tuples of the function f for the learning data set by using thorough simulations. In Sec. V, we then discuss how the function $\hat{f}$ is learned based on this data set. For the data generation we use the p2p system simulator PeerfactSim.KOM [18], [19]. As the main overlay we use Chord, which is modified to allow the adaptation of configuration parameters at runtime. The values of the metrics are measured via the simulator itself. In real networks this can be done by a monitoring approach, e.g. SkyEye.KOM [4]. At the end of each simulation run, we get tuples of the form $(m_1, ..., m_r, e_1, ..., e_s, o_1, ..., o_t)$, with $m_i \in M$, $e_j \in E$ and $o_k \in O$.

To compute a suitable approximation, one needs to vary the values of each parameter and run a simulation with this modified setting (called full factorial design). For computing all possible combinations of parameter values for parameters p_i, the total number of combinations is $\prod_{i=1}^{n} |p_i|$. Having 13 different parameters with a few of four variations each, we get $4^{13} = 67{,}108{,}864$ different combinations. It is infeasible to simulate this many configurations nowadays.

To decrease the number of different combinations we build up all possible combinations only for the environment parameters. For every such combination of environment parameters we vary every overlay parameter only once while setting all other overlay parameters to predefined default values. We call this the mixed factorial design. Thus, the number of all combinations with environment parameters e_j and overlay parameters o_k is $\prod_{j=1}^{s} |e_j| \cdot \sum_{k=1}^{t} |o_k|$. The default values for this mixed factorial design are shown in the second column of Table IV in bold print. With this adjustment, we end up with $2{,}100 \cdot 31 = 65{,}100$ combinations. This number is simulatable and seems to be a good compromise.

Furthermore, we apply a basic setup and vary just one overlay parameter more granular (called one factorial design). The values for the variations of the overlay parameters are shown in the third column of Table IV. All parameters are set to default values (bold print) and just the overlay parameter we

TABLE I
CONFIGURATION PARAMETERS

Code	Overlay Parameter	Unit	default
o1	Message Timeout	s	10
o2	Message Resend	#	3
o3	Operation Timeout	s	120
o4	Operation Max. Redos	#	3
o5	Max Hop Count	#	50
o6	Upd. Finger Table Intv.	ms	30
o7	Upd. Neighbors Intv.	ms	30
o8	Upd. Successor Intv.	ms	30

TABLE II
ENVIRONMENTAL PARAMETERS

Code	Env. Parameter	Unit	default
e_1	Node Count	#	1000
e_2	Churn Factor	#	0
e_3	Mean Session Length	s	∞
e_4	Bandwidth	MB/s	OECD
e_5	Random Lookup Rate	1/h	30

TABLE III
METRICS

Code	Metrics	Unit
Messages		
m_1	Avg. Network Message In	#
m_2	Avg. Network Message Out	#
m_3	Avg. Transport Message In	#
m_4	Avg. Transport Message Out	#
m_5	Avg. Forwarded Queries	#
m_6	Avg. Service Message Throughput	#/s
m_7	St. Dev. Service Message Throughput	#/s
m_8	Avg. Service Message Count	#
m_9	St. Dev. Service Message Count	#
Traffic		
m_{10}	Avg. Network Bytes Sent	kB
m_{11}	Avg. Transport Bytes Sent	kB
m_{12}	Avg. Free Upload Bandwidth	kB/s
Performance		
m_{13}	Avg. Hop Count	#
m_{14}	Avg. Lookup Hops	#
m_{15}	St. Dev. Lookup Hops	#
m_{16}	Avg. Lookup Duration	s
m_{17}	St. Dev. Lookup Duration	s
m_{18}	Avg. Operation Duration	s

TABLE IV
PARAMETER VARIATIONS

Code	Mixed Factorial	One Factorial
o_1	5, **10**, 20	2,3,4,5,8,**10**,12,15,18,20
o_2	0, 1, **3**, 10	0,1,2,**3**,4,5,7,8,9
o_3	60, **120**, 300	60,90,**120**,150,180, 210,240,270,285,300
o_4	0, 1, **3**, 10	0,1,2,**3**,4,5,7,8,9,10
o_5	5, 10, 25, **50**, 100	3,5,7,10,17,35,**50**,75,100
o_6	3, 10, **30**, 60	3,5,7,10,15,20,**30**,40,50,60
o_7	3, 10, **30**, 60	3,5,7,10,15,20,**30**,40,50,60
o_8	3, 10, **30**, 60	3,5,7,10,15,20,**30**,40,50,60
e_1	10, 33, 100, 330, 1000, 3300, 10000	1000
e_2	0, $\frac{1}{10}$, $\frac{3}{10}$	0
e_3	30, 60; 180, ∞	∞
e_4	OECD, random: 1-2 5-10, 10-30, 1-30	OECD
e_5	0, 1, 3, 6, 30	30

want to predict is varied. The environment parameters are not varied. We apply ten variations to each of the eight parameters in O, so we end up with $10 \cdot 8 = 80$ combinations. Every combination is simulated five times, each with a different random seed and each tuple from each seed is used to train the neural network. Thus, it can happen that the neural network receives different metric values for the same parameter setting. Having this data set with a large number of configurations varied, we apply our neural network-based learning approach to learn how to set configurations in order to reach desired quality levels in p2p overlays.

V. DATA PREDICTION

In this section, we approximate $\hat{f} : E \times M \rightarrow O$. For every overlay parameter $o_i \in O$ we construct a neural network which takes the environment parameters E and the metrics M as input and has one overlay parameter as output, so we get $|O|$ different neural networks. To train a neural network n_i for o_i we ignore all overlay parameters in the tuples of generated data except o_i, so for the training of each neural network n_i we get $(m_1, \ldots, m_r, e_1, \ldots, e_s, o_i)$.

We also apply feature selection algorithms before constructing a neural network to reduce the number of input attributes presented to the neural network. Feature selection algorithms try to find attributes which are redundant or can be omitted without a big loss of information. We use Correlation-based Feature Selection [20] and the Principal Component Analysis [21], which are part of WEKA [22]. To construct and train the neural networks we use the encog framework [23], especially the Feed-Forward Neural Network implementation with resilient backpropagation.

The abstract structure of the neural networks is as follows: The input layer consists of input neurons for every metric and every environment parameter. The hidden layer(s) may contain arbitrarily many neurons and the output layer consists of only one neuron, the overlay parameter we want to predict. Thus, we need one neural network of this structure for every chord parameter. In our evaluation we use one or two hidden layers with up to 100 neurons and make encog choose a good configuration. Depending on whether we apply a feature selection algorithm and which features are found important, the number of metrics and environment parameters in the input layer may vary. After training the neural network, the number of input neurons cannot be changed and only the attributes used while learning are presented to the neural network. Using feature selection algorithms reduces the number of necessary attributes while maintaining the prediction quality, thus leading to less computational overhead. Neuronal networks can only predict parameters for reachable metric constellations, which should be defined by experts.

The learning process is structured as follows: 1) Split the labeled data into 70% training data (T), 20% validation data (V) and 10% prediction data (P). 2) Train the neural network with T. 3) Compute the error on V. 4) If stopping criterion is not met, go to 2. 5) Compute the error on P.

As stopping criterion we use any of the following conditions: a) Perform at most 10,000 training steps. b) The error rate on V did not change significantly or got worse in the last 30 training steps. c) The error rate is low enough (< 0.01).

The split of the data in Step 1 is necessary as neural networks tend to overfit. Stopping criterion b) prevents overfitting, as this indicates that the neural network is starting to learn the training data by heart and loses the ability to abstract the data characteristics. At the end we also want to know the error rate on data that did not influence the training phase, i.e. not on V. Therefore we compute the error on P in Step 5.

The learning process is done for every overlay parameter with Correlation-based Feature Selection (*cfs*), Principal Component Analysis (*pca*) and without feature selection (*no fs*). Furthermore, we repeat this process with raw, normalized and standardized data. The results for normalized data perform best, the results of the others are therefore left out.

First, we evaluate our approach on the whole data set with 65,100 combinations (cf. Sec. IV). Unfortunately, the mixed factorial approach leads to results that are biased towards the default values. Since we vary only one overlay parameter

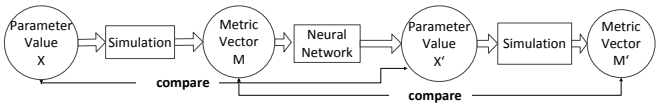


Fig. 1. Steps of Evaluation

while leaving the other overlay parameters set to default values, these default values occur more often than the non-default values. Recall that a tuple for training the neural network n_i to predict the overlay parameter o_i looks like $(m_1, \ldots, m_r, e_1, \ldots, e_s, o_i)$. In the majority of such tuples, the value of o_i is the default value for o_i. The best approximation the neural network finds is to always predict the default value. With this behavior, the overall error rate is very low but the error rate for a non-default value is very high. This is not intended, but a reasonable trade-off between the number of combinations and the accuracy of the results.

VI. Evaluation

In the following we present the results of the one factorial setup. We use the Root-Mean-Squared error to measure the quality of our approach in learning the interdependencies of parameters and metrics, as well as the quality of $\hat{f}$. For the latter, we measure the difference of the learned and the predicted parameter setting for a desired quality metric tuple.

First, we investigate the error rates on the validation data set and prediction data set. Fig. 2 shows the error rate on the validation data set V for normalized data with and without feature selection. We observe that the Principal Component Analysis (*pca*) is performing worse for o_1, o_6 and o_8 in comparison to Correlation-based Feature Selection (*cfs*) and no feature selection (*no fs*). In average its error rates are higher. The approaches *cfs* and *no fs* perform equally for most parameters, while *no fs* is slightly better except for o_2. Fig. 3 shows the error rate on the prediction data set P for normalized data with and without feature selection. We observe even larger error rates with the *pca* compared to Fig. 2. Errors larger than 100% are reached as the neural network is used for regression. Both, the *cfs* and *no fs* approaches show good results.

Next, we focus on the prediction of overlay parameter values for a given p2p overlay metric. We display our steps of evaluation in Fig. 1: First, we choose a parameter p and a value X for this parameter. All other parameters are set to predefined default values. With this parameter setting we start a simulation and get a value for every metric, denoted by a metric vector M. The metric vector M represents the desired state for the p2p network. We then use our trained neural network to predict a value X' for overlay parameter o based on the M. The value X' describes the parameter configuration for o, which should result in the network quality denoted by M. In order to check the quality of X', we simulate the p2p overlay with X' and all other parameter values as default again and get another metric vector M' which is compared to M. The quality of the learning approach is then judged on the difference of X and X', as well as M and M'. In the following one metric value is shown for each overlay parameter, selected from those metrics which change if we adapt the overlay parameter.

TABLE V
Comparison of X, X', M and M'

Timestamp	$\mathbf{X}$	$\mathbf{X'}$	$\mathbf{M}$	$\mathbf{M'}$	$\lvert\mathbf{1}-\frac{\mathbf{M}}{\mathbf{M'}}\rvert$	$\lvert\mathbf{1}-\frac{\mathbf{X}}{\mathbf{X'}}\rvert$
For o_1 and m_{16}						
w_1	6s	6s	0.14	0.15	0.07	0.00
w_2	11s	11s	0.18	0.20	0.10	0.00
w_3	19s	18s	0.24	0.27	0.11	0.06
For o_6 and m_2						
w_1	4s	4s	175.00	173.66	0.01	0.00
w_2	25s	26s	39.60	38.34	0.03	0.04
w_3	55s	54s	24.50	25.51	0.04	0.02
For o_7 and m_2						
w_1	6s	4.6s	65.80	77.27	0.15	0.30
w_2	25s	27.5s	36.60	35.98	0.02	0.09
w_3	51s	55s	31.70	31.63	0.00	0.07
For o_8 and m_2						
w_1	4s	3.9s	60.21	60.90	0.01	0.03
w_2	37s	36s	34.27	34.10	0.00	0.03
w_3	55s	52s	33.21	33.00	0.01	0.06

We concentrate on parameters which had acceptable error rates for the test and validation set without feature selection. Thus, we select *message timeout* (o_1), *update fingertable interval* (o_6), *update neighbors interval* (o_7) and *update successor interval* (o_8). The corresponding error rates for the test and validation set are always below 20%. Other parameters had higher error rates, due to low interdependencies or an environmental setting in which effects are rare. For every parameter we measure the metric three times in a time gap of ten minutes after each measurement to make sure that the metric values do not change significantly after a change of the parameter value. In the following we always present nine measurements with three measurements belonging to one parameter change. The measurements belonging to the same parameter change are labeled w_i. We average the three measurements and compare the exact values for every investigated parameter. The predicted metric value is marked as "$\times$" while the optimal value is marked as "+". We compare the optimal and predicted parameter values and also the corresponding metric values, aiming at a low difference.

A. Results

Table V shows the prediction of three different parameter values for the parameters *message timeout* (o_1), *update fingertable interval* (o_6), *update neighbors interval* (o_7) and *update successor interval* (o_8) and the corresponding values of the metrics *average lookup duration* (m_{16}) for o_1 and *average network messages out* (m_2) for o_6, o_7 and o_8. Figs. 4 to 7 show the corresponding plots.

The error rate of the planned and reached metric quality, $|1-\frac{M}{M'}|$, as well as the difference between X and X', $|1-\frac{X}{X'}|$, is in most cases below 15%, thus acceptable. For o_1 the values of the metric m_{16} change slightly over time due to network fluctuations, resulting in an error rate around 10%. For o_7 and m_2, in the first measurement series w_1 the error rate is 15%. For small values of o_7 the influence on m_2 might be large, for increasing o_7 the influence might vanish. This effect is supported by the prediction quality in w_2 and w_3, as well as the large error $|1-\frac{X}{X'}|$ for w_1. While the result is quite good, the neural network could do better.

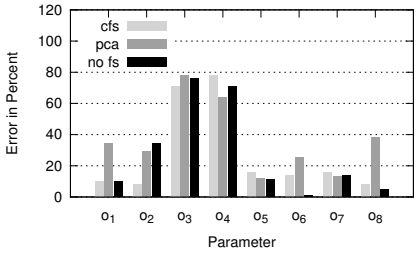


Fig. 2. Prediction Error on Validation Data

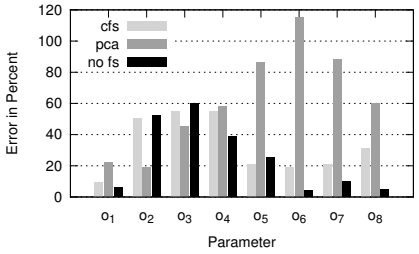


Fig. 3. Prediction Error on Prediction Data

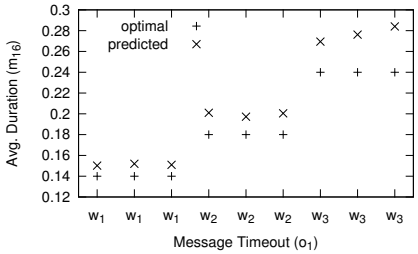


Fig. 4. Prediction of o_1: Message Timeout

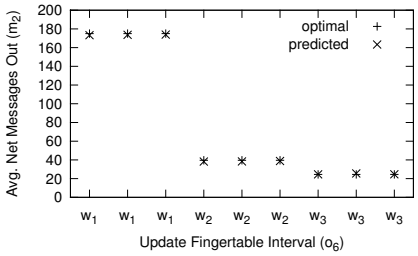


Fig. 5. Prediction of o_6: Update Fingers Interval

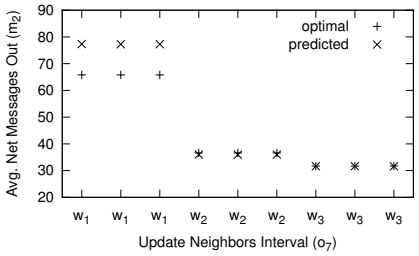


Fig. 6. Prediction of o_7: Update Neighors Interval

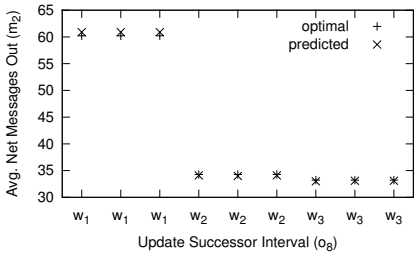


Fig. 7. Prediction of o_8: Update Successor Interval

Thus, our approach demonstrates that for a given metric tuple, a configuration parameter is found, which when applied, results in the desired quality metric. The difference between the desired metric value and the reached metric value is very small. Using the presented learning approach, one can calibrate the p2p overlays offline or online, identify quality ranges and reach desired quality levels.

VII. Conclusions

Distributed control loops in p2p overlay networks allow to operate overlays that are able to monitor their performance statistics, analyze these statistics and adapt through reconfiguration in order to reach and hold desired quality metrics. In this paper we address the issue on how to set the configuration parameters in a p2p overlay at runtime, if a desired performance state is not yet reached. For that we systematically simulate the configuration parameters of Chord and create a data set, which is then used to learn the interdependencies between parameters and quality metrics using feed-forward neural networks with resilient backpropagation. The neural networks learn which configuration parameters result in which quality metrics. Evaluation shows in several proof-of-concepts, that for a given reachable quality metric a configuration parameter is found which can be applied in the p2p overlay. Applying this configuration parameter leads to the desired quality. On the long term, we envision a p2p overlay that is able to learn how to reach any valid quality level and, by analyzing the current needs of the users, to also set the quality goals accordingly.

References

[1] N. Liebau, K. Pussep, K. Graffi, S. Kaune, E. Jahn, A. Beyer, et al., "The Impact Of The P2P Paradigm," in *Proc. of AMCIS*, 2007.

[2] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," http://www.research.ibm.com/autonomic/, 2001.

[3] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregate Information," in *Proc. of IEEE FOCS*, 2003.

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

[5] K. Graffi, D. Stingl, J. Rueckert, et al., "Monitoring and Management of Structured Peer-to-Peer Systems," in *Proc. of IEEE P2P*, 2009.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. of ACM SIGCOMM*, 2001.

[7] R. van Renesse, K. P. Birman, and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining," *ACM Trans. on Comp. Sys.*, vol. 21, no. 2, 2003.

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

[9] Z. Zhang, S. Shi, and J. Zhu, "SOMO: Self-Organized Metadata Overlay for Resource Management in P2P DHT," in *Proc. of IPTPS*, 2003.

[10] M. Jelasity and A. Montresor, "Epidemic-Style Proactive Aggregation in Large Overlay Networks," in *Proc. of IEEE ICDCS*, 2004.

[11] M. Jelasity, A. Montresor, and Ö. Babaoglu, "Gossip-based Aggregation in Large Dynamic Networks," *ACM Trans. on Comp. Sys.*, vol. 23, 2005.

[12] V. Rapp and K. Graffi, "Continuous Gossip-based Aggregation Through Dynamic Information Aging," in *Proc. of IEEE ICCCN*, 2013

[13] M. Ryu and U. Ramachandran, "DynaStream: Adaptive Overlay Management for Peer-to-Peer Video Streaming," in *Proc. of ICCCN*, 2010.

[14] X. Liao, F. Guo, and H. Jin, "Service Quality Assurance Mechanisms for P2P SIP VoIP," in *Proc. of IFIP NPC*, 2011.

[15] S. Haykin, *Neural networks and learning machines.* vol. 3, New York: Prentice Hall, 2009.

[16] G. Cybenko, "Approximation by Superposition of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, vol. 2, 1989.

[17] M. Riedmiller, et al., "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *Proc. of IEEE ICNN*, 1993.

[18] A. Kovacevic, S. Kaune, H. Heckel, A. Mink, K. Graffi, O. Heckmann, and R. Steinmetz, "PeerfactSim.KOM - A Simulator for Large-Scale Peer-to-Peer Networks," TU Darmstadt, Tech. Rep. Tr-2006-06, 2006.

[19] K. Graffi, "PeerfactSim.KOM: A P2P System Simulator – Experiences and Lessons Learned," in *Proc. of IEEE P2P*, 2011.

[20] M. A. Hall and L. A. Smith, "Feature Selection for Machine Learning: Comparing a Correlation-Based Filter Approach to the Wrapper," in *Proc. of AAAI FLAIRS'99*, 1999.

[21] K. Pearson, "On Lines and Planes of Closest Fit to Systems of Points in Space," *Philosophical Magazine*, vol. 2, no. 6, 1901.

[22] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[23] Heaton Research, "Encog Machine Learning Framework." [Online]. Available: https://github.com/encog