Congestion Control for Distributed Hash Tables [*]

Fabius Klemm, Jean-Yves Le Boudec, and Karl Aberer
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
{fabius.klemm, jean-yves.leboudec, karl.aberer}@epfl.ch

Abstract

Distributed Hash Tables (DHTs) provide a scalable mechanism for mapping identifiers to socket addresses. As each peer in the network can initiate lookup requests, a DHT has to process concurrently a potentially very large number of requests. In this paper, we look at congestion control for DHTs. Our goal is to control the flow of lookup requests that are routed in the overlay network. We first show that congestion control is essential for certain applications with high lookup rates. We then present two congestion control mechanisms for DHTs and compare their performances in different network conditions.

1 Introduction

Distributed Hash Tables (DHTs) are used to efficiently store and lookup data in a peer-to-peer (P2P) network. Data is addressed by an *identifier* (or *key*) from a common identifier space. Each peer in the network is responsible for a subset of identifiers. For increased reliability, multiple peers can be responsible for the same identifiers. The main operation of a DHT is *lookup(id)*, which returns socket addresses (i.e. the IP addresses and port numbers) of responsible peers for id. Each peer maintains a routing table to forward requests it is not responsible for. Most DHTs, such as Chord, Pastry, or P-Grid [11, 9, 1] create routing tables of size $O(log(n))$, where n is the number of peers in the system. The routing entries are chosen in such a way that the resulting graph has small-world properties [5, 4]. Resolving any lookup request started at any peer is then guaranteed to take $O(log(n))$ overlay hops on average.

There are two ways of resolving lookup requests in a DHT: when *recursive routing* is applied, a peer that is not responsible for a requested identifier id_x forwards the request to a next hop found in its routing table. We call a peer that forwards a request a *relay peer*. This forwarding process continues until a responsible peer for id_x is found, which returns a reply to the requester (i.e. the initiator of the lookup request for id_x). Whereas, in *iterative routing* instead of forwarding a lookup request, a relay peer returns the socket address of the next hop to the requester. The requester then contacts the next hop directly. This process continues until the requester knows the address of the responsible peer for id_x.

A lookup request contains the socket address of the requester to allow the responsible peer to return a direct reply. However, in some cases due to firewalls, Network Address Translation (NAT), or link failures, direct IP communication between two peers is not always possible. In such cases, the reply is relayed back to the requester by one or several intermediate peers.

In this paper, we study the problem of congestion control (CC) for DHTs. Dealing with CC for DHTs might not be necessary if only a moderate number of messages is routed in the system: for file sharing, for example, a peer would typically provide some hundreds to a few thousands of files, which are indexed using only a few keywords. Bursts in indexing traffic could be avoided by gradually inserting the index. Further routing traffic is caused by user queries that are very likely to stay well below one query/s per peer and are therefore not heavily loading the network. In such a scenario, a DHT without any CC mechanisms will work acceptably most of the time.

In the past years, however, the usage of DHTs has passed simple file sharing. A large research community is now working on providing full-text retrieval on top of DHTs, e.g. [6, 12]. Performing information retrieval on top of a DHT (P2P-IR) drastically increases the load of the system by at least two orders of magnitude, compared to file sharing. In P2P-IR each peer extracts several thousands of keys from its local document collection to be inserted into the DHT for indexing. We will show in section 3 that such a load will lead to a congestion collapse of a DHT if no precautions are taken. When a DHT has to handle a very large

[*]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**).

number of requests, CC is essential to guarantee the correct and efficient functioning of the system.

In this paper, we will present two CC mechanisms for recursive routing, Credit System Congestion Control (CSCC) in section 2.1 and Back-Pressure Congestion Control (BPCC) in section 2.2. CSCC detects packet loss to determine the routing capacity of a DHT, whereas BPCC propagates congestion state towards sending peers, which then react accordingly. We will explain how both mechanisms can be used in current DHT implementations. In section 3 we will analyze the performance of both mechanisms. In sections 4 and 5 we present related work and our conclusions.

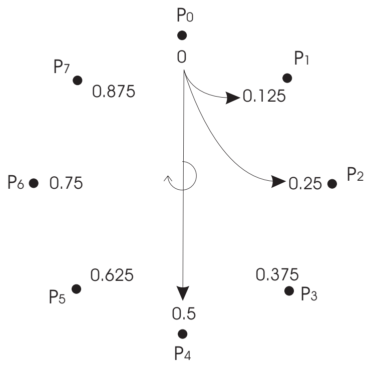

Figure 1. Ring topology

2 Congestion Control for DHTs

Controlling the flow of lookup requests in a DHT becomes necessary when there is heavy load and the peers in the DHT have different processing capacities. Consider figure 1: Assume that peer P_0, for example, sends messages to peer P_4 with rate λ_1 and peer P_4 processes messages with rate λ_2. If $\lambda_1 > \lambda_2$ the incoming queue of peer P_4 will start to fill up. Once the queue limit is reached there are two options: a) P_4 drops further incoming messages or b) P_4 informs sending peers (here P_0) that it temporarily cannot accept more messages. We will use option a), which is similar to TCP/IP, for CSCC (section 2.1) and option b), which is referred to as back-pressure CC, in our second mechanism called BPCC (section 2.2). Both options have been studied in the computer networking domain for end-to-end traffic flows. Our proposals for CC in DHTs re-use successful principles whenever applicable. We shall see, however, that there are many unique features of CC in DHTs.

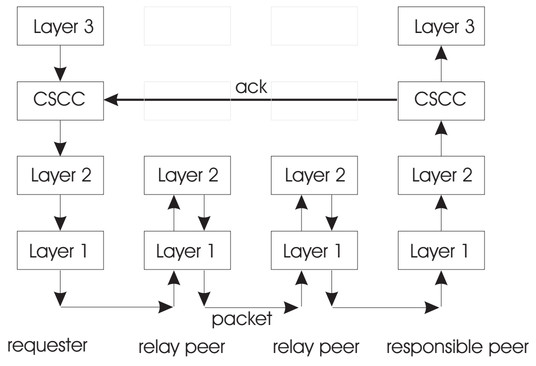

Figure 2. Architecture with CSCC

2.1 Credit System Congestion Control (CSCC)

For ease of explanation, we will structure a peer into multiple layers (cf. figure 2). Layer 1 provides communication between two neighboring peers, which is done using TCP. Layer 2 performs recursive routing in the overlay network, i.e. it decides to which neighbor messages are forwarded. Applications using the DHT, such as information retrieval, are on layer 3.

Our first congestion control mechanism, CSCC, works on top of overlay routing, i.e. between layer 2 and 3. It regulates the insertion rate of lookup requests initiated at layer 3. CSCC passes messages to layer 2, where they are (recursively) routed via several relay peers to the responsible peer (cf. figure 2). At the responsible peer, for each received message, CSCC returns an acknowledgment (ACK) *directly* to the socket address of the requester. In our implementation an ACK contains only a 4 byte ACK id. We therefore chose UDP as the transport protocol to send the ACK. Opening a TCP connection to the requester to return the ACK would be overkill, in particular because in a large network nearly every ACK goes to a different requester and thus the chances of reusing an open TCP connection are very low. Another option would be to route the ACK back using the DHT. This option is more expensive ($O(log(n))$ messages), however it might be necessary in certain environments, where direct IP communication is limited (e.g. by NATs and firewalls). In all cases the ACK messages can be used to piggyback small replies, e.g. the socket address of the responsible peer. Notice that all peers in the system are requester, relay, and responsible peer at the same time.

Each peer buffers incoming messages in a queue that is situated in layer 2. For CSCC, once a specified queue limit is reached, further incoming messages are dropped silently (as in an IP router). Neighbor-to-neighbor communication on layer 1 is implemented using TCP, i.e. all messages are reliably transmitted to the next hop. However, a forwarding peer will not take notice whether a message is dropped at the next hop. TCP CC between two neighbors deals with congestion in the IP network and is therefore independent of CSCC.

2.1.1 Credits

The CSCC layer at each peer maintains credits that represent outstanding messages, i.e. requests for which the peer has not received an acknowledgment yet. CSCC does not maintain any destination-specific state. The number of available credits represents the capacity of the whole DHT. For setting the credits, we use the standard textbook algorithm for setting a TCP send window: for each received acknowledgment, the requester will increase its credits c as follows:

```
if (c < ssthresh) c := c + 1;
else c := c + 1 / c;
```

ssthresh is the slow start threshold. CSCC changes c and ssthresh in case of detected packet loss:

```
if (c > ssthresh) ssthresh := 0.8 * c;
else ssthresh := 0.8 * ssthresh;
c := 5;
```

When packet loss occurs, ssthresh is set to 0.8 * c if c had passed ssthresh, otherwise its decreased by multiplication with 0.8. c is set to 5 to instantly relieve congestion. As long as c < ssthresh, c will quickly increase. Once ssthresh is reached, CSCC limits the speed of growth of c to 1/c for each acknowledged message.

2.1.2 Retransmission Timeout

CSCC retransmits a message if it has not been acknowledged within a certain time. We maintain a RTT estimate (RTT_est) that is the exponential average of measured RTTs. We also estimate the RTT variance (errorEst). The timeout is the estimated RTT plus 10 times the estimated error, which empirically gives a good tradeoff between timely detection of loss and low number of unnecessary early retransmissions.

```
RTT_est := 0.875 * RTT_est + 0.125 * last_RTT;
errorEst := 0.75 * errorEst + 0.25 * lastErr;
timeout := RTT_est + 10 * errorEst;
```

Notice that these constants to increase and decrease the credits, as well as to calculate the RTT are empirically chosen for the environment we will present in section 3. A more detailed study about the effects of changes of these numbers in different environments is part of future work.

2.1.3 Discussion

CSCC has similarities with CC in TCP. The most important difference is that TCP is used for long-lasting flows between two end systems. In DHTs, such long-lasting overlay flows do not exist, as lookup requests are for different destinations. Given two different identifiers, a peer does not know whether two or only one peer is responsible. Once a requester knows the socket address of a responsible peer P_x for an identifier, it can communicate directly with P_x (with UDP or TCP) without having to use the DHT. As TCP is end-to-end and delivers messages in sequence, it uses a sliding window: the reception of an acknowledgement increases the set of messages that can be sent only if the acknowledgement includes the *oldest* outstanding message. CSCC, on the contrary, cannot guarantee in-order delivery and therefore uses a credit system. It can send a new message whenever an outstanding one has been acknowledged.

A requester retransmits a message when it does not receive an acknowledgement within a certain time. Peers can therefore receive the same message several times. To detect duplicate messages, each peer would have to keep track of which messages were received from which peer. The cost of keeping this sender-specific state would be *O(n)*. CSCC therefore does not guarantee that each message is delivered exactly once to the application layer but only at least once (in the absence of exceptional errors). Exactly-once semantics has to be implemented by the application if needed.

2.2 Back-Pressure Congestion Control (BPCC)

We will now introduce our second CC mechanism using back-pressure. When back-pressure is used, no messages are dropped at layer 2. Instead, a peer signals its current queue state to sending peers. Congestion in the DHT thus propagates back to requesting peers that adjust their lookup rates accordingly. Back-pressure requires neighbor-to-neighbor flow control that can be implemented by using TCP at layer 1. When the queue of a relay peer P_r is full, instead of dropping packets, it will stop reading packets from the TCP sockets of incoming connections. TCP flow control will automatically slow down all peers forwarding messages to P_r.

2.2.1 Risk of Deadlocks

One problem in a back-pressured network is the risk of deadlocks. A sending peer is blocked when the incoming queue at the receiver is full. It has to wait for the receiving peer to process queued messages. However, the receiving peer itself might be waiting for another peer. A deadlock can occur when there is a cycle in the buffer waiting graph. Consider again routing in the Chord-ring [11] of figure 1: assume that P_0 is sending packets to P_5 via P_4, P_4 is sending to P_7 via P_6, and P_6 to P_1 via P_0. This scenario can lead to a deadlock, when P_0 is waiting for P_4, P_4 is waiting for

P_6, and P_6 is waiting for P_0. In a distributed environment such deadlocks are extremely difficult to detect and resolve.

2.2.2 Deadlock-Free Back-Pressure

A known principle in packet-switched networks is that a routing scheme is deadlock free if and only if there are no cycles in the buffer waiting graph [10, 7]. There have been many proposals for routing in DHTs, e.g. [11, 9, 1], many of which build a ring or a tree topology. We will show in the following that for ring and tree topologies it is possible to obtain a deadlock-free overlay network if each peer has an individual queue per neighbor. Each peer thus maintains $O(log(n))$ queues for outgoing messages. As before, a sending peer is blocked when the queue it inserts into (at the next hop) is full. However, other sending peers can still insert messages into the remaining queues if there is space.

2.2.3 Ring

Ring topologies are used, for example, in the Chord DHT. We will show that the ring topology is deadlock-free when each peer maintains a separate queue for each neighbor. The Chord routing protocol prescribes that at peer P_a a message for identifier id is forwarded in a clockwise direction to the neighbor P_b in the routing table that is closest to id with the condition that P_b lies between P_a and id (going in clockwise direction starting at P_a). Consider again the network in figure 1: if peer P_0 wants to route a message with identifier $id = 0.375$, the next hop would be P_2 as it lies between P_0 and id.

A ring topology with all routing entries correctly set is deadlock-free if there is one outgoing queue per neighbor: each message in a given queue will travel on its next hop at most half the distance of its previous hop. A queue of a link of distance d_1 can only wait on a queue of a link of distance $d_2 \leq 1/2 \cdot d_1$. Therefore, there are no cycles in the buffer waiting graph.

2.2.4 Tree

P-Grid [1] routes using a tree structure. Figure 3 shows a P-Grid tree with eight peers (P_0 to P_7). Each peer has three routing entries. A routing entry can be filled with any peer from the corresponding opposite subtree. Consider the example shown for peer P_2: entry 1 can be filled with any peer from the largest opposite subtree, i.e. P_4 to P_7 (here P_6 was chosen); entry 2 can be filled with either P_0 or P_1, and entry 3 is P_3. Routing is performed by forwarding into the smallest known subtree of the destination: if P_2 routes a message for destination P_4, it will forward to a peer in the right subtree, in this case P_6. P_6 will forward the message into the next smaller subtree (here either P_4 or P_5). If each peer has a queue per routing entry, the P-grid routing protocol is deadlock free: when a peer forwards a message to the next hop, it is inserted into a queue that will not lead back to the subtree the message is coming from. The buffer waiting graph is therefore cycle-free.

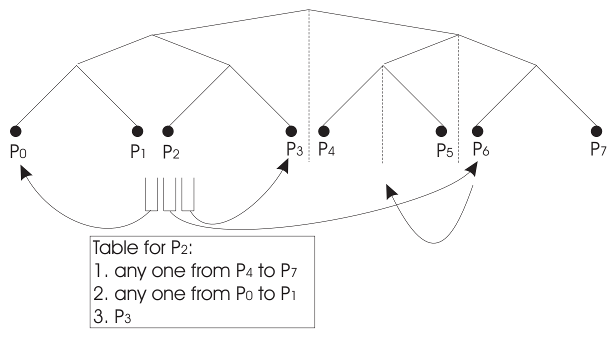

Figure 3. Routing in P-Grid.

3 Evaluation

In this section we will present our experimental results. The goal was a) to show that without CC, the system goes to a congestion collapse state when peers increase the load. Furthermore, we would like b) to evaluate the performance of CSCC and BPCC.

We implemented a fully functional P2P client in Java using the Chord routing protocol and both CC mechanisms. The experiments were performed in a cluster of five machines using ModelNet[1]. ModelNet allows us to emulate a wide-area network, where several peer client applications run on the same physical machine, however each of them receives its own IP address. We set the link capacity per peer to 1 Mbit/s with 30 ms delay between any two peers and zero loss.

3.1 a) Congestion collapse

The first experiment evaluates the performance of the DHT under an increasing load. Starting at the same time, all peers in the network perform each 2000 lookup requests for random identifiers. 2000 requests per peer is still a small value compared to the number of requests necessary to index a document collection of reasonable size. The experiment was performed with 16 peers. Such a relatively small number of peers is already sufficient to cause a congestion collapse. The same effect is visible in larger networks.

We first run the experiment with CC disabled. We gradually increase the lookup request rate (offered load) per peer

[1]http://issg.cs.duke.edu/modelnet.html
Make sure Java uses IPv4 (java -Djava.net.preferIPv4Stack=true).

from 15 msg/s to about 100 msg/s. Figure 4 shows the results averaged over 5 runs with mean deviation error bars. Without CC, up to an offered load of ∼80 msg/s per peer, the achieved throughput matches the offered load. When the peers further increase their sending rates the achieved throughput starts to drop instead of further increasing. This behavior is called a congestion collapse.

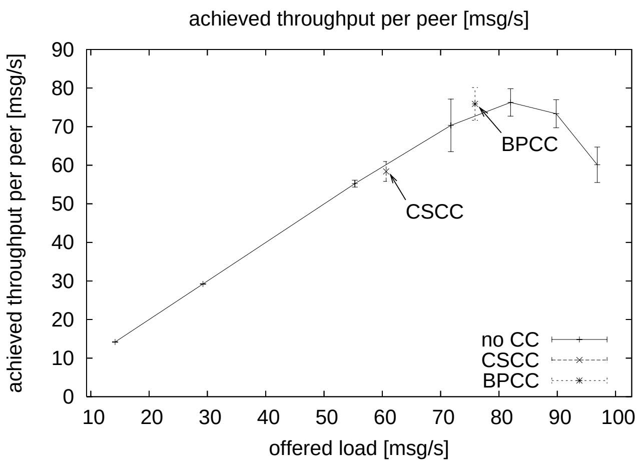

Figure 4. CSCC achieves a rate of ∼60 msg/s, BPCC ∼80 msg/s. Without CC, the system suffers congestion collapse when the offered load per peer exceeds 80 msg/s.

We then enable either CSCC or BPCC. Now all peers perform requests as fast as possible and CC regulates the rate to match the capacity of the DHT. CSCC achieves a rate of ∼60 msg/s, with approx. 5% retransmissions (4% caused by drops and 1% by early timeouts). BPCC performs better with a rate of ∼80 msg/s, which matches the maximum rate when no CC is applied. With 16 peers, BPCC achieves thus a global rate of almost 1300 lookup requests per second. One reason for the better performance of BPCC is that it is loss-free and thus does not have to retransmit dropped messages. A large difficulty in CSCC, however, is strongly varying RTTs as each packet has a different destination. Such variations make it harder for CSCC to find the optimal capacity of the DHT.

Obviously, in such a small network, each peer could cache the socket addresses and according id ranges of all other peers instead of performing a fresh lookup for each request. The larger the network, however, the less effective such a cache would be. Our goal is to study the behavior of the overlay network under heavy load. We therefore do not cache any lookup replies.

3.2 b) BPCC and CSCC under Cross-Load

We perform further experiments to compare BPCC and CSCC under more hostile conditions. We simulate varying cross-loads at each peer, which could be caused by e.g. other processes running on the same machine together with the peer client. When a peer is cross-loaded it will stop completely (i.e. pause) forwarding messages. Cross-load can also be seen as simulation of delays caused by failed routing entries that have to be repaired before routing at a peer can continue. We choose an exponentially distributed pause-time with average of 2 s. Figure 5 shows the result for 0% pause-time (no cross-load) to 15% pause time (i.e. each peer pauses routing traffic for 15% of the time on average). We perform the experiments with total buffer sizes of 100 and 400 messages per peer (i.e. in BPCC, each peer has four outgoing buffers of either size 25 or 100 messages, whereas in CSCC each peer has one buffer of size 100 or 400 messages).

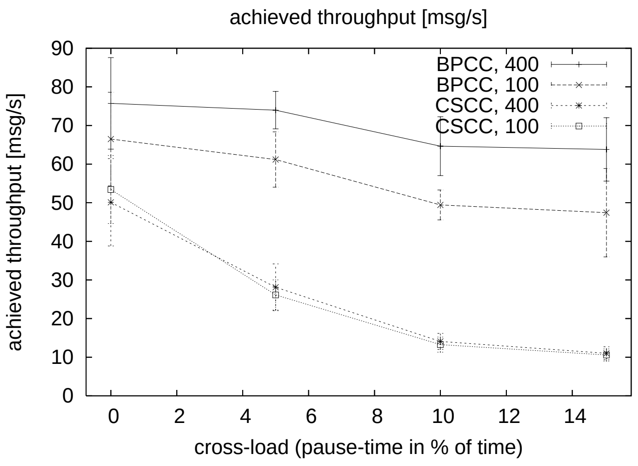

Figure 5. CSCC and BPCC under cross-load with different buffer sizes. BPCC largely outperforms CSCC.

With increasing cross-load BPCC more and more outperforms CSCC. Cross-load largely increases RTT variations as well as message drops, which severely diminishes the performance of CSCC. Increasing the buffer size helps BPCC to deal with cross-load, whereas it has no effects on CSCC.

3.3 Discussion

CSCC and BPCC can regulate the request rate of peers under heavy load and successfully avoid a congestion collapse. Our initial results showed that BPCC performs better than CSCC. However, BPCC might not be the right choice for extremely distributed and uncontrolled systems as there is a risk of deadlocks as soon as even a few peers behave maliciously (for cooperative peers BPCC is deadlock-free as shown in section 2.2). The DHT routing protocols presented in section 2.2 are no longer deadlock-free when, for example, a responsible peer (illicitly) returns a lookup reply using overlay routing instead of sending a direct message to the socket address of the requester. Techniques to detect and avoid such scenarios are beyond the scope of this paper.

The experiments we present can at best indicate the performance of the two CC mechanisms. Both mechanisms are in an initial state and have to be further developed. For a complete analysis other factors, such as churn, ungraceful leaves, and the behavior in large heterogeneous peer populations have to be taken into account. Due to the complexity of performing large-scale experiments with thousands of peers these extensions are part of ongoing and future work.

4 Related Work

Few papers have discussed CC in DHTs so far. For *recursive routing* many DHT proposals use either TCP for neighbor-to-neighbor routing or a simplified re-implementation of TCP using UDP, e.g. in Tapestry [14]. However, it is not further specified whether these systems contain DHT-wide CC mechanisms and how they behave under heavy loads. Most of the current DHT implementations are designed for low to moderate loads and CC has not been an issue yet.

For *iterative routing*, in DHash++ [3] presents the Striped Transport Protocol (STP), which has similarities with CSCC as it maintains DHT-wide destination-independent congestion state. The difficulty in iterative routing, however, is that a peer has to directly communicate with many other peers for which reliable RTT estimates are not available. STP uses a mix of Vivaldi latency predictions and past delays to calculate retransmission timeouts. However, when the network becomes congested, latency predictions might not be accurate anymore. Furthermore, it becomes difficult to distinguish between failed and overloaded peers.

[2] proposes component-based transport protocols for highly distributed applications. They provide higher flexibility to perform application-specific optimizations regarding (among other things) congestion control in peer-to-peer networks.

CC in DHTs is orthogonal to most of the load balancing work that has been done for P2P systems. [8], for example, proposes mechanisms to reduce the imbalance of data items stored at peers. Though important for DHTs, these mechanisms cannot avoid congestion caused by routing traffic in a DHT.

Further work on P2P CC is in the area of multi-cast overlay networks. [13], for example, studies the performance of a back-pressure mechanism.

5 Conclusions

In this paper, we have introduced congestion control for DHTs. To the best of our knowledge, this is the first paper that specifically deals with CC in DHTs. Our goal is not to present a final solution for CC in DHTs. We rather want to demonstrate that CC in DHTs is essential for certain DHT applications that send very large numbers of small messages, such as in peer-to-peer information retrieval. We have presented two initial CC mechanisms, CSCC and BPCC, which successfully prevent a congestion collapse. Preliminary experimental results with a real DHT implementation in Java show that our CC mechanisms work and can sustain high loads, even under low to moderate cross-load. There are still many possibilities for increasing the performance of our proposals. Future work includes a throughput analysis of DHTs for different network conditions, further experimental results with more peers, and implications of malicious peers on CSCC and BPCC.

6 Acknowledgements

We would like to thank Patrik Bless and Steven Dropsho for setting up a ModelNet cluster at EPFL.

References

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

[2] T. Condie, J. M. Hellerstein, P. Maniatis, S. Rhea, and T. Roscoe. Finally, a Use for Componentized Transport Protocols. In *Proceedings of the Fourth ACM Workshop on Hot Topics in Networks (HotNets-IV)*, 2005.

[3] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris. Designing a DHT for low latency and high throughput, 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] 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.

[7] G. D. Pifarre, L. Gravano, G. Denicolay, and J. L. C. Sanz. Adaptive deadlock- and livelock-free routing in the hypercube network. *IEEE Trans. Parallel Distrib. Syst.*, 5(11):1121–1139, 1994.

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

[9] 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.

[10] L. Schwiebert and D. N. Jayasimha. A universal proof technique for deadlock-free routing in interconnection networks. In *SPAA '95: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 175–184, New York, NY, USA, 1995. ACM Press.

[11] 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.

[12] C. Tang and S. Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. *First Symposium on Networked Systems Design and Implementation (NSDI'04), San Francisco*, 2004.

[13] G. Urvoy-Keller and E. W. Biersack. A congestion control model for multicast overlay networks and its performance. In *NGC'2002, 4th international workshop on network group communication, 23-25 October, 20002 - Boston, USA*, Oct 2002.

[14] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Jan. 2004.