Improving the Throughput of Distributed Hash Tables Using Congestion-Aware Routing *

Fabius Klemm, Jean-Yves Le Boudec, Dejan Kostić, Karl Aberer
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Abstract

Advanced applications for Distributed Hash Tables (DHTs), such as Peer-to-Peer Information Retrieval, require a DHT to quickly and efficiently process a large number (in the order of millions) of requests. In this paper we study mechanisms to optimize the throughput of DHTs. Our goal is to maximize the number of route operations per peer per second a DHT can perform (given certain constraints on the lookup delay). Each peer receives congestion feedback from the DHT, which it uses to adjust its routing decisions. This way, peers can avoid routing through slow parts of the overlay network and hence increase the rate at which they insert new messages into the DHT. We provide a numerical analysis of congestion-aware routing in DHTs and show that considerable improvements in throughput are possible compared to DHTs with proximity neighbor selection and strictly greedy routing.

1 Introduction

Distributed Hash Tables (DHTs), such as Chord [13], Pastry [12], P-Grid [1], and many more, provide a scalable means to map identifiers (ids) to socket addresses (i.e. IP address and port). The basic operation is $route(id, data)$, which routes $data$, e.g. queries, inserts, updates, to a peer responsible for id.

Peer-to-peer (P2P) environments have rapidly changing performance characteristics as peer client software often shares CPU and network resources with other processes running on the same machine. Moreover, churn in the DHT causes routes to change often, which entails further performance variations.

The goal of this paper is to study mechanisms to increase the throughput of DHTs by quickly adapting routing according to performance changes in the overlay network. Until now, most DHTs perform greedy routing in the id space, i.e. peers always select the closest neighbor to the searched (destination) id when forwarding messages. So far, the goal was to reduce the number of hops to resolve an id, without taking the routing capacity of peers in the overlay network into account. Proposals to select neighbors taking proximity in the underlying network into account can avoid building extremely inefficient routes, e.g. routing back and forth between different continents. However, these proposals are unable to react to rapid performance changes in a DHT.

Optimizing the routing performance of DHTs is an important requirement for advanced applications, such as P2P Information Retrieval (P2P-IR). In P2P-IR, a DHT has to quickly process a large number of, usually fairly small, messages. Some P2P-IR approaches, such as those presented in [7, 8], require a DHT to handle in the order of millions of messages in a short period of time during indexing and searching.

The paper is structured as follows: we first review related work in section 2. We then describe the P2P environment and explain how peers adapt their message insertion rates to the currently available routing capacity of a DHT (section 3). Section 4 introduces congestion-aware routing to increase the throughput of a DHT. Each peer monitors congestion feedback, which it receives when inserting new messages into the DHT. This feedback allows peers to quickly adapt routing to avoid overloaded parts of the DHT. We present a numerical analysis and an evaluation of our algorithm in section 5. Finally, we provide a discussion of our work in section 6 and conclusions in section 7.

2 Related Work

For recursive routing in DHTs, there has been very few work that focuses on maximizing the routing throughput as the main measure of performance. [3] proposes mechanisms to reduce latency and increase throughput for the Chord DHT, however, it concentrates on data delivery.

*The work presented in this paper was carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European FP 6 STREP project ALVIS (**002068**).

Other studies introduce methods for exploiting network proximity in DHTs, e.g. [2]. One option is proximity routing, as proposed for CAN [11] and Chord [3], where each peer considers both progress towards the searched id in the space *and* physical proximity when forwarding a message. This mechanism, as well as proximity neighbor selection during the construction of a DHT, work well for relatively stable performance characteristics. However, they cannot handle short-lived performance changes, which are very likely to be stronger in P2P than in controlled environments.

Load balancing algorithms proposed for DHTs, e.g. [10], assure that a peer's capacity (e.g. storage) corresponds approximately to its responsibility in the id space. These algorithms, however, do not consider routing load and are therefore orthogonal to our work.

3 Description of the Environment

We now introduce the basics of Distributed Hash Tables (DHTs) in section 3.1 and congestion control in DHTs in section 3.2.

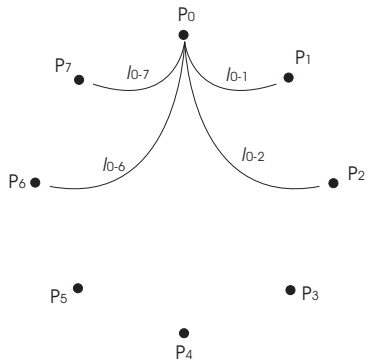

Figure 1: Ring topology with bidirectional links shown for peer P_0.

3.1 Routing in DHTs

We use a ring topology as shown in figure 1. With n peers, each peer maintains connections to $O(log(n))$ other peers (called neighbors) in the network ($2 \cdot [log_2(n) - 1]$ in our case). In principle, we build the routing tables as in other popular $O(log(n))$ DHTs (i.e. [1, 12, 13]): the probability that a peer P_x chooses a peer P_y as routing entry is inverse proportional to the distance between P_x and P_y in the id space, i.e. P_y is more likely to be chosen if it is close to P_x. The resulting network has small-world properties [4, 5]. Routing is done recursively and in a greedy fashion: when a peer receives a message for an id for which it is not responsible, it chooses the neighbor closest to the searched id to forward the message. At any peer, a route request for any id reaches a responsible peer with only $O(log(n))$ overlay hops. Two peers communicate using TCP. All links are therefore bidirectional.

3.2 Congestion Control in DHTs

As peers in a DHT relay messages, there can be congestion if new messages are inserted at a rate faster than the DHT can handle. Limiting factors are the CPU and network capacities of peers. To avoid a congestion collapse, load-intensive applications, such as P2P-IR, have to adapt message insertion rates to the current routing capacity of a DHT.

We use additive increase, multiplicative decrease (AIMD) congestion control (CC), as presented in [6]. Note that this CC avoids congestion in the *overlay network* and is thus independent of TCP CC between two neighboring peers, which takes care of not overloading the underlying IP network.

CC for DHTs is similar to TCP using ECN [9] and works as follows: each message routed in the DHT has an explicit overlay congestion notification (ECN_o) bit in its header, which is initially set to 0. When a message is relayed by a congested peer, the ECN_o bit is set to 1 to indicate congestion on this path. For each received message a responsible peer returns an *overlay acknowledgement* (ack_o) to the initiator of the message (either as direct UDP packet or routed in the DHT). This ack_o contains the information whether the ECN_o bit in the message header was set when arriving at the responsible peer. The ack_o also indicates that a message successfully arrived at the responsible peer. Note that overloaded peers drop messages. In case of a missing ack_o, a peer can reinsert messages. A peer increases or decreases its insertion rate according to the ECN_o feedback as explained in the following subsections.

3.2.1 Notation

We use the following notation:

p	Prob. of the ECN_o bit set
x	Message insertion rate per peer
c_{pos}	Constant for additive increase
c_{neg}	Constant for multiplicative decrease

3.2.2 Adaptation of Insertion Rates

In case of positive feedback from the overlay network, a peer increases its message insertion rate x in the following way:

$$x = x + c_{pos},\ c_{pos} > 0$$

In case of negative feedback a peer decreases its rate x as follows:

$$x = x \cdot c_{neg},\ 0 < c_{neg} < 1$$

The constants c_{pos} and c_{neg} determine the aggressiveness of a source and thus depend on the environment. We shall see in section 5.3 how they affect the throughput.

With p_t being the probability of a peer receiving negative feedback (ECN_o bit set) at time t, we can calculate the change Δ_t of the rate x_t for a peer:

$$\begin{aligned} \Delta_t &= p_t \cdot (-x_t \cdot c_{neg}) + (1 - p_t) \cdot c_{pos} \\ x_{t+1} &= x_t + \Delta_t \end{aligned} \tag{1}$$

4 Congestion-Aware Routing

In this section we first introduce our congestion-aware routing algorithm. We then describe an analysis to calculate potential throughput gains for DHTs in section 4.2.

4.1 Overview

The basic idea is to loosen the rule in greedy routing of strictly selecting the neighbor closest to the searched id when forwarding a message: each peer considers the k closest neighbors (if closer than itself to the searched id) and monitors their performance. When inserting new messages into the DHT, each peer tracks the feedback it receives (carried in overlay acknowledgements as explained in section 3.2) to evaluate the option of routing through certain neighbors. The information which neighbor a peer had taken to forward a new message is stored together with a number identifying each outstanding message (e.g. ack_o seq. no.).

To simplify the presentation, we take the case $k = 2$, i.e. a peer considers the closest and second-closest neighbor when forwarding a message. Table 1 shows the routing options P_0 has when forwarding messages (cf. figure 1). Take the third line, for example: for destination P_3, the first choice of P_0 is to route via P_2 using link l_{0-2} and the second choice is via P_1 using link l_{0-1}. If the next hop is responsible for the searched id, there is no second choice (N/A). If two neighbors are equally distant to the searched id (e.g. to P_4), we choose the left-hand neighbor as first option.

destination	1st choice	2nd choice
P_1	l_{0-1}	N/A
P_2	l_{0-2}	N/A
P_3	l_{0-2}	l_{0-1}
P_4	l_{0-2}	l_{0-6}
P_5	l_{0-6}	l_{0-7}
P_6	l_{0-6}	N/A
P_7	l_{0-7}	N/A

Table 1: Routing options for peer 0

4.1.1 Adaptive Routing

Each peer observes the ECN_o feedback it receives when using the first or second choice for routing. Therefore, for each pair of routing choices, each peer maintains k (here $k = 2$) *observed probabilities* (op_i) of the ECN_o bit set in the acknowledgement as shown in table 2 for P_0.

pair	1st choice	2nd choice
$(l_{0-2} - l_{0-1})$	op_1	op_2
$(l_{0-2} - l_{0-6})$	op_1	op_2
$(l_{0-6} - l_{0-7})$	op_1	op_2

Table 2: Options with observed probabilities

These probabilities are the only state each peer has to maintain for congestion-aware routing. It grows $O(log(n))$ per peer and thus scales well with the number of peers in the network.

4.1.2 Shifting Traffic

We now come to an important part in increasing the throughput of a DHT: shifting traffic between the considered neighbors for each routing decision. Each peer considers up to k possible options to forward a message. We first look at the case $k = 2$. As seen in table 2, each peer tracks observed probabilities op_1 and op_2 of negative feedback for all routing options. Depending on the observed probabilities, a peer sends a fraction f_1 of the traffic via the first choice and a fraction f_2 via the second choice ($f_1 + f_2 = 1$). A peer updates f_1 and f_2 in the following way (cf. eq. 2): δ is the amount of traffic shifted from the first to the second option. It is the difference of the two observed probabilities times a constant ϕ, which determines how fast a peer adapts to changes. If δ is negative, traffic is shifted back to the first option. Each fraction is at least f_{min}. Therefore, each peer receives feedback for both options of each pair.

$$\begin{aligned} \delta &= (op_1 - op_2) \cdot \phi \\ f_1 &= f_1 - \delta,\ f_2 = f_2 + \delta \\ f_{min} &\leq f_i,\ f_j \leq 1 - f_{min} \end{aligned} \tag{2}$$

For $k > 2$ we first calculate op_{mean}, the mean of all observed probabilities op_i. We then increase or decrease each f_i depending on how far the according op_i is under or over op_{mean}.

A peer updates its observed probabilities whenever initiating new messages. When a peer forwards messages from other peers, it splits the traffic according to the calculated

fractions for the different choices. Thus, on each hop, an optimization choice is made, overall leading to higher throughput.

Whenever a peer does not choose the closest neighbor for forwarding a message, the number of routing hops necessary to reach a responsible peer will increase. However, if the routing tables are large enough, the number of hops will only slightly increase as we also show in the numerical analysis in section 5.

4.2 Analysis

We now describe an analysis of congestion-aware routing to calculate the throughput gains we can achieve. We show how to calculate the message insertion rates per peer for a DHT with given capacities when AIMD congestion control is used. The following subsections use the notation in table 3.

P_i	Peer i
x_i	Message insertion rate of peer i
l_{ij}	Link between P_i and P_j
x_{ij}	Traffic of link$_{ij}$
c_{ij}	Capacity of link$_{ij}$
p_{ij}	Probability of link$_{ij}$ setting an ECN$_o$ bit

Table 3: Notation

4.2.1 Calculation of p

Each peer observes the probability p of receiving negative feedback (i.e. an ECN$_o$ bit set). p is the average of the feedback received in a certain time frame. It depends on the paths the traffic generated by a peer uses and the link capacities and loads on these paths.

A link l_{ij} has a capacity c_{ij} of messages it can process per time unit. A peer sets an ECN$_o$ bit in the header of a message for link$_{ij}$ with probability p_{ij}, which depends on the load x_{ij} of the link, and its capacity c_{ij}. We use the following formula to set $0 \leq p_{ij} \leq 1$. The more x_{ij} approaches c_{ij}, the faster p_{ij} goes to 1:

$$p_{ij} = 1 - \sqrt{1 - \frac{x_{ij}}{c_{ij}}},\ 0 \leq x_{ij} \leq c_{ij} \tag{3}$$

To calculate the probability p_i of negative feedback for a peer P_i, we have to take into account how much of the traffic generated by P_i flows through each link l_{ij}. We assume that searched ids are uniformly randomly distributed, which is valid after certain load balancing, randomized hashing, and caching techniques have been applied. Thus, each peer P_i generates x_i/n fresh traffic for each of the n peers in the network. We can thus calculate x_{ij} and p_{ij} for all links, which gives us p_i for all peers.

4.2.2 Rate Calculation

To calculate the insertion rate x_i per peer, we perform the following iteration until the change Δ_t of the rate of eq. 1 is small (e.g. $< \varepsilon$).

Initialization: All peers start with rate $x_i = 0$. All link traffic x_{ij} is thus zero and hence $p_{ij} = 0$. The link capacities c_{ij} are initialized according to a chosen network topology (cf. section 5.1).

Iteration steps:

1. For each peer P_i, calculate the probability p_i of ECN$_o$ bits set (avg. over all destinations) when sending requests at rate x_i as described in 4.2.1 and using eq. 3. If congestion-aware routing is enabled, for each peer and all neighbor pairs, calculate the traffic shift as shown in eq. 2.

2. Given p_i, a peer changes its rate x_i by $\Delta_{i,t}$ according to eq. 1. If $\forall i, \Delta_{i,t} < \varepsilon$, stop the iteration. Otherwise, continue with step 3.

3. After the insertion rates x_i for all peers have been changed, the traffic x_{ij} for all links is updated according to the fraction of traffic each peer sends through each link. Start again with step 1.

5 Numerical Analysis of an Example

We have seen in section 4 how to calculate the message insertion rates for all peers. We now present an example with 256 peers. Each peer has 14 bidirectional links to peers chosen as described in section 3.1. We first discuss different strategies to construct a locality-aware DHT and show how we model link capacity variations. We then study the performance of congestion control and congestion-aware routing.

5.1 Locality-Aware DHTs

We initialize the DHT as follows: Each peer has a uniformly random chosen coordinate in a 5-dimensional space. The link capacity between two peers decreases with increasing distance in the space. We construct the DHT using the following locality scenarios:

NL: No Locality Some DHTs do not provide proximity neighbor selection. In this case, we build a random ring among the peers without taking proximity into account and choose the routing entries as described in section 3.1.

L1: Locality-aware I We simulate a locality-aware join algorithm, which builds the ring in such a way that physically close peers tend to be close on the ring.

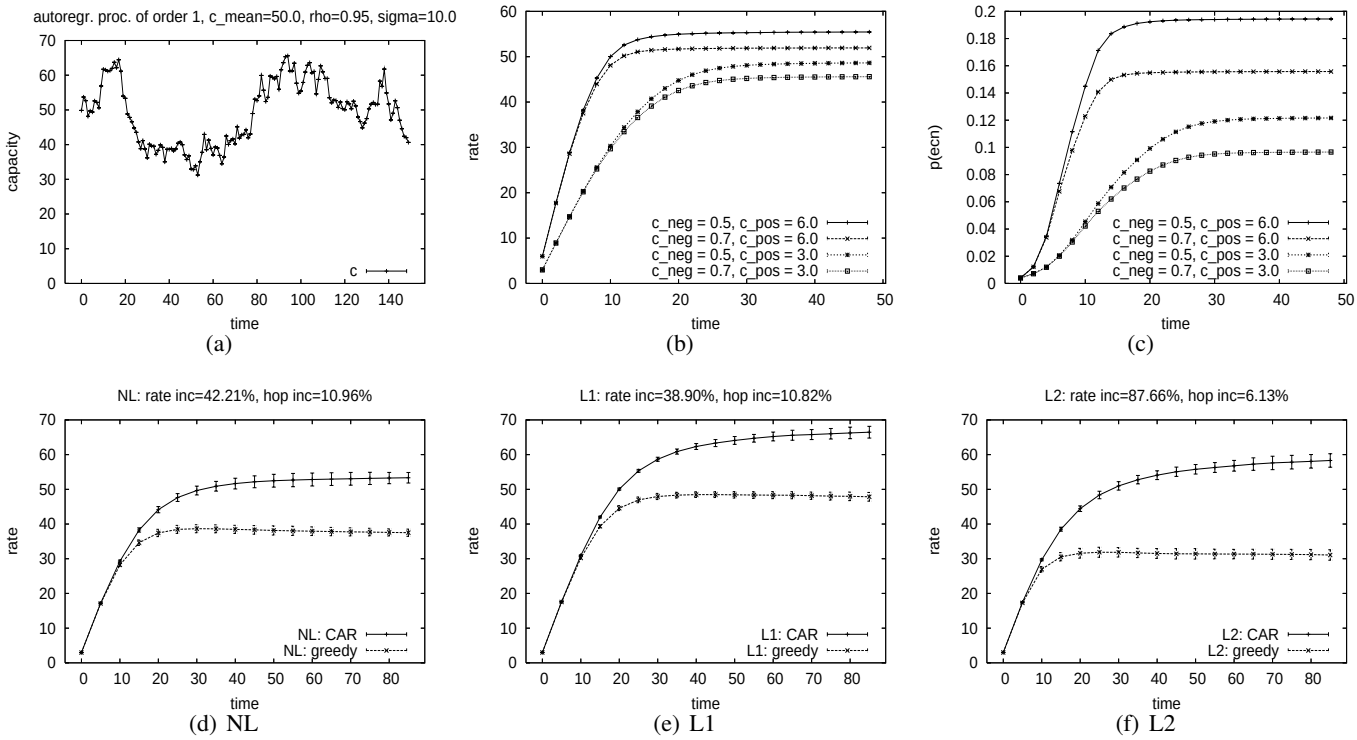

Figure 2: Example of an autoregressive process (a), analysis of AIMD (b-c), and congestion-aware routing (CAR) (d-f)

L2: Locality-aware II We simulate the algorithm for selecting routing entries as proposed in [2], where peers select physically close *long-range* links. In consequence, direct neighbors on the ring tend to be physically distant.

5.2 Modeling Of Varying Link Capacities

We change the link capacities over time according to an autoregressive process of order one, which is rich enough to allow us to model correlation of changes, mean capacity, and standard deviation. The link capacity at time t is c_t, which changes as follows:

$$c_{t+1} = \rho \cdot c_t + (1-\rho) \cdot \bar{c} + \sqrt{1-\rho^2} \cdot N \cdot \sigma,$$

where ρ the correlation, $\bar{c}$ the mean capacity, N a normally distributed random variable with mean 0 and standard deviation 1, and σ is the standard deviation of c_t. Figure 2(a) shows an example for mean link capacity $\bar{c} = 50$, correlation $\rho = 0.95$, and standard deviation $\sigma = 10$.

5.3 Effects of c_{pos} and c_{neg}

We first study the functionality of the AIMD congestion control algorithm and the influence of its constants. We use a stable network, i.e. the link capacities are not changing. The choice of the topology has no effect as the goal of this experiment is to study the influence of c_{pos} and c_{neg} of section 3.2.2. Figure 2(b) shows the change of the average peer insertion rate x, which stabilizes (according to eq. 1) between 40 and 60 message insertions per peer per time unit. Then, $8 - 20\%$ of overlay acknowledgements have the ECN bit set (figure 2(c)). A larger c_{pos} lets a peer quicker find the available capacity as the increase for positive feedback is stronger. c_{neg} determines the aggressiveness of a peer: a smaller value lets a peer back-off less in case of negative feedback. Thus rate and p are higher for larger c_{pos} and smaller c_{neg}.

5.4 Benefits of Congestion-Aware Routing

Figures 2(d) to 2(f) compare the achieved throughput (i.e. avg. insertion rate x) for three locality scenarios (NL, L1, and L2). For the model of varying link capacities we choose $\rho = 0.95$, $\sigma = 0.2 \cdot \bar{c}$, where $\bar{c}$ is set according to the locality scenario. Furthermore, the AIMD constants are $c_{pos} = 4$ and $c_{neg} = 0.5$. We compare greedy routing with congestion-aware routing (CAR). For CAR, we set $k = 2$, i.e. we consider two options for adaptive routing. The plots show the increase of the avg. insertion rate over time. We also calculate the increase in hops, which is caused when CAR does not choose the neighbor closest to the searched id but the second option. As the throughput depends on the randomly chosen

topology and random link capacity variations, we present the average of five runs with mean deviation error bars.

In all three locality scenarios, CAR can considerably increase the throughput, while only moderately increasing the hop count. Locality scenario L1 shows the best performance, as progress towards the target id in the id space directly entails progress in the physical network. The low performance of L2 first seems surprising. The reason is that L2 introduces imbalance in the number of routing entries each peer has. Therefore, some peers become routing bottlenecks. However, CAR manages to avoid these bottlenecks and thus achieves considerable performance gains in L2.

6 Discussion

The analytical model shows that considerable improvements are possible with congestion-aware routing. However, the model does not cover all aspects: first, as congestion feedback is returned in overlay acknowledgements, it is delayed by 0.5 to 1 round trip time (RTT). Second, this feedback delay is not stable as it depends a) on the destination and b) on varying queuing delays, which are not considered in our model. Furthermore, for a good estimate of the probability of negative feedback p, a peer has to perform a sufficient number of requests per time unit. Despite these shortcomings, we believe that congestion-aware routing will lead to substantial gains in a real implementation.

Our model does not account for latency in the DHT. Latency is largely influenced by queuing delays in congested peers. Our algorithm avoids congested peers and is thus expected to decreases queuing delays. Furthermore, the choice of c_{pos} and c_{neg} of AIMD in section 3.2.2 determines the aggressiveness of peers, and thus the number of outstanding messages. As shown in section 5.3, these constants can be used to trade-off between high throughput and low delay.

The strength of our scheme is that it allows peers to quickly adapt to performance changes in the overlay network, without generating extra messages, and requiring only few additional state to be maintained.

7 Conclusions

This work presents the first analysis of throughput in DHTs. We are aware that it is in an early state. However, we believe that such an analysis is an essential first step for augmenting DHTs with mechanisms to increase the throughput, an important requirement for demanding DHT applications. Our analysis shows that adaptive routing using explicit congestion notifications can potentially achieve significant improvements.

The immediate next steps of this work include testing the performance in a prototype implementation. Furthermore, the model can be easily extended to take peer processing capacities (e.g. CPU) into account.

Some open research questions for future work are: what is the influence of the number of considered routing choices? How can we take network dependencies of overlay links into account? After introducing some simplifications, is a theoretical analysis of the throughput of a DHT with millions of peers possible? Can other joint congestion control and routing algorithms from the networking domain be applied in DHTs?

8 Acknowledgements

We would like to thank Wojciech Galuba and Vasilios Darlagiannis for their helpful comments.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Sixth International Conference on Cooperative Information Systems*, 2001.

[2] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables. In *International Workshop on Future Directions in Distributed Computing (FuDiCo)*, 2002.

[3] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2004.

[4] S. Girdzijauskas, A. Datta, and K. Aberer. On small world graphs in non-uniformly distributed key spaces. 2005.

[5] J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.

[6] F. Klemm, J.-Y. Le Boudec, and K. Aberer. Congestion control for distributed hash tables. In *The 5th IEEE International Symposium on Network Computing and Applications (IEEE NCA06)*, 2006.

[7] J. Li, T. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. On the Feasibility of Peer-to-Peer Web Indexing and Search. *IPTPS03*, 2003.

[8] I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Scalable Peer-to-Peer Web Retrieval with Highly Discriminative Keys. In *IEEE 23rd International Conference on Data Engineering (ICDE)*, 2007.

[9] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ecn) to ip. rfc 2481, 1999.

[10] A. Rao, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica. Load balancing in structured p2p systems. In *IPTPS*, pages 68–79, 2003.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.

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