Using Recurring Costs for Reputation Management in Peer-To-Peer Streaming Systems

Michael Rossberg and Guenter Schaefer and Thorsten Strufe
Telematics and Computer Networks Group
Technische Universität Ilmenau
michael.rossberg[at]stud.tu-ilmenau.de, [guenter.schaefer, thorsten.strufe][at]tu-ilmenau.de

Abstract—**Due to the dependency on preceding nodes in overlay live streaming systems, only highly reliable nodes should be chosen to occupy vital positions in the overlay topology, serving large numbers of succeeding participants. Otherwise, the highly dynamic and potentially hostile environment with frequent arrivals and departures of participants may lead to high packet loss rates and a significant decrease in quality of service. Hence, a high resilience towards failure of participants and especially deliberate sabotage through malicious parties is a prerequisite for this content distribution scheme to gain acceptance by users and the market. In order to incorporate the reliability of nodes into the topology construction process a stable metric for assessing the reliability of nodes has to be defined that preserves the anonymity of the subscribers and allows coping with their stochastic behavior. In this paper we present *eLeumund*, an algorithm, which utilizes recurring costs as a means to compute a dependable reputation value representing a node's reliability for the service. Our scheme maintains the privacy of all participants.**

Index Terms—**Peer-To-Peer, Security, Reputation, Anonymity, Privacy, Recurring Costs, Client Puzzle.**

I. Introduction

Over the last years Cooperative Live Streaming, sometimes called application layer multicast (ALM) [1], has received increasing attention as an alternative scheme for live content distribution. Systems implementing ALM make use of the resources of end hosts, as each subscriber in turn offers to relay the received packets to other participants, thus raising the overall available bandwidth of the service. Each packet of the stream hence is delivered following the topology of a spanning tree consisting of all participants in the service. As the content is forwarded to the receivers by other participants of the system, each subscriber depends on the correct service of all preceding participants on the overlay path to the source. In this distribution scheme, the failure of a participant (a *parent node*) to deliver a packet to a succeeding receiver (the *child node*) affects all successors in the respective branch until the topology is repaired by placing a different node in the void position. In order to reduce the impact of a single failing parent or indirect predecessor most ALM systems arrange the overlay such that nodes are served different parts of the stream by different parents and preferably on differing and node-disjoint overlay paths.

Due to the heterogeneity of the participating nodes with different processing and networking capacities, it is important to choose participants with sufficient resources as inner nodes of the spanning trees, in order to reduce the overall delays and packet loss rates. However, in the potential presence of attackers, the selection of reliable nodes is even more vital as attackers have to be prevented from ascending to positions in the topology, where they serve large groups of succeeding nodes or are able to gather information about other nodes with a high number of successors. When selecting nodes for important positions in the topology preference should therefore be given to highly cooperative participants that have delivered a good service to a large number of other nodes over a long period of time.

In order to be able to judge which node qualifies best, the reputation of nodes is measured by the satisfaction of their successors. As different children and indirect successors benefit from the service, it is vital for a node to aggregate reputation that is attested by different nodes. In order to have a metric, which allows a good comparison between any two nodes, the aggregation should lead to a linear gain in reputation.

Some existing peer-to-peer applications, such as eMule and BitTorrent [2], already use trust databases in order to collect information on how cooperative different clients act. This approach is similar to reputation systems as the reputation of a node is simply the aggregated trust of others in the considered node. However, as these classifications are made on direct observations only. The potential transitive knowledge is neither gathered nor aggregated. The primary reason for this lack is the difficulty of constructing a satisfactory and yet secure alternative.

Another existing approach to avoid the impact of malicious nodes are client puzzles [3]. In this approach, nodes requesting a service have to solve a small task before they are served. The willingness to invest into solving the task approves that the requesting node has a substantial interest in receiving the service as it already invested resources in the form of computational effort itself. This indicates the node can be "trusted" in a way that the request is not a bogus one. However, conventional client puzzles cannot be used for measuring the reputation of a node in overlay live streaming systems as they provide no metric to measure the quality of the solutions. Additionally, the tasks cannot be split into subtasks nor can the results be aggregated and successors of nodes cannot vouch for their parents.

With *eLeumund*, we propose an algorithm that introduces the task of guessing the preimage to a given hash value as the client puzzle that receiving peers have to solve in order to prove their sincerity in the service. This task can easily be partitioned as the search space can be analyzed in parallel by multiple children as the subtasks, which they have to solve in order to vote for a parent. Aggregation again is simple as each parent can select the best guess from the set of preimages it receives from its children. This approach can be used to measure the satisfaction of a group of receivers with the service a specific node provides, as they can gauge the amount of effort they invest for a parent according to the perceived quality of the received service. It still does not disclose the identity of the nodes involved in solving the puzzle, as the invested processing time is an anonymous and yet significant value.

The rest of the paper is organized as follows: in Section II we give some background on the problem of calculating reputation in overlay live streaming and present some assumptions and prerequisites needed to be able to develop a practicable reputation system in this scenario at all.

Section III outlines the requirements to a reputation system and the objectives it has to meet in order to be useful in the domain. The main idea behind our proposed reputation algorithm *eLeumund* is presented in Section IV as well as the short description of its implementation. In Section V an evaluation of eLeumund is given in two parts. The approach is assessed by analyzing to what extent it meets the objectives at first. As some objectives cannot be assessed analytically a simulation study shows the performance of eLeumund regarding the remaining requirements. In Section VI we put eLeumund into the context of related work and in Section VII we finally draw a conclusion and present directions for future studies of reputation systems for overlay live streaming.

II. Background

In Overlay Live Streaming [4], [5], a single source broadcasts a video program or any other continuous data stream. For reasons of robustness the stream is divided into several parts before transmission. These so called *stripes* are sent to and consecutively relayed by independent children of the source, thus forming a distribution tree by distributing the packets of the stripes to all participants of the service. If a new node joins the system, it will be inserted as a leaf into the distribution tree of every stripe. Whenever a forwarding node leaves the system, it is replaced by some other node. A small glitch in the stripe transmission for the successors may occur in this case. This can be compensated either by a forward error correction scheme or by error tolerant software. Analyzing comparable scenarios [6] it becomes apparent that the system has to deal with highly dynamic behavior of node arrivals and departures. This is a very hostile environment for a reputation system because nodes only need a short period of time to gain good reputation values – a fact that significantly lowers the required time potential attackers have to invest in order to achieve a competitive reputation.

Nonetheless, reputation systems promise to offer a desirable security gain and some axioms can be gathered from the scenario described. They form the constraints for the reputation algorithm to work in:

1) **The stream source is trustworthy.** Because of the single source scenario this superior node could obviously send malicious data or stop transmission completely. However, this behavior is highly unlikely and cannot be prevented by a reputation system.
2) **A secure location service is available.** Nodes joining the system must be able to locate a node that is already part of the topology. This locating service is a vital part of a peer-to-peer system. However, it is out of the scope of this paper.
3) **The chance for a node to leave does not increase over time.** Reputation systems try to derive information of future performance from past behavior. Hence, they depend on this correlation. A negative example would be a wireless sensor network where extremely cooperative nodes usually starve first. If a non-specialized reputation system was used in this scenario, it would always put the eldest cooperative node in the most important position. Thus creating an unstable network topology as its battery will be drained quickly.
4) **Secure and reliable communication between any two nodes is possible.** This paper depends on several transport layer services but their mechanisms are well known and therefore not considered in this work.
5) **Initial clients are trusted.** Reputation systems do not have enough information to perform their task right after the start of a new peer-to-peer network. Therefore, no statements can be made to protect the system at this stage. Another problem in this early stage is the high weight of a single attacker as the count of honest clients is relatively low. Potential attackers can gain much influence in this phase.

III. Objectives for a proposed reputation system

The requirements for a reputation system in overlay live streaming are either aiming at goals for the distribution topology or at security goals.

A. Network Topology Objectives

The use of a reputation system can only be justified, if it is able to provide information that can be used to optimize the topology of the peer-to-peer network towards given goals. Therefore, the following optimization targets are defined:

1) **The height of distribution trees shall be small.** The chance of any intermediate system to fail depends on the count of such systems. Therefore, this value is to be kept low.
2) **Inner nodes need to be the eldest.** In a hostile environment like the internet, long cooperative participation is the major evidence for the trustworthiness of nodes. Therefore, the inner nodes shall be as old as possible.

3) **Inner nodes need to be the most cooperative.** Inner nodes should not be selfish nodes for two reasons. On the one hand, such a behavior shall not be encouraged. On the other hand, choosing cooperative and high-performing nodes on the other hand helps to keep the tree height small, thus supporting objective (1).
4) **Nodes shall not ascent in multiple trees.** Whenever a node holds positions close to the source in more than one stripe, it becomes a vital part of the system. Such nodes gather too many successors and their failure has a major impact on the overall service. The reputation system shall prevent nodes from ascending to high layers in multiple trees.
5) **The "conservatism" of the reputation algorithm needs to be adjustable.** Any system that can tune the gain of reputation can react better to changes of the environment. Such a change could for an example result from an observed attack.
6) **Scalability.** The system, including means to manage the reputation of nodes, has to scale with the number of participants.

B. Security Objectives

General security objectives, such as confidentiality, integrity, accountability, controlled access, and availability are relevant for reputation systems. However, additional objectives are needed in the given context.

1) **Anonymity.** Following a simple rule: "A malicious user will not be able to attack if he does not know whom". The major security demand in this paper is extensive confidentiality of the identity of all participants.
2) **Limited knowledge about the network topology.** For the same reason as in security objective (1) it is important to keep the own position within the network confidential. This objective additionally extends the request for identity protection. Potential attackers shall not know how important they themselves and other nodes in their vicinity are. This makes the planning of the optimal time and target for attacks more difficult.
3) **Gain by attacks on known nodes.** A malicious user should not be able to elevate his reputation by attacking the availability of any other member.
4) **Restricted benefit by Byzantine attacks.** Reputation gain by giving wrong information in the protocol procedure shall be kept low.
5) **Resistance against Sybil attacks.** Perhaps one of the most significant attack development in the last years was the Sybil attack by Douceur [7]. It is basically a special case of the collaborative Byzantine attack. By replicating itself a malicious or selfish peer tries to overrule fair nodes and therefore gain an advantage. The resulting gain in reputation has to be low.
6) **Resistance against Whitewasher attacks.** Another widely discussed attack on reputation system is the Whitewasher attack. It is mainly used by selfish peers. They can try to simply reenter the system whenever they have gotten a bad reputation and start with a neutral record again. The effect of Whitewashers has to be negligible.
7) **Low impact of Sleepers.** Just like in real life, sleepers can be very powerful enemies in a reputation system. A node that initially exhibits normal and cooperative behavior can provide attackers with good reputation or suddenly change to becoming an attacker. The proposed system has to consider and avoid a large impact of potential sleepers.

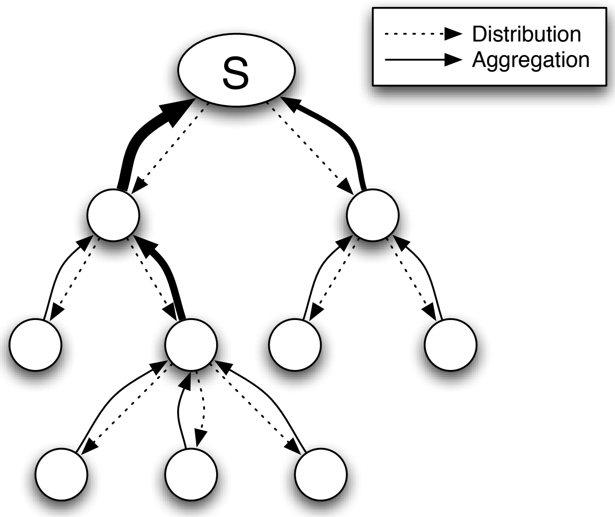

Fig. 1. Phases of the eLeumund system

IV. THE ELEUMUND SYSTEM

This section outlines the conditions that a reputation system has to incorporate and gives a short description of the core algorithm of eLeumund. The main idea behind eLeumund is the periodic distribution of approximation problems over the stripe trees. Each node tries to calculate a solution of the highest possible quality for the given task. The solutions after a short fixed period of time are aggregated using a reverse-multicast mechanism. The reputation of a successor in the tree is represented by the average quality of solutions it delivers over time. A simple outline of the process is given in Figure 1.

By distributing different problems for each stripe at the same time, nodes need to weight how much of their available calculation time they donate to which predecessors. In this way, nodes in higher layers can detect which of their successors delivers the best service, measured by the amount of satisfied successors or the degree of their satisfaction.

A. Characterization of $\mathcal{V}$

The distributed approximation problem must fulfill some criteria to meet the security goals of eLeumund. Hereafter, this class of problems will be called $\mathcal{V}$. Each problem $\mathcal{P} \in \mathcal{V}$ can be described as a generic approximation problem with

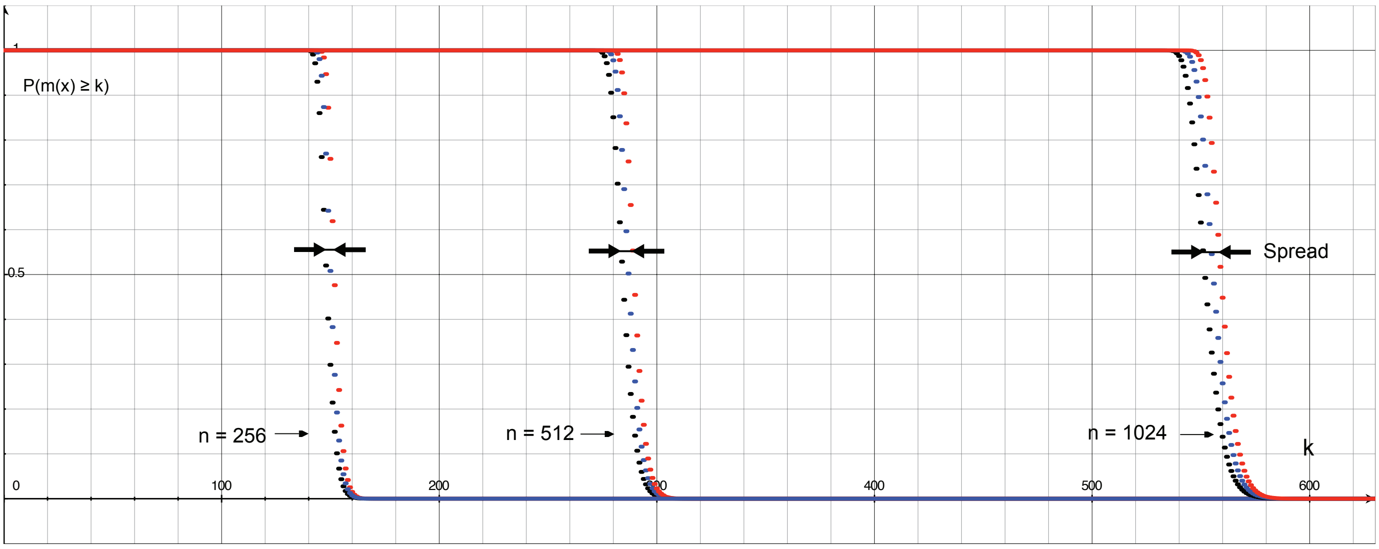

Fig. 2. Probabilistic spread of the reputation values for varied hash length ($n = \{256, 512, 1024\}$) and calculation count ($t = \{100, 200, 400\}$)

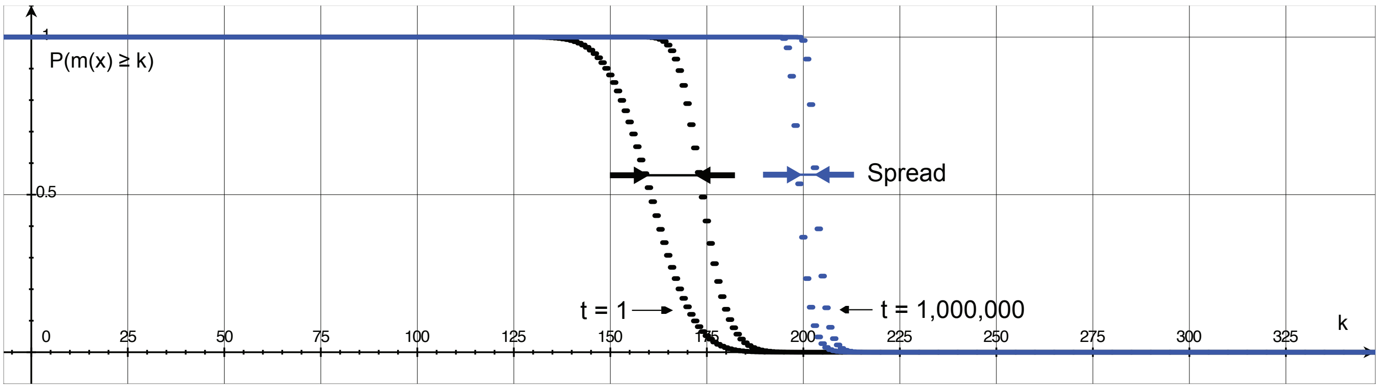

Fig. 3. Probabilistic spread of the reputation values for fixed hash length ($n = 320$) and varied calculation count ($t = \{1; 10; 100k; 1,000k\}$)

$(I, SOL, m, goal)$ – I being a set of well formulated instances, SOL a set of solutions, m a function calculating the quality of a given solution and $goal$ an optimization target. SOL^* is the optimal solution and m^* the optimal quality respectively. The additional criteria are:

1) An $x \in I$ and $m(x)$ can be computed efficiently.
2) The algorithm for the approximation of $\mathcal{P}$ shall be known to be the fastest available. If this property was not given, an attacker could use a faster algorithm and cheat on the system. Therefore, instances of $\mathcal{V}$ are to be based on a difficult, known mathematical problem. An example would be the discrete logarithm.
3) The value of $m(x)$ has to be an indication for how much effort was invested. Ideally, a linear correlation exists in order to keep values comparable over the whole space of reputation values.
4) The approximation has to be parallelizable.
5) Partial solutions have to be aggregatable. In the simplest case this can be done by selecting the best solution from a given set of solutions.
6) Elements of SOL need to be constructible in a way that additional information like nonces can be embedded. The creation of many different ($> 2^{80}$) solutions must be possible to avoid dictionary attacks.
7) It shall be possible to restrict I to certain values in order to embed for an example counters.
8) $m^*(x)$ is of interest.

B. *Utilizing Hash-Functions for $\mathcal{V}$*

The application of a technique similar to client puzzles is the main concept for eLeumund. It achieves the given criteria with certain caveats to be circumvented. The whole problem persists in the calculation of a value X for a given Z such that $\text{HASH}(Z \,||\, X) = Y$. Z is dynamically given by the source in eLeumunds distribution process. A particular instance of the cryptographic hash function HASH can be chosen at compile-time. Its output length is called n.

The source also distributes a second value, V, which is the optimal solution. V is needed to define the quality indicator $m(x) := \sum_{i=0}^{n} (y_i \ominus v_i)$, where $(v_0, v_1, .., v_n)$ and $(y_0, y_1, .., y_n)$ are the binary representations of V and Y. $\ominus$ represents the binary equal operator.

Analyzing eLeumund by the identified criteria leads to the following facts:

1) Values of X are pseudo-random bit strings of the length $n - |Z|$ and can be calculated in polynomial time. $m(x) \in O(n\ log\ n)$
2) Parts of the output of a cryptographic hash function cannot be predetermined by definition. Therefore, it is not possible to conclude a good X from Y in another way than through trying systematically.
3) Unfortunately, the quality of the estimated solution, $m(x)$, does not linearly correlate with time complexity. Although there is a statistical connection, its resolution is low for common hash functions ($\leq 512{-}bit$). Another adverse property of the proposed approach is that nodes by chance can guess solutions with a y being very close to the optimal v, which leads to positive mavericks in their reputation in seldom cases.
4) Calculation is parallizable by letting processes randomly choose values or by dividing the searched problem space.
5) The aggregation of approximated solutions cannot be done straightforward as a simple concatenation of collected approximations, would lead to a high traffic volume. In consequence, choosing the best approximation is the only practical solution.
6) It is possible to construct 2^n solutions.
7) Different values of Z allow the creation of different solution spaces.
8) The optimal solution and its quality are defined as $SOL^* := V$ and $m^*(x) := n$.

The values of $m(x)$ are randomly distributed. The probability of getting a solution with quality k is $P(m(x) = k) = 1 - \left(1 - \left(\binom{n}{k} * 0.5^n\right)\right)^t$. Parameters are t (trial count) and n (output length of the hash function). Figure 2 shows that the distance between different ratings is very small, even if the calculation effort is doubled. However, it increases with the output length of the hash function. Therefore, the construction of functions with longer output is needed to gain better results. Fortunately, no additional entropy is needed, hence it is possible to choose from a number of cryptographic primitives:

- LONGHASH(k) := (HMAC$(k, 1)$ || HMAC$(k, 2)$ || .. || HMAC(k, i))
- Output of a block cipher in output-feedback mode with the key $Z \,||\, X$ and a defined seed
- Output of a suitable stream cipher with the key $Z \,||\, X$

Another available property for optimization is to reduce the number of trials to a bare minimum. In this case, all nodes still need to be able to vote, but the gain between rates with doubled effort drops with rising numbers of trials (Figure 3).

C. Implementation

After having demonstrated the existence of suitable approximation problems, the actual reputation algorithm can be built. It is assumed that a hash-algorithm-like function is used for $\mathcal{V}$. Even though the mechanism has to deal with the problems discussed above, it represents the only known problem class to fulfill the security objectives. The environment is assumed to have loosely synchronized clocks as the algorithm presented does not handle timeouts. These can be introduced straightforward for a production system.

The algorithm of eLeumund consists of two different parts. One part of the algorithm that generates the tasks runs on the source node and is shown in Algorithm 1.

Algorithm 1 Cyclic valuation algorithm for the source.

```
CYCLICVALUATIONSOURCE()
 1  foreach $k$ in $M_{succ}$ do
 2      $precondition_k \leftarrow$ ID($stripe_k$) ||
 3              GENRANDOMBITS(PRE_SIZE)
 4      $postcondition_k \leftarrow$ GENRANDOMBITS(POST_SIZE)
 5  foreach $k$ in $M_{succ}$ do
 6      SEND($k$, {
 7        precondition => $precondition_k$,
 8        postcondition => $postcondition_k$,
 9        rating => 1.0
10        } )
11  wait for CALC_TIME
12  foreach k in $M_{succ}$ do
13      quality ← 0
14      response ← RECEIVE($k$)
15      if VALID(response{result}, $precondition_k$,
16              response{keychain})
17        then
18        post ← CALCPOSTCONDITION(response{result})
19        quality ← CALCM(post, $postcondition_k$)
20      $reputation_k \leftarrow reputation_k *$ DECREASE +
21              quality * (1 − DECREASE)
```

The procedure CYCLICVALUATIONSOURCE is executed periodically. It starts with the generation of tasks (lines 1–4) that are transmitted to all successors (5–10). Then, the source has to wait for the clients to perform their voting (11). The value for CALC_TIME has to be chosen a little higher than the round-trip-time from the source to all leafs. Too large values for CALC_TIME burden the clients more than necessary. Even worse, it will make the distribution of the reputation values smaller. Small values of CALC_TIME will keep lower-layer nodes from correct voting.
After collecting results during CALC_TIME the responses are checked. (12–21). The review for correctness has to verify whether *postcondition* is a valid solution and if the response came in a timely manner. However, it is important to note that the reputation of a successor is derived from the solution quality utilizing exponential smoothing. The DECREASE factor determines the rate of reputation establishment.

The counterpart of the source algorithm is placed in every participating node (Algorithm 2). It begins with the reception and forwarding of the received tasks to the respective successors (2–12). A division of the solution space into distinct

Algorithm 2 Cyclic valuation algorithm for the nodees.

```
CYCLICVALUATIONVERTEX()
foreach s in $M_{stripes}$ do parallel
    task ← RECEIVE($predecessor_s$)
    myKeyChain ← $\emptyset$
    foreach k in ($k \in M_{succ} \mid stripe_k = s$) do
        $key_k$ ← GENRANDOMBITS(KEY_SIZE)
        $precondition_k$ ← ID($s$) ||
                HMAC($key_k$, task{precondition})
        SEND($k$, {
            precondition => $precondition_k$
            postcondition =>task{postcondition},
            rating => MIN(task{rating}, VOTE($s$))
            } )
    SETTASKPRIORITY(MIN(task{rating}, VOTE($s$)))
    myResponse ← CALCULATEVCONSTTIME(
            task{precondition},task{postcondition})
    SETTASKPRIORITY(1.0)
    foreach k in ($k \in M_{succ} \mid stripe_k = s$) do
        quality ← 0
        response ← RECEIVE($k$)
        if VALID(response{result}, $precondition_k$,
                response{keychain})
            then
            post ← CALCPOSTCONDITION(response{result})
            quality ← CALCM(post,task{postcondition})
            myResponse ← AGGREGATEV(myResponse,
                    response{result})
            if CHANGED(myResponse)
                then
                myKeyChain ← {$key_k$} ∪ response{keychain}
        $reputation_k \leftarrow reputation_k *$ DECREASE +
                quality * (1 − DECREASE)
    SEND($predecessor_s$, {
        result => myResponse,
        keychain => myKeyChain
        } )
```

fractions for all successors would give the nodes an opportunity to guess their position in the distribution tree. Hence, each successor is supplied with a subtask comprising the complete solution space. It is important to modify the actual task at this point to prevent Sybil attacks. Currently, an HMAC function is used.
After completing the setup, the actual approximation takes place (13–16). The priority of the task is reduced to the minimum of either the nodes' trust in its predecessors or the predecessors' trust in higher layer nodes. The involved VOTE(s) function returns a value between 0.0 and 1.0 that represents the satisfaction of the node in its parents service. This includes both the length of the reception period and the quality of the given service.

Subsequently, a more complex version of the sources' reputation collection is executed (17–30). At this point it becomes evident that the nodes need to reproduce the entire hash chain in order to validate the presented solutions of their successors. If a valid solution is received, it will be aggregated by selecting the one with the highest quality.

At the end of the cycle the best solutions are forwarded to the parent nodes. For hash chain reproduction, the used keys are included as well.

V. Evaluation

The evaluation of the presented algorithm is split into two parts. Wherever possible the approach is evaluated analytically. However, some complex aspects require a simulation study.

A. Analysis

The analysis of eLeumund starts with the network topology objectives, as it is its primary task to optimize the network topology.

1) **Ascent in multiple trees:** In order to get reputation in more than one stripe a node has to split its calculation efforts and networking capacity. Hence it cannot gain as much reputation as an equally strong node that specializes on calculating for one particular stripe only.
2) **Adjustable rate of reputation gain:** Two aspects of eLeumund control the speed of reputation gain. The DECREASE constant determines the aging of the reputation information. The calling frequency CYCLICVALUATIONSOURCE influences the gain, too. The combination of a high calling frequency and a small decrease value on the one hand yields a high robustness against mavericks in the reputation system. A low calling frequency with a high decrease value on the other hand reduces the system load.
3) **Scalability:** The reputation system does not impair the scalability of the overall system. The size of the *keyChain* array grows with the total node count. This affects both the networking overhead and the computing power. Anyway, with the utilized user behavior model, this overhead is acceptable as it grows logarithmically in the amount of participants only.

In addition to the network topology objectives the algorithm has to meet the security objectives, as they are imperative preconditions. As the source is considered trustworthy the security goals apply to the node algorithm only.

1) **Anonymity:** Within the presented eLeumund algorithm no identities or node addresses are exchanged. However, since a connectionless network protocol is assumed, nodes will learn the addresses of their predecessors and successors. This is an optimal outcome as they are known by the streaming algorithm anyway.
2) **Limited topology knowledge:** Through monitoring downstream messages and evaluating the packet loss rates in different stripes, nodes in overlay live streaming systems can generally derive an estimated distance to the source. However, with eLeumund the observed values of the *rating* parameter can most likely additionally be statistically evaluated as they are strictly monotonic decreasing. Nodes can estimate the number of levels to the bottom of the tree, too. This is accomplished by keeping track of the maximum keychain length. The same estimation could probably also be done utilizing

statistical observations on reputation values of the successors.
The possibility to launch probabilistic attacks leads to the deduction that eLeumund does not fully meet this objective. The design of a mechanism to reasonably change a topology without detailed knowledge about it, seems to be fairly difficult if not impossible at this point.

3) **Gain by attacks on known nodes:** If a node attacked a successor, it would not benefit, as the lost reputation in the next valuation cycle would be the consequence. The only other known nodes are predecessors. Differentiation between two cases is needed here: On the one hand the predecessor could be an ordinary node. In this case the node would be disconnected from the stripe and would not benefit otherwise than disrupting the service for the joined branch of the tree. On the other hand the predecessor could be the source itself. This is the worst case to happen as the resulting failure is unrecoverable and a simulation study has to show how well eLeumund performs in preventing this attack.
4) **Benefit of Byzantine attacks:** Nodes can only lie when they send data. This happens at two stages of the algorithm: during the forwarding of messages and in the aggregation phase. The first send operation holds the parameters $precondition$, $postcondition$, and $rating$. A change of the $condition$ values would make the calculated reputations unsuitable as they are checked by the predecessors. The $rating$ parameter allows nodes to express their mistrust in their parent nodes to their children. By setting this value to 1.0 attackers would only get an advantage if their successors trusted them. This will most likely lead to ascendence of the attacker in a bad-performing tree. Later the nodes are supposed to transmit the best result they calculated. The answer consists of two parameters. Lying by sending a void, a random, or a suboptimal result parameter will lead to a decreasing reputation as the parent node will not be able to deduct a meaningful quality value. The transmission of any different $keyChain$ parameter will cause the sanity check in the parent node to fail because the HMAC will produce random invalid results. Therefore lying within the eLeumund protocol sequence does not lead to significant advantages.
5) **Resistance against Sybil attacks:** Running a Sybil attack against the eLeumund algorithm requires linearly more calculation power as well as network capacity.
6) **Resistance against Whitewashers:** In eLeumund the Whitewasher problem is coped with by the abdication of blames. Nodes entering the system start with lowest possible reputation of 1.0. Therefore, participants with a low reputation cannot exploit a reentrance.
7) **Impact of Sleepers:** No node can use its own reputation to elevate the reputation of others. The transformation of the $preconditon$ values inhibits the interchange of solutions. A sleeper could work like a slave for some attacker but this causes the attacker to loose reputation, as it has no time to solve the puzzle for itself. The impact of a sleeper that attacks itself is evaluated using simulation, which is described in the next section.

B. Simulation

After evaluating the objectives through analysis where possible, only four are left. These are the impact of sleepers, the height of the distribution tree, the age, and the cooperation of inner nodes. These remaining objectives have been evaluated using event-based simulations with OMNeT++. For reasons of simplicity the simulation was limited to a single stripe and the overlay network only. The user behavior is assumed to be similar to an average user of video streams as described in [6]. In this model the clients join the system following a Poisson distribution with an average arrival rate of 0.3 seconds. The length of their stay is approximated by a lognormal distribution with an average residence time of only about 10 minutes. In consequence, most users leave the stream already after 20 seconds. In all simulations this user model was used, the DECREASE factor was set to 0.2, and a reputation calculation was initiated in a 5 second cycle.

1) Cooperation of Inner Nodes: In order to estimate the quality of the reputation system it is required to compare the cooperation of the nodes at different levels. The nodes on the higher layers should be more cooperative and selfish peers should not ascent. To evaluate if eLeumund meets these requirements an experiment with nodes of equal performance was conducted. They differed in their network capacity only, which was uniformly distributed between 1 and 10.

The simulations show that after 30.000 seconds of simulation time the cooperative nodes dominate in the higher levels as they can collect the reputations from more successors, as expected. Figure 4 displays the average results of 100

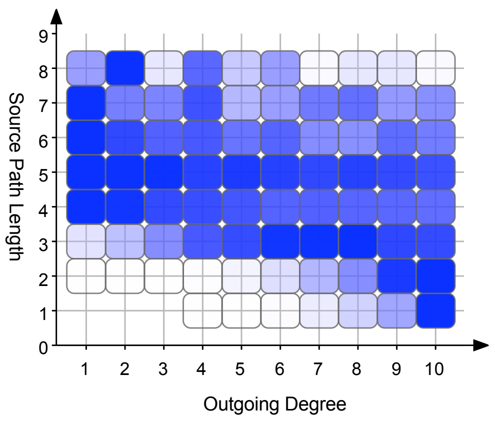

Fig. 4. Dependency between Node Level and Cooperation

simulation runs. Darker spots represent a higher relative occurrence of nodes with a particular outgoing degree in a node of the respective level. Level one represents nodes that are directly served by the source. These nodes are predominantly of the highest outgoing degree. Selfish-nodes with a single successor can almost only be found in level four and lower.

The results in layers six to nine are not representative as they are caused by mavericks. Consequently, eLeumund fulfills the required network topology objective of giving more reputation to cooperative nodes, thus leading to the fact of their ascent in the stripe tree.

2) Age of Inner Nodes: The request that inner nodes shall be elder than leaves is another major goal of the reputation system in the presented scenario. Therefore, another experiment using the same parameters as in the preceding experiment has been conducted. This was done to evaluate how much time nodes need to gain enough reputation to be chosen as inner nodes.

Nodes need time to gather reputation in eLeumund, which is caused by the exponential smoothing and the slow process of collecting additional successors. This characteristic leads to the expectation that in the evolving topologies nodes with a higher age are going to be located closer to the source than nodes that joined the system later.

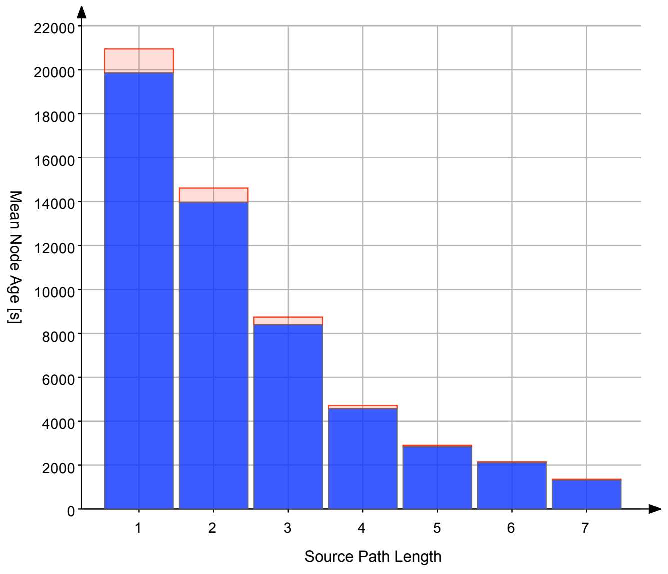

Fig. 5. Age of Nodes Per Distribution Tree Level, 99% confidence

The simulation shows that the average age constantly decreases from about 20,000 seconds in the highest layer to 1,500 seconds in layer seven (Figure 5), which is the last significantly occupied layer, as expected. Thus, eLeumund surrounds the source with old nodes, as desired.

3) Distribution Tree Height: In order to estimate the quality of the distribution tree height, no intuitive yet meaningful metrics exist: The maximum tree height is not very robust against mavericks and the average tree height suffers from a pyramid effect, since lower layers can accommodate more nodes than the higher layers. Partitioning the objective, three factors can be identified to indicate to what extent the objective is reached: the overall node count, the outgoing degree of the inner nodes and the density of the tree. While the total node count is derived from the user scenario and therefore a given value, the last two values have to be determined through experiments. A high outgoing degree of inner nodes, which leads to a low distribution tree height, additionally gives evidence to what degree the network topology objective (3) (*Inner nodes need to be the most cooperative*) is reached and has already been analyzed in Section V-B1

In consequence, another simulation study is conducted to evaluate the occupation of the layers in the evolving distribution trees. In the experiment, the tree density was measured in each layer, as a high density in the upper layers is a precondition for a minimum tree height.

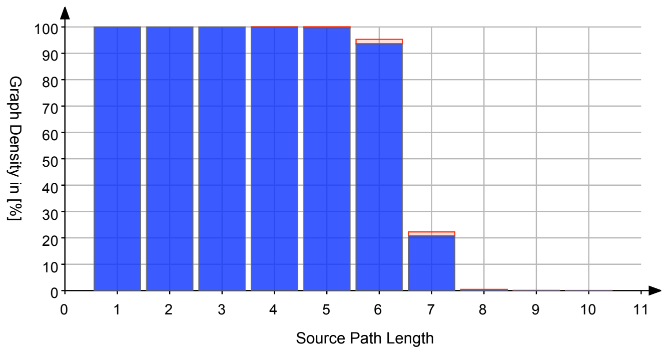

Fig. 6. Density of the Distribution Tree, 99% confidence

In the experiment all nodes had equal processing performance and a transient phase was removed before taking snapshots for the evaluation. Figure 6 illustrates that eLeumund in this case performs as expected. The upper node layers are fully occupied and it can be concluded that the algorithm creates distribution trees that are almost optimally filled. The mavericks in layer eight to ten are caused by reconnecting nodes and negligible.

4) Impact of Sleepers: With regard to the security objectives the impact of sleepers remains to be evaluated by simulation. In order to assess the effort an attacker has to invest to ascend to a relevant position in the topology different experiments were conducted. In the given scenario an attacker is expected to aim at identifying the source as it represents the only single point of failure. A conceivable attack through sleepers therefore consists of following two steps. First, the source node needs to be found by a sleeper attack. Second, it must be eliminated utilizing for an example a denial of service attack. The reputation system can only help in making the first step lasting for as long as possible.

In order to see how much time is needed for an attacker to discover the source, attackers were provided with different capabilities. In the simulations conducted they were inserted after the completion of the transient phase.

In the first experiment the attackers were assumed to be of equal performance as the average nodes. All nodes had three successors. As expected all sleepers gathered enough reputation to identify the source after a period of time. The exact time span depends on the number of attackers and is exponentially decreasing from a mean of about 110,000 seconds to 35,000 seconds (Figure 7). Both, the aggregated

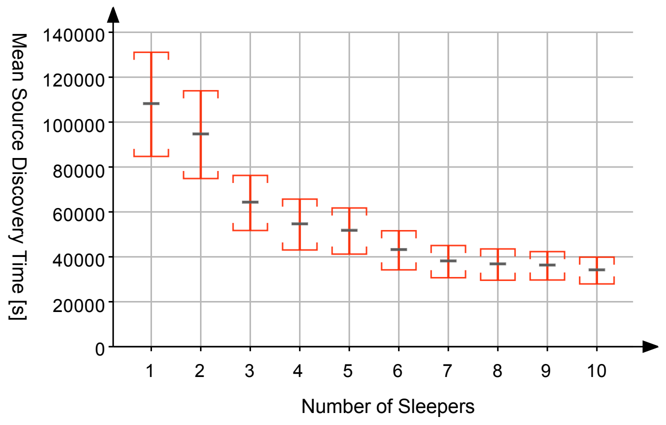

Fig. 7. Time to source discovery - Variation of attacker count, 99% confidence

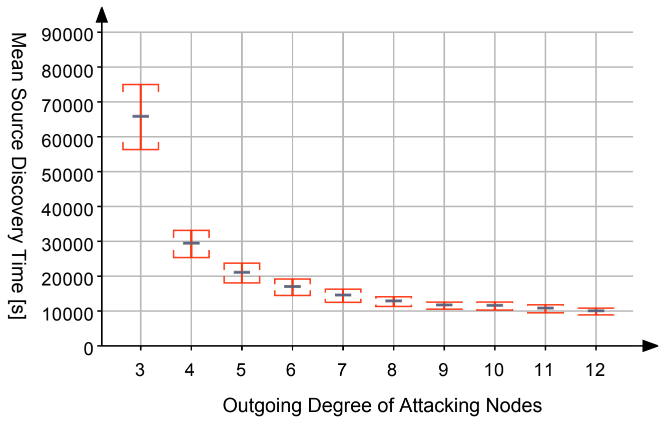

Fig. 9. Time to source discovery - Variation of attacker network capacity, 99% confidence

calculation power and utilized network capacity, which the sum of the attackers had to invest, increased by a factor of nine to achieve this speed up in identifying the source. From the results the following conclusions can be drawn: a single sleeper will need more than a day on average to identify the source, if not specially equipped. However, the confidence intervals are very large, which indicates that in seldom cases less time is needed.

Assuming that an attacker does not utilize unlimited resources all subsequent experiments are conceived so that an attacker controls three nodes as sleepers. This seems to be a good trade-off for an offender. If it had more capacities, they would better be spend on more powerful sleepers as will be shown in the following.

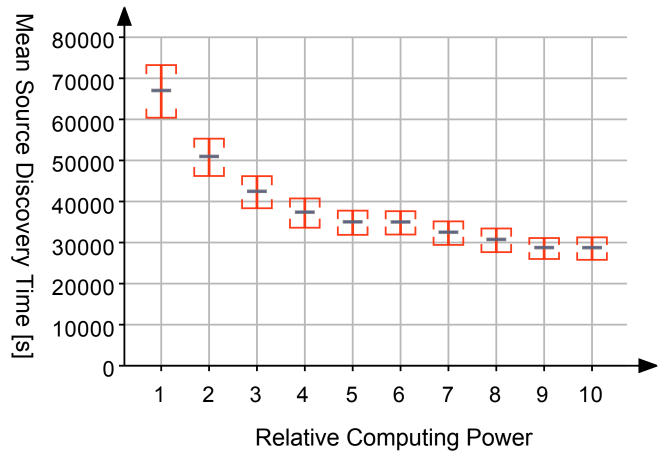

Fig. 8. Time to source discovery - Variation of attacker computing power, 99% confidence

Figure 8 shows the results of the next experiment. The number of attackers was held constant at three. However, these attackers could calculate faster than normal nodes. The illustrated behavior was expected, too. The required time drops from about 78,000 seconds down to 25,000 seconds when the calculation power increases to a factor of ten.

The last experiment revealed a bad side-effect of the network topology optimization goals. Target of this particular experiment was the determination of the coherence between network capacity and reputation. Aggressors had the same computing performance as average nodes but a higher outgoing degree. The results in Figure 9 show an exponential decrease of the time before a sleeper identified the source. This was expected to happen. However, a factor of four in the network capacity leads to only 10,000 seconds before the source is identified and hence to a gain of a factor of 10 in the speed of locating the source. While three hours are still long compared to the average online time of the nodes there is certainly room for further improvements regarding the dependency between network capacity and reputation.

The reason for this behavior is the fact that the reputation system exhibits a pyramid characteristic. The higher outgoing degree of the attackers causes them to have more direct and indirect successors. This observation leads to the discovery of a contradiction in the objectives. The effect is actually wanted, being a direct consequence of the demand for a low distribution tree height. For future developments an adjustable trade-off will be needed at this point. The main idea is the involvement of suboptimal decisions with regards to the network topology. Nodes having a lot more reputation than all other successors would get a penalty in such a scheme and therefore keep attackers with high network capacities on lower levels of the distribution tree.

VI. Related Work

A range of other approaches, such as Hashcash [8] and client puzzles [3] utilize the calculation of hash-collisions. However, instead of forming reputation systems they constitute mechanisms to counter resource exhaustion attacks on individual nodes. As they focus on load control the resolution of hash-collisions, which is identified as a problem in our current work, does not pose a problem in these mechanisms as is explicitly stated in [9]. Another approach to use client puzzles in multi-user scenarios is described in [10], which primarily aims at avoiding Sybil attacks. In the other client puzzle approaches each node calculates its own solutions and trust is established in one-to-one relationships. The scheme proposed in this paper additionally extends the idea to a group investing effort in order to vouch for one entity.

Closer to the scope of our work are other reputation systems that can be categorized by the way they cope with Whitewasher- and Sybil attacks. This leads to a classification that allows outlining different building blocks and approaches.

The Whitewasher attack has been thoroughly discussed and two different countermeasures have been proposed. One class of solutions involves admission control systems. Pseudonyms or accounts can be utilized to prevent a user from reentering the system as long as their creation is either expensive or is limited somehow. The other class of solutions prevents Whitewashers by allocating a minimal reputation to new participants by default. Another more common approach is to prohibit negative reputation votes [11] as they can also be used for discrediting [12] single participants. The eLeumund algorithm follows the second paradigm and does not provide means to discredit other nodes.

Straightforward protections against the Sybil attack cannot easily be found. [13] elucidates different mechanisms. These are comprised of trusted certifications, resource testing, recurring costs, trusted devices, and observation. While trusted certifications and resource testing may work very well under certain circumstances [14] they are bound to raise privacy concerns in the disperse environment of overlay live streaming. This is the reason for eLeumund to follow the scheme of recurring costs. Another publication suggests a privacy enhanced reputation system utilizing trusted devices [15], but the required TPM chips are not available today nor widely accepted.

VII. Conclusion

In difference to common certificate-based reputation management solutions, eLeumund follows the approach of calculating the reputation through measuring the quality of preimage guesses for given hash values. This approach allows for a group of succeeding nodes to vouch for a predecessor depending on the quality of the relaying service in the overlay. With eLeumund it is possible to show the effective exploitation of tree structures in the overlay network.

However, some issues for eLeumund remain to be solved. The probably most vital issue is the extension of security. In particular the obfuscation of the network topology shall be secured against statistical attacks. Additionally, improved robustness against sleepers has to be achieved.

In order to better understand the effects of transmission delays, timeouts, message loss, and shifted clocks the simulations have to be extended to include packet-level-models.

In another line of research it is a challenge to find better optimization problems. Tasks that do not depend on pure computing performance, but for example on memory access times, would be more fair to older computers.

Currently, inter-stripe aspects are completely undiscussed and ways have to be found to transfer reputation information between stripe trees.

Last but not least, reputation is collected over a short period in our scenario. However, the storage of reputation information is desirable because of the brief lifecycles of nodes.

References

[1] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1456–1471, Oct 2002.

[2] B. Cohen, "Incentives build robustness in bittorrent," in *First Workshop on the Economics of Peer-to-Peer Systems*, 2003. [Online]. Available: http://www2.sims.berkeley.edu/research/conferences/p2pecon/papers/s4-cohen.pdf

[3] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *Proceedings of NDSS '99 (Networks and Distributed Security Systems)*, 1999, pp. 151–165. [Online]. Available: http://www.rsasecurity.com/rsalabs/node.asp?id=2050

[4] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth content distribution in a cooperative environment," in *Proceedings of (IPTPS'03)*, 2003, pp. 298–313.

[5] T. Strufe, G. Schäfer, and A. Chang, "BCBS: An Efficient Load Balancing Strategy for Cooperative Overlay Live-Streaming," in *Proceedings of the IEEE-ICC*, 2006.

[6] E. Veloso, V. Almeida, W. Meira, A. Bestavros, and S. Jin, "A hierarchical characterization of a live streaming media workload," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. New York, NY, USA: ACM Press, 2002, pp. 117–130.

[7] J. R. Douceur, "The sybil attack," in *Proc. of the International Workshop on Peer-to-Peer Systems*, 2002. [Online]. Available: http://www.cs.rice.edu/Conferences/IPTPS02/101.pdf

[8] A. Back, "Hashcash - a denial of service counter-measure," Tech. Rep., 2002.

[9] T. Aura, P. Nikander, and J. Leiwo, "Dos-resistant authentication with client puzzles," in *Revised Papers from the 8th International Workshop on Security Protocols*. London, UK: Springer-Verlag, 2001, pp. 170–177. [Online]. Available: http://research.microsoft.com/users/tuomaura/Publications/aura-nikander-leiwo-protocols00.pdf

[10] N. Borisov, "Computational puzzles as sybil defenses," in *Peer-to-Peer Computing, 2006. P2P 2006. Sixth IEEE International Conference on*, 2006, pp. 171–176. [Online]. Available: http://www.crhc.uiuc.edu/ nikita/publications.html

[11] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2003, pp. 640–651.

[12] P. Michiardi and R. Molva, "CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks," in *CMS'2002: Communication and Multimedia Security 2002*, Aug 2002.

[13] B. N. Levine, C. Shields, and N. B. Margolin, "A survey of solutions to the sybil attack," University of Massachusetts Amherst, Amherst, MA, Tech report 2006-052, October 2006. [Online]. Available: http://prisms.cs.umass.edu/brian/pubs/levine.sybil.tr.2006.pdf

[14] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *Proc. of the 9th ACM Conference on Computer and Communications Security*, Washington, DC, USA, November 2002.

[15] M. Kinateder and S. Pearson, "A privacy-enhanced peer-to-peer reputation system," in *Proceedings of the 4th International Conference on Electronic Commerce and Web Technologies (EC-Web 2003)*, ser. LNCS, K. Bauknecht, A. M. Tjoa, and G. Quirchmayr, Eds., vol. 2738. Prague, Czech Republic: Springer-Verlag, Sep. 2003, pp. 206–215. [Online]. Available: ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2003-18/INPROC-2003-18.pdf