Probabilistic Knowledge Discovery and Management for P2P Networks

Dimitrios Tsoumakos
Department of Computer Science
University of Maryland, College Park
dtsouma@cs.umd.edu

Nick Roussopoulos
Department of Computer Science
University of Maryland, College Park
nick@cs.umd.edu

***Abstract*— The Peer-to-Peer (P2P) paradigm dictates a distributed network model which enables the sharing of resources between its participants. In many cases, the location of these resources is a non-trivial task with network-wide effects. In this work, we describe the *Adaptive Probabilistic Search* method (*APS*) for search in unstructured P2P networks. *APS* utilizes feedback from previous searches to probabilistically guide future ones. Besides being a very cost-efficient technique, it enables the distribution and adaptation of search knowledge over the network. Based on that, we provide examples where this scheme can prove useful in more demanding environments.**

I. INTRODUCTION

Peer-to-Peer (hence P2P) networking has been growing rapidly in the last few years. Its success, originally boosted by some popular file-sharing applications (e.g., [1]), led to the emergence of numerous systems that utilize P2P technology (e.g., [2]–[5]).

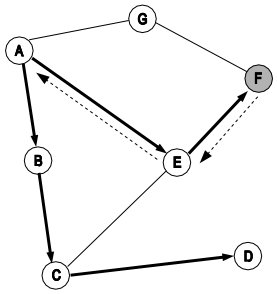
These systems have had a considerable and multi-dimensional impact. Focusing on technical aspects, ref. [6] reported that bandwidth consumption attributed to popular file-sharing applications amounts to a considerable fraction (up to 60%) of the total Internet traffic. So, while there exists a vast amount of "untapped" potential around the Internet, current resource-sharing applications consume huge amounts of bandwidth. P2P technology can play a key role in our efforts to tackle both issues, if carefully applied. In all cases, the first step involves the efficient discovery of the various resources inside a network.

Today, the most popular P2P applications operate on *unstructured* networks. The basic characteristics of these networks are the lack of 1) system control over object placement, 2) guarantees about search complexity and success. We should also note that in such systems peers arrive and depart at will, connecting in an ad-hoc fashion.

In [9], we described the problem of resource discovery in unstructured P2P networks as well as many proposed solutions. The shortcomings of most current methods relate to either excessive message consumption (during object location or index updates) or inability to adapt to dynamic workloads and environments.

In our work with *APS* [7], we proposed a new search algorithm that achieves high performance at low cost. In *APS*, a node deploys k walkers for object discovery, but the forwarding process is probabilistic instead of random. Peers effectively direct walkers using feedback from previous searches, while keeping information only about their neighbors. This scheme exhibits many plausible characteristics, namely high success rates, low bandwidth consumption and robust behavior in fast-changing environments.

In this paper, we describe this probabilistic forwarding scheme based on walker success/failure rates and how it can prove an efficient solution for the general search problem. We go further to identify various other knowledge resources that can be used by our algorithm, both individually and jointly. It is a fact that many contemporary applications require (or would benefit from) the utilization of more advanced features instead of a single one. We identify some possible cases and present modifications to *APS* in order to meet these goals.



Indices	Initially	Walkers' finish	After updates
A→B	30	20	20
B→C	30	20	20
C→D	30	20	20
A→E	30	20	40
E→F	30	20	40
A→G	30	30	30

Fig. 1. Search for an object using *APS*. The table depicts the change in index values, where X→Y denotes the index value stored at node X for neighbor Y relative to the requested object.

II. THE *APS* METHOD

A. Algorithm Description

In *APS*, each node keeps a local index consisting of one entry for each object it has requested, or forwarded a request for, per neighbor. The value of each entry reflects the relative probability of this node's neighbor to be chosen as the next hop in a future request for the specific object.

Searching is based on the simultaneous deployment of k walkers and probabilistic forwarding: The requester chooses k out of its N neighbors (if $k \geq N$, the query is sent to all neighbors) to forward the request to. Each of these nodes evaluates the query against its local repository and if a hit occurs, the walker terminates successfully. On a miss, the query is forwarded to one of the node's neighbors. This procedure continues until all k walkers have terminated, either with a success or a failure. So, while the requesting node forwards the query to k neighbors, the rest of the nodes forward it to only one. In the forwarding process, a node chooses its next-hop neighbor(s) not randomly, but using the probabilities given by its index values. At each forwarding step, nodes append their identifiers in the search message and keep a soft state about the search they have processed. If two walkers from the same request cross paths (i.e., a node receives a duplicate message due to a cycle), the second walker is assumed to have terminated with a failure and the duplicate message is discarded.

Index values stored at peers are updated in the following manner: When a node forwards the request to one or k of its neighbors, it pro-actively either increases the relative probability of the peer(s) it picked, assuming the walker(s) will be successful (*optimistic* approach), or it decreases the relative probability of the chosen peer(s), assuming the walker(s) will fail (*pessimistic* approach).

Upon walker termination, if the walker is successful, there is *nothing* to be done in the *optimistic* approach. If the walker fails, index values relative to the requested object along the walker's path must be corrected. Using information available inside the search message, the last node in the path sends an *"update"* message to the preceding node. This node, after receiving the update message, *decreases* its index value for the last node to reflect the failure. The update procedure continues along the reverse path towards the requester, with intermediate nodes decreasing their local index values relative to the next hops for that walker. Finally, the requester decreases its index value that relates to its neighbor for that walker. If we employ the *pessimistic* approach, this update procedure takes place after a walker succeeds, having nodes increase the index values along the walker's path. There is nothing to be done when a walker fails.

Figure 1 shows an example of how the search process works. Node A initiates a request for an object stored at node F using two walkers. Assume that all index values relative to this object are initially equal to 30 and the *pessimistic* approach is used. The paths of the two walkers are shown with thicker arrows. During the search, the index value for a chosen neighbor is reduced by 10. One walker with path (A,B,C,D) fails, while the second with path (A,E,F) finds the object. The update process is initiated for the successful walker on the reverse path (along the dotted arrows). First node E, then node A increase the value of their indices for their next hops (nodes F, E respectively) by 20 to indicate

object discovery through that path. In a subsequent search for the same object, peer A will choose peer B with probability $2/9$ ($=\frac{20}{20+40+30}$), peer E with probability $4/9$ and peer G with probability $3/9$.

B. Characteristics and Performance

Along the paths of all k walkers, indices are updated so that better next hop choices are made with bigger probability. Our learning feature includes both positive and negative feedback from the walkers in both update approaches. In the *pessimistic* approach for example, each node on the walker's path decreases the relative probability of its next hop for the requested object along the search path. If the walker succeeds, the update procedure increases those index values by more than the subtracted amount (positive feedback). Therefore, both *learning* and *unlearning* is performed during the search process: *Learning* helps in achieving high performance and discovery of newly inserted objects. *Unlearning* helps our process adjust to object deletions and node departures, redirecting the walkers elsewhere.

As an immediate consequence of that, more knowledge is obtained with more questions. The more feedback from the walkers, the more precise the indices become. That particularly suits the discovery of popular objects in the P2P network, which, according to studies [8], constitute over 60% of all searches. Another similar observation is that all nodes participating in a search will benefit from the process. This is a distinctive feature of our method, with indices being constantly updated using search results and not after object updates. Therefore, a node that has never before requested an object but is "near" peers that have done so, inherits this knowledge by proximity.

APS requires no message exchange on any dynamic operation such as node arrivals or departures and object insertions or deletions. The nature of the indices makes the handling of these operations simple: If a node detects the arrival of a new neighbor, it will associate some initial index value with that neighbor when a search will take place. If a neighbor disconnects from the network, the node simply removes all relative entries from its memory. No action is required after object updates, since indices are not related to file content. The *s-APS*

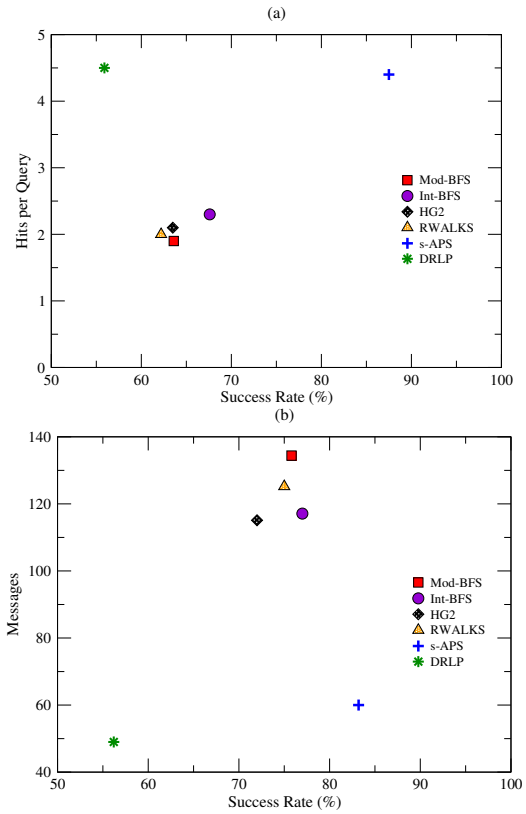


Fig. 2. Direct comparison of s-APS with various methods having (a) similar messages or (b) similar hits

improvement further reduces message consumption by adaptively switching between the optimistic and pessimistic strategies and minimizing the update messages. In [7] we also experimented with various update functions for the index values.

Our main performance metrics are the success rate, the average message consumption and the number of discovered objects per query. Most methods are effective in one or two of these metrics, but usually behave badly in the remaining one(s). While there is no method that meets all requirements, a good all-around performance is desirable. To illustrate this point, we present results where we compare *s-APS* with the methods described in [10]–[14]. In Figure 2(a) we show the success rate and hits per query for all algorithms when they display very similar message consumption. We would like our search schemes to be located around the upper-right corner of this field. In Figure 2(b) we display the success rate and messages per query for the algorithms when they discover a similar amount of objects. In this case, we would like a scheme to be

located around the lower-right corner of the graph. We clearly notice that *s-APS* proves a most reliable solution compared to these methods.

III. EXTENDING TO OTHER METRICS

The *APS* algorithm, as described above, utilizes a single source of knowledge, specifically path success or failure for requested objects. We showed that taking advantage of this property results in good performance for the general resource location problem. The specific characteristics of the indexing scheme that we would like to capitalize relate to the distributed computation of knowledge (knowledge can be shared and reinforced by multiple peers), small memory and bandwidth requirements and the ability for both learning and unlearning. An interesting question now is how could this indexing scheme be applied, so that more information can be utilized and more complex requirements be met.

One example where *APS* can be readily applied relates to the problem of load balancing. This problem is well-known especially in the client-server environment. In P2P networks, it is the case that peers play the roles of both client and server. Data replication allows frequent discovery of multiple peers that hold a particular object. Naturally, not all peers provide services of the same quality, so they may differ in the number of concurrent connections that they allow, their upload bandwidth, the quality of content they store, etc. Peers that share (locally or temporally) popular content or peers that store large numbers of objects usually receive a large number of requests. This results in performance degradation as perceived by the requester nodes. The *APS* scheme can be actively used to "direct" searches to different parts of a neighborhood, thus implementing a form of local load-balancing. Peers can change index values and re-direct walkers to non-congested parts of the network. In this case, index semantics will be associated to congestion information for the overlay links.

In reality, the decision of whether to choose a particular path/host/object is not based on a single metric. *APS* can be naturally extended to take into account network or application-dependent information. The interesting point in integrating more information is the fact that the scheme allows for its concurrent re-computation. The more searches are performed, the more accurate our indices become for a specific metric. As an example, consider that we want to incorporate a trust model into our system. Each peer holds a value $t_i \in [0,1]$ for each neighbor i. These values can have the meaning of how much this specific peer is satisfied by its recent transactions with its neighbors. Of course, index semantics can be interpreted in many different ways. The great advantage of such a scheme is the ability to share information among peers (a peer that just entered the network will take advantage of already built indices in neighbors). Moreover, indices along paths or located inside certain areas can be aggregated for more complex computations (e.g., computation of trust between distant peers, voting scheme inside a peer-neighborhood, etc).

Of course, any combination of different (even conflicting) metrics can be incorporated. All or some of the following parameters could be considered (depending on the application): The capacity of overlay links, the "trustworthiness" of each neighbor (however this may be defined), its sharing ability, network congestion, geographic location, etc. The weight to be assigned to each of these properties is application-specific; in a system where we worry about bogus content, we should give preference to peers with high trust values. If we plan on sharing large amounts of data (e.g, ISO images for a new Linux distribution), then we should aim for fast, reliable connections (high-capacity links together with large peer uptime).

We believe that this cost-efficient scheme can be a basis for many large-scale, distributed communication protocols. For our future work, we plan on pursuing the directions mentioned above and report on the relative advantages and disadvantages compared to existing schemes.

IV. SUMMARY

In this article we presented the *APS* method for search in unstructured P2P networks. *APS* deploys probabilistically directed walkers by utilizing information from past searches. The key characteristic of the scheme is that it allows for fast, joint learning, while being extremely bandwidth-efficient. We went further to propose the extension of knowledge integrated into the forwarding/learning process, as well as some possible applications that could benefit

from this technique. We believe that this light-weight probabilistic scheme can produce efficient applications for many low-guarantee networks.

REFERENCES

[1] "Napster website: http://www.napster.com," .

[2] "SETI@home web site: http://setiathome.ssl.berkeley.edu/," .

[3] I. Clarke, O. Sandberg, B. Wiley, and T. Hong, "Freenet: A Distributed Anonymous Information Storage and Retrieval System," *Lecture Notes in Computer Science*, 2001.

[4] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," in *SOSP*, 2001.

[5] R. Dingledine, M. Freedman, and D. Molnar, "The Free Haven Project: Distributed Anonymous Storage Service," *Lecture Notes in Computer Science*, 2001.

[6] "The impact of file sharing on service provider networks. An Industry White Paper, Sandvine Inc.," .

[7] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," in *P2P2003*.

[8] J. Chu, K. Labonte, and B. Levine, "Availability and locality measurements of peer-to-peer file systems," in *SPIE*, 2002.

[9] D.Tsoumakos and N. Roussopoulos, "A Comparison of Peer-to-Peer Search Methods," in *WebDB*, 2003.

[10] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A local search mechanism for peer-to-peer networks," in *CIKM*, 2002.

[11] S. Daswani and A. Fisk, "Gnutella UDP extension for scalable searches (GUESS) v0.1," .

[12] M. Stokes, "Gnutella2 specifications part one," .

[13] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS*, 2002.

[14] D. Menascé and L. Kanchanapalli, "Probabilistic Scalable P2P Resource Location Services," *SIGMETRICS Perf. Eval. Review*, 2002.