AGNO: An Adaptive Group Communication Scheme for Unstructured P2P Networks

CS-TR-4590, UMIACS-TR-2004-33

Dimitrios Tsoumakos
Department of Computer Science
University of Maryland
College Park, MD 20742, U.S.A.
dtsouma@cs.umd.edu

Nick Roussopoulos
Department of Computer Science
University of Maryland
College Park, MD 20742, U.S.A.
nick@cs.umd.edu

***Abstract*— We present the *Adaptive Group Notification* (*AGNO*) scheme for efficiently contacting large peer populations in unstructured Peer-to-Peer networks. *AGNO* defines a novel implicit approach towards group membership by monitoring demand for content as this is expressed through lookup operations. Utilizing search indices, together with a small number of soft-state shortcuts, *AGNO* achieves effective and bandwidth-efficient content dissemination, without the cost and restrictions of a membership protocol or a DHT. We present several simulation results over different topologies using both synthetic and real workloads, to evaluate its performance. Our method achieves high-success content transmission at a cost at least two times smaller than proposed techniques for unstructured networks.**

I. Introduction

A multicast transmission is defined as the dissemination of information to several hosts within a network. These hosts are interested in receiving the same content from an authority node (such as a web server) and naturally form a group. The lack of deployment of multicast communication in the IP layer has led to the development of various application-level multicast protocols, in which the end hosts are responsible for implementing this functionality. One-to-many communication is a very useful mechanism for a variety of network applications (e.g., [1], [2]).

Peer-to-Peer (hence P2P) computing represents the notion of sharing resources available at the edges of the Internet. The P2P paradigm dictates a fully-distributed, cooperative network design, where nodes collectively form a system without any supervision. As the applications that embrace the P2P paradigm grow, a number of methods have also been proposed to implement multicast communication utilizing some popular P2P overlays, e.g., [3]–[6]. Nevertheless, these approaches take advantage of the structure that DHTs (distributed hash tables) provide. In many realistic scenarios, the topology cannot be controlled and thus DHTs cannot be used (e.g., ad-hoc networks or current large-scale unstructured overlays). Other approaches require frequent communication overhead between group members and explicit membership protocols. These schemes often prove unsuitable because of the generated traffic for large and dynamically changing group populations.

Today, the most popular P2P applications operate on *unstructured* networks, where peers connect in an ad-hoc fashion, the location of the documents is not controlled by the system and no guarantees for the success or the complexity of a search are offered to the users. More important, peers obtain only local knowledge of a network where nodes enter and leave frequently. For such systems, contacting large numbers of nodes is implemented by either broadcast-based schemes (e.g., Gnutella [7], Modified-BFS [8]), or *gossip*-based approaches, e.g., [9]–[11]. Both produce large numbers of messages by contacting many peers inside the network.

In this paper, we present the *Adaptive Group Notification* (*AGNO*) method for this problem. Our work is motivated by the following observation: Group membership can be implicitly defined through the interest that peers show by conducting searches. Locally maintained state during these

lookup operations can assist in the process of disseminating content to interested peers. Regardless of the low-guarantee nature of our target environments, we aim at producing a protocol that is:

- *Efficient:* Our method should be able to contact a high percentage of interested peers with low message overhead.
- *Scalable:* The scheme should be able to scale to very large group sizes (over 1,000 peers).
- *Robust:* We would like to avoid the necessity of a single point of contact or group leader as well as the burden of costly message exchanges in case of member arrivals and departures.
- *Adaptive: AGNO* should adapt to changes in the group size and to dynamic workloads.

A. Motivation and Overview of our Approach

P2P computing is gaining an increasing amount of attention from both the academic and the large Internet community. The number of applications utilizing this technology is constantly increasing (web caching [12], instant messaging [13], e-mail [14], etc). The importance and applications of group communication schemes have been well-defined in past and recent research work (e.g., [2], [3], [5], [10]). Our work aims at providing peers in dynamic, unstructured environments with an effective yet inexpensive mechanism to disseminate information to groups of nodes interested in their content.

We assume a fully distributed and unstructured system, where peers share and request resources replicated inside the network. Users are interested in mutable objects, for example results of a sports meeting in real time, current temperature or weather maps, stock quotes, etc. There exist some nodes (similar to the web servers or mirror sites in the Internet) that provide with fresh content, but their connectivity or availability varies, as happens with all other network nodes. Peers that are interested in retrieving the newest version of the content conduct searches for it in order to locate a fresh or closer replica. In this environment, interest in a specific object is tied to the lookups generated for it. We argue for a push-based approach, where a server node can forward notifications (or other object-specific information) towards the interested hosts. Our assumption is that peers which have recently searched or retrieved an object would also be interested in receiving such content. For example, it is safe to assume that a host frequently querying for the price of a quote or the temperature of an area would like to be informed about an update or another object-related notification.

It is important to note here that peers still search and retrieve objects in a distributed manner. The notification itself may or may not be directly related to a specific object: A severe weather alert to be effective in the next 3 hours is not related to the current area temperature. A change in the scores or quote prices, on the other hand, is directly linked to the content of the object. Group communication (especially for large groups) requires a considerable amount of bandwidth. Content providers can assess the importance of various updates/notifications and choose to push those that would be the most beneficial.

On a more technical note, the forwarding path between any two given peers in a DHT remains the same with high probability. This is a feature that many approaches utilize in order to construct efficient multicast paths. This is not the case for unstructured P2P networks: Peers have multiple (and dynamically changing) communication paths with each other. Therefore, a notification scheme for such networks can also be used to simulate that functionality and identify reverse paths from the destination (replica location) to the requesters. This information can in turn be used in a variety of problems (e.g., load balancing and replication [15]).

Our approach combines the utilization of state accumulated during the search process together with probabilistically stored shortcuts. The first indicates the amount of demand for a specific object and can be used to infer membership and guide our content dissemination. Note that our method builds its knowledge by only monitoring the independently conducted lookups and does not produce any artificial or control messages. By also allowing peers to locally store a constant amount of requester addresses (called *backpointers*), we show that *AGNO* achieves a robust, scalable behavior in a variety of environments and group sizes. Our method utilizes a simple *binning* scheme as well as adaptive index *aging* to adjust its performance to different workloads and member joins/leaves. *AGNO* does not require any global knowledge, existence of a special contact

node or any membership message exchange. Finally, its performance can be easily tuned to fit specific application requirements.

The contributions of our work are two-fold: First, we propose a group communication algorithm specifically designed for unstructured networks. To the best of our knowledge, this is the first work that couples searches to membership information for such environments. Second, we present detailed simulation results over a variety of topologies and dynamic workloads, showing that *AGNO* can achieve high-success and low-overhead delivery for many realistic and highly dynamic scenarios.

The rest of this paper is organized as follows: Section II presents the related work. In Section III we describe our scheme in detail, while in Section IV we present the simulation results. Finally, Section V contains our conclusions.

II. Related Work

The problem of distributing content to multiple hosts is well-studied. We categorize existing methods into general application-layer multicast protocols, multicast for structured P2P overlays and, finally, approaches for unstructured networks.

A. Application-layer Multicast

Proposed approaches roughly fall into three categories: The mesh-first category (e.g., Narada [16]), where nodes form a random mesh between them and then compute unicast paths for each pair of members. This approach requires control overhead quadratic to the group size with refresh messages. In the tree-first approach (e.g., Yoid [17]), peers directly form a data delivery tree and also maintain a few extra links to exchange control messages. Finally, in the implicit approach (e.g., NICE [2]), both control and delivery structures are implicitly defined by the underlying routing protocol. For example, NICE arranges members into a hierarchy of layers and clusters and defines processes for member arrival/departure and cluster merge/split. All these approaches require the existence of a designated host to initiate the membership process, periodic exchange of control messages and also significant overhead for member joins/leaves.

B. Multicast over P2P Overlays

The algorithm described in [6] describes a broadcast mechanism that operates over CAN [18]. Nodes forward to their neighbors in the d-dimensional space, as this is defined in CAN. There are also provisions made to eliminate duplicate messages and prevent looping of the packets around the coordinate space.

Scribe [4] is implemented on Pastry [19]. Interested hosts route their requests towards the node responsible for the group's key (the root). Each node on the path checks if it is a current member of the group. If this is the case, it registers the source node as its child in the multicast tree and stops the forwarding process. Otherwise, it stores the ID of the source and makes a join request towards the root. Scribe is a decentralized and scalable protocol that takes advantage of the overlay structure to produce a balanced delivery tree.

Bayeux [3] is implemented on Tapestry [20]. The difference with Scribe is that join/leave operations go through the root of the tree, making it less scalable. Overcast [5] also requires coordination with the root node, while it builds its multicast tree in a manner similar to Yoid. The work in [21] contains thorough descriptions and performance comparisons for representative schemes from this category.

C. Group Communication in Unstructured Overlays

Many search schemes for unstructured P2P networks have been proposed that implement flooding or its modifications in order to contact large numbers of nodes. Examples include the gnutella flooding algorithm [7], the modified-BFS scheme [8], the iterative deepening method [22], etc. All these techniques produce a large number of messages, cannot adapt to variable group sizes and use blind forwarding, which results in many non-members receiving the message.

An alternative solution to the problem is presented by a variety of gossip algorithms, where each member is responsible for forwarding a notification to a randomly selected subset of the group. These approaches have been used in a variety of different scenarios (e.g., distributed databases [23], publish-subscribe systems [24]) and have proved to be a robust solution in the face of member/network

failures at the cost of inducing extra traffic on the network.

In Lpbcast [24], membership is achieved by a periodic gossiping of subscriptions: peers transmit a set of subscriptions that they recently heard to a random subset of their locally known group members. Upon receiving such a message, nodes replace a random subscription from their local lists with the new one. To achieve the probabilistic guarantees offered by similar schemes, the size of the group and the local list size must be fixed, which is not the case in highly dynamic networks.

SCAMP [11] is a decentralized membership protocol that utilizes gossiping. Joining members subscribe by contacting a random existing member. Upon receiving a subscription request, a member forwards it to all the members in its local repository. Nodes decide probabilistically whether to store or forward the subscription. For the unsubscription process, a node notifies the locally known members to replace its ID with the IDs of the members it has received messages from. Group communication is performed in the standard gossip-based manner. SCAMP is shown to converge to a local state of slightly over $log(n)$ member IDs, which guarantees with high probability that all members will receive a notification.

In [10], the push phase of an update algorithm for unstructured P2P networks is a rumor-spreading scheme: each peer receives an update message along with a partial list of other peers to which the update has been sent. If the update has not been received before, it is forwarded to a different subset of members with a certain probability. In [9], peers that have received a message less than F times, forward it to B randomly selected neighbors, but only those that the node knows have not yet received it. The deterministic version of that algorithm requires global knowledge of the overlay. Nodes forward messages to all neighbors with degree equal to 1, plus to B remaining neighbors that have the smallest degrees.

In contrast, our approach requires no special group management process, while its forwarding scheme is an adaptive selection between neighbors and shortcuts, relative to the quality of the local search knowledge.

III. The AGNO Protocol

In this section we present *AGNO*. We first describe our general framework and also give an overview of the *APS* search algorithm which is utilized by our approach. We then present the *AGNO* scheme in detail.

A. Our Framework

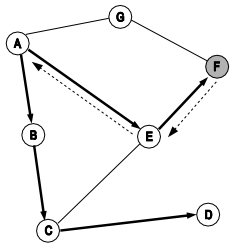
We assume a pure P2P model, with no imposed hierarchy over the set of participating peers. All of them may equally serve and make requests for various objects. Each peer retains its own collection of objects which are locally maintained. Peers and objects are assumed to have unique identifiers. Ignoring physical connectivity and topology from our talk, we assume that peers are aware of their one-hop neighbors in the overlay. The system can generally exhibit a dynamic behavior, with peers entering and leaving at will and also updating their local repositories. We should also note that we do not expect the overlay structure to be static, since nodes are not guaranteed to connect to the same neighbors each time.

A multicast transmission in this setting (also referred to as the notification or push phase hereafter) is initiated by an object holder (or *server* node) and its target is to reach as many group members (or requester nodes) as possible with the least amount of messages over the overlay. The focus of this work is to describe an efficient mechanism for such transmissions and not to define their content.

B. Overview of the APS Method

In *APS* [25], each node keeps a local index consisting of one entry for each of its neighbors on a per-object basis. 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 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



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

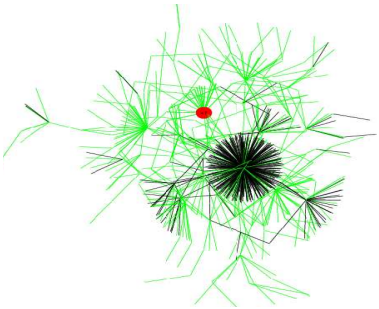


Fig. 1. Search for an object with the APS method. X→Y denotes the index value stored at node X for neighbor Y relative to the requested object.

a failure. The paths of the walkers are not chosen at random, but using the probabilities given by the index values of the intermediate peers. These values are updated in the following manner: Upon forwarding, a node pro-actively increases or decreases the index values for the peer(s) it selected, assuming the walker(s) will be successful or not (*optimistic* and *pessimistic* approaches respectively).

Upon walker termination, if the walker is successful, there is nothing to be done in the optimistic approach. If the walker fails, the update procedure is initiated with nodes along the reverse path decreasing their local index values relative to the next hops for that walk. For the pessimistic approach, this procedure takes place only after a walker succeeds.

Figure 1 shows an example of how the search process works. Node A initiates a request for an object served by node F using two walkers. All index values for this object are initially equal to 30 and the *pessimistic* approach is used. During the search, the index value for a chosen neighbor is reduced by 10. The walker with path (A,B,C,D) fails, while the second (A,E,F) succeeds. The update process is initiated on the reverse path (along the dotted arrows), with nodes E and A increasing the index value for nodes F, E respectively by 20. 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 $3/9$.

APS exhibits a learning feature with both positive and negative feedback from the walkers. Positive feedback helps in achieving high performance and discovery of new objects, while negative feedback helps our process adjust to object deletions and node departures, redirecting the walkers. Knowledge is refined as more questions are made inside the network. Another characteristic is that all nodes participating in a search will benefit from the process. Therefore, a node that has never before requested an object but is "near" peers that have done so, inherits this knowledge by proximity.

To better understand the state that *APS* builds, we present a part of a 4,000-node power-law graph on the right part of Figure 1. The oval represents a server node and arcs represent links to or from the server and 400 randomly selected nodes that search for the object. Links drawn with light lines represent high index values (i.e. many successful searches through them), while dark black links show paths with low probability of success. After only a small number of requests, most paths that connect the server to the requesters comprise of light-colored lines.

C. AGNO Protocol Description

The rationale behind *AGNO* relates to the observation that efficient group communication comes at a cost. In current approaches, this cost is paid by either a membership management protocol or an overlay infrastructure. Our goal is to provide with the missing state that can allow for content dissemination to a group of peers, but in a way consistent to the nature of an unstructured P2P system. Our approach couples search knowledge with the information necessary to contact interested peers. In *AGNO*, the equivalent of group membership is the demand for an object (or group of them), realized through searches and object sharing that are *independently* conducted by peers. The granularity can be as coarse or fine-grained as the application requires. For the remainder of this paper we assume

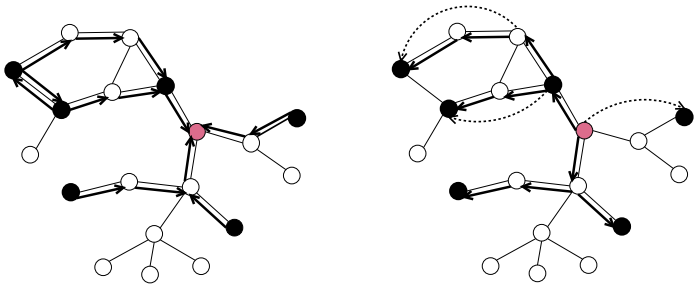


Fig. 2. Search for an object stored at the gray node and the push phase from this node towards the requesters (black nodes)

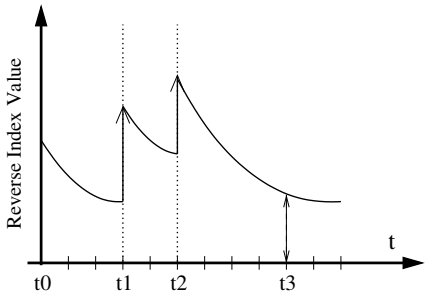


Fig. 3. Example of the reverse index value update process

that groups are formed on a per-object basis.

After each search with the *APS* scheme, peers accumulate knowledge about the relative success of a search through each of their neighbors. Intuitively, overlay paths that comprise of high index values are the ones most frequently used to connect requesters and object holder(s). In *AGNO*, nodes utilize those indices in order to forward group messages towards possible group members during the push phase. Note here that, although the *APS* method is used as a means to provide with the soft state, our approach can be used with a variety of search mechanisms, as long as they support a similar demand incentive.

We now describe the nature of the index values that are stored at each peer. *APS* keeps a local view (an index value) for each neighbor. For *AGNO*, each peer P needs to maintain the index values that P's neighbors hold relative to P. If A→B denotes the index value stored at node A concerning neighbor B for a particular object, then peer P must know $X \rightarrow P$, for each neighbor X. These values can be made known to P either implicitly or explicitly: In the first case, peer P can infer the index $Q \rightarrow P$ if it knows about the update process used (optimistic or pessimistic) and the initial value. In the explicit approach, whenever a search is conducted and Q forwards to P, it piggybacks $Q \rightarrow P$. We call these new stored values the *reverse indices*, to distinguish them from the indices used by *APS* in searches. For the rest of our discussion, we assume that the explicit approach is used.

Reverse indices are not the only state that our method utilizes. During the search, intermediate nodes decide with probability p_r whether to store the requester's ID or not. For a search path h hops long, the (ID, address) pair of the requester will be stored on hp_r peers on average. With this scheme, we create a number of soft-state shortcuts called *backpointers* along the search paths which point to group members. Each peer can individually decide on the maximum number of backpointers stored. For simplicity, we assume that all nodes can store a maximum of c backpointer values. Backpointers are soft-state that gets invalidated after some amount of time.

In the push phase, a peer that receives a group message forwards it to its neighbors using the reverse index values. We consider the following forwarding schemes:

- Forward to one or more neighbors chosen with probabilities given by the reverse indices or to those with the k largest values
- Forward to all neighbors with reverse index value larger than a defined threshold

Notifications carry a TTL field which is decremented whenever it reaches a node. A group message is discarded either when its TTL value reaches zero or if it is a duplicate (a node receives the same notification more than once due to a cycle). Moreover, a peer forwards to each of its valid backpointers with probability p_n. These messages have a $TTL = 1$ and do not travel further. Therefore, our scheme combines a selective, modified-BFS forwarding augmented with shortcuts in order to contact the group members. This is shown pictorially in Figure 2.

We now discuss how the aforementioned state is maintained at each peer. The backpointer values expire after a certain amount of time. Since our incentive to push a message is the demand on a per-object basis, new backpointers replace the oldest valid ones (if a node already has c valid back-

pointers). As searches take place inside the system, the backpointer repositories get updated, while the probabilistic fashion in which they are stored guarantees a diverse collection of (ID, address) pairs. Reverse indices get updated during searches, but this is not enough: There may be peers that have searched for an object and built large index values, but are no longer interested in receiving notifications (i.e., stop querying for that object). If searches are no longer routed through those peers, the reverse index values (which reflect *APS* indices) will not be updated and will remain high.

To correct this situation, we add an *aging* factor ξ to the reverse indices, which forces their values to decrease with time. Peers need to keep track of the time that a reverse index was last updated in order to acquire its correct value before using it. When a peer receives a search message, it sets the corresponding reverse index to the piggybacked value and its last modified field to the time of receipt. Figure 3 shows how this process works. The value of the index decreases exponentially, while two searches at times t_1, t_2 reset its value. A push message received at time t_3 will use the value as shown in the figure. The last modified value is also reset when a reverse index is used, since a peer computes its current value before using it.

In the next section we describe in more detail how our protocol proceeds in the computation of the various parameters described above.

D. Protocol Specifics

1) Space Requirements: The amount of space required by the peers is $O(2d+c)$ per object, where d is the average node degree in the overlay and c is the maximum number of backpointers stored. Even if nodes want to keep track of large numbers of objects, the space requirements are in the order of a few tens of megabytes, definitely affordable by the vast majority of modern hosts (typical 1GB of main memory configurations). For about 1 million objects, assuming $c = d = 4$, each peer would need approximately 48MB of memory for *AGNO*.

2) Forwarding: Nodes use a threshold parameter *Thresh* in order to choose the neighbors to which a notification will be forwarded. Neither the probabilistic or the top-k value schemes are suitable, as they fail in a variety of cases. Consider for example a peer with very low values for all its neighbors. Thresholding enables peers to forward to the most "promising" (active in searches) parts of the overlay. A good first approximation is for each peer to use the average of all its neighbors' indices as *Thresh*. Nevertheless, both the average and the median values fail as well in various circumstances (e.g., when all indices have a very close low or high value).

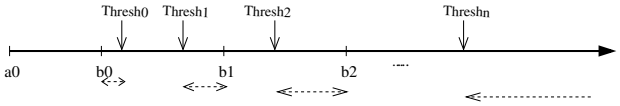


Fig. 4. Sample binning scheme with the respective threshold values for each interval

3) Local Threshold Computation: After each peer computes the average of its neighbors' reverse index values at time t (aiv_t), it uses a globally defined *binning* scheme to come up with the actual value for *Thresh*. The binning method divides the space of index values into a set number of bins, $\{Bin_i = ([a_i, b_i), Thresh_i)\}$. Bin_i is characterized by its lower and upper limit values a_i, b_i ($a_0 < b_0 = a_1 < b_1 = a_2 ...$) and a $Thresh_i$ value. The final threshold value is $Thresh = Thresh_i$, if $aiv_t \in [a_i, b_i)$. For example, assume we use a 2-bin scheme, $\{Bin_1 = ([0,50), 40), Bin_2 = ([50,\infty), 100)\}$. If $aiv_t = 75$, that node will forward to all neighbors with reverse index value over 100. Bins represent an approximation that maps reverse indices to a value representing their quality. Higher numbered bins represent higher quality indices.

Values $Thresh_i$ are chosen such that $Thresh_{i-1} - b_{i-1} > Thresh_i - b_i$ and $Thresh_{i-1} < Thresh_i$. For small i values we should pick few neighbors (therefore a high threshold relative to the bin's interval), while for large i (i.e., high quality bins), most of the neighbors need to be chosen. Note that we do not require $Thresh_i$ to belong to $[a_i, b_i)$, nor do we require that $b_i - a_i = b_j - a_j, i \neq j$. As a simple heuristic for selecting the $Thresh_i$ values, given $Thresh_0$ near or larger than b_0, we set $Thresh_i = (2Thresh_{i-1} + b_i - b_{i-1})/2$. Figure 4 gives a graphic description of our binning scheme. Its granularity, controlled by the number of defined bins, can be as fine-grained or coarse as our application requires.

4) Reverse Index Aging: APS updates its index

values after either a success or a failure, achieving both learning and unlearning. The latter is very important for *AGNO* as well: Peers that lose interest in an object should be left out of the push phase as quickly as possible. Our scheme uses the aging factor ξ together with the last modified time of each reverse index to reduce the influence of inactive ones. Assuming index $P \rightarrow Q$ was last modified at time t_{last}, its value at time $t \geq t_{last}$ is: $P \rightarrow Q(t) = (1-\xi)^{t-t_{last}} P \rightarrow Q(t_{last})$, where $\xi \in [0,1]$. For $\xi = 0.2$, a reverse index value will be 80% of its last modified after one time unit.

The value of ξ dictates how aggressive our aging will be. It depends on the rate at which requests (and therefore index updates) occur: The larger the rate of searches, the more aggressive the aging can be. Nevertheless, it is still application-dependent, since the rate at which notifications are issued (or even their content) largely affects the aging factor. For example, in sharing stock market data, for the duration of a peer's online time it can be assumed that a user is always interested in her portfolio. We define λ_r, λ_n to be the average rates at which a peer or server makes requests or issues notifications respectively.

For the remainder of this paper, we assume that peers use the same value for ξ which satisfies the inequality: $(1-\xi)^T max\_reduced\_Thresh < \min_i(Thresh_i)$ (1). In effect, we pick ξ such that any reverse index with value less or equal to *max_reduced_Thresh* will be reduced below the lowest threshold (and thus will not be selected) if not used for T time steps (T is defined as our "tolerance" parameter). The maximum $Thresh_i$ represents the minimum high-quality index value. Therefore, by setting $max\_reduced\_Thresh = \max_i(Thresh_i)$, we choose ξ such that all reverse indices up to that level of quality are discarded after a period of time T without getting updated. Choosing larger *max_reduced_Thresh* values results in a more aggressive aging. The same is true for choosing smaller T values. Assuming that, in the vast majority of cases, notifications are considerably less frequent than requests, we set $T = O(1/\lambda_r)$, which defines the tolerance interval to be in the order of the average request interarrival period. This is done in order to quickly identify and decrease idle indices in the overlay.

5) Estimation of λ_r: In order for our scheme to work without requiring a priori knowledge of the request rate but also to be able to adapt to changes in the workload, we need an effective yet inexpensive mechanism to estimate its value and compute the new ξ before each push. This value is then piggybacked downstream and used by all receiving nodes. In order to estimate λ_r, we need the zeroth and first frequency moment of the request sequence arriving at a server. F_0 is the number of distinct IDs that appear in the sequence, while F_1 is the length of the sequence (number of requests). Servers can easily monitor the number of incoming requests inside a time interval. Many efficient schemes to estimate F_0 within a factor of $1 \pm \varepsilon$ have been proposed (e.g., [26], [27]). We use one of the schemes in [26], which requires only $O(1/\varepsilon^2 + \log(m))$ memory bits (only at servers), where m is the number of distinct node IDs. In reality, m is in the order of the distinct peers within TTL hops from a server, since only these nodes can reach it. After each push phase, both estimates are reset and a new estimation cycle begins.

6) Backpointer Selection: Finally, we specify which backpointers are used by a node that receives a group notification message. Clearly, following the same number of backpointers at different peers and times is not efficient. Our method utilizes the local thresholding computation to assist in the process of selecting valid backpointers. As we mentioned before, the threshold value is representative of the average quality of a peer's reverse indices (higher bins choose on average more neighbors to forward to).

Given that a peer's threshold bin is i at time t, the probability with which each stored backpointer will be followed is p_{ni}, given from the set $\{p_{n1}, p_{n2}, ...p_{ni}, ...\}$ (i.e., one p_n value for each bin). We choose those values such that $p_{ni} > p_{nj}\ \forall i < j$, since better quality bins forward to more neighbors and need not waste more bandwidth. With this scheme, *AGNO* adaptively balances the amount of forwarded messages per peer between the shortcuts and the neighbors according to the current quality of its reverse indices.

7) Summary: AGNO is a probabilistic group notification scheme that integrates search indices with a constant amount of shortcuts to effectively route

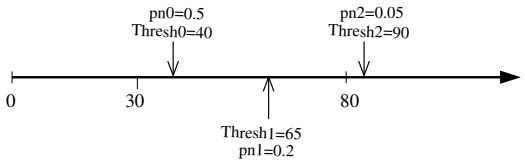


Fig. 5. Our binning scheme with the respective threshold and p_n values for each interval

messages in an unstructured overlay. It utilizes a tunable binning scheme to choose between the exact amount of useful information from each source and an aging mechanism to gracefully adapt to member departures, requiring no explicit cooperation on their part.

IV. Simulation Results

A. Simulation methodology and compared methods

We use a message-level simulator written in C (about 2,100 lines of code) which runs on a linux-based platform using an Athlon 2.1GHz processor and 1GB of main memory. Requesters make searches for objects using *APS* at rate λ_r (exponentially distributed interarrival times), while servers initiate push transmissions at rate λ_n. At each run, we randomly choose a node that plays the role of a server and a number of requesters, also uniformly at random. Results are averaged over several hundred runs.

We present results for both *random* and *power-law* graphs. There has been strong evidence [28] that connects large-scale unstructured P2P networks to a power-law topology. We utilize the *BRITE* [29] and *Inet-3.0* [30] topology generators to create the random and power-law graphs respectively. We consider 10K node graphs with average node degrees around 4 (similar to gnutella snapshots [28]). Results for graphs up to 50K nodes and larger average degrees are qualitatively similar.

We use the following metrics to evaluate the performance of a scheme: The *success rate*, which is the ratio of contacted group members versus the total number of group nodes and the bandwidth *stress*, which we define as the ratio of the produced messages over the minimum number of messages in order to contact all members.

AGNO Parameters: We choose to set $c \approx d$, which reserves an amount of space for backpointers roughly equal to the average node degree. Ref. [28] shows that over 90% of the node pairs in gnutella are around 5 hops away. Given this value as an estimate for the TTL parameter, we set $p_r \geq \frac{1}{TTL}$, so that at least one peer on the search path can store the requester's address. Given that the default index value for *APS* is 30, increased by 10 for each successful walk and linearly decreased after a failed one, we employ a 3-bin scheme with backpointer probabilities as shown in figure 5. The first bin represents indices below the initial value (very few or no successes), the second those with some hits and the last those with even more successes. While we experimented with various configurations, using more bins and different thresholds, we prefer to study the performance of our method with this simple scheme. From (1) and setting $T = 2T_r$ (where $T_r = 1/\lambda_r$) we have: $\xi = 1 - 0.44^{0.5\lambda_r}$. The value of λ_r (and therefore ξ) is estimated right before each server push using $\varepsilon = 0.1$.

We compare our method against 3 algorithms: The SCAMP membership protocol [11] and the two rumor-spreading schemes in [9]: *Rumor Mongering* (RM) and its deterministic version (det-RM), where peers have complete topology information. For SCAMP, we first run the membership phase, in which we favor the method by assuming joining peers know all already joined members. The parameters for those three methods are the *branching factor B*, which represents how many other peers shall be contacted per forwarding step and the *seen* value F that represents how many times a peer can receive the same message before dropping it.

Finally, we also designed and implemented a pure shortcut selection scheme (*Shortcuts*) inspired by the DHT-based multicast tree creation. Search packets carry the (ID, address) values of the last node along the path interested in the object so far. Initially, this pair contains the requester node's information. During the search, an interested peer that receives a search message, decides with probability p_r whether to store the last member's ID or not. Moreover, it replaces this ID with its own before forwarding the request. With this scheme, we create a small sub-overlay of soft-state backpointers with direction from the object holders towards the group members. For simplicity, we assume the same maximum number of shortcuts as in *AGNO*. In the

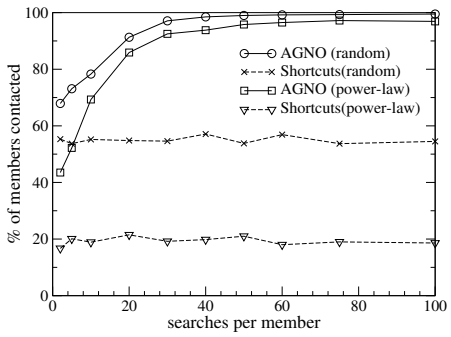


Fig. 6. Success over variable number of searches

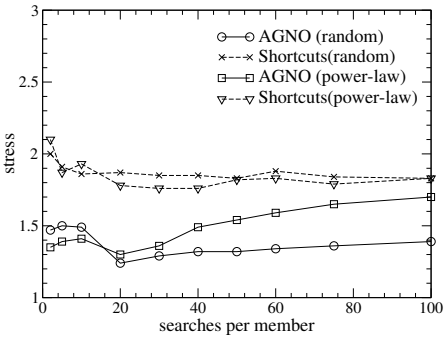


Fig. 7. Stress over variable number of searches

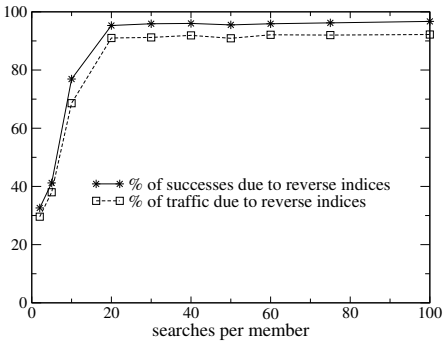


Fig. 8. Utilization of pure forwarding vs. backpointers

push phase, a peer forwards to all valid shortcuts, using the standard TTL scheme (unlike *AGNO*, where backpointers are contacted with a $TTL = 1$).

B. Basic performance analysis

In this first set of experiments, we try to quantify the ability of our method to contact requesters without considering time-related aspects (i.e., take a snapshot of its operation in time, or $\xi = 0$). We first try to identify the effect that *APS* has on the performance of *AGNO*. For a group size of 500 peers, we vary the number of requests each of them makes before a single push phase occurs. We report the stress and success rates averaged over 20 random 10,000-Node topologies and 20 10,000-Node power-law topologies ($d = 4$ and $d = 4.1$ respectively) with 1,000 runs for each graph. Figures 6 and 7 present the results for *AGNO* and *Shortcuts* which are affected by the number of searches.

We notice that the pure shortcut scheme, while capable of identifying a non-negligible number of members, cannot provide an efficient notification method. *AGNO* quickly contacts the majority of requesters after only a few searches take place, while maintaining a low stress factor. As our scheme adapts to increased quality indices, there exists a slight variation in the stress. This is due to the fact that after a certain number of queries, peers switch to a different (higher) bin on average. Figure 8 shows the percentage of contacted members and messages of *AGNO* purely attributed to forwarding (not backpointers). As we move from less to more precise reverse indices (from fewer to more queries), our method uses a decreasing number of backpointers. These results also depict the usefulness of the backpointer scheme as for less accurate indices they can provide with over 50% of the contacted members.

TABLE I

SUCCESS RATE AND STRESS RESULTS FOR THE REMAINING METHODS WITH 500 REQUESTERS

	SCAMP	**RM**	**det-RM**
10K random	$(0.89, 2.7)$	$(0.89, 34.5)$	$(0.98, 31.1)$
10K PLAW	$(0.68, 2.1)$	$(0.27, 13.6)$	$(0.65, 10.8)$

TABLE II

EFFECT OF PARAMETER c

	10 queries/member		20 queries/member	
	success	**stress**	**success**	**stress**
c=1	68.7%	1.17	90.3%	1.16
c=2	73.5%	1.27	91.5%	1.20
c=4	77.9%	1.42	91.6%	1.23
c=8	79.6%	1.80	92.5%	1.37
c=16	81.2%	2.80	92.9%	1.49

The distinctiveness of the power-law topologies, where about 34% of the peers have degree one, forces fewer paths to be used compared to the random topologies. This, combined to the fact that no aging is performed, explains why the stress for *AGNO* slighty increases with more requests in Figure 7. The respective results for the remaining methods (not affected by searches) are shown in Table I. *AGNO* proves very accurate (in the big majority of runs) and also the most bandwidth-efficent of the compared methods. All three rumor-spreading schemes show considerably worse numbers in the power-law topologies. *det-RM* is much more effective than *RM* in such graphs, which is in accordance to the findings of [9].

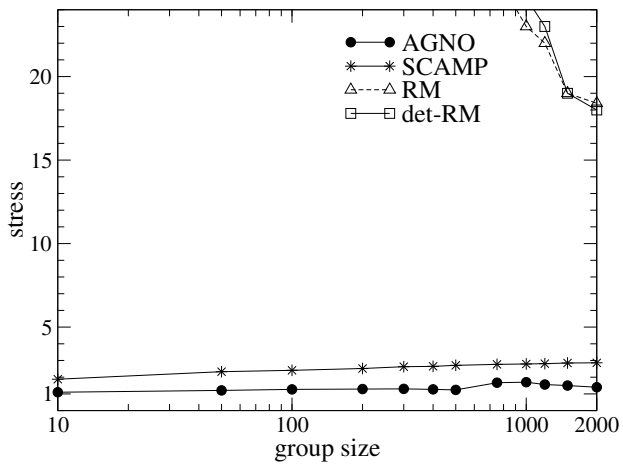


Fig. 9. Stress values over variable group size

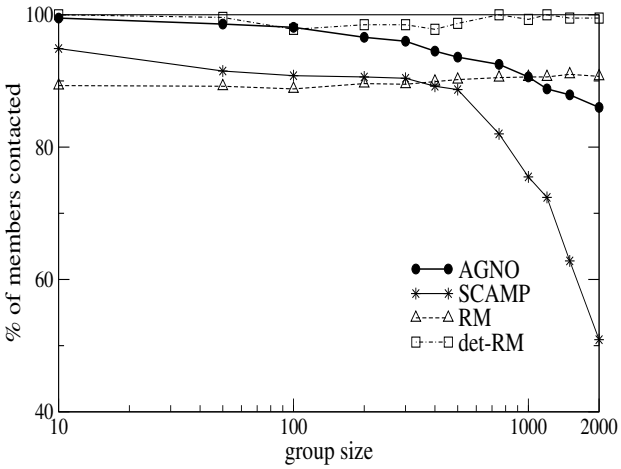


Fig. 10. Success over variable group size

Table II summarizes the effect that a change in the number of maximum stored backpointers (c) has on the performance of *AGNO*. We select two runs from the previous experiment, where each of the 500 members make 10 or 20 queries in the random topologies. For 10 queries/requester, many peers fall into bins 1 and 2 on average, while the majority of nodes operate on bin 3 with twice as many queries. With less queries (and larger backpointer usage), the increase in the success rate over our selected $c = 4$ is very small compared to the stress increase. As the indices get more accurate, the method becomes almost insensitive to the value of c.

Next, we try to measure the scalability of our method relative to various group sizes, ranging from 10 to 2,000 peers, using the random topologies. Requesters make only 10 searches on average, immediately followed by a single push phase from the server node. For SCAMP, the membership protocol is run before each different group size. For RM, det-RM and SCAMP, we set $B = 3, F = 1$, which proves the best combination taking into consideration both the success rate and stress metric. Figures 9 and 10 present the results.

Our method is very successful in all group sizes, deteriorating only slightly as the members increase. This happens because with more requesters, their average distance from the server increases (the number of peers reachable from a node increases exponentially with the hop distance). This makes *APS* searches (and its indices) less accurate for some requesters. The RM schemes produce a similar number of messages regardless of the group size (average stress between 1600 and 20), while the closest competitor (SCAMP) has roughly twice the stress value of *AGNO*, without including the overhead of the membership phase. Our method manages to contact a very high percentage of the members (86-99.5%) using an almost constant message ratio over the group size.

C. Sensitivity to λ_r

In this section, we try to evaluate the effectiveness of our λ_r estimator and the computed ξ values over the random topologies. Results for the power-law graphs are qualitatively similar.

The value of T defines how aggressive the aging is. The smaller it gets, the bigger ξ becomes and thus the bigger the reduction in the reverse index values. Figure 11 shows how the success rate of *AGNO*, given 1,000 peers making requests at $\lambda_r = 1/sec$ (and $T_n = 10sec$), varies by changing the value of T relative to the average request period $T_r = 1/\lambda_r$. Our default choice for $T = 2T_r$ yields very good results, while choosing values close to the request period also produces fast learning. As T decreases more, the success rates increase at much smaller rate. Surprisingly, even if we employ twice as aggressive an aging as the average request rate, over 80% of the members will be contacted after three *AGNO* pushes. Nevertheless, it is not safe to assume that the larger the value of T the better. This would be the case if, for example, we had a static group size (no aging necessary); a significant

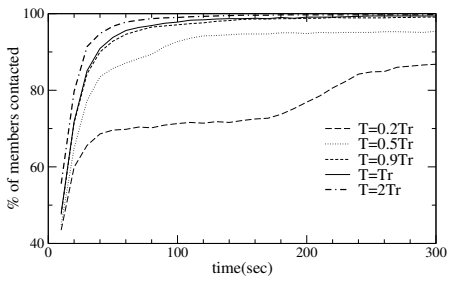


Fig. 11. Success for different values of T ($T_n = 10sec$)

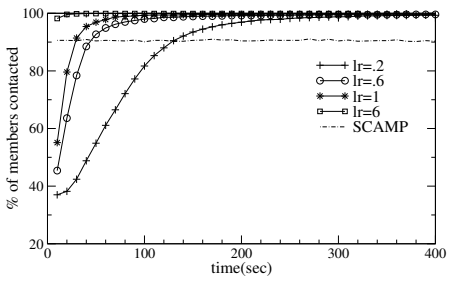


Fig. 12. Success over variable λ_r values ($T_n = 10sec$)

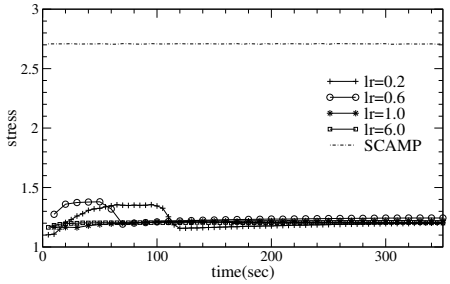


Fig. 13. Stress over variable λ_r values ($T_n = 10sec$)

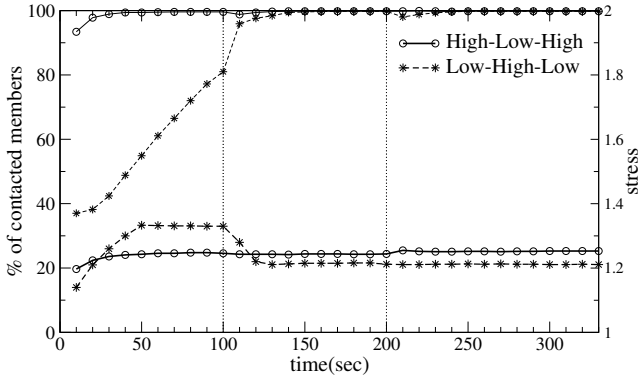


Fig. 14. Adaptation to a change in λ_r by a factor of 20

number of member departures combined with a large value for T would delay the adaptation to the new group size and cause more messages to be created than necessary.

In the next experiment, assuming a group size of 1,000 peers, we try to evaluate the performance of *AGNO* for different λ_r values. Figures 12 and 13 show the results. Not surprisingly, the bigger the value of λ_r, the faster the increase in the success rate, since indices get accurate faster. Another observation is that, regardless of the average request rate, our method asymptotically manages to contact all interested peers and reach a very low stress level (below 1.3). For most realistic scenarios ($T_n >> T_r$), the choice of T_n does not affect *AGNO*'s performance. In the very rare cases that $T_n < T_r$, we just set $T = O(T_n)$ to achieve comparable adaptation. In all cases, our adaptive aging mechanism selects a suitable value for ξ such that the stress remains almost static and below 1.4, half the value of the best of the remaining schemes (SCAMP). For small request rates, peers adapt using initially low and then higher quality bins (thus the slight variation in stress). The smaller the value of λ_r, the longer this adaptation takes.

Finally, Figure 14 shows how effective our adaptive λ_r estimation scheme is. We simulate the extreme case where the 1,000 requesters suddenly change their query rates by a factor of 20 (from $\lambda_r = 4/sec$ to $\lambda_r = 0.2/sec$ and vice versa). Our goal for the transition from high to low rate is to quickly decrease ξ so that our success rate is not affected. For the transition from low to high rate, we wish to quickly adjust the new ξ value according to the increased requests, such that no more than the necessary indices increase their value. We name our two runs high-low-high and low-high-low respectively: Starting with a rate of $\lambda_r = 4/sec$ (0.2/sec), requesters drop (increase) their average number of requests to 0.2/sec (4/sec) at time $t = 100sec$. At time $t = 200sec$, they increase (decrease) their rates to 4 queries/sec (0.2/sec). The top two lines correspond to success rates while the bottom two to the respective stress values. The maximum observed decrease in the success rates at 100 or 200 seconds is only 2%, while the stress values remain almost unaffected (increase equal to 0.01).

D. Changes in group sizes

We now evaluate the performance of *AGNO* under dynamic changes in the group size. Our goal is to allow for members to join or leave the group with the minimum amount of message exchange and performance degradation. Employing this approach that ties group membership to the interest (or lack thereof) of peers for objects, we require no coordination between members nor any single authority node.

Our protocol uses the aging scheme in order to

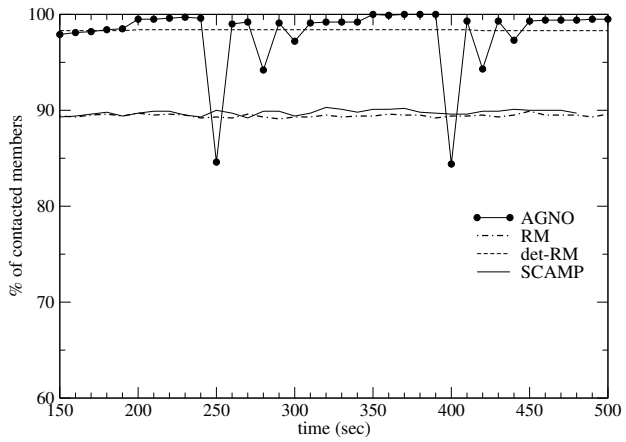


Fig. 16. Success rates after a series of member departures and arrivals ($\lambda_r = 0.5, T_n = 10$)

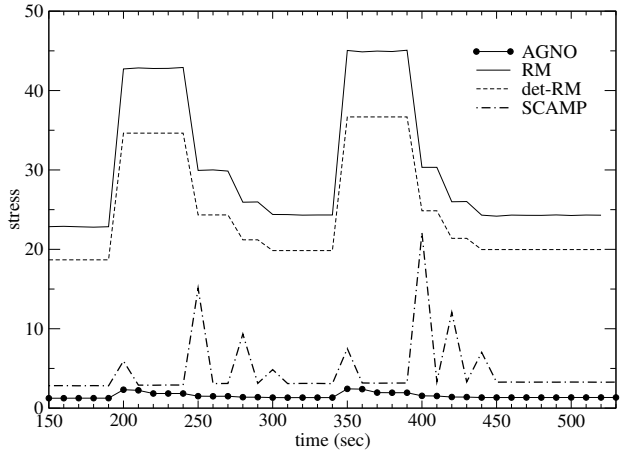


Fig. 17. Stress after a series of member departures and arrivals ($\lambda_r = 0.5, T_n = 10$)

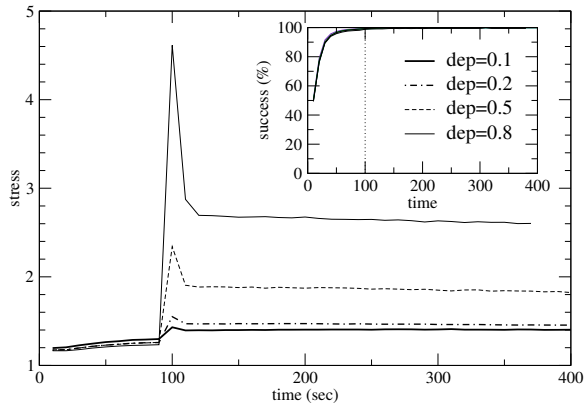


Fig. 15. Stress and success rates when a different ratio of peers depart at time t=100sec ($\lambda_r = 1, T_n = 10sec$)

identify and ultimately stop contacting disinterested peers, while it takes advantage of the cooperative nature of *APS* in order to quickly learn new members. Figure 15 shows how our two metrics are affected by having 10-80% of the 1,000 requesters leaving the group (stop making queries) at time $t = 100sec$. We take the worst-case scenario and assume that all these nodes jointly and instantly decide to leave from the group.

In all runs, the stress value peaks at the time of the departures, since the same number of peers are notified but fewer are now considered as members. The size of the departing sub-group directly affects the stress increase. The stress value instantly drops due to our aging mechanism, but it does not reach its previous value. This is due to the fact that a peer's indices get updated not only when it makes a request but also when any request passes through it. Therefore, while shortcuts for departing peers expire, indices leading to them may still have large values, depending on the relative positions of other requesters in the overlay. The amount of increase for $\{10\%, 20\%, 50\%$ and $80\%\}$ of the members departing is $\{7\%, 12\%, 38\%$ and $100\%\}$ respectively. The amount of increase decreases as the original group size gets smaller, which proves our previous point: Assuming 200 initial members instead, the respective stress increase percentiles are $\{7\%, 9\%, 16\%$ and $25\%\}$. On the other hand, as the included graph shows, our success rate is not affected at all. We show next that the decrease in stress after new members join compensates for the increase after peer departures.

Figures 16 and 17 display the performance of the compared methods under a combination of member joins and leaves. At times $t = \{200, 350\}sec$, 50% of the current group members decide to leave. At $t = \{250, 280, 300, 400, 420, 440\}sec$, 50% of the non-active requesters re-join the group. Members make requests at $\lambda_r = 0.5$, while the group notification phase is performed every 10 secs.

The success rate shows an instant decrease at the exact time of arrival which is proportional to the number of joining peers. Nevertheless, always more than 85% of the current members are contacted, and *AGNO* has learned of their presense by the exact next transmission. In the next push phases,

the method quickly reaches its previous levels. On the other hand, the value of stress is decreased after member joins and balances the small increase that occurs after member departures.
SCAMP and the two rumor spreading schemes show big variations in the stress metric. For RM and det-RM, this happens because of the change in the group size (same number of messages regardless of peer membership), while for SCAMP this is due to the subscription and unsubscription processes. *AGNO* contacts the vast majority of members at a cost 1 to 10 times lower than the closest compared method (SCAMP).

E. Real traces

We now present results from using real traces to our simulation environment. In our first experiment, we monitor the change in content for two very popular web sites, CNN and BBC news. We retrieve their home pages (*http://www.cnn.com* and *http://news.bbc.co.uk* respectively) at a minute granularity and record the time that their content has been modified. To determine that, we extract the official *Last Updated* string from the page and also directly compare the files [1]. Each page is preprocessed with *HTML Tidy*. Taking advantage of the fact that the overall structure of the same page rarely changes, we discard code, advertisements and pictures that change after each browser refresh, focusing on content. We monitor the changes over a period of 2 weeks, from Feb. 16th to Mar. 1st, 2004.

The CNN home page changes every 18.1 minutes on average, while BBC's news page every 8.6 minutes. In our experiments, we use the same 10,000-Node power-law graphs of the previous sections and a group size of 1,000 requesters, making requests with exponentially distributed interarrival times ($\lambda_r = 0.1/min$) for those two pages. The notification phases occur each time a page is updated, as given by our collected data. At exponentially distributed intervals (an average of 1 every 15 minutes), we choose with equal probability among the following events: 10% of the members stop requesting the pages; 80% of inactive members resume their requests; and nothing happens. On average, we vary our setup over 60 times per run. Figure 18 shows the results over the 14 1-day periods (averaged over all graphs with multiple runs for each). *AGNO* manages to exhibit very high accuracy and adapts its notification mechanism such that the stress value always remains stable between 1.6 and 1.7.

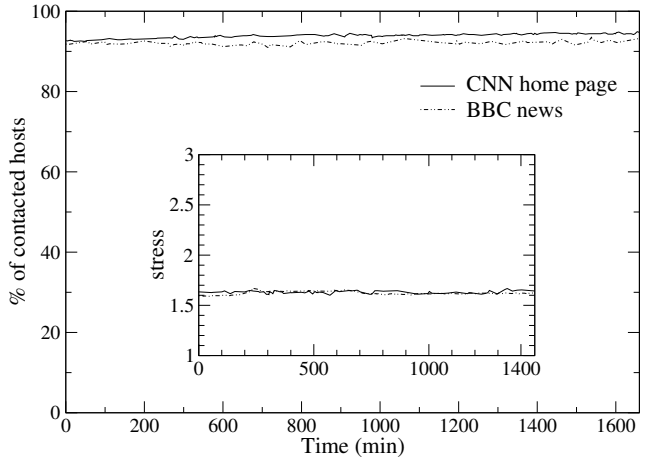


Fig. 18. Average results for one-day periods for the CNN and BBC news front pages

Finally, we test the behavior of our scheme in a much more dynamic environment. We use real traces taken from NYSE stock trades, which describe the accesses, volumes and values of all quotes in a 10-day period (Apr. 3-14, 2000). Aggregating to minute granularity, we monitor quote activity (accesses-updates) during a busy time interval (11:00-11:59am) each day. For our simulation, using the same power-law topologies as in the previous experiment, we assume a standard client population (group members) equal to the maximum number of accesses recorded at any minute per individual quote. We model our system such that, given there were Q accesses at a given minute, only the first Q clients are assumed to query for that object. This is equivalent to having a variable request rate for each member. Pushes were conducted whenever a quote's value was updated, with a maximum of one notification per minute.

Figure 19 shows the results for three of the most active quotes, SUNW (Sun Microsystems Inc.), MSFT (Microsoft Corp.) and ORCL (Oracle Corp.) The statistics for each of these quotes are presented in Table III. The interesting statistic here is the

[1]This method was developed as part of a project for the CS724 Database graduate course in University of Maryland

	Mean	Max	STD
SUNW	148	1037	118
MSFT	240	1171	184
ORCL	165	1137	101

TABLE III

ACCESS STATISTICS FOR THE THREE QUOTES

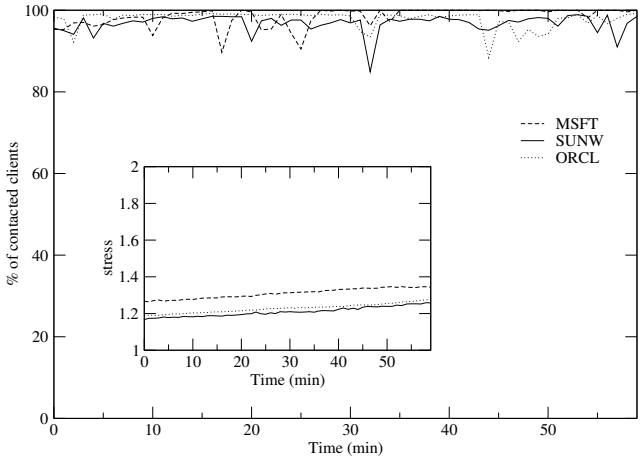


Fig. 19. Results for a 7-day period for the Microsoft, SUN and Oracle quotes between 11:00am and 11:59am

high standard deviation value for all three quotes, which translates to a wide range of different λ_i rates for each requester i in our experiments. Updates (=push transmissions) were performed once a minute. For all three datasets, *AGNO* achieves a high success rate with few small spike-shaped decreases occurring. A more detailed analysis of the data shows that these coincide with sudden increases (often more than 400%) in the group size (or accesses per minute), as were observed in the data. Given traces for more days, those spikes would have less weight on the averages. We also depict the average stress values for the quotes, which are kept at a very low level throughout the whole interval. These results also show that our adaptive forwarding and aging mechanisms work effectively even in the most dynamic environments. Results for less popular quotes or for time intervals outside high-access periods are qualitatively similar and were not selected since the average group size was less than 100.

V. CONCLUSIONS

In this paper we present *AGNO*, an adaptive and scalable group communication scheme for unstructured Peer-to-Peer networks. Our method integrates knowledge accumulated during searches to enable content-providers contact the large majority of interested peers with very small overhead. We described in detail our adaptive mechanisms to regulate message forwarding according to the quality of existing knowledge as well as to ensure efficient performance in all group operations. A variety of simulations using both synthetic and real traces showed that *AGNO* adapts quickly to variable request rates and group sizes, being at least twice as bandwidth-efficient as the compared methods.

VI. ACKNOWLEDGEMENTS

This material is based upon work supported by, or in part by, the U.S. Army Research Laboratory and the U.S. Army Research Office under contract/grant number DAAD19-01-1-0494

REFERENCES

[1] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the internet using an overlay muilticast architecture," in *SIGCOMM*, 2001.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *SIGCOMM*, 2002.

[3] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of NOSSDAV*, 2001.

[4] A. Rowstron, A. Kermarrec, M. Castro, and P. Druschel, "Scribe: The design of a large-scale event notification infrastructure," in *NGC*, 2001.

[5] J. Jannotti, D. Gifford, K. Johnson, F. Kaashoek, and J. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *OSDI*, 2000.

[6] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level multicast using content-addressable networks," *Lecture Notes in Computer Science*, 2001.

[7] "http://www.gnutella.com," Gnutella website.

[8] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti, "A Local Search Mechanism for Peer-to-Peer Networks," in *CIKM*, 2002.

[9] M. Portmann and A. Seneviratne, "Cost-effective broadcast for fully decentralized peer-to-peer networks," *Computer Communications*, vol. 26, 2003.

[10] A. Datta, M. Hauswirth, and K. Aberer, "Updates in highly unreliable, replicated peer-to-peer systems," in *ICDCS*, 2003.

[11] A. Ganesh, A. Kermarrec, and L. Massoulie, "Peer-to-peer membership management for gossip-based protocols," *IEEE Trans. Comp.*, 2003.

[12] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," in *PODC*, 2002.

[13] "http://web.icq.com/," ICQ web site.

[14] J. Kangasharju, K. Ross, and D. Turner, "Secure and Resilient Peer-to-Peer E-Mail: Design and Implementation," in *IEEE Intl Conf. on P2P Computing*, 2003.
[15] D. Tsoumakos and N. Roussopoulos, "A Framework for Sharing Voluminous Content in P2P Systems," in *PDPTA*, 2004.
[16] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *SIGMETRICS*, 2000.
[17] P. Francis, "Yoid: Extending the internet multicast architecture," 2000.
[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content Addressable Network," Tech. Rep. TR-00-010, University of Berkeley, CA, 2000.
[19] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, 2001.
[20] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, UC Berkeley, 2001.
[21] M. Castro, M. Jones, A. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *INFOCOM*, 2003.
[22] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," in *ICDCS*, 2002.
[23] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *PODC*, 1987.
[24] P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kouznetsov, "Lightweight probabilistic broadcast," in *DSN*, 2001.
[25] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks," in *IEEE Intl Conf. on P2P Computing*, 2003.
[26] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *RANDOM*, 2002.
[27] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *STOC*, 1996.
[28] M. Ripeanu and Ian Foster, "Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems," in *IPTPS*, 2002.
[29] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *MASCOTS*, 2001.
[30] C. Jin, Q. Chen, and S. Jamin, "Inet: Internet Topology Generator. Technical Report CSE-TR443-00, Department of EECS, University of Michigan," 2000.