Havelaar: A Robust and Efficient Reputation System for Active Peer-to-Peer Systems

Dominik Grolimund, Luzius Meisser, Stefan Schmid, Roger Wattenhofer
{grolimund@inf., meisserl@, schmiste@tik.ee., wattenhofer@tik.ee.}ethz.ch
Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 8092 Zurich, Switzerland

***Abstract*— Peer-to-peer (p2p) systems have the potential to harness huge amounts of resources. Unfortunately, however, it has been shown that most of today's p2p networks suffer from a large fraction of free-riders, which mostly consume resources without contributing much to the system themselves. This results in an overall performance degradation. One particularly interesting resource is bandwidth. Thereby, a service differentiation approach seems appropriate, where peers contributing higher upload bandwidth are rewarded with higher download bandwidth in return. Keeping track of the contribution of each peer in an open, decentralized environment, however, is not trivial; many systems which have been proposed are susceptible to false reports. Besides being prone to attacks, some solutions have a large communication and computation overhead, which can even be linear in the number of transactions—an unacceptable burden in practical and active systems. In this paper, we propose a reputation system which overcomes this scaling problem. Our analytical and simulation results are promising, indicating that the mechanism is accurate and efficient, especially when applied to systems where there are lots of transactions (e.g., due to erasure coding).**

I. Introduction

The power of *peer-to-peer (p2p) computing* is based on the resource contribution of the network's constituent parts, the peers. Therefore, the success of a system in practice crucially depends on its ability to cope with selfish peers which aim at only using resources without contributing anything themselves. One resource of special interest is *bandwidth*. Thereby, a *service differentiation* approach seem appropriate: peers providing higher upload bandwidth (and for longer time periods!) should be rewarded with higher download bandwidths in return.

When faced with the task of implementing a fairness scheme for our distributed p2p storage system research project *Kangoo*[1], we could not find a solution which perfectly fits our needs. In Kangoo, *erasure codes* are employed to achieve high data availability with moderate redundancy. Files are divided into blocks, which are further divided and recoded into redundant fragments. Therefore, lots of transactions are necessary to reconstruct a file.

In this paper, we present the reputation system *Havelaar*, which we have developed and implemented for Kangoo. Unlike many existing solutions, Havelaar does not rely on *transitivity of trust*, and achieves a *high robustness to attacks by design*. This is achieved by a novel technique in which a peer u always reports directly or indirectly observed contributions to the same set of peers (the so-called *successors* of peer u). These successor peers are determined a set of hash functions $h(u)$ on the identifier (e.g., the IP-address) of u. Thus, a successor is able to detect and defend against any abuse (e.g., reporting too large values, reporting too often, etc.). This is different from so-called *distributed hash table (DHT) approaches* where a peer u benefitting from a peer v reports v's contribution value to peers determined by a hash function $h(v)$ (depending on v rather than u).

Our results are promising: Havelaar is not only robust to attacks, but also efficient and—unlike many other solutions—scales well in the number of transactions. Hence, we believe that Havelaar is well-suited for other active p2p systems with many transactions (e.g., due to erasure coding).

The rest of this paper is organized as follows. In Section II, related work is reviewed. Section III describes how Havelaar rewards peers given their uploading reputations. The reputation system is then presented in detail in Section IV. In Section V, we analyze how many observations a peer has to accumulate in order to achieve an accurate approximation of the real contributions. Means to reduce the message sizes in the Havelaar system are studied in Section VI. We evaluate the communication complexity by simulation and compare it to a benchmark system in Section VII. Section VIII considers some classic attacks on fairness systems and shows how Havelaar copes with these attacks. The accuracy of Havelaar's reputation values is studied by simulation in Section IX. Finally, Section X concludes the paper.

II. Related Work

It is not hard to find evidence of selfish behavior in existing p2p systems [2], [11]. The problem has already spurred a large body of research, and the field is still very active [7], [10], [14], [18], [24].

Perhaps the simplest approach of a fairness mechanism is to directly incorporate contribution monitoring into the client software itself. For example, in the popular file-sharing system *Kazaa*, the client records the contribution of its user. However, such a solution can simply be bypassed by implementing a different client, which hard-wires the contribution level of the user to the maximum, as has been demonstrated by *Kazaa Lite*.

In systems such as *BitTorrent* [6] in which peers upload to the same peers from which they also download, a simple *tit-*

[1]To be released in autumn 2006.

for-tat mechanism [3] is feasible. When interactions between the same pairs of peers are less frequent, however, such barter systems [28] fail.

Inspired by real economies, some researchers have also proposed to use some form of virtual money for each transaction. However, these *monetary or credit based approaches* typically have a substantial overhead in terms of communication costs and infrastructure, and hence are inefficient [9], [29]. Often these systems also require market regulation mechanisms [27] to cope with inflation or deflation—a complex issue. Additionally, monetary based systems may provide disincentives to the users to participate [19].

If a peer has too few own direct observations to judge the contribution of some other peer, it has to take into account indirect observations of other peers [13]. Such systems are generally called *reputation systems* (sometimes also *reciprocity based* systems). They are well-known from auctioning applications such as eBay. However, second-hand observations introduce the problem of *false reports* [4], [13]. Many proposals have been made to mitigate these effects [1], [8], [12]), most of them relying on *trust-transitivity*, where observations are weighted by the reputation of the reporter. However, these systems can be exploited by peers with good reputations.

Additionally to the problem of false reports, also an infrastructure to exchange the second-hand observations is needed [13]. In most most reputation-based systems, second-hand observations are either requested before a transaction from other peers [1], [4], or they are simply flooded through the system. Alternatively, the reputation values can be stored in a *distributed hash table* (DHT) [21], [23], [26]. For systems with lots of transactions, where contribution values are updated constantly, however, this results in an unacceptable communication overhead: checking a peer's reputation entails costly (wide-area) DHT lookups [17], [20].

In contrast, in the Havelaar reputation system, a peer is able to compute the reputation of a requesting peer *locally*. This is particularly beneficial in systems where there are lots of transactions. By aggregating the contribution values of a large number of peers, our system tackles also the problem of trust-transitivity. Finally, Havelaar is able to check the credibility of *reports* locally, thus preventing many reputation attacks by design.

III. Rewarding Mechanism

Havelaar is mainly a *reputation system* (cf Section IV) and therefore independent of any concrete rewarding mechanism. That is, given the reputation values of Havelaar, many strategies can be applied to allocate bandwidth to the peers.

However, to complete the picture, we briefly sketch the approach we have chosen for our distributed storage system Kangoo. In Kangoo, we make use of the mechanism proposed in described in [15]. However, as performance is crucial in Kangoo, we only apply fairness mechanisms in situations of *contention*, i.e., when several peers want to download from the same node concurrently. Assuming that bandwidth is free and lost when not used, the maximum possible bandwidth will always be allocated to a requesting peer, and no artificial limits are used. For our system, this is the desired behavior because we do not want to provide any disincentives to participate and download in the network, as there would be with monetary-based systems. We only limit the resource allocation for excessive downloaders, as will be described in Section IV.

IV. Reputation Mechanism

In this section, we describe the main ideas behind Havelaar. Basically, Havelaar has three goals: (1) robustness against selfish peers, (2) accurate estimation of the real contribution values of other peers, and (3) efficiency.

In our system, the reputation of a peer u should reflect on the peer's contribution C_u, which in turn depends on the bandwidth b_u it provides, *and* the size s of the corresponding fragments. Hence, the total contribution value is given by $C_u = \sum_{\forall \text{ transactions t}} (b_u(t) \cdot s_u(t))$. Note that the contribution value will only be increased *after a complete upload* in order to reward proper transactions only.

So how does Havelaar track this contributions? Each peer u maintains a vector $\vec{o}$ (observations) of size n (number of peers in the network), where it stores the contributions of other peers which it has directly experienced itself. That is, after each download, u updates its $\vec{o}$ vector accordingly.

Even in active systems with lots of transactions and erasure coding, a peer typically only gets in touch with a subset of all peers. Therefore, the private observations are sent to other peers once in a while. Basically, to achieve this, Havelaar employs a *round-based aggregation technique*. Thereby, once in a round, each peer u sends its observation vector $\vec{o}$ to a small number k (e.g., 7) of other peers in the system, called u's *successors* (similarly, we will refer to u as a *predecessor* of such a successor peer). The successor peers are determined by a set of k hash functions $h(u)$ on u's identifier.[2] In the following, we will assume that a *round* is roughly one week. Moreover, note that it does not matter, when exactly in this time interval a peer sends its reports to the successors, and hence, churn is not an issue here.

Observe that by this scheme, a peer u always informs the same set of peers, *independently* of which peers contributed resources to u; hence, upon receiving a vector, a peer can check whether it has been sent by a correct predecessor by verifying the hash function[3]—otherwise, it can simply drop the vector. Note that the "observed" contribution value of the successor is not taken into account, so that the attack with the biggest incentive is simply made impossible by design. What is more, each peer can also ignore the vector of a peer that sends too frequently (more than once in a round). If

[2]Due to the hash function, some peers will have slightly more, others slightly less predecessors.

[3]To verify the predecessor identifier, public/private-key pairs among peers are presumed.

an observation vector is susceptible to be a false report (cf Section VIII), it is not taken into account either. Finally, "extremely large contribution values" are culled as well, due to the danger of false praise. Thus, already by this simple mechanism, the potential influence of an attacker is limited.

Unfortunately, sending direct observations to the successors is still not enough in order to accurately estimate the contributions of all peers. Therefore, we extend the mechanism as follows: Upon receiving the observation vectors from its k predecessors, each peer aggregates them with its own observations, and sends the new vector to its k successors. Thus, one vector can summarize a large number of observations. However, we have to make sure that the values in the vectors do not increase forever, and do not contain too many observations from the past which do not reflect the current behavior of each peer. What is needed is a scheme where old observations can be truncated in the observation vector, while still containing enough observations to update the contribution vector after each round.

Therefore, the Havelaar finally works as follows. In every round, a peer puts its own observations into a vector $\vec{o}_0$ (so far simply called $\vec{o}$). After each round, it sends a message to its k successors containing its own (direct) observations from this round ($\vec{o}_0$), the aggregated observations of its k predecessors from the last round $\vec{o}_1$, the aggregated observations of the k predecessors of its own k predecessors $\vec{o}_2$ from the round before the last round, and so forth. The message thus contains an $n \times r$-matrix $O := [\vec{o}_0, \ldots, \vec{o}_{r-1}]$. Upon receiving the matrix $O_i := [\vec{o}_1, \ldots, \vec{o}_r]$ from predecessor $i \in [1, k]$ (note that when sending, the index runs from $0 \ldots r-1$, and when receiving, it is renamed to $1 \ldots r$), a peer aggregates all observations and updates its contribution vector $\vec{c}$ accordingly. Observe that the vectors from previous rounds aggregate an exponentially growing number of observations. Furthermore, also observe that at any moment, the oldest observations lie r rounds in the past.

A simplified description of the Havelaar reputation system is given in Algorithm 1. The algorithm extends the method described so far by an *aging factor* $\gamma \leq 1$. However, note that since the aggregation vectors already include many observations from the past, using $\gamma = 0$ is fine for our purpose, but can be increased if longer absences from the system should not result in a complete loss of the previous contribution value.

Algorithm 1 Simplified Havelaar Reputation System

1: **observe** $\vec{o}_0$;
2: **receive** $O_1 := [\vec{o_1}, \ldots, \vec{o_r}], \ldots, O_k$ **from** predecessors;
3: **for** $j \in [1, r]$: $\vec{o}_j = \sum_{i=1}^{k} O_{ij}$;
4: $\vec{c} = \gamma\vec{c} + (1-\gamma)(\sum_{i=0}^{r} \vec{o}_i)$;
5: **send** $O := [\vec{o_0}, \ldots, \vec{o_{r-1}}]$ **to** k successors;

In our system, a peer u increases the contribution values only after downloading fragments from other peers, but it never decreases any contribution values if it has to provide upload bandwidth to some peer. This has the drawback that if two peers have contributed to the system equally, they will be allocated the same amount of download bandwidth, independently of their downloading behavior—it is questionable whether this is fair. However, as mentioned, we do not want to provide any disincentives for downloading in our network, so that this is acceptable.

However, the behavior of *excessive downloads* should be discouraged. Such downloads could have the disadvantage of a vicious circle: because of excessive downloads, the network is congested, which in turn encourages other users to download in advance, resulting in an even more congested network. Therefore, in our system, each peer u additionally maintains a second vector $\vec{d}$ (downloads). After another peer downloads from u, u will increase the download value of that peer. As opposed to the observation vector, the download vector will not be sent around. However, before allocating resources among competing peers, the download values will be subtracted from the respective contribution values from $\vec{c}$, and only then used to allocate the bandwidth. For excessive downloaders, repeated interactions are more probable, and hence excessive downloaders are eventually slowed down. Note that the download values need to age. However, one could also treat downloads differently, for instance, not at all, or by a similar mechanism as we use it for uploads.

V. Analysis

A. Overview

Assume that two peers u and v compete for the same upload bandwidth of a given peer w. In order to achieve the desired fairness, peer w should allocate the bandwidth to u and v according to their (global) contribution values C_u^g and C_v^g, respectively, i.e., with respect to all transactions to which they have contributed. However, in Havelaar, peer w does not have precise information about C_u^g and C_v^g, but only knows the local approximations C_u^l and C_v^l (values from its observation vector $\vec{c}$). Hence, if $\frac{C_u^g}{C_v^g}$ is five, also $\frac{C_u^l}{C_v^l}$ should be roughly five, such that peer u indeed receives approximately five times more bandwidth than peer v.

In this section, we analyze how many observations x are needed in order that the values in the local vectors $\vec{c}$ are an acceptable approximation of the global contributions. Consequently, we can compute the number of rounds r that are necessary in Havelaar to achieve the required number of (aggregated) observations. In the following, we use C_u and C_v to denote C_u^l and C_v^l, respectively.

Our network consists of n peers, not all of which are always online. We simplify the analysis by assuming that at any time exactly $m < n$ peers are online. Furthermore, we assume that each peer downloads t fragments from other peers, which are chosen uniformly at random among all peers. Hence, when downloading a fragment, the probability that a given peer is chosen is simply given by $p = \frac{1}{m}$. Finally, assume that these transactions are distributed uniformly over time.

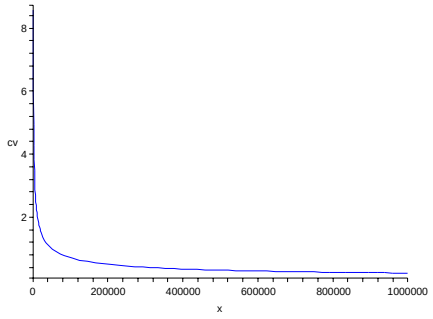

Fig. 1. The coefficient of variation converges to 0 for $x \to \infty$.

B. Basic Analysis

Since we aggregate all x observations of peer u in the variable C_u, it does not matter where these observations are made. For the basic analysis, we assume that bandwidth and fragment sizes are equal to 1, that is, the C_u is increased by 1 after each download from u. As explained before, peer u is chosen with probability p for every download. Therefore, C_u can be regarded as a random variable. What is the probability distribution of C_u after x downloads?

This situation corresponds to a *balls-into-bins problem* [16], where x balls are tossed into m bins, and where $p = \frac{1}{m}$ is the probability that a tossed ball lands in any given bin. For a concrete bin u, the ball tossing process can be viewed as a sequence of x random, independent *Bernoulli trials*, each with a probability p of success. Therefore, the random variable C_u follows a *binomial distribution* $C_u \sim Bin(x, p)$, where $\mu_{C_u} = E(C_u) = x \cdot p$ and $\sigma^2_{C_u} = Var(C_u) = x \cdot p \cdot (1-p)$.

For small x and large m, the *coefficient of variation* $\frac{\sigma_{C_u}}{\mu_{C_u}}$ is large. However, for $x \to \infty$, it quickly converges to 0, as shown in Figure 1. This indicates that for lots of transactions, relative estimates become accurate enough. However, estimating the actual contribution of u is a difficult task, since it depends on many factors. Instead, we are only interested in the ratio of the contribution values of two peers competing for resources at the same time. Therefore, let us introduce the random value Z, reflecting this ratio:

$$Z = \frac{C_u}{C_v}.$$

What is the expected value and the variance of Z? Since C_u and C_v are independent, the following approximations are reasonable [22]:

$$\mu_Z = E(Z) \approx \frac{\mu_{C_u}}{\mu_{C_v}} + \sigma^2_{C_v} \frac{\mu_{C_u}}{\mu^3_{C_v}} \tag{1}$$

$$\sigma^2_Z = Var(Z) \approx \sigma^2_{C_v} \frac{\mu^2_{C_u}}{\mu^4_{C_v}} + \frac{\sigma^2_{C_u}}{\mu^2_{C_u}} \tag{2}$$

Obviously, for $x \to \infty$, the variance converges to 0. Thus, for lots of observations x, Z will be a very good approximation to the ratio of the real contributions of peers u and v.

Our goal is to have an acceptably small coefficient of variation $\frac{\sigma_Z}{\mu_Z}$ (or, similarly, a small *variance-to-mean ratio* $\frac{\sigma^2_Z}{\mu_Z}$). In the following, we will make use of another helpful approximation of the coefficient of variation [25]:

$$\left(\frac{\sigma_Z}{\mu_Z}\right)^2 \approx \left(\frac{\sigma_{C_u}}{\mu_{C_u}}\right)^2 + \left(\frac{\sigma_{C_v}}{\mu_{C_v}}\right)^2 \tag{3}$$

However, before we conclude how many transactions are necessary, we want to extend our model such that it incorporates more aspects arising in reality.

C. Online Time

So far, we have assumed that peers u and v are online all the time and can thus be chosen for all x transactions. In reality, however, some peer u might be online much longer than some other peer v. Therefore, u is likely to be involved in more transactions and will hence also contribute more to the network. This should clearly be reflected in the local approximations C_u and C_v.

Let us assume that peer u is online with a fixed probability of p_u, and peer v with probability p_v. Based on the assumption that the transactions are distributed uniformly over time, peer u will only be chosen for $x_u = p_u x$ transactions in expectation, and peer v for $x_v = p_v x$ transactions. It is run of the mill to extend our random variable appropriately to $C_u \sim Bin(x_u, p)$, with $\mu_{C_u} = E(C_u) = x_u \cdot p = p_u \cdot x \cdot p$ and $\sigma^2_{C_u} = Var(C_u) = x_u \cdot p \cdot (1-p) = p_u \cdot x \cdot p \cdot (1-p)$, and similarly for C_v. Clearly, the mean contribution value is double for peers having double the online time of other peers.

D. Bandwidth and Fragment Size

The goal of the Havelaar system is to encourage high upload bandwidths; consequently, the provided bandwidth must be included in the model. Let us assume that peer u uploads fragments with a fixed bandwidth b_u, and peer v with bandwidth b_v. Instead of adding 1 to the contribution value of each peer, we add the respective bandwidth. This corresponds to the multiplication of a random variable with a constant. The expected value of C_u changes therefore to $\mu_{C_u} = E(C_u) = b_u \cdot p_u \cdot x \cdot p$, and the variance to $\sigma^2_{C_u} = Var(C_u) = b^2_u \cdot p_u \cdot x \cdot p \cdot (1-p)$. Note that the variance is now multiplied twice (b^2_u); thus, the variance in the contribution C_u does not increase linearly in the bandwidth, but quadratically.

We could also extend the model to variable fragment sizes. However, in order to keep things simple, we omit this generalization in our analysis; note that this would also influence the variance of the ratio Z as well.

E. Observations

With the complete model, we can now estimate the number of observations necessary for a good approximation, that is, for small coefficients of variation of Z. Plugging μ_{C_u}, σ_{C_u}, μ_{C_v}, and σ_{C_v} into Equation (3) yields

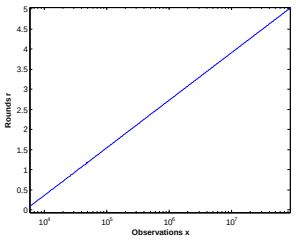

Fig. 2. Number of rounds r to aggregate x observations.

$$\left(\frac{\sigma_Z}{\mu_Z}\right)^2 \approx \left(\frac{\sqrt{b_u^2 p_u x(p-p^2)}}{b_u x p_u p}\right)^2 + \left(\frac{\sqrt{b_v^2 p_v x(p-p^2)}}{b_v x p_v p}\right)^2$$
$$= \frac{1-p}{x p_u p} + \frac{1-p}{x p_v p} \quad (4)$$

We can solve Equation (4) for x, and get the following fact.

Fact 5.1: Havelaar requires

$$x \approx -\frac{p_v(p-1)+p_u(p-1)}{\left(\frac{\sigma_Z}{\mu_Z}\right)^2 p_u p_v p}$$

observations in order to compute a good approximation of the global contribution values.

As an example, for a network of $n = 100,000$ peers, where at any time $m = \frac{1}{4}n = 25000$ peers are online and if we assume that $p_u = p_v = \frac{1}{4}$, $x \approx 10^7$ observations are required for an acceptable coefficient of variation of 0.1.

F. Rounds

Knowing how many observations x are approximately needed for an acceptable accuracy, we can now determine how many aggregation rounds are necessary. Assuming that every peer makes t transactions (= observations), the number of rounds r is simply given by $r = \log_k \frac{x}{t}$.

For the above example of $x = 10^7$ observations, assuming that each peer makes $t = 5000$ transactions and sends its observations to $k = 7$ peers, $r = 3.9 \approx 4$ rounds are needed. Since x depends on the size of the network n, more observations need to be aggregated as well. Because of the multiplicative increase of observations per round, this is certainly feasible, as shown in Figure 2. On the other hand, one can imagine that t is larger in larger networks, as more interesting fragments can be downloaded.

G. Contention

Z follows a *log-normal distribution* which does not contain negative values and is skewed to the left. Knowing the distribution of Z, we could define a 95%-confidence interval depending on t and compute the relative error. However, since our model is based on several simplifications, this would be overly exact. Instead, we are interested in a rough estimate of x in order to determine the number of rounds necessary of Algorithm 1.

Especially when taking into account another aspect of reality: *contention*. In times of contention, the fairness mechanism applies and bandwidth is allocated according to a mechanism similar to the one outlined in Section **??**. If so, the allocated upload bandwidth will be much smaller than the actually provided one, and hence the increase in the contribution value does not reflect this correctly.

However, assuming that contention is distributed uniformly, this aspect affects each download with equal probability, and with lots of observations, it will be smoothed out. But it certainly introduces another variance to the model, which ultimately influences the variance of Z. We can see from Equation (2) that a linear increase in the variance of the contribution values results in a quadratic increase of the ratio Z. However, since we can increase the aggregation size multiplicatively (factor k) by using just one more round, integrating that necessary amount of observations becomes feasible.

VI. Coding and Compression

The Achilles' heel of Havelaar is its message size: Every peer has to send—for example, once a week—the aggregated observations to its successor. Although we believe that this is a tolerable burden in many practical systems, some forms of compression are still very desirable. Therefore, in this section, we explore some coding and compression techniques that we can apply to reduce the size of the messages.

Contribution Values: So far, we have not said anything about the size of a contribution value. Of course, if we allow the contribution values only to be within a certain range of 2^l discrete (integer) values, using l bits per observation is enough. Of course, this introduces another variance, which is however small with respect to the analysis in Section V.

Moreover, contribution vectors from older rounds will have higher contribution values on average because they are aggregated from much more observations. This entails a great range of different contribution values. Therefore, it could be worth normalizing these values first. This is done by dividing each value by the sum of all values in the vector, or by dividing each value of vector i by k^i. Before aggregating, the values need to be multiplied by that factor of course. This clearly reduces the accuracy, especially for values from older rounds. However, for the purpose of the fairness system, this is acceptable, as this variance is smaller than the one due to the factors analyzed in Section V. However, the gain in the size reduction is huge.

Analyzing a histogram of all values shows that a further reduction is possible. Since not every value is equally likely, it would be worth encoding the values with different lengths. This could for instance be done using a *Huffman encoding*, using the relative frequency of each value, which would encode the values close to their entropy. Another, more generic approach is to fit a distribution to the histogram,

and use this distribution to encode the values, which renders it unnecessary to send the code alphabet. In the following, however, we will not take into account any possible reductions from encoding techniques, but instead assume that the contribution values can be encoded with $l = 8$ bits each (in a simulation, the entropy was only ≈ 7).

Sparseness and Bloom Filter: The matrix O contains r observation vectors, each corresponding to the observed aggregation from the predecessors of the respective round. In particular, $\vec{o_0}$, the first column vector, is the observation vector of only one peer. Therefore, because $t \ll n$, this vector will contain many empty entries and is thus very sparse. Of course, it is unnecessarily expensive to send all these empty entries. Instead, we will only send entries for which at least one observation could be made. That is, we need to identify the corresponding entry of the vector using $\lceil logn \rceil$ bits. If $t \ll n$, this is much cheaper than sending the entire vector. We can generalize this also to the other rounds. Again, assuming that each peer makes t transaction, the column vector from round r will contain $x = k^r t$ observations. How many entries are empty in this case? The answer corresponds to the number of empty bins in a balls-into-bins game with n bins and x balls. The expected number of non-empty bins is given by $f(n, x) := n - n \cdot (1 - \frac{1}{n})^x \approx n - n \cdot e^{-\frac{x}{n}}$. Therefore, we will send the entire vector $\vec{o_i}$ only if it is more efficient in terms of size than when sending the single entries individually, which is expressed by

$$\min\{f(n, k^{i-1} \cdot t) \cdot (\lceil \log n \rceil + l), n \cdot l\}$$

A further reduction can be achieved; instead of addressing each peer by $\lceil \log n \rceil$ bits, we could encode them using a *Bloom filter* [5] with $c \cdot f(n, x)$ bits, where c would be small constant (e.g., 8)—a small false positive rate is certainly acceptable for the purpose of this reputation system if it results in a large efficiency gain. As an example, for encoding $250,000$ identifiers using a Bloom filter with 5 hash functions and $2,000,000$ bits, we would have a false positive rate of only 0.021, with a reduction in size of $\frac{\lceil logn \rceil}{8}$. Putting this together yields the size of the vector $\vec{o_i}$ by

$$\min\{f(n, k^{i-1} \cdot t) \cdot (c + l), n \cdot l\}$$

Conclusion: To conclude, we define the expected message size χ, depending on the number of rounds r and the number of transactions t in the system. We will use $l = 8$ and $c = 8$. In a simulation, $l = 7$ was sufficient.

Fact 6.1: The average size in bytes of messages in Havelaar is approximately given by

$$\chi = \frac{1}{8} \cdot \sum_{i=0}^{r} \left[\min\{f(n, k^{i-1} \cdot t) \cdot (8 + 8), n \cdot 8\}\right].$$

VII. Evaluation of Communication Cost

As mentioned, besides Havelaar's strengths—e.g., being robust to attacks, and computing fair contribution values—, one of the major concerns is the amount of information which has to be transmitted per round. Therefore, in this section, we analyze the communication costs. Moreover, we compare them to a reputation system which is based on DHTs, i.e., where the contribution value of a given peer u is stored in the DHT based on a hash function $h(u)$. Note that this is only meant as a benchmark in terms of communication overhead: unlike Havelaar, the DHT approach is typically already vulnerable to simple attacks.

A. DHT Benchmark

DHT, because peers will not always remain online. There are basically two approaches: First, we could store the value redundantly at different peers. Second, we could move the contribution value to the next closest peer whenever a peer leaves.

is especially important for large data. However, it suffers from the problem that the data is read-only; updating all redundant copies would be too expensive. Updating the contribution value is obviously necessary. However, we could argue that it does not matter if not all copies would be updated when increasing the contribution value of a peer. After all, also in our system, the contribution value does not need to be exact. However, in this case, the variance in the contribution value would depend on the online behavior of these peers, which cannot be controlled and could thus vary very much. To balance this, we would need to introduce an even further redundancy.

purpose; however, it does result in an increased communication overhead, especially under high churn rates. Furthermore, there is a huge potential for attacks.

performance evaluation, since we do not take this extra communication costs into account. Instead, we state the costs of the DHT approach as "lower bound" for exactly this reason.

Exact DHT: In the DHT approach, the contribution value of peer u is stored at a peer which is chosen based on a hash function $h(u)$. Before uploading data to peer u, its value must be retrieved from the DHT. After downloading data from peer u, its value must be updated.

Let us assume that every message has an overhead of $\mu = 40$ bytes (e.g., message header plus packet type). We assume $\lceil \log n \rceil$ bits for identifying each peer, and we assume that each contribution value uses 8 bits. Furthermore, a message can be routed in $\log n$ steps, which is typical for many DHTs [21], [23], [26], [30]. Finally, we consider a scenario where each peer makes t downloads and therefore also approximately t uploads.

Thus, since the contribution value of each peer needs to be requested before an upload, and needs to be stored after a download, the communication costs of this approach are approximately given by $2 \cdot t \cdot \log n(\mu + \frac{\lceil \log n \rceil}{8} + 1)$.

Note that these are lower bound costs; in reality, the DHT approach would need to move the contribution value to another peer whenever a peer joins or leaves the system,

so that it is always available. Storing redundant copies does not work, because not all replicas can be updated on a change.

For $t \ll n$, there is only a very small probability that peer u has contact with the same peer again. For large t, however, we could—similarly to our reputation system—only update and retrieve the contribution value of each individual peer once a round. As described in Section VI, this corresponds to the number of non-empty bins in a balls-into-bins game and can be described by the function $f(n,t)$. Therefore, we will extend our lower bound cost function for the exact approach (DHT_{EL} for DHT exact lower bound) accordingly:

$$DHT_{EL} = 2 \cdot f(n,t) \cdot \log n \cdot (\mu + \frac{\lceil \log n \rceil}{8} + 1)$$

Approximate DHT: In the exact approach, the contribution value of each peer is reflected accurately. In Havelaar, however, we accept that the value is only approximate, and hence the comparison is not entirely fair yet, as we could also reduce the communication overhead of the DHT approach: Instead of updating the contribution value after each transaction, we could only update it with a probability p. This approach has been suggested in [20], where the authors have shown that the results are accurate enough. A value of $p = 0.05$ would result in a variance that is in the same order as in our case. However, these costs can only be saved for updates; before every download, we still need to retrieve the contribution value. Thus, the adapted costs for this approach (DHT_{AL} for DHT approximate lower bound) are:

$$DHT_{AL} = (1+p) \cdot f(n,t) \cdot \log n \cdot (\mu + \frac{\lceil \log n \rceil}{8} + 1)$$

B. Results

We can now compare the costs of the DHT approach to Havelaar. In the Havelaar reputation system, every peer sends the observation matrix O to its k successors. The size χ of such a message has been computed in Section VI. The total communication costs in our Havelaar system are therefore

$$Havelaar = k \cdot \chi$$

Note that we keep r in this simple evaluation. However, as has already been mentioned, r is always quite small, and changes slowly (logarithmical) in the total number of peers.

3 compares Havelaar to the exact and approximate DHT benchmarks for a network of size $n = 100,000$. It shows the communication overhead in megabytes (MB) depending on the number of transactions. For small number of transactions ($t < 2000$), the communication overhead of our reputation system is higher than when a DHT approach is used. However, it clearly indicates that it scales very well with the number of transactions. Furthermore, observe that the both DHT approaches represent a lower bound—in reality, further communication costs occur.

4 compares the communication costs of Havelaar to the exact DHT approach for a fixed number of $t = 20,000$ transactions depending on the size of the network. brauchts da noch conclusion?

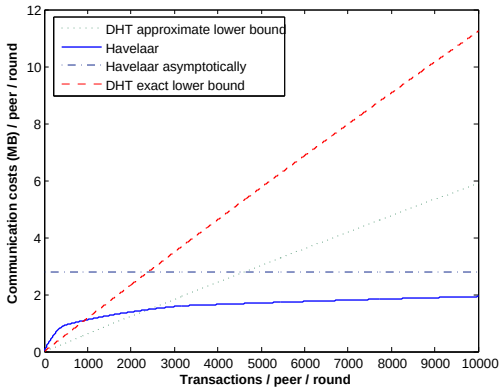

Fig. 3. Comparison of the communication complexity of our reputation system Havelaar to the DHT approach for a network of $n = 100,000$

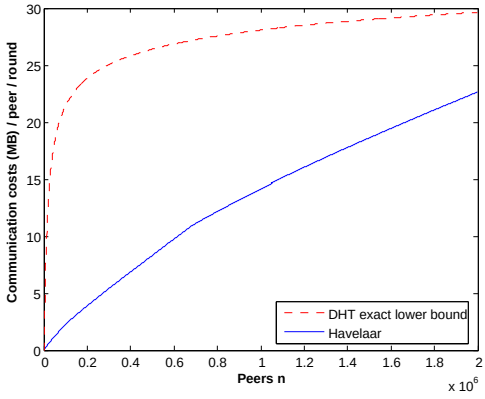

Fig. 4. Comparison of the communication overhead for $t = 20,000$ transactions depending on the size of the network.

In conclusion, we see that although the costs of Havelaar can be high, they are still tolerable in many practical systems. Moreover, our system compares well to DHT approaches, especially if there are many transactions per round (i.e., per week).

VIII. Attacks

Havelaar is designed to cope with peers aiming at selfishly consuming larger shares of resources than other peers. The fact that every peer can send its observations only to its k successors makes *local defenses* possible.

The peer uses several defense mechanisms. First a receiver will check whether a sender is one of its predecessors, and otherwise ignores the report. Moreover, it can make sure that a peer does not report contributions too often.

Of course, a peer does not take into account observations of the predecessor itself. Thus, a most attractive attack is made hard *by design*: It is only possible to falsely "praise" or "accuse" *another* peer.

In addition to limiting the range of possible values, other measures are taken in our system to detect and ignore false reports. Before updating the local contribution vector $\vec{c}$ on the basis of the k observation matrices, for each value the average and variance is calculated. If one value is extremely large, it is considered an outlier with respect to the other $k-1$ values, and is dropped. Then, the average of the other $k-1$ values is taken as the input to the update function.

Clearly, one can think of several further local defense mechanisms. For instance, statistical measures could be included to detect a possible false report by studying the distribution (histogram) of the observation vector, e.g., by checking whether the histogram is spiked. In any of the above cases where the successor is suspicious of an attack, it could reduce the trust value associated with each predecessor. The trust value can be used to either drop observations by suspected peers, or to weigh their observation values accordingly. However, so far, we make not use of these techniques.

Besides the advantages of local defense, the robustness of Havelaar comes from the *extensive aggregation*: A single wrong value can hardly influence the overall outcome. In this sense, also the damage which can be done by a small fraction of colluding peers is limited. In particular, as successor peers are determined by hash functions, becoming a predecessor of a specific peer is difficult, and hence colluders cannot efficiently leverage their accumulated power either.

In summary, Havelaar's design—which is based on local defense and extensive accumulation, and which (unlike many other approaches) does not rely on trust transitivity—makes it hard to falsely report in this system.

IX. Simulation

We have performed several simulation of Havelaar which fortify our results. In this section, we want to present the most interesting findings.

In Figure 5, the real ratio of the contribution values of two peers are compared using the ratio from the local approximation in several rounds in a network of size $n = 100,000$, with $k = 7$ successors and $r = 4$ rounds. In each round, the peers can change their upload bandwidth. In the first round, for instance, peer u contributed exactly three times more than peer v. Note that the approximation is shifted to the right; this is due to the fact that contribution values are only updated once a round, based on observations from the past. Note also that the standard deviation of the approximation is generally low—being higher when peers change their behavior abruptly, again because of observations from the past.

Figure 6 plots the local contribution vector for the same network against the real contribution vector. Both vectors are normalized by dividing each entry by the sum of the whole vector. Therefore, each contribution value reflects the proportional contribution of the whole network. Again, the local approximation reflects the real contribution very accurately.

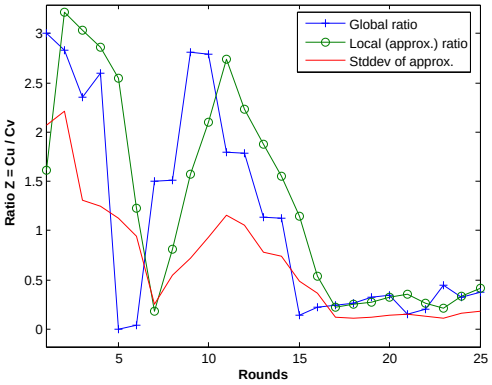

Fig. 5. Local approximation vs. real contribution value of two peers in several rounds.

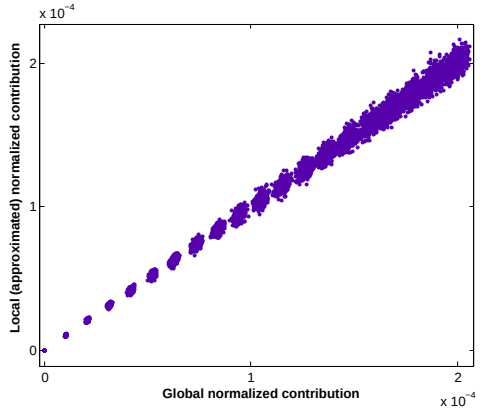

Fig. 6. Real normalized contribution value plotted against the locally approximated.

For peers with higher contribution, however, the variance becomes larger. The reason for this is that the variance is multiplied by the square of the bandwidth, as described in Section **??**. Observe, however, that the coefficient of variation remains constant for every peer because it does not depend on the bandwidth.

Finally, in Figure 7, we plot the locally computed contribution ratios of all peers in a histogram. In reality, peer u has contributed twelve times more than peer v, in the same network as above. The histogram shows that the distribution is slightly skewed to the left. Fitting a log-normal distribution indicates that the ratio Z is in fact log-normally distributed. Figure 8 depicts the probability plot as an indication of the accuracy of the fit.

X. Conclusions

The main goals of the Havelaar reputation system are (1) robustness to false reports, (2) efficiency, especially in the absence of contention, and (3) a communication complexity which does not depend on the number of transactions. This is achieved by a novel design approach in which peers always

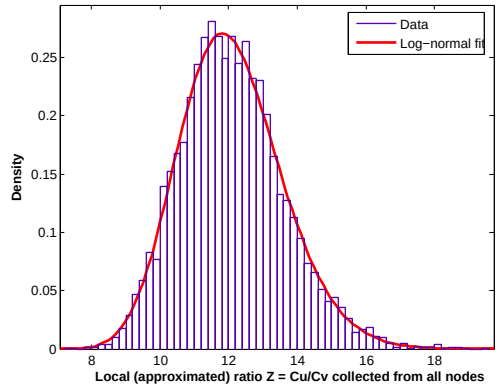

Fig. 7. Histogram and log-normal fit of the locally approximated ratio $Z = \frac{C_u}{C_v}$ collected from all peers.

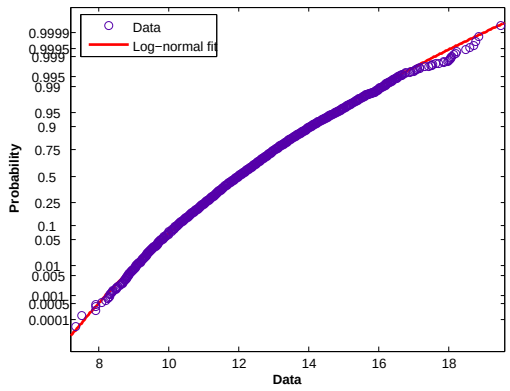

Fig. 8. Probability plot of the log-normal fit to show its accuracy.

report the reputation values the same set of peers, which allows for a local control of a peer's behavior. Encouraged by our results, we have now integrated Havelaar in our distributed storage system (Kangoo). However, we believe that Havelaar is a good choice for many active p2p systems requiring a fairness mechanism.

REFERENCES

[1] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In H. Paques, L. Liu, and D. Grossman, editors, *Proc. of the 10th Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 310–317, 2001.

[2] E. Adar and B. Huberman. Free Riding on Gnutella. *First Monday*, 5(10), 2000.

[3] R. Axelrod. The Evolution of Cooperation. *Science*, 211(4489):1390-6, 1981.

[4] D. Banerjee, S. Saha, S. Sen, and P. Dasgupta. Reciprocal Resource Sharing in P2P Environments. In *Proc. 4th Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2005.

[5] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, 1970.

[6] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. Workshop on Economics of Peer-to-Peer Systems*, 2003.

[7] M. Feldman and J. Chuang. Overcoming Free-Riding Behavior in Peer-to-Peer Systems. *ACM Sigecom Exchanges*, 6, 2005.

[8] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *Proc. ACM Conf. on Electronic Commerce (CECOMM)*, 2004.

[9] F. D. Garcia and J.-H. Hoepman. Off-Line Karma: A Decentralized Currency for Peer-to-peer and Grid Applications. In *Proc. 3rd Applied Cryptography and Network Security (ACNS)*.

[10] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives in Peer-to-Peer File Sharing. In *Proc. 3rd ACM Conf. on Electronic Commerce (EC)*, 2001.

[11] D. Hughes, G. Coulson, and J. Walkerdine. Free Riding on Gnutella Revisited: The Bell Tolls? *IEEE Distributed Systems Online*, 6(6), 2005.

[12] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proc. WWW*, pages 640–651, 2003.

[13] K. Lai, M. Feldman, J. Chuang, and I. Stoica. Incentives for Cooperation in Peer-to-Peer Networks. In *Proc. Workshop on Economics of Peer-to-Peer Systems (P2PEcon)*, 2003.

[14] Q. Lianz, Y. Pengx, M. Yangx, Z. Zhangy, Y. Daix, and X. Li. Robust Incentives via Multi-level Tit-for-tat. In *Proc. 5th Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[15] R. B. Ma, S. M. Lee, J. S. Lui, and D. Y. Yau. A Game Theoretic Approach to Provide Incentive and Service Differentiation in P2P Networks. In *SIGMETRICS*, pages 189–198, 2004.

[16] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge Press, 2005.

[17] MMAPPS. Market Management of Peer-to-peer Service. *EU/IST Project*, 2004.

[18] S. J. Nielson, S. Crosby, and D. S. Wallach. A Taxonomy of Rational Attacks. In *Proc. 4th Int. Workshop on Peer-to-Peer Systems (IPTPS)*, pages 36–46, 2005.

[19] A. M. Odlyzko. The Case Against Micropayments. In *Financial Cryptography*, pages 77–83, 2003.

[20] T. G. Papaioannou and G. D. Stamoulis. Reputation-based Policies that Provide the Right Incentives in Peer-to-Peer Environments. *Computer Networks*, 50(4):563–578, 2006.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proc. of ACM SIGCOMM 2001*, 2001.

[22] J. A. Rice. *Mathematical Statistics and Data Analysis*. Duxbury Press, 1995.

[23] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proc. 18th IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[24] J. Shneidman and D. C. Parkes. Rationality and Self-Interest in Peer to Peer Networks. In *Proc. 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[25] W. Stahel. *Statistische Datenanalyse. Eine Einfuehrung fuer Naturwissenschaftler*. Vieweg, Braunschweig, 2000.

[26] 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. ACM SIGCOMM Conference*, 2001.

[27] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: A Secure Economic Framework for P2P Resource Sharing. In *Proc. Workshop on Economics of Peer-to-Peer Systems (P2PEcon)*, 2003.

[28] W. Wang and B. Li. Trust Based Incentive in P2P Network. In *Proc. Int. IEEE Conf. on E-Commerce Technology for Dynamic E-Business (CEC-East)*, pages 302–305, 2004.

[29] W. Wang and B. Li. Market-driven Bandwidth Allocation in Selfish Overlay Networks. In *Proc. IEEE Conf. on Computer Communications (INFOCOM)*, pages 36–46, 2005.

[30] B. Y. Zhao, L. Huang, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), 2004.