University of Zurich
Department of Informatics

Thomas Bocek
David Hausheer
Reinhard Riedl
Burkhard Stiller

Introducing CPU Time as a Scarce Resource in P2P Systems

TECHNICAL REPORT — No. 2006.01

January 2006

University of Zürich
Department of Informatics (IFI)
Winterthurerstrasse 190, CH—8057 Zürich, Switzerland

Thomas Bocek, David Hausheer, Reinhard Riedl, Burkhard Stiller:
Introducing CPU Time as a Scarce Resource in P2P Systems
Technical Report No. 2006.01, January 2006
Communication Systems Research Group
Department of Informatics (IFI)
University of Zürich
Winterthurerstrasse 190, CH—8057 Zürich, Switzerland
URL: http://www.ifi.unizh.ch/csg/

Introducing CPU Time as a Scarce Resource in P2P Systems

Thomas Bocek[1], David Hausheer[1], Reinhard Riedl[2], Burkhard Stiller[1,3]

[1]Communication Systems Group, Department of Informatics IFI, University of Zurich, Switzerland
[2]Information Systems Group, Department of Informatics IFI, University of Zurich, Switzerland
[3]Computer Engineering and Networks Laboratory TIK, ETH Zürich, Switzerland

E-Mail: [bocek|hausheer|riedl|stiller]@ifi.unizh.ch

Abstract

Peer-to-peer (P2P) systems are flexible, robust, and self-organizing resource sharing infrastructures which are typically designed in a fully decentralized manner. However, a key problem of such systems are freeriders, i.e. *peers overusing a resource. One approach to solve this problem is by introducing a central element taking care of resource accounting. Another approach is the use of a trust management system where resource usage information is collected and evaluated by peers. Finally, the last approach covers resource trading schemes which limit resource usage.*

This paper presents a fully decentralized scheme against freeriding in P2P systems, which does not require a priori information about a peer. The approach developed is based on a scarce resource trading scheme (SRTCPU) which utilizes CPU time as a form of payment. Thus, providing incentives to provide CPU time in return of consuming a scarce resource. A distributed Domain Name System (DNS) has been implemented as an example application that uses the new trading scheme SRTCPU.

1 Introduction

An economic market is based on the exchange of goods and services. One of its properties is that these goods and services are scarce. Overuse is limited in theory by the supply and demand principle. If goods are requested in larger quantities the price will rise and only those who are willing to pay the higher price will obtain these goods. If the resources are not scarce the price is independent of the demand, and a low as well as a high demand will result in the same price. Thus, overuse is not limited by the supply and demand principle. This problem affects peer-to-peer (P2P) systems when requesting and supplying resources.

A P2P system is fault-tolerant, robust, and usually does not require any special administrative arrangements [3]. Currently, several P2P infrastructures exist which can be used for large-scale P2P applications, *e.g.*, CAN [23], Chord [29], Pastry [26], or Tapestry [31]. P2P applications based on these P2P systems cannot only suffer from malicious nodes that may stop a P2P application to operate by deploying a sibyl attack [10] or by pseudo spoofing [9], but also suffer from overusing resources [7].

A node in a P2P system has typically a limited amount of resources that it may share with other nodes, *e.g.*, bandwidth, storage space, or central processing unit (CPU) power. However, if these resources are overused by other nodes an imbalanced load is created so that nodes which actively contribute to the network are punished [14]. To achieve a fair allocation, every node should be given the possibility to use a resource in the same way as it provides resources. Three different accounting concepts can be used to achieve such a fair allocation:

- A ***central element*** can account for resource usage of every node in a P2P system. However, a central element is not a good option as the specific flexibility and robustness properties introduced by a decentralized P2P system would be weakened.
- A ***decentralized collection of information*** about a node's behavior is an accounting concept that requires cooperation. Nodes collect information about resource usage and behavior of other nodes and provide other nodes with this information. Thus, decentralized collection of information is cooperative because the information is provided partly by third party nodes.
- ***Decentralized resource trading*** allows nodes to be independent from other nodes. Resource trading denotes the concept that a node receives a resource in return for providing a resource [6]. Resource trading is fully decentralized, as a node can trade with its own resources and can decide on its own whether or not to provide resources. Thus, the node is independent of other nodes.

In this paper, a fully decentralized resource trading scheme is developed, which is called SRTCPU (Scare Resource Trading with CPU Time). The scheme prevents overuse of resources in a decentralized system by introducing a payment for services, which is based on calculations using CPU time, thus, providing incentives to provide CPU time in return of consuming a scarce resource. It maintains a self-regulated, self-organized, and sustainable resource exchange allowing for a fair resource allocation.

Although other scarce resources, such as bandwidth or storage space, could have been used, however, SRTCPU focuses on CPU time. The major reason for this decision is that, *e.g.*, the use of storage space as a scarce resource would only provide benefits to the users if data is stored for a period of time. However, the final control mechanism for storage over a period of time is much more complex, although the concept would be similar. Therefore, as a reference, a possible mechanism for storage trading is described in [20]. The node storing data randomly picks nodes claiming to have replicas. After picking a node, it asks for a hash of an arbitrary block of the data. The node can only return the correct hash value if the data has been stored.

The SRTCPU approach proposed is evaluated using naming and directory services as examples. The traditional DNS

[18] serves as an example of a large-scale centralized system. Thus, an extension, a Distributed Domain Name System (DDNS) prototype has been built that overcomes the resource overuse problem and operates in a decentralized manner.

In general, a DNS in decentralized manner — thus a DDNS — will suffer without any countermeasures a Denial-of-Service (DoS) problem, which can be solved by applying the developed SRTCPU approach. To detail this solution, it is important to remember that in general naming and directory services can be seen as a list of so-called name value pairs. For example, the name in the DNS is a domain name and the value is an IP address. Usually, such a name and its value have a size of not more than some bytes, *e.g.*, *www.unizh.ch* and *130.60.68.124*. Although a first approach in developing a name service using P2P networks was presented in [7], the key problem described there is the problem of *insertion denial of service*, which can be considered as a resource overuse problem. This problem deals with the possibility of a massive insertion of name value pairs. A node reserving every possible name combination in an endless loop could induce a halt in the network operation as new names can no longer be inserted. In contrast, within the DNS it is not possible to reserve such a large number of name combinations unless a huge amount of money is spent, since every single name has a price to be paid for. Therefore, the SRTCPU scheme applied acts as a payment for name insertion in a DDNS. However, unlike crypto puzzles, which determine a specific type of calculation in order to keep the CPU busy, SRTCPU can use the CPU for any type of calculation. This enables to contribute CPU time to, *e.g.*, a grid or any project that is in need of computational power.

The remainder of this paper is organized as follows. While Section 2 compares related work, Section 3 designs SRTCPU. Section 4 discusses the implementation of the Distributed DNS prototype and validates key requirements. Finally, Section 5 summarizes and discusses future work.

2 Related Work

To prevent overuse of a resource in a decentralized system three concepts can be applied: introduction of a central element accounting for the use of resources, symmetric and asymmetric resource trading, and decentralized collection of information about resource usage.

A review of related work has revealed a number of different approaches tackling the problem of resource overuse. Key characteristics taken into account cover the level of decentralization and the dependency on other nodes. These characteristics clearly distinguish the three concepts. Consequently, related work is discussed and compared according to the following dimensions:

- **Existence of a central element**. This indicates whether a central element is present or not.
- **Identification of fair resource usage based on local data.** Trust is good, control is better. A decision taken based on local data judging if a node is overusing a resource is always better then to trust other, potentially malicious nodes.
- **Efficient resource usage**. Counting back a hash is an exhaustive calculation. CPU time cannot be contributed to calculating anything else, *e.g.,* SETI@home [27].
- **Resource symmetry.** Resource symmetry denotes the trading with the same kind of resources, for example bandwidth in exchange for bandwidth.

2.1 Comparative Dimensions.

Distributed systems applying decentralized or centralized schemes do show different behaviors. Thus, a comparative study as depicted in Figure 1 has been performed. The notion of *cooperation* has been introduced to denote a high degree of dependency. In cooperative systems, *e.g.*, transitive trust management, a node is dependent on more nodes than in independent systems. For example in Tit-For-Tat (TFT), a node can make a decision about sharing a resource based on the interaction between itself and another node.

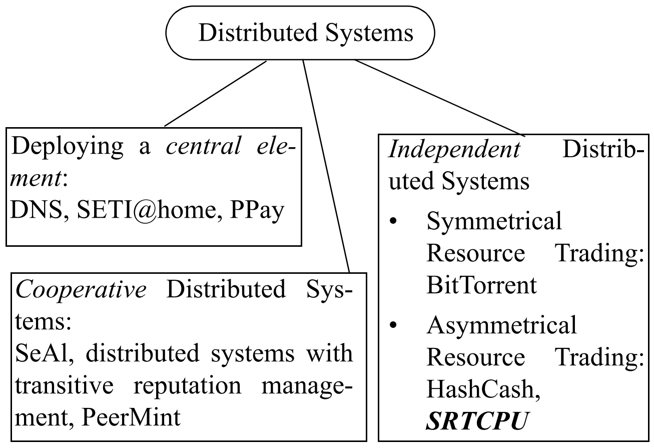
Distributed Systems
Deploying a *central element*: DNS, SETI@home, PPay
Cooperative Distributed Systems: SeAl, distributed systems with transitive reputation management, PeerMint
Independent Distributed Systems
Symmetrical Resource Trading: BitTorrent
Asymmetrical Resource Trading: HashCash, ***SRTCPU***

FIGURE 1. Classification of related work

Resource trading can be divided into symmetric and asymmetric resource trading. Symmetric resource trading happens when the same kind of resource is traded between two nodes, *e.g.*, if one node requests *x* data sets from a second node the first node has to provide the second node with *x* data sets too. Bittorrent [5] is an example of symmetrical resource trading with bandwidth taking place. However, symmetric resource trading only works if the first node is interested in the resource of the second node.

If symmetric resource trading is not possible another resource available for trading has to be found. Asymmetric resource trading happens when one type of resource is traded for a different type of resource, *e.g.*, storage space in return for CPU power. This implies dealing with exchange rates for trading different resources with different values. In a decentralized system, no supervisor or resource allocator is present. Therefore, decentralized resource trading has to happen in a completely self-regulated and self-organized way.

2.2 Deployment of Central Elements

The first group of approaches deals with centralization.

2.2.1 Centralized Accounting

Authentication, authorization, accounting, auditing, and charging (A4C [28], as well as A^x [24]) can be used to limit resource overuse. With A4C, usually used by service providers (SP), a central element is introduced to keep track of resource usage. Also new nodes can be tracked and controlled in a central manner. Charging is an important component, which must be present to limit service usage. Charging a high price will result in a lower demand and a lower price will result in a higher demand. This way, a SP can steer demand and limit resource usage. Without a charging component, resource usage can only be accounted for and blocked by the SP if the usage is exceeding a limit. This limitation can be undermined if a user is able to have multiple identities.

However, in a P2P system with a decentralized design, introducing a central element weakens the specific flexibility and robustness properties introduced by a decentralized system.

2.2.2 Payment and Micropayment with Digital Cash

Payment with real money as, *e.g.*, in the DNS shows that an overuse can be prevented. However, this is only possible with a central element [30]. PPay [30] defines a micropayment protocol for P2P systems that achieves a great reduction of communication with the central broker issuing the money. Hence, in PPay there is still a central element, and it does, therefore, not comply with the requirement of a fully decentralized system. Even when reducing the traffic with the central broker a shutdown of a broker can stop the system to operate. Micropayment with digital cash as described for various schemes in [9] cannot be applied either, because in a decentralized system, using transferable cash is not feasible without a central element [30].

2.3 Cooperative Distributed Systems

The second group of approaches deals with cooperation among components of a distributed system.

2.3.1 Trust Management

Trust management as outlined in [1], [8], and [15] can be described as collecting and evaluating information about a node's behavior, such as download and upload statistics or amount of shared objects. The evaluation of collected data represents the trustworthiness of a node. [11] and [15] uses the notion of transitive trust management. A node that collects data from a neighbor tells other neighbors about its findings. These neighbors tell their neighbors about those findings. A transitive characteristic in the scheme shows that eventually all nodes will share the same information. However, this may take a long time, as propagating this information depends on the number of nodes within a system.

Trust management can only be used to limit resources when limited joining is allowed. Without this limit, a malicious user can undermine any limit of a resource by inserting many new nodes and whitewash the rating with a new identity. When using CPU time as a scarce resource the limit cannot be bypassed by inserting a new node, because the limit is bound to the CPU. Requesting many resources with many nodes needs the same CPU time as requesting the same resources with just one node.

2.3.2 Decentralized Accounting

PeerMint [13] is a decentralized approach to account for resource usage. An additional charging scheme can be set up to limit the usage. Main technical characteristics, as shown in Figure 2, are third-party nodes, so-called mediators, which keep track of resources traded during an exchange, and peer account holders who hold the complete account data of a node.

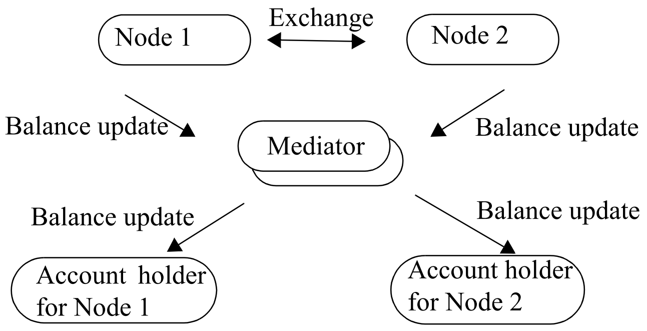

FIGURE 2. Basic principle of PeerMint.

If two peers exchange a resource the mediators keep track of this exchange and the resulting resource bill is transferred to the peer's account holders. Redundancy of account holders provides a guarantee to a certain degree that the account data is preserved, even if an account holder leaves the system or behaves maliciously. The mediators are determined based on a secure hash calculation, and can thus not be freely chosen by the nodes.

SeAl [21] proposes another decentralized accounting mechanism. A management/accounting layer collects receipts of transactions and these receipts can be traded (as favors). The trading is enforced, so a peer cannot be selfish. However, SeAl can only trade symmetric resources and favors are passed to other nodes in a cooperative way.

2.4 Independent Distributed Systems

The third group of approaches deals with independency of multiple components within a distributed system.

2.4.1 Tit-for-Tat

Tit-for-tat (TFT) is a strategy used for example in BitTorrent to discourage freeriders and provide incentives to upload data at a high rate [5]. A participant in a BitTorrent network wanting to download data has to offer data for upload as well. TFT is not suitable in systems exchanging different kind of resources.

2.4.2 Hashcash

Hashcash [2] is a way to use CPU time as a scarce resource. The purpose of hashcash is to prevent spam mailings being sent. Figure 3 shows the basic principle of hashcash to count back a hash to its original value. The recipient of an e-mail sends an MD5 hash of a random value to the sender. The sender has to count back the original value from the hash, which takes some time and uses far more CPU time than creating the hash. Afterwards, the value is sent back to the recipient. If the transferred value matches the random value, the e-mail is accepted. Thus, a spammer wanting to send a huge amount of e-mails runs out of CPU power. The CPU time is used to limit the amount of e-mail messages being sent to the public.

By re-labeling the sender to node *a* and the recipient to node *b* in Figure 3, the hashcash method can also be used to prevent the overuse of a resource in a P2P system. Before a resource such as bandwidth or storage space can be used on node *b*, node *a* has to count back a hash. After confirmation of the correct result, node *a* may use the bandwidth or storage space. However, the CPU time is only used for an exhaustive calculation to find the original value from a hash. SRTCPU uses a similar approach, but with the difference that the CPU time can be used for any type of calculation. In SRTCPU it is not important *what* has been calculated, but *that* something has been calculated correctly.

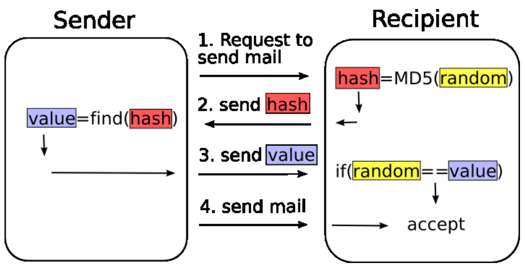

FIGURE 3. Basic principle of hashcash [2].

In order to prevent a parallelization a time crypto puzzle as described in [25] could be applied. This computational task cannot be parallelized because the task is “intrinsically sequential” [25]. However, using parallel resources can be tolerated as this also applies to the economic market. As an example, the DNS also allows the usage of parallel resources, i.e. to have more names registered in the DNS, more money has to be paid. A similar scheme is applied to SRTCPU: To register more names in DDNS more CPU power has to be used.

2.5 Comparison

The supply and demand principle applies when trading with scarce resources. All of the mentioned related work is based on that principle and all related work is using a scarce resource. Trust management and central accounting use a central element to make a resource scarce. The others use the scarce resource CPU time or bandwidth.

With respect to SRTCPU, TFT, and hashcash no central elements are necessary and they are not dependent on data from other nodes for verifying fair resource usage. With SRTCPU, the CPU time can be used for arbitrary calculations, *e.g.*, for solving large-scale computation problems as in SETI@home [27] by creating incentives to offer CPU time. In contrast, hashcash can only keep the CPU busy with a very specific computational task without any freedom of choosing an arbitrary calculation. This can be seen as a waste of CPU time.

In contrast, hashcash keeps the CPU busy with a very specific computational task without any freedom of choosing an arbitrary calculation. This can be considered as a waste of CPU time. Finally, in contrast to TFT, no resource symmetry is required for SRTCPU. Thus, SRTCPU defines an optimal approach to prevent the overuse of a resource in a fully decentralized system.

The full comparison of those approaches investigated is shown in Table 1, where a “+” indicates the presence of the characteristic, while a “-” indicates its absence.

TABLE 1. Comparison of different strategies for resource trading.

	No central element	Identification of fair resource usage based on local data	Efficient resource usage	No resource symmetry required
Tit-for-tat	+	+	+	-
Trust Management	-[a]	-	+	+
Hashcash	+	+	-	+
Centralized Accounting	-	-	+	+
PeerMint	+	-	+	+
SRTCPU	+	+	+	+

a. Trust Management has to limit the number of possible identities of a client. [10] show that without resource trading, a limitation is only possible with a central element.

3 Example-driven Design

Based on the investigation of related work, the key requirements for SRTCPU have been derived. Therefore, the detailed design of SRTCPU includes the set of mechanisms in charge of sending computational tasks, which needs CPU time to solve, in exchange for another resource to become as scarce as the CPU time. The basic design of such a use of CPU time includes a randomly chosen third party acting as a task provider. The payment for the resource does not happen in monetary form, as, *e.g.*, in the DNS, but in terms of CPU time. Binding a resource to a second resource, which is scarce, makes the first resource as scarce as the second. This means for SRTCPU that, *e.g.*, the storage of a name is as expensive as calculating a computational task.

In case of Internet Service Providers, the trading of bandwidth, *e.g.*, for a certain period of time, implicitly takes place with storage trading. However, in the area of trading bandwidth with an arbitrary data transfer — also with non-persistent data — more research has to be performed. A first step into this direction is PeerMart [12], a distributed auction platform. Another step utilizes reputation management systems, which could take the bandwidth into account as well.

3.1 Technical Prerequisites and Requirements

To describe the operation of SRTCP’s core idea, key prerequisites and requirements are outlined.

A node ID (identifier) identifies a node, and SRTCPU relies on the fact that a node ID cannot be freely chosen. A prevention of arbitrary node ID selection can, *e.g.*, be achieved by constructing the node ID based on the IP address [29]. A second possibility includes the use of a central element, as proposed by [10], to assign node IDs. SRTCPU uses the first method, i.e. a node ID cannot be freely chosen and all nodes available in the system are able to verify that this precondition holds.

The following 7 requirements have been defined to enable the design of a fully decentralized asymmetric resource trading scheme which will meet the key goals of a distributed system: (a) efficiency and scalability with respect to the number of resources being dealt by and (b) the number of users utilizing the scheme developed:

- **Fair**. Provide a fair allocation of resources, i.e. prevent resource overuse.
- **Fully decentralized**. No central elements should be present.
- **Load balanced**. A node should not become a bottleneck.
- **Fault tolerant**. A failure of nodes should not affect the service.
- **Self-organized**. The scheme is able to adapt to changes in the network.
- **Efficient**. The scheme should consume as little resources as possible.
- **Trustworthy**. In order to reduce the risk of attacks, minimal trust relations should be deployed.

3.2 Main Challenge

The main challenge of SRTCPU's architectural design is to ensure that a peer has calculated the task by itself. Without this assurance the node could re-label the task and send it as a task provider to another node. To prevent such negative behavior a checksum of what has been calculated is being determined. The checksum is calculated using those instructions that have been performed and the node ID of the node that has commissioned the task. A malicious node that re-labels a task and sends it back as a new task will receive a different checksum from that re-labeled task compared to the original task. The node which received the re-labeled task will use the node ID from the malicious node to calculate the checksum. As stated as a technical prerequisites, a node ID cannot be freely chosen and other nodes can verify that.

Another concern may be that a node, which is sending a task, has to verify that the calculation received is correct. *E.g.*, Hashcash offers a direct and fast verification, because the result of calculating back the hash must be the same as the original value. In SRTCPU, however, it is necessary to introduce redundancy to verify the correctness of a calculation. This means the same calculation is sent to more than one node and those results received from these nodes are compared. As a consequence, some overhead is created.

3.3 Communication Design

To enable a view on the sequence of actions and the interoperation between peers involved in an SRTCPU-based scheme, the following examples are provided and discussed to achieve an efficient communication paradigm.

Three participants can be identified in Figure 4: A task provider that needs a calculation to be outsourced, a requesting node (node 2) and the node that provides a resource. (node 1). In general, the number of requesting nodes is denoted *r*, the percentage of malicious nodes is *m*.

The communication starts with node 2 requesting a resource (1). In order to get that resource node 2 must be willing to provide CPU time for that resource. Node 1 sends this request to a task provider (2). Node 2 receives the task (3) indicating that this task was requested from node 1 and node 2 calculates the task. During the task, a checksum is generated, including the node ID of node 1 and the instructions that have been executed in the task. The result and the checksum are sent back to the task provider (4). Node 1 receives the checksum from node 2 and the task provider (5), (6). This allows node 1 to determine if the task provider has received the result. If the checksum is not received from the task provider, an error has occurred either in node 2 or in the task provider. Node 1 will mark both in a local list as potentially malicious. If a certain threshold in this list has been reached because a node continues to behave incorrectly that node will not be considered in the future.

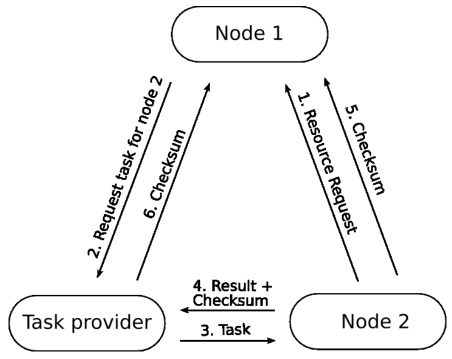

FIGURE 4. The communication design with three participants (r = 1, m = 0).

The task provider that announced himself to node 1 must also compute a task to get listed. Otherwise a task provider could create many new nodes with the result that other task providers would be excluded from providing tasks. The task provider will be selected randomly by node 1. A task provider can register on any node.

The communication design with three participants and $r = 1$ is not possible, if a malicious node sends arbitrary data and checksums back. Therefore, more nodes ($r > 1$) have to be introduced. In Figure 5 with $r = 2$, node 4 has been introduced that is calculating the same task as node 3. It is obvious that a task must be deterministic. Node 1 receives r checksums that must be equal. If they are not, one node has made a mistake and both are marked as potentially malicious. By reaching a certain threshold, a malicious node can be detected. This scheme works only when $m < 0.5$ in an ideal P2P system.

Node 1 is also able to send a computational task, if no task provider is available or $r = 1$ with $m > 0$. The task looks similar to hashcash, calculating back a hash, i.e. node 1 sends node 2 a hash value of an initial value. Node 2 has to calculate back a possible initial value for that hash value.

Figure 5 shows the network communication design with $r = 2$. It is also possible to have $r > 2$. With more checksums sent back, node 1 can determine faster if a node has behaved maliciously. However, with more nodes, more overhead is created.

With the introduction of $r > 1$ a certain amount of malicious nodes cannot overuse resources in the system since these malicious nodes will be detected and ignored. The amount of malicious clients that can be present in the system without overusing resources can vary from none to any, depending on the scheme used. When using a crypto puzzle the amount of malicious nodes is irrelevant, while no malicious client may be present in the schema with $r = 1$ (cf. Figure 2). SRTCPU could be customized by varying the schemes to address different requirements in decentralized networks (e.g. every 5th task is a crypt puzzle).

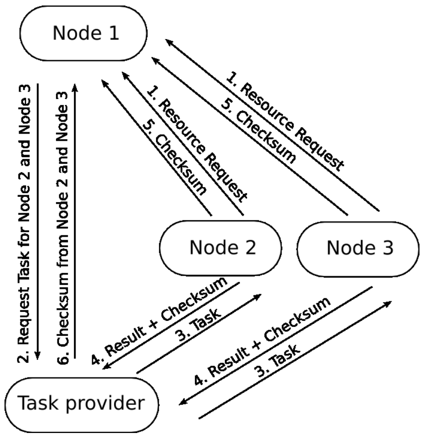

FIGURE 5. Extended communication design with four participants; 2 nodes (node 2, node 3) have received the same task (r = 2)

3.4 Self-Regulating Task Difficulty

A resource can be paid for by calculating a task. The amount of payment is defined by difficulty of the task which is an important factor in making a resource scarce. If the difficulty is set too low overuse will be the result, if it is set too high, underuse will happen. Along with every checksum sent back, the number of processed instructions is also sent to indicate if more computation is needed.

Node 1 knows how much node 2 and node 3 have calculated by indicating the number of processed instructions that has been returned to node 1. Node 1 now needs to determine the average CPU time per inserted name value pair in the system to keep the system self-regulated. In this way, the system keeps track of an increase in CPU power over time [19].

Node 1 can determine the average CPU time with a benchmark on itself, assuming to have an average CPU processor. A tolerance factor will be applied to allow for a wider range of different CPU power. The tolerance factor determines how homogeneous — in terms of CPU processing speed — the system can be. This is an important factor, since not every node has the same CPU power. Small and embedded devices have usually lower CPU power. However, SRTCPU focuses mainly on homogeneous systems, where a similar amount of resources are available on every node.

To calculate the average CPU time per inserted data, node 1 must also know the average amount of data stored in the P2P system. Again, the node assumes that the stored data is the average. With the average CPU time and the average stored data, node 1 can decide, if node 2 and node 3 have calculated sufficiently enough to be allowed to store the data on node 1. Hence, the system is self-regulated.

4 Implementation and Evaluation

DDNS and SRTCPU have been implemented to validate the feasibility of a decentralized naming/directory service by using SRTCPU.

All calculations have been executed in a virtual machine, which has also been implemented in Java and performs mathematical tasks only. For the sake of simplicity, a prototypical MathVM [4] has been implemented in Java. The MathVM can update the checksum on every processed instruction, it uses its own language, and it has an assembler that translates the source code into instructions (opcodes). A task description in a MathVM has a fixed size of several kilo byte, but a single task description has been implemented to simulate a crypto puzzle. Therefore, the implementation will only work without any malicious nodes, which does not prevent a proof of concept. DDNS is based on a Distributed Hash Table (DHT) with an XOR (exclusive or) metric, similar to [16], providing a good performance for any lookups in *O(log n)* time, where *n* is the number of nodes considered. The prototype has been tested with 1000 nodes to validate that the search function scales gracefully [4].

4.1 DDNS Efficiency Analysis

The two most important commands in DDNS are *get* and *store*. A *get* is a data lookup that takes *O(log n)* time in a structured overlay network [16]. A *store* performs first a lookup in *O(log n),* in order to find the node, where to store the data. For all necessary details of the communication protocol, please refer to [4].

Let *n* be the number of participating nodes. It is assumed that the number of queries scales with *O(1)*, which means that the number of queries a user performs are constant over time. The analysis of the *store* and *get* commands shows in the following that the message complexity remains *O(log n)* as in the Kademlia network [16].

The *get* command has been modified for SRTCPU to obtain always more than one single result, but this does not affect its performance of *O(log n).* Based on the fact that the number of queries *p(n)* is *O(1)*, the overall number of messages, which equals *p(n) * O(log n)*, will also be *O(log n)*. The *store* command has also been modified. Every *store* performs a *get* to decide, if the name to be inserted exists. For every *store* command the respective message complexity is *p(n) + (p(n) * O(log n))*. Again *p(n)* is *O(1)* and the resulting efficiency is *O(log n)*.

Even if *p(n)* were *O(log n)*, the system with the overall complexity of $O(\log n)^2$ remains. However, with *p(n) = O(n)* the complexity would be *O(n * log n)*, which is too high. But on the other hand, with the same situation that *p(n) = O(n),* the complexity of the DNS would be *O(n) * O(1) = O(n)*, which is also too high for a scalable system. Thus the efficiency achieved is highly comparable with existing systems.

As a result of this analysis, it can be concluded that all modifications do not exceed *O(log n)* and the lookup remains scalable. Nevertheless, some optimizations still can boost the performance. One optimization includes a clustering approach similar to SHARK [17]: The system is divided into parts. A part is built by grouping similar nodes. Similar in this context means that nodes searching the same names are similar. Let *p(n)* be the amount of nodes which searches within its parts and let *q(n)* be the amount of nodes which searches in other parts. Having constant part sizes and growing numbers of parts, a *get* command in such a part is *O(1)*. The *get* command for searching inside and outside of a part is *p * O(1) + q * O(log n)*. The complexity remains *O(log n)*, but if a lot of searches are within a single part, the *get* command becomes faster. The second optimization addresses caching, where the same situation arises. When caching a large number of entries the *get* command becomes faster, but there will also be many uncached names, consequently the system remains *O(log n)*.

4.2 DDNS Experiment

The prototype implemented has been tested and several simulation experiments with up to 100 nodes per machine have been made. In total 10 separate Pentium III Machines with 1.6 GHz and 256 MByte RAM have been used to simulate a smaller network. These experiments have confirmed those results of the theoretical efficiency analysis as discussed above in Section 4.1.

The result for the number of messages in total to be exchanged for the commands *get* and *store* are depicted in Figure 6. This figure indicates that the system communication grows with *O(log n)*. The system in this simulation run stored 15 names and made 100 *get* requests. The *get* requests were made once and the *store* commands were repeatedly called thereafter. After every name had been stored and 100 requests had been made, the run was aborted. The numbers of nodes tested are 10, 20, 60, 100, 150, 200, 500, and 1000. In each step, 2 runs have been made and the mean has been calculated. A node can fail with a probability of 10%. No malicious nodes have been used ($m = 0$), no task providers are present and $r = 1$.

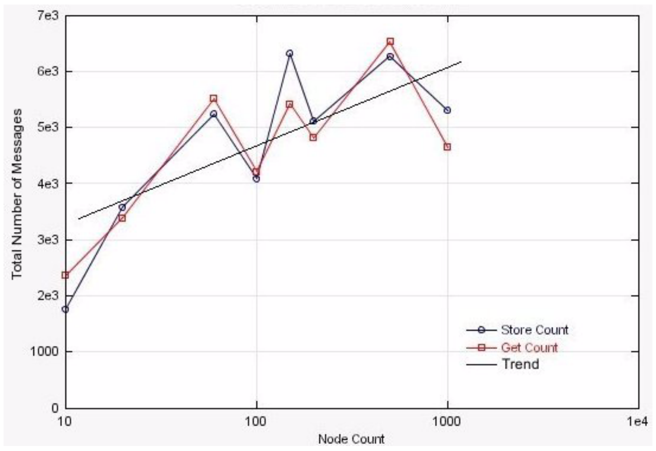

FIGURE 6. Simulation experiments of the prototype.

Due to the implementation of the prototype with just one thread per communication channel, only a small number of nodes (ca. 150 nodes) has been tested per machine.

4.3 Validation and Discussion

Key requirements discussed above have been met.

- **Fair**. Fairness is achieved since everyone has to pay about the same amount of CPU time for a resource. The amount of CPU time to be paid is self-organizing. Resource overuse is limited through the introduction of SRTCPU offering the possibility of trading different types of resources. In DDNS, a name value pair is traded for CPU time.
- **Fully decentralized**. SRTCPU is fully decentralized. No central element is necessary to account for resource usage.
- **Load balanced and fault tolerant**. Due to redundant storage a failure does not affect the scheme and the load is balanced. Additionally, a caching mechanism in DDNS optimizes the balance especially with popular names.
- **Self-organized**. With the use of CPU time to prevent overuse of a resource, every node can decide how much to charge for its resource. A central element is not necessary and a self-regulating task difficulty establishes a balanced charge of CPU time with respect to increasing processing power (Moors' law [19]) over time. Therefore, even when the resources change over time the system itself remains self-organized.
- **Efficient**. Although CPU time has to be spent, SRTCPU adds a benefit over a crypto puzzle by making the CPU time available for other calculations as well.
- **Trustworthy**. Trust can be measured based on local data. Thus, minimal trust relations have been achieved.

A smaller number of further issues remain for discussion. A potential problem arises when the network is very heterogeneous, i.e. with a lot of different CPU capabilities. [10] states that *large-scale distributed systems are inevitably heterogeneous.* Especially embedded and small devices have usually fewer resources to allocate. However, if a participation in a system requires certain resources the problem of heterogeneity is less relevant. If a node wants to join the network, it has to make sure that it has sufficient resources. If there are not sufficiently large the node cannot join and has to be upgraded, for example with more storage or a faster processor. The final result is that every SRTCPU participant will have a similar amount of resources available.

Furthermore, efficiency is limited by the possible presence of malicious nodes and the absence of nodes for the verification of a computational task. Overhead is created as SRTCPU has to send a task to multiple nodes to verify the result. Therefore, the calculation is not as efficient as in a supervised system. A fallback strategy similar to hashcash is used when just one node requests a resource. In this case, the CPU time cannot be contributed to calculate anything else. Details need to be investigated further.

Additionally, the bandwidth of a node, with which it is connected to the network, may become a bottleneck, if the task description is very large. Therefore, in the DDNS prototype and its experiments, the task description was limited to a fixed size. Without this limitation, the required bandwidth would be considered as part of the payment.

Finally, although in the traditional DNS the uniqueness of names can be guaranteed, this is not possible in a fully decentralized system, since all peers behave autonomous.

However, these ambiguities in DDNS may exist only for a limited period of time: If two names are inserted at the same time, but from different peers, the name propagated faster will prevail [4]. Thus, no following concern remains.

5 Summary and Future Work

Since Peer-to-peer (P2P) systems are flexible, robust, and self-organizing resource sharing infrastructures which are typically designed in a fully decentralized manner, the key problem of freeriders, i.e. peers overusing a resource, have to be solved. In general, three strategies can be applied to prevent the overuse of a resource in a decentralized system: (a) introducing a central element, (b) symmetric and asymmetric resource trading, and (c) decentralized collection of information about resource usage. The approach SRTCPU (Scare Resource Trading with CPU Time) introduced in this paper uses asymmetric resource trading, since this can be designed and implemented in a fully decentralized manner. Anyone who wants to use a critical resource has to pay for it with CPU time.

This approach taken can be generalized. Introducing CPU time as a scare resource in a P2P system can make other resources as scare as the CPU time by binding these two resources together. As an example application of SRTCPU, Distributed Domain Name System (DDNS) has been implemented. DDNS is an architectural concept of a decentralized naming service based on a P2P network. It implements SRTCPU to protect the critical resource *storage space for name value pairs*. The names can be retrieved in *O(log n)* due to the underlying Distributed Hash Table. With the introduction of task providers, one can use the CPU time, resulting from name insertions, for any task. It could also be used in a grid, to solve large-scale computation problems similar to SETI@home. Contrary to the altruistic donation of CPU time to large-scale computation communities, DDNS creates an incentive to provide CPU time to a grid. The feasibility of DDNS has been validated, its efficiency has been analyzed, and a prototypical implementation has been performed as well as evaluated.

Future distributed architectures may be based on other resources than CPU time. For example, a decentralized web page system including a naming service could make storage space and bandwidth available as the scarce resource. In order to store a name of a web page, storage space would have to be provided for other nodes.

To achieve a detailed understanding of the impact of malicious nodes and to obtain further data from the simulation, the simulation experiments are expected to be extended toward a testbed of a larger number of machines. This would include the varying of the parameter r and the introduction of malicious nodes into the system.

Acknowledgments

This work has been performed partially in the framework of the EU IST Network of Excellence EMANICS “Management Solutions for Next Generation Networks” (FP6-2004-IST-4-026854-NoE).

References

[1] K. Aberer, Z. Despotovic. *Managing trust in a peer-2-peer information system*. H. Paques, L. Liu, and D. Grossman (edts): Tenth International Conference on Information and Knowledge Management (CIKM’01), New York, USA, November 2001, pp 310—317.

[2] A. Back. *Hash cash — a denial of service counter-measure*, URL: http://www.hashcash.org/papers/hashcash.pdf, August 2002.

[3] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica. *Looking up data in P2P systems*. Communications of the ACM, Vol. 46, No. 2, February 2003, pp 43—48.

[4] T. Bocek. *Feasibility, Pros and Cons for Distributed DNS,* Master Thesis, University of Zürich, Department of Informatics IFI, May 2004.

[5] B. Cohen. *Incentives Build Robustness in BitTorrent*. 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, California, U.S.A., June 2003.

[6] Brian F. Cooper, Hector Garcia-Molina, *Peer-to-Peer Resource Trading in a Reliable Distributed System*, Lecture Notes in Computer Science, Volume 2429, January 2002, pp 319—327.

[7] R. Cox and A. Muthitacharoen, R. Morris, *Serving DNS using a peer-to-peer lookup service*, 1st International Workshop on Peer-to-Peer Systems (IPTPS’02), Cambridge, Massachusetts, U.S.A., March, 2002.

[8] E. Damiani, De Capitani di Vimercati, S. Paraboschi, P. Samarati, F. Violante. *A reputation-based approach for choosing reliable resources in peer-to-peer networks*. 9th ACM Conference on Computer and Communications Security, Washington DC, U.S.A., November 2002, pp 207-216.

[9] R.Dingledine, M.Freedman and D.Molnar, “Accountability”, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'REILLY Press, 2001, Chapter 16, pp.271—340.

[10] J. R. Douceur. *The sybil attack*. 1st International Workshop on Peer-to-Peer Systems (IPTPS’02), Cambridge, Massachusetts, U.S.A., March 2002.

[11] M. Feldman and K. Lai and I. Stoica, J. Chuang, *Robust Incentive Techniques for Peer-to-Peer Networks,* ACM Electronic Commerce, New York, U.S.A., May 2004.

[12] D. Hausheer, B. Stiller, *Decentralized Auction-based Pricing with PeerMart,* 9th IFIP/IEEE International Symposium on Integrated Network Management, Nice, France, May 2005.

[13] D. Hausheer, B. Stiller, *PeerMint: Decentralized and Secure Accounting for Peer-to-Peer Applications,* IFIP Networking 2005, Waterloo, Ontario, Canada, May 2005.

[14] S. Kamvar, M. Schlosser, and H. Garcia-Molina. *Incentives for Combatting Freeriding on P2P Networks*. International Conference on Parallel and Distributed Computing (Euro-Par), Klagenfurth, Austria, August 2003.

[15] S. D. Kamvar, M. T. Schlosser, H. Garcia-Molina. *The eigentrust algorithm for reputation management in p2p networks.* Twelfth International World Wide Web Conference (WWW), Budapest, Hungary, May 2003.

[16] P. Maymounkov and D. Mazieres. Kademlia: *A peer-to-peer information system based on the xor metric*. 1st International Workshop on Peer to Peer Systems (IPTPS’02), Cambridge, Massachusetts, U.S.A., March 2002.

[17] J. Mischke, B. Stiller: *Rich and Scalable Peer-to-Peer Search with SHARK*, Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03), Seattle, Washington, U.S.A., June 2003.

[18] P. Mockapetris, *Domain Names - Concepts and Facilities*, Request for Comments: 1034, Network Working Group, November 1987.

[19] Moore’s Law, URL: http://www.intel.com/technology/mooreslaw/index.htm, August 2005.

[20] T.-W. J. Ngan and D. S. Wallach and P. Druschel. *Enforcing fair sharing of peer-to-peer resources*. 2nd International Workshop on Peer-to-Peer Systems (IPTPS), Berkeley, California, February, 2003.

[21] Nikos Ntarmos, Peter Triantafillou. *SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Network*s. Fourth International Conference on Peer-to-Peer Computing (P2P'04), Zürich, Switzerland, August 2004, pp 116—123.

[22] OASIS UDDI, *UDDI Technical Committee Draft,* URL: http://uddi.org/, August 2005.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. *A scalable content addressable network*. ACM SIGCOMM, San Diego, California, U.S.A., August 2001.

[24] C. Rensing, Hasan, M. Karsten, B. Stiller: *AAA: A Survey and a Policy-based Architecture and Framework;* IEEE Network Magazine, Vol. 16, No. 6, November/December 2002, pp 22—27.

[25] R. L. Rivest and A. Shamir, D. A. Wagner, *Time-lock Puzzles and Timed-release Crypto,* Technical Report, MIT Laboratory for Computer Science, 1996.

[26] A. Rowstron, P. Druschel. *Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems*. 18th IFIP/ACM Conference on Distributed Systems Platforms, Heidelberg, Germany, November 2001.

[27] SETI@home, URL: http://setiathome.ssl.berkeley.edu/, August 2005.

[28] B. Stiller, J. Fernandez, Hasan, P. Kurtansky, W. Lu, D.-J. Plas, B. Weyl, H. Ziemek, B. Bhushan: *Design of an Advanced A4C Framework,* Whitepaper, EU IST Project Daidalos, http://www.ist-daidalos.org/, August 2005.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: *A scalable peer-to-peer lookup service for internet applications*. ACM SIGCOMM, San Diego, California, U.S.A., March 2001, pp 149-160.

[30] B. Yang, H. Garcia-Molina. P*Pay: Micropayments for Peer-to-Peer Systems*. Technical report, Stanford University, 2003.

[31] B. Y. Zhao, J. D. Kubiatowicz, A. D. Joseph. *Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing*, Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.