Handling Very Large Numbers Of Messages In Distributed Hash Tables

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

***Abstract*—The principal service of Distributed Hash Tables (DHTs) is $route(id,\ data)$, which sends $data$ to a peer responsible for id, using typically $O(\log(\#\ of\ peers))$ overlay hops. Certain applications like peer-to-peer information retrieval generate billions of small messages that are concurrently inserted into a DHT. These applications can generate messages faster than the DHT can process them. To support such demanding applications, a DHT needs a congestion control mechanism to efficiently handle high loads of messages. In this paper we provide an extended study on congestion control for DHTs: we present a theoretical analysis that demonstrates that congestion control for DHTs is absolutely necessary for applications that provide elastic traffic. We then present a new congestion control algorithm for DHTs. We provide extensive live evaluations in a ModelNet cluster and the PlanetLab test bed, which show that our algorithm is nearly loss-free, fair, and provides low lookup times and high throughput under cross-load.**

I. Introduction

Distributed Hash Tables (DHTs), such as [1], [13], [14], [16], provide a scalable mechanism to map ids from a common id space to physical machines. Each peer in the network is responsible for a subset of ids. Given an id, a DHT can route any data value v to a peer responsible for the id. Depending on the application, v can be a lookup, insert, update, delete, etc. request. With n number of peers in the system, the expected cost of such a request is $O(\log n)$ overlay hops when each peer maintains a routing table of size $O(\log n)$.

There are two basic mechanisms for resolving a request: with *recursive routing*, when a peer p_r receives from a neighbor p_a a message[1] for id_x, p_r will, if it is not responsible for id_x, forward the message to a neighbor selected from its routing table. In the case of *iterative routing*, on the contrary, p_r will return the address of the next hop back to p_a instead of forwarding the message. p_a then contacts the next hop directly. In this paper, we focus on recursive routing.

Initially, DHTs have been developed as efficient and scalable replacement for Gnutella file-sharing networks. However, in recent years, new demanding applications have been proposed on top of DHTs, such as P2P Information Retrieval (P2P-IR) [17], [18] and P2P Database Management Systems (P2P-DBMS) [4]. These are examples of applications that generate so-called elastic traffic, where the request rate of the application adjusts to fill the available capacity of the DHT. In P2P-IR, for example, indexing a small document collection requires tens to hundreds of thousands of requests that have to be processed by a DHT [9]. These new applications can generate requests faster than a DHT can process them and therefore require the DHT to have special congestion control mechanisms to work efficiently.

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 project NEPOMUK No FP6-027705.

[1]We will use the terms request and message interchangeably.

A. Principles of Congestion Control for DHTs

Congestion control for DHTs is independent of TCP congestion control: TCP takes care of avoiding congestion in the underlying IP network, whereas DHT congestion control deals with avoiding overloading peers in the DHT.

Congestion in a DHT happens when a peer receives messages at a higher rate than it can process them. A cause of congestion can be that peers insert new messages faster than the DHT can process them. Skewed traffic patterns and heterogeneous capacities (CPU, bandwidth) can deteriorate congestion. The goal of DHT congestion control is to limit the request rates of peers to available (routing) capacities in the DHT.

There are two principle mechanisms of dealing with congestion. a) An overloaded peer drops all messages it cannot process and sources adjust their request rates accordingly. b) An overloaded peer signals its current processing rate to all peers that forward messages to it, to avoid that it receives messages at a faster rate than it can process them. This mechanism is called back-pressure and does not drop any messages.

B. Contribution of the Paper

We presented an initial study and small-scale evaluation of congestion control for DHTs in [7]. We proposed two congestion control mechanisms: 1) Credit System Congestion Control (CSCC), which uses mesage loss as congestion indicator and 2) Back-Pressure Congestion Control (BPCC), which propagates congestion state hop-by-hop in reverse direction to the message flows in the DHT.

The two mechanisms had the following weaknesses: CSCC performed well in terms of lookup delay, however, particularly in the presence of cross-load, the throughput decreased considerably due to an increased number of message loss. BPCC performed better in terms of throughput in the presence of cross-load. However, there is an inherent risk of deadlock using back-pressure congestion control if there are cycles in the buffer waiting graph [10], [15], which cannot be completely excluded in a highly decentralized, uncontrolled P2P environment. Furthermore, BPCC exhibited high lookup delays.

In this paper, we propose a new congestion control algorithm to remedy these problems. It uses marking of messages instead of loss to detect an onset of congestion. Live evaluation in a controlled emulation environment shows that it clearly outperforms the two initial algorithms in terms of lookup delay, throughput, fairness, and robustness. Furthermore, we provide a theoretic analysis of a DHT under high load and show that without congestion control, a DHT can suffer a congestion collapse. We also evaluate our new congestion control algorithm in the PlanetLab test bed.

C. Organization

After discussing related work in section II, we provide, in section III, an analysis of the behavior of a DHT under high load. In section IV, we present our new congestion control algorithm for DHTs. In section V, we provide live evaluation in a ModelNet cluster and using the PlanetLab test bed. We end with a discussion (section VI) and conclusions (section VII).

II. Related Work

Dabek et al. [3] present DHash++ and the Striped Transport Protocol (STP). The authors show experimental results for single-client fetch operations using STP for iterative routing and TCP back-pressure for recursive routing.

In contrast to [3], our DHT usage scenario is different as we consider routing a very large number of small messages as the main load (such as created by P2P-Information Retrieval or P2P-Database Management Systems applications). In such a scenario, all peers permanently generate new requests, which leads to a constant message flow in the DHT. Under such high-load conditions, there is a risk of deadlock when recursive routing is used in combination with TCP back-pressure as we show in [7]. Furthermore, iterative routing has the disadvantage that each peer performs very short interactions with many different peers in the network. For these interactions, it is difficult to get good RTT estimates, which are necessary to schedule a possible retransmission as well as to detect failed peers. Virtual coordinates can help estimating RTTs, however, they become imprecise in the case of network congestion and with varying peer capacities (e.g. available CPU).

For communication between two peers, [20] propose a re-implementation of TCP using UDP. However, they do not specify how congestion in the overlay network is avoided. Most DHT proposals so far were designed for low load, where congestion is not an issue.

Congestion control in DHTs is orthogonal to load balancing mechanisms that have been proposed for P2P systems, e.g. [12], which reduces the imbalance of data items stored at peers. These mechanisms cannot avoid congestion in the DHT caused by routing messages.

Further work on congestion control for P2P systems can be found in the area of component-based transport protocols [2] and multicast overlay networks [19].

III. Analyzing Congestion Collapse

In this section, we provide an analysis of the throughput of a DHT without congestion control under high load. We shall see that even in an idealized case, where all peers in the network have the same capacities, not restricting the rate at which a peer initiates requests will lead to a congestion collapse.

Let us consider an application in which each peer performs requests at a rate that is only limited by its own capacity (i.e. CPU or access link to the network) and without taking the congestion state of the DHT into account. Our analysis is for recursive routing. Fresh requests and relayed requests coming from other peers contend for available capacities in the P2P system. If the offered traffic of requests exceeds a peer's capacity, the peer has to drop requests. In this case of overload, all offered traffic contending at the bottleneck is reduced approximately proportionally.

A. Overview

The goal of this analysis is to study the achieved throughput $\mathcal{A}$ of requests in a DHT when each peer initiates new requests with a given rate x. Each request has a probability p of being successfully processed at a peer. $p = 1$ if the peer is not overloaded, otherwise $0 < p < 1$. Whether a peer is overloaded depends on the load offered to the bottleneck, which can be the access link or the CPU of a peer. We therefore have to calculate the total load $\mathcal{O}$ offered to the bottleneck and compare it to a given capacity c, which allows us then to calculate p and the achieved throughput of requests $\mathcal{A}$ as a function of x. p is a dimensionless number, while x, $\mathcal{O}$, and $\mathcal{A}$ can be measured in messages per second.

B. Definitions

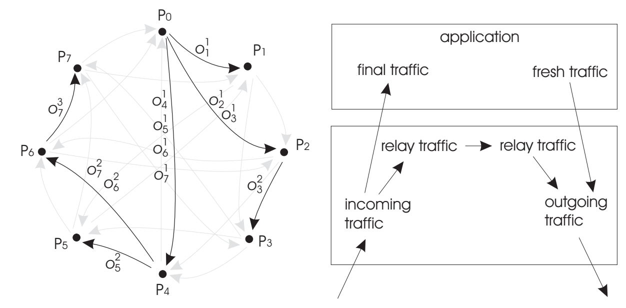
1) Offered Traffic: Each peer initiates requests, which are routed in the network. To calculate the total offered load $\mathcal{O}$, we first define *offered traffic*:

Definition 3.1: o_d^h is the offered traffic with final destination d before taking its h^{th} hop.

As we consider an idealized case, where all peers have the same capacities, all peers will generate the same traffic. For this reason, we defined the offered traffic independent of the source.

For ease of explanation, we first consider a small Chord [16] topology. However, our analysis is generally valid for any $\log n$ DHT. Figure 1(a) shows the offered traffic generated by P_0.

Each peer has $log_2 n = l$ (here $n = 8$ and $l = 3$) routing entries, each pointing to a peer (called neighbor) at distance $2^i, i = 0...(l-1)$. Each peer performs greedy routing, i.e. it selects the neighbor closest to the searched id when forwarding a request. P_0 generates new requests for the remaining seven peers in the network. Some of the traffic is relayed by other peers: o_7^1, for example, is the offered traffic at P_0 with destination P_7 before the first hop (P_0-P_4).



(a) Offered traffic o_d^h generated by P_0. (b) Incoming and outgoing traffic.

Fig. 1. Traffic flows in a DHT

2) Achieved Throughput: Each peer has a capacity c to process offered traffic. c depends on a peer's CPU and on its access link to the Internet. If the total offered traffic does not exceed a peer's capacity, all offered traffic is processed. Otherwise, a fraction of the offered traffic is dropped. We define achieved throughput as:

Definition 3.2: a_d^h is the achieved throughput with final destination d after taking its h^{th} hop.

$a_7^1 \leq o_7^1$, for example, is the amount of traffic with destination P_7 after the first hop, i.e. the traffic arriving at P_4.

3) Processing Probability: The fraction of processed traffic is defined as:

$$p = \min\left(1, \frac{\mathrm{c}}{\mathcal{O}}\right), \tag{1}$$

where c is the capacity of a peer and $\mathcal{O}$ the total offered load, which we further define in the following subsections. As long as a peer is not overloaded, i.e. $\mathcal{O} \leq c$, all offered traffic is processed, i.e. $p = 1$, otherwise $p < 1$. If the offered traffic is twice as big as the capacity of the peer, for example, then the processing probability for a message is 0.5.

If the traffic to a destination flows over several hops, only the fraction that arrives at a peer can be offered on its next hop:

$$o_d^{h+1} = a_d^h = o_d^h \cdot p \tag{2}$$

C. Models

Each peer has to process incoming and outgoing offered traffic (cf. figure 1(b)): incoming traffic consists of final traffic, $\mathcal{O}_{\text{final}}$, which is handed to the application, and relay traffic, $\mathcal{O}_{\text{rel}}$, which is forwarded to neighbors. Outgoing traffic consists of fresh traffic, $\mathcal{O}_{\text{fresh}}$, generated by the application and relay traffic, $\mathcal{O}_{\text{rel}}$. We consider two bottleneck scenarios:

1) Model 1: Upstream Link is the Bottleneck: Many home PCs have asymmetric Internet access with usually higher downstream than upstream bandwidth. Therefore, the outgoing traffic determines the capacity of the peer. However, we also consider a fraction α (e.g. $\alpha = 0.05$) of the incoming traffic, as TCP acknowledgements use some upstream bandwidth. The offered load to the bottleneck is thus:

$$\mathcal{O}_{\text{M1}} = \mathcal{O}_{\text{fresh}} + \mathcal{O}_{\text{rel}} + \alpha(\mathcal{O}_{\text{final}} + \mathcal{O}_{\text{rel}})$$

2) Model 2: Peer Processing Capacity is the Bottleneck: If a peer has enough up- and down-link bandwidth the bottleneck is the peer's processing speed (i.e. CPU). We get:

$$\mathcal{O}_{\text{M2}} = \mathcal{O}_{\text{fresh}} + \mathcal{O}_{\text{rel}} + \mathcal{O}_{\text{final}}$$

D. Total Offered Load $\mathcal{O}$

We now have to calculate $\mathcal{O}_{\text{M1}}$ and $\mathcal{O}_{\text{M2}}$ for our DHT to be able to determine the processing probability p for a given capacity c using equation 1. We consider the symmetric case, i.e. all peers initiate uniform random requests at a rate x. Here, x also includes requests that have to travel zero hops to be resolved, i.e. the initiating peer is responsible. We shall first discuss the case for eight peers, then the general case.

1) For Eight Peers: In the case of eight peers $x/8$ requests travel zero hops, while $7/8 \cdot x$ requests have to be routed through the network to be resolved. The offered load on the first hop is therefore:

$$\forall d, 0 < d \leq 7, o_d^1 = 1/8 \cdot x \tag{3}$$

We now calculate the total traffic $\mathcal{O}$ offered at each peer using the traffic flows generated by P_0 (shown as black arrows in figure 1(a)). We simplify using eq. 2 and 3:

Fresh traffic is the traffic generated by the application for all destinations in the DHT before taking its first hop:

$$\mathcal{O}_{\text{fresh}} = o_1^1 + o_2^1 + o_3^1 + o_4^1 + o_5^1 + o_6^1 + o_7^1 = \frac{7x}{8}$$

Final traffic is achieved traffic to all destinations after its final hop. It is handed to the application and not further relayed:

$$\begin{aligned}\mathcal{O}_{\text{final}} &= a_1^1 + a_2^1 + a_3^2 + a_4^1 + a_5^2 + a_6^2 + a_7^3 \\ &= \frac{3x}{8}p + \frac{3x}{8}p^2 + \frac{x}{8}p^3\end{aligned}$$

Relay traffic is forwarded to the neighbors:

$$\mathcal{O}_{\text{rel}} = a_3^1 + a_5^1 + a_6^1 + a_7^1 + a_7^2 = \frac{4x}{8}p + \frac{x}{8}p^2$$

Incoming and outgoing traffic can thus be written as:

$$\mathcal{O}_{\text{out}} = \mathcal{O}_{\text{fresh}} + \mathcal{O}_{\text{rel}} = \frac{7x}{8} + \frac{4x}{8}p + \frac{x}{8}p^2 \tag{4}$$

$$\mathcal{O}_{\text{in}} = \mathcal{O}_{\text{final}} + \mathcal{O}_{\text{rel}} = \frac{7x}{8}p + \frac{4x}{8}p^2 + \frac{x}{8}p^3$$

2) For n Peers: For any $\log n$ DHT (such as [1], [16]) with $n = 2^l$ peers, each peer has $l = \log_2 n$ neighbors (links). The offered load per peer for all destinations d on the first hop is thus:

$$\forall d, 0 < d < n, o_d^1 = \frac{x}{n} = \frac{x}{2^l}$$

We get the following offered loads:

Fresh traffic:

$$\mathcal{O}_{\text{fresh}} = \sum_{d=1}^{n} o_d^1 = \frac{n-1}{n}x \tag{5}$$

Final traffic, also called achieved throughput $\mathcal{A}$ as it represents all successfully routed requests. Looking at the way a $\log n$ DHT selects the routing entries, we can use Pascal's Triangle to calculate the number of peers that are a certain number of hops away: line 3 in the triangle, for example, is 1, 3, 3, 1, i.e. there is 1 peer 0 hops away, 3 are 1 hop, 3 are 2 hops, and 1 is 3 hops away. With l neighbors there are thus

$$\binom{l}{h} = \frac{l!}{h!(l-h)!}$$

destinations that are reached with h hops. On each hop, a message is relayed with probability p. We thus get:

$$\mathcal{A} = \mathcal{O}_{\text{final}} = \frac{x}{2^l} \sum_{h=1}^{l} \binom{l}{h} p^h \tag{6}$$

Notice that $h \geq 1$, i.e. we consider only the traffic that has to be routed in the network. We assume that requests for which the peer itself is responsible, i.e. $h = 0$, can be answered with no cost.

Relay traffic: To determine the relay traffic, we calculate the total outgoing traffic and subtract the fresh traffic:

$$\mathcal{O}_{\text{rel}} = \mathcal{O}_{\text{out}} - \mathcal{O}_{\text{fresh}} \tag{7}$$

The total outgoing traffic can be also determined using Pascal's Triangle. However, we consider all hops and not just the final hop as for $\mathcal{O}_{\text{final}}$. We thus get:

$$\mathcal{O}_{\text{out}} = \frac{x}{2^l} \sum_{h=1}^{l} \binom{l}{h} \sum_{j=1}^{h} p^{j-1} \tag{8}$$

We demonstrate eq. 8 for $l = 3$ and observe that we obtain the expected result as in eq. 4:

$$\begin{aligned} \mathcal{O}_{\text{out}} &= \frac{x}{2^3} \sum_{h=1}^{3} \binom{3}{h} \sum_{j=1}^{h} p^{j-1} \\ &= \frac{x}{8} \left(3 \cdot 1 + 3 \cdot (1+p) + 1 \cdot (1+p+p^2)\right) \\ &= \frac{x}{8} \left(7 + 4p + p^2\right) \end{aligned}$$

E. Example for one Million Peers

We show an example of the achieved throughput $\mathcal{A}$ for growing request rates x for $l = 20$, i.e. $n = 2^{20} \approx 1$ million peers. Each peer has a capacity of processing $c = 120$ requests per second determined by its bottleneck. We consider the two proposed models M1 (uplink is bottleneck) and M2 (CPU is bottleneck). We choose $\alpha = 5\%$ for M1, i.e. 5% of the incoming traffic is used by TCP acknowledgements on the uplink.

Given the eq. 1 and 5 to 8 we can numerically calculate the processing probability p for a given 2^l peers with capacity c and request rate x (e.g. using Mathematica's 'solve' function). With p and eq. 6 we can calculate $\mathcal{O}_{\text{final}}$, i.e. the achieved throughput $\mathcal{A}$ of requests for the DHT.

Figure 2 shows the achieved throughput $\mathcal{A}$. The max. achieved throughput per peer is reached for $x_{optM1} = 11.4$ request/s for model 1 and $x_{optM2} = 10.9$ request/s for model 2. Once the offered load exceeds x_{opt}, the achieved throughput drops quickly. This behavior is called a congestion collapse.

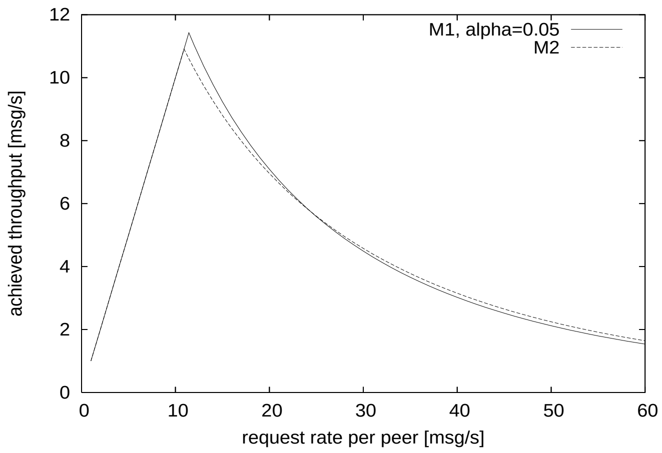


Fig. 2. Achieved throughput $\mathcal{A}$ vs. request rate per peer x for a DHT with one million peers for the bottleneck scenarios M1 and M2. The DHT suffers a congestion collapse once the offered load exceeds a certain rate x_{opt}.

F. Analysis of the Asymptotic Behavior of $\mathcal{A}$

We now look at the asymptotic behavior of the achieved throughput $\mathcal{A}$ for a saturated DHT (i.e. $p \rightarrow 1$) and an extremely overloaded DHT (i.e. $p \rightarrow 0$).

We first simplify eq. 5 to 8: using the equations $\sum_{j=0}^{h-1} p^j = (1-p^h)/(1-p)$, $\sum_{h=1}^{l} \binom{l}{h} p^h = (1+p)^l - 1$, and $\sum_{h=1}^{l} \binom{l}{h} = 2^l - 1$ we get:

$$\begin{aligned} \mathcal{O}_{\text{fresh}} &= \frac{2^l-1}{2^l} x = \frac{x}{2^l(p-1)} (2^l-1)(p-1) \\ &= \frac{x}{2^l(p-1)} (2^l p - 2^l - p + 1), \end{aligned}$$

$$\begin{aligned} \mathcal{O}_{\text{final}} &= \frac{1}{2^l} x \textstyle\sum_{h=1}^{l} \binom{l}{h} p^h \\ &= \frac{x}{2^l} \left[(1+p)^l - 1\right] \\ &= \frac{x}{2^l(p-1)} \left[(1+p)^l - 1\right](p-1) \\ &= \frac{x}{2^l(p-1)} \left[(1+p)^l p - (1+p)^l - p + 1\right], \\ \mathcal{O}_{\text{out}} &= \frac{x}{2^l} \textstyle\sum_{h=1}^{l} \binom{l}{h} \sum_{j=0}^{h-1} p^j \\ &= \frac{2^l - (1+p)^l}{2^l(1-p)} x \\ &= \frac{x}{2^l(p-1)} \left[(1+p)^l - 2^l\right], \end{aligned} \tag{9}$$

$$\begin{aligned} \mathcal{O}_{\text{rel}} &= \mathcal{O}_{\text{out}} - \mathcal{O}_{\text{fresh}} \\ &= \frac{x}{2^l(p-1)} \left[(1+p)^l - 2^l - 2^l p + 2^l + p - 1\right] \\ &= \frac{x}{2^l(p-1)} \left[(1+p)^l - 2^l p + p - 1\right]. \end{aligned}$$

We now calculate the behavior of the achieved throughput $\mathcal{A}$ for processing probability $p \rightarrow 1$ and $p \rightarrow 0$. We show the detailed calculations for M1 and provide only the results for M2. We first simplify $\mathcal{O}_{\mathrm{M1}}$:

$$\begin{aligned}
\mathcal{O}_{\mathrm{M1}} =\ & \mathcal{O}_{\mathrm{out}} + \alpha(\mathcal{O}_{\mathrm{final}} + \mathcal{O}_{\mathrm{rel}}) \\
=\ & \frac{x}{2^l(p-1)}\Big\{(1+p)^l - 2^l + \alpha\big[(1+p)^l p - (1+p)^l \\
& \qquad - p + 1 + (1+p)^l - 2^l p + p - 1\big]\Big\} \\
=\ & \frac{x}{2^l(p-1)}\left\{(1+p)^l - 2^l + \alpha p\left[(1+p)^l - 2^l\right]\right\} \\
=\ & \frac{x}{2^l(p-1)}(1+\alpha p)\left[(1+p)^l - 2^l\right],
\end{aligned}$$

We can now distinguish two cases: 1) the network is saturated, but not overloaded (i.e. $p \to 1$, all traffic is relayed), and 2) the offered load largely exceeds the peer's capacities (i.e. $p \to 0$):

1) The DHT Is Saturated ($p \to 1$): For $\mathcal{O} \geq c$ the network is saturated or not fully loaded implying $p \leq 1$ and $\mathcal{O} = \frac{c}{p}$ (eq. 1). Using l'Hôpital's rule we calculate $\mathcal{O}$ for the saturated case $p \to 1$.

$$\begin{aligned}
\lim_{p\to 1} \mathcal{O}_{\mathrm{M1}} &= \frac{x}{2^l} \lim_{p\to 1} \mathrm{d}\left\{(1+\alpha p)\left[(1+p)^l - 2^l\right]\right\}/\mathrm{d}\,p \\
&= \frac{x}{2^l} \lim_{p\to 1} \Big\{(1+\alpha p) l (1+p)^{l-1} \\
&\qquad + \alpha\big[(1+p)^l - 2^l\big]\Big\} \\
&= \frac{x}{2}(1+\alpha) l.
\end{aligned}$$

For model 1 we get thus:

$$x = \frac{2c}{(1+\alpha) l}$$

For model 2 the calculations are similar. We get:

$$x = \frac{2^l c}{2^l + 2^{l-1} l - 1}$$

2) The Extremely Overloaded DHT ($p \to 0$): The relation $\frac{1}{1-p} = 1 + p + p^2 + \ldots$, valid for $|p| < 1$, allows us to expand $\frac{c}{x} = \frac{p\mathcal{O}}{x}$ into a power series $a_0 + a_1 p + a_2 p^2 + \ldots$ of p and conversely p into a power series $b_0 + b_1 x^{-1} + b_2 x^{-2} + \ldots$ of x^{-1}.

$$\frac{c}{x} = \frac{p}{2^l(p-1)}(1+\alpha p)\left[(1+p)^l - 2^l\right] = \frac{2^l - 1}{2^l} p + a_2 p^2 + \ldots,$$

$$p = \frac{2^l c}{2^l - 1} x^{-1} + b_2 x^{-2} + \ldots.$$

We set p into $\mathcal{O}_{\mathrm{final}}$ (eq. 9) and use

$$\begin{aligned}
(1+p)^l &= \sum_{k=0}^{l} \binom{l}{k} p^k \\
&= \binom{l}{0} p^0 + \binom{l}{1} p^1 + \binom{l}{2} p^2 \ldots \\
&= 1 + lp + \ldots
\end{aligned}$$

For model 1 and 2 we get for $p \to 0$:

$$\mathcal{A} = \mathcal{O}_{\mathrm{final}} = \frac{x}{2^l}\left[1 + \frac{2^l c}{2^l - 1} x^{-1} - 1\right] = \frac{c}{2^l - 1}$$

G. Conclusions of the Analysis

We have shown for two scenarios, M1, when the uplink capacity is the bottleneck and M2, when CPU is the bottleneck, that a DHT can suffer a congestion collapse if peers increase their request rates without taking the capacity of the DHT into account. This analysis was done for $\log n$ DHTs. It can be applied to any DHT by adapting the offered load $\mathcal{O}$ and the achieved throughput $\mathcal{A}$ to the specific routing function of the DHT.

This analysis shows that even in an ideal, symmetric environment, a congestion collapse can occur. Our live evaluation in section V shows a congestion collapse behavior as predicted by this analysis. In a heterogeneous environment, the capacities at different peers can vary strongly, which makes the DHT even more susceptible for congestion. The goal of congestion control is to limit peer request rates to match the currently available capacity in the DHT.

IV. Congestion Control for DHTs using Marking

We explained two congestion control strategies in the introduction: 1) congestion control using back-pressure, which is not suitable for highly uncontrolled P2P environments because of the risk of deadlock. 2) A strategy using message loss as signal for congestion, which is also not optimal, as a source can only react when there is already congestion in the network.

In the following subsections, we introduce congestion control using marking, which outperforms the first two strategies as we will demonstrate in the evaluation section V.

1) Behavior of a Queue: On each hop, messages are queued. We use queues to set congestion feedback in a message-header field h, in such a way that each peer receives a fair share of the bottleneck resource without overloading it. The size of h is one bit, which is unset, i.e. initialized to 'false', at the source of the message. $h =$ 'false' signifies 'no congestion' and $h =$ 'true' signifies 'congestion'. Each peer puts its current request rate x in the message header. Furthermore, each peer maintains a running average sending rate x_{avg} over the source rates x of the last k messages it has received. A queue sets h to 'true' with probability q, which depends on the average queue size and whether the message source has a higher or lower rate than the average rate x_{avg} perceived by the peer.

q is calculated as follows:

s	Number of messages in the queue
δ	Smoothing value, e.g. 0.9
$\bar{s}$	Average number of messages in the queue
t	Maximum feedback threshold
γ	Fairness parameter

$$\bar{s} = (1-\delta)\cdot s + \delta \cdot \bar{s}$$

$$q = min\left[\frac{\bar{s}}{t \cdot (t - \bar{s})} \cdot \left(\frac{x}{x_{avg}}\right)^{\gamma}, 1\right] \quad (10)$$

Eq. 10 consists of two parts: the first part is for congestion control: the closer the average queue size $\bar{s}$ gets to the maximum feedback threshold t, the faster the probability for negative feedback (i.e. $h =$ 'true') increases. If $\bar{s} \geq t$, $q = 1$, i.e. h is set with a probability equal to one. The maximum feedback threshold t is chosen to be considerably smaller than the queue capacity (e.g. $t = 20$, $queue\ capacity = 200$), to allow for buffering of short message bursts. Furthermore, we average the queue size s with parameter δ to avoid that a queue returns negative feedback in case of short variations of the message arrival rate.

The second part of eq. 10 is for fairness: when the rate x of a message source is smaller than the average rate x_{avg} perceived by the peer, the probability of negative feedback decreases. If $x > x_{avg}$, q increases, i.e. the source is punished for taking a larger part of the bottleneck resource than average. The parameter γ regulates how much fairness should be taken into account. For $\gamma = 0$, fairness is turned off. The higher γ the stronger a deviation of a source rate from the average rate is taken into account.

Figure 3 shows how q of eq. 10 changes for different average queue sizes $\bar{s}$ with $\gamma = 4$. It shows q for three sources, one that is 25% higher than x_{avg}, one 25% lower, and one that equals x_{avg}.

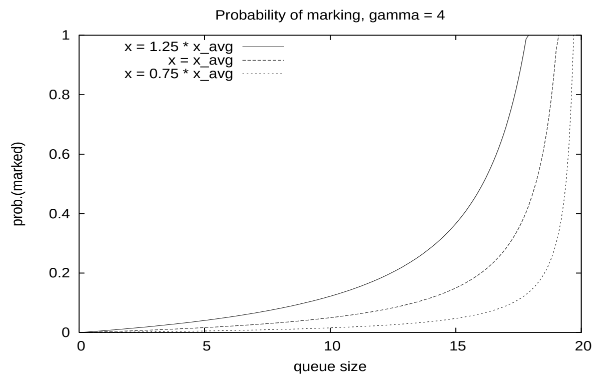


Fig. 3. Average queue size $\bar{s}$ vs. probability q of giving negative feedback for different source rate - average rate ratios with threshold $t = 20$.

Once h is set, it will never be unset on the path to the destination of the message. The destination copies h into an overlay-acknowledgement, which is returned to the source, either using UDP or by routing in the DHT. The source will behave as explained in the following subsection.

2) Behavior of a Source: A peer receives for each issued request an overlay-ack with congestion feedback h. For positive feedback (i.e. $h =$ 'false', no congestion), a peer increases its rate x as follows:

$$x := x + \frac{c_+}{x}$$

c_+ is a small positive constant (e.g. 2) that determines the aggressiveness of a source to test for extra bandwidth when it receives positive feedback.

For negative feedback (i.e. $h =$ 'true', congestion), a peer decreases its request rate x as follows:

$$x := x \cdot c_-,$$

where $0 < c_- < 1$ is a constant (e.g. 0.5). It determines how strongly a source backs off when receiving negative feedback.

We thus have an additive increase, multiplicative decrease scheme of changing a peer's request rate as it is done in TCP. The choice of c_+ and c_- depends on the RTT and the routing capacity of a DHT. As for TCP, these constants have to be fixed for an expected usage scenario. Making c_+ and c_- adaptable to a changing environment has implications on the fair sharing of bottleneck resources in the DHT. We leave such improvements as future work.

3) Round Trip Time Estimates: Each peer maintains an RTT estimate for requests: for each request it measures the time until the overlay-ack arrives. This RTT estimate is DHT-wide, i.e. destination-independent, as in large DHTs it is likely that each request goes to a different peer. Thus, the RTT varies strongly. If the retransmission timeout for an outstanding request fires, a peer decreases its rate (as shown in IV-2) and retransmits the request.

4) Analysis of Marking: We now show that the marking scheme does avoid a congestion collapse. We assume a very large network of peers and use the modeling framework of [6]. We first consider model M2 from section III, i.e. peers are CPU constrained. Let x_j be the rate at which a peer j generates requests. Further, assume that the probability that a peer i marks a request is a function $f(\mathcal{O}_i)$ of the total rate of requests $\mathcal{O}_i = \sum_j x_j p_{j,i}$ received by peer i ($p_{j,i}$ is the probability that a request originated by peer j is relayed by peer i; it is computed in detail in section III). A direct application of [6], theorem 3 shows that the rates x_i obtained with the marking scheme maximize the global utility function

$$\sum_i \sqrt{\frac{c_+}{1-c_-}} \arctan\left(x_i \sqrt{\frac{1-c_-}{c_+}}\right) - \sum_i F(\sum_j x_j \cdot p_{j,i}), \quad (11)$$

where we define F by $F(x) = \int_0^x f(u)du$. The maximum of eq. 11 is obtained for $x_i = x^*$, where x^* is independent of i, by the symmetry assumption. Further, we can lower bound x^* as follows: it is reasonable to assume that the marking probability f (and therefore F) is negligible except when $\mathcal{O}_i$ is close to ηc, where c is the capacity of a peer and η is a safety margin factor, e.g. $\eta \approx 0.80$. If, for some i, $x_i < \eta x_{opt}$, we can always increase the utility in eq. 11 as the first term increases while the second remains 0. Thus the maximum of eq. 11 is reached for $x_i \geq \eta x_{opt}$ and thus

$$x^* \geq \eta \cdot x_{opt}.$$

In other words, the rate achieved by our marking scheme in a symmetric network of any size is at worst a fraction η of the peak rate x_{opt} in figure 2.

For model M1, the reasoning is similar, after replacing $p_{j,i}$ by the probability that a request issued by peer j is handled by i, multiplied by $(1+\alpha)$ for $i \neq j$ (non fresh traffic).

V. Experimental Results

In this section, we present extensive experimental results. We implemented a DHT in Java to test the performance of our congestion control mechanism. We choose the routing entries in a Chord-like fashion. Requests are routed recursively. Overlay-acknowledgements are returned in a UDP message.

We evaluate the two algorithms in [7], i.e. 'back-pressure' and congestion control using lost packets as congestion indication ('loss'), as well as the new algorithm introduced in section IV, which uses marking as congestion indication ('marking').

A. Environment

We present a live evaluation of our prototype in a ModelNet[2] cluster and in the PlanetLab[3] test bed. The ModelNet cluster consists of 21 dual 3.4 GHz Xeon processors with 2 GB RAM, 1 GBit LAN. We use ModelNet because it allows us to test our prototype in an environment with Internet-like behavior. Each peer client runs in a virtual node. Several virtual nodes are hosted on one physical machine. All IP packets are routed through a ModelNet core, which applies the network characteristics, such as the bandwidth, delay, and loss between virtual nodes. The advantage of ModelNet is that live experiments can be repeated with the exact same network characteristics, which makes it possible to compare different algorithms.

B. Demonstration of a Congestion Collapse

The following experiments are performed in the ModelNet cluster with 128 peers. Each peer has 600 kbps available bandwidth and there is a 20 ms delay between any two peers. The goal is to demonstrate the behavior of a DHT with and without congestion control.

We first disable congestion control. All peers perform requests at a constant rate x and we measure the achieved throughput of successful requests. We repeat this experiment fixing x to values between 5 and 80 requests per peer per second (figure 4). Without congestion control, the achieved throughput increases as long as the capacity of the DHT is not reached. Once the peers perform requests faster then the DHT can handle, (at $\sim$65 request/s per peer), we enter a congestion collapse state and the achieved throughput drops for increasing x.

We repeat the experiment without limiting x but enabling one of the congestion control mechanisms, either 'back-pressure', 'loss', or 'marking'. All three algorithms successfully prevent a congestion collapse and limit the request rate of the application (i.e. the offered load) at 65-70 request/s. With 128 peers, the global request rate of the DHT is thus $\sim$8300 request/s.

[2]http://modelnet.ucsd.edu/

[3]www.planet-lab.org

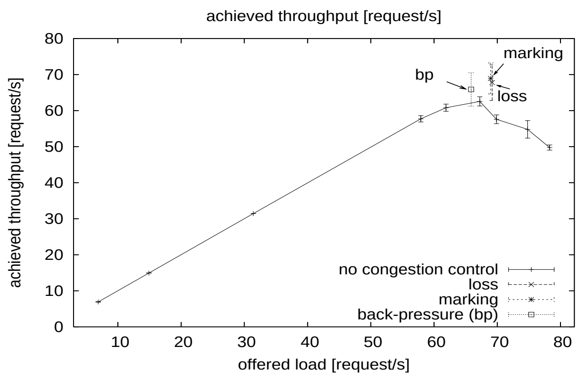


Fig. 4. Achieved throughput of requests for increasing offered load. No congestion control leads to a collapse once a rate of $\sim$70 requests per second is reached. The congestion control algorithms 'back-pressure', 'loss', and 'marking' limit the rate between 65 and 70 request/s.

C. Performance of Different Congestion Control Strategies with Cross-Load

We now evaluate the performance of the three different congestion control strategies in an environment with *cross-load*. We simulate cross-load by letting peers stop relaying messages for short periods of time. Cross-load is likely to appear when a peer client runs on a PC on which other applications are running. Cross-load can also be seen as delay caused by stale routing entries when a peer has to stop relaying messages to replace a stale entry. We selected cross-load, i.e. the intervals when a peer stops forwarding, to be exponentially distributed with an average of 500 ms.

The following experiments are in the ModelNet cluster with 128 peers, with access link capacities and delays varying between 600-1000 kbps and 5-15 ms (both randomly assigned).

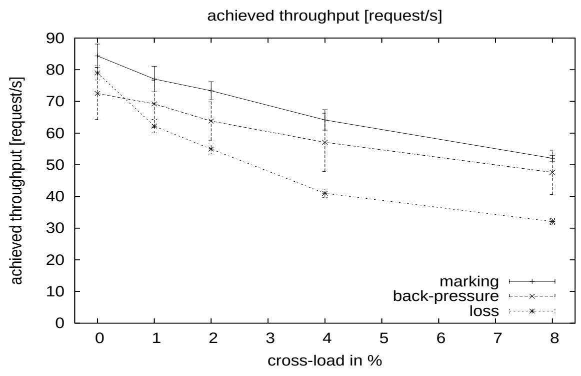
Figure 5 shows several performance characteristics for increasing cross-load. A cross-load of e.g. 5% means that the peer pauses sending messages on average 5% of the time.

Figure 5(a) shows the achieved throughput of requests per peer per second. The throughput decreases for all three strategies, however, 'marking' clearly outperforms 'loss' when cross-load increases.

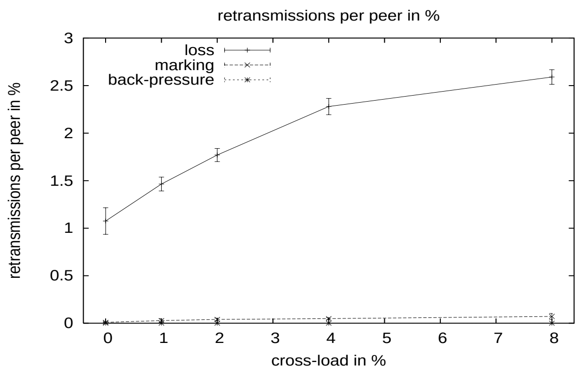
Figure 5(b) shows the % of retransmitted requests: for 'back-pressure' there are zero retransmissions as peers never drop messages. 'marking' shows also very little retransmissions (less than 0.2%) as it successfully prevents that queues overflow. 'loss' clearly suffers under heavy cross-load and requires many retransmissions caused by an increased number of dropped messages.

Figure 5(c) shows that the delay to resolve a request is considerably higher for 'back-pressure' than for the other two congestion control strategies. The reason for the delay-increase with 'back-pressure' is the high number of outstanding requests (i.e. requests that are buffered in the network) as shown in figure 5(d).

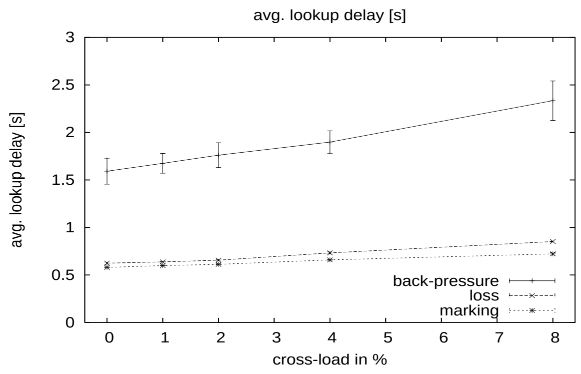
Overall, we observe a considerable performance improvement in terms of throughput and delay with 'marking'. Furthermore, as already discussed in the introduction, 'back-pressure' does not classify for distributed and uncontrolled



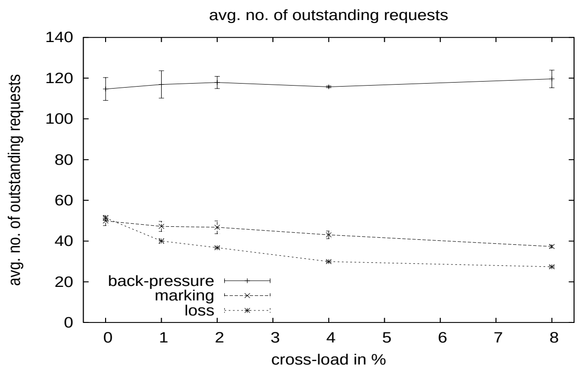
(a) Congestion control using 'marking' clearly outperforms congestion control using 'loss' in the presence of heavy cross-load.



(b) Congestion control using 'marking' needs almost zero retransmissions.



(c) 'Back-pressure' suffers high lookup delays.



(d) The number of outstanding messages is considerably higher with 'back-pressure'.

Fig. 5. Performance of congestion control using 'back pressure', 'loss', and 'marking' for increasing cross-load.

P2P environments as there is a risk of deadlock under heavy load when routing tables are not correct.

D. Fairness

The congestion control strategy 'marking' has the additional advantage that it supports fairness as presented in section IV-1. In a setup as presented in the previous subsection and choosing $\gamma = 16$, we were able to decrease the mean deviation of request rates among all peers by 40% compared to 'marking' without fairness (i.e. $\gamma = 0$).

E. PlanetLab Experiments

We performed experiments in PlanetLab to test our congestion control mechanism using 'marking' in a demanding environment. PlanetLab machines are heavily overloaded (CPU load averages vary between 2 and 30). In addition, the available bandwidth between different machines varies strongly. We thus have a scenario that is both CPU and bandwidth constrained. This PlanetLab experiment serves as a proof-of-concept of our DHT implementation. The following test runs with 64 peers, each peer on a separate machine. We chose 64 machines with relatively stable performance distributed around the world. All peers constantly perform requests for random ids using 'marking' as congestion control mechanism. Replies are routed back in the DHT as we experienced high loss rates with UDP. Figure 6 shows the average rate per peer over a period of over 4 hours: it varies around 3.5 requests per second per peer. In total, all peers together perform thus roughly 3.3 million requests. Notice that for this experiment we do not use proximity neighbor selection, which could increase the throughput.

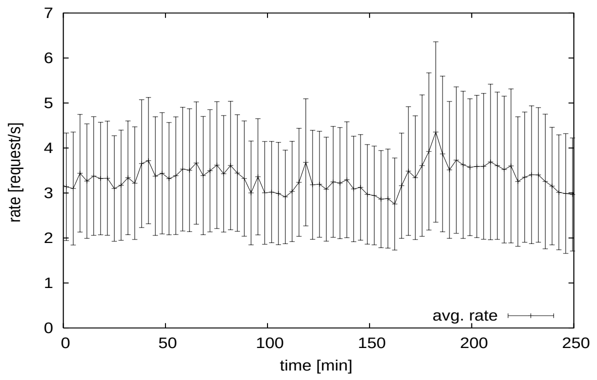


Fig. 6. Average request rate per peer per second in PlanetLab

VI. Discussion

Congestion control in DHTs using marking has similarities with TCP congestion control and proposed extensions using explicit congestion notifications [11]. The most important difference is that in TCP congestion control, the communication is only between two entities. In a DHT, each peer communicates with many other peers in the network depending on request destinations. In a large DHT, it is likely that each feedback that a peer receives with an overlay-acknowledgement is returned by different peer. Thus, RTTs vary strongly. In addition, guaranteeing in-order request delivery and detecting duplicate requests in a DHT requires to keep state of size $O(n)$ and is thus not supported by DHTs.

Fairness in traditional TCP assumes that all packets of a connection take the same IP route. In a DHT, each request is likely to travel a different path. Therefore, DHTs require a different mechanism for fairness, which we presented in section IV-1.

We presented experimental results using a Chord-like routing topology. Experiments of all three congestion control strategies in tree- and randomized small-world DHT topologies showed results that are comparable to those presented in this paper.

The proposed congestion control and fairness strategies are susceptible to greedy peers in the network that ignore congestion feedback. A similar problem also exists in TCP/IP. Strategies to detect such peers are outside the scope of this paper.

Congestion control forces sources to slow down to the bottleneck capacity in the DHT. Slow peers in the network can thus considerably reduce the throughput of the whole DHT. Adaptive routing around such bottlenecks (cf. [8]) would be a suitable mechanism to increase the throughput of a DHT. Such mechanisms (as well as other load-balancing mechanisms) are orthogonal to congestion control: given a routing scheme, congestion control regulates the message flows in a DHT to avoid a congestion collapse and to get as close as possible to the maximal throughput. This maximal throughput can also vary considerably in time as it depends on the available CPU and bandwidth.

Mechanisms like XCP [5], where the bottleneck specifies the exact sending rate for each flow passing through it, cannot be applied to DHTs. Different peers can have different (unknown) fractions of their requests traversing the bottleneck peer. Specifying the fair share for each peer that is using the bottleneck peer would require to keep state for all flows traversing the bottleneck, which does not scale.

VII. Conclusions

In this paper, we have provided an extensive study about congestion control for DHTs. We have demonstrated in analysis that congestion control in DHTs is essential for applications that send very large numbers of small messages, such as P2P-IR or P2P-DBMS. We have presented a new congestion control strategy that achieves high throughput and low delays in the presence of heavy cross-load. Extensive experiments with a real DHT implementation in a ModelNet cluster and in the PlanetLab test bed show that our mechanism works and can sustain high loads.

Future work includes a comparison of congestion control mechanisms for iterative and recursive routing.

References

[1] K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS*, pages 179–194, 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. F. Kaashoek, and R. Morris. Designing a dht for low latency and high throughput. In *NSDI*, pages 85–98, 2004.

[4] R. Huebsch, B. N. Chun, J. M. Hellerstein, B. T. Loo, P. Maniatis, T. Roscoe, S. Shenker, I. Stoica, and A. R. Yumerefendi. The architecture of pier: an internet-scale query processor. In *CIDR*, pages 28–43, 2005.

[5] D. Katabi, M. Handley, and C. E. Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM*, pages 89–102, 2002.

[6] F. Kelly. Mathematical modelling of the internet. In *Mathematics Unlimited – 2001 and Beyond*, pages 685–702. 2001.

[7] 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.

[8] F. Klemm, J.-Y. Le Boudec, D. Kostic, and K. Aberer. Improving the throughput of distributed hash tables using congestion-aware routing. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[9] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *IPTPS*, pages 207–215, 2003.

[10] G. D. Pifarré, 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.

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

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

[13] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, pages 161–172, 2001.

[14] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, pages 329–350, 2001.

[15] L. Schwiebert and D. N. Jayasimha. A universal proof technique for deadlock-free routing in interconnection networks. In *SPAA*, pages 175–184, 1995.

[16] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[17] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *NSDI*, pages 211–224, 2004.

[18] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM*, pages 175–186, 2003.

[19] G. Urvoy-Keller and E. Biersack. A multicast congestion control model for overlay networks and its performance. In *Networked Group Communication*, pages 141–147, 2002.

[20] 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*, pages 41–53, 2004.